Intel® Integrated Performance Primitives for Intel® Architecture

Reference Manual

Volume 2: Image and Video Processing

August 2008

Document Number: A70805-023US

World Wide Web: http://www.intel.com

Version	Version Information	Date
-1001	Documents Intel® Integrated Performance Primitives (Inte® IPP) release 1.0 beta.	07/2000
-1002	Documents Intel IPP 1.0 beta 2 . Alpha composition, color twist, gamma correction, and FFT/DFT/DCT functions have been added.	09/2000
-1003	Documents Intel IPP 1.0 final release. Includes new functionality: wavelet transforms, computer vision functions, and extended geometric transforms. New data initialization, arithmetic, and color conversion functions have also been added.	02/2001
-1101	Documents Intel IPP release 1.1 beta. JPEG codec functions have been added.	04/2001
-2001	Documents Intel IPP release 2.0 beta. Video processing functions for H.263+ and MPEG-4 decoders and wavelet transform functions for JPEG codec have been added.	08/2001
-2002	Documents Intel IPP release 2.0 gold. Library common functions have been added. New flavors of arithmetic, conversion, filtering, statistics, and JPEG codec functions are included.	11/2001
-3001	Documents Intel IPP 3.0 pre-beta. Function flavors to support compatibility layer with domain library have been added. JPEG200 codec functions have been extended. Video decoding functions have been added.	04/2002
-3002	Documents Intel IPP 3.0 beta release.	06/2002
-3003	Documents Intel IPP 3.0 beta update. Video encoding functions were added.	09/2002
-3004	Documents Intel IPP 3.0. The set of MPEG-4 video processing functions was extended.	11/2002
-4001	Documents Intel IPP 4.0 beta. New functions for cross-architecture development added.	05/2003
-4002	Documents Intel IPP 4.0 release.	10/2003
-013	Documents Intel IPP 4.1 beta release.	04/2004
-014	Documents Intel IPP 4.1 release. Added new color conversion functions and flavors for computer vision functions. Updated descriptions of H.264 video coding functions.	07/2004
-015	Documents Intel IPP 5.0 beta release. Added new image support and arithmetic functions, as well as morphological, filtering, and video coding functions. Considerably extended the set of color conversion, computer vision, and image transform functions.	04/2005

Version	Version Information	Date
-016	Documents Intel IPP 5.0 release. Added new data exchange, image color converion, filtering, image statistics, image geometric transform, and computer vision functions and function flavors.	08/2005
-017	Documents Intel IPP 5.1 beta release. Added advanced morphology, deconvolution, new computer vision, image arithmetic and geometric transform, and video coding functions, new FFT and DFT function flavors.	03/2006
-018	Documents Intel IPP 5.2 beta release. Added new image statistics, filtering, color conversion and image compression functions; new computer vision primitives for image inpainting and segmentation; new H.264, DV, and VC-1 decoder functions and function flavors.	10/2006
-019	Documents Intel IPP 5.2 release. Added new image data exchange, statistics, filtering, color conversion, geometry transforms, image compression, video coding (H.264) functions and flavors. Added new code examples for different functions.	01/2007
-020	Documents Intel IPP 5.3 beta release. Added new image data exchange, filtering, color conversion, image compression, computer vision, video coding (VC-1 encoder) functions and flavors. Added new code examples for different functions (filtering, computer vision, image compression, video coding, image transforming).	06/2007
-021	Documents Intel IPP 5.3 release. Added new color conversion, filtering, arithmetic, thresholding and deinterlacing functions. Added description of new AVS video codec.	09/2007
-022	Documents Intel IPP 6.0 beta release. Added new color conversion, computer vision, denoising and filtering functions. Added new functions for super-resolution algorithm.	02/2008
-023	Documents Intel IPP 6.0 release. Added new color conversion, image processing functions; new functions for video coding (general functions, H.264 and AVS codecs). Added texture compression functions.	08/2008

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

MPEG-1, MPEG-2, MPEG-4, H.261, H.263, H.264, MP3, DV, MJPEG, AC3, and AAC are international standards promoted by ISO, IEC, ITU, ETSI and other organizations. Implementations of these standards, or the standard enabled platforms may require licenses from various entities, including Intel Corporation.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Inside, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2000 - 2008, Intel Corporation. All rights reserved

Contents

Chapter 1 Overview

About This Software 1-1
Hardware and Software Requirements 1-2
Platforms Supported 1-2
Cross Architecture Alignment 1-2
API Changes in Version 5.0 1-2
Technical Support 1-3
Intel® IPP Code Samples 1-4
About This Manual 1-4
Manual Organization 1-4
Function Descriptions 1-5
Audience for This Manual 1-6
Online Version 1-6
Related Publications 1-6
Notational Conventions 1-6
Font Conventions 1-6
Naming Conventions 1-7

Chapter 2 Intel® Integrated Performance Primitives Concepts

Basic Features 2-1
Function Naming 2-2
Data-Domain 2-2
Name 2-3
Data Types 2-3
Descriptor 2-5

Parameters 2-6
Function Prototypes in Intel IPP 2-6
Integer Result Scaling 2-8
Error Reporting 2-9
Structures and Enumerators 2-14
Image Data Types and Ranges 2-20
Major Operation Models 2-22
Neighborhood Operations 2-22
Regions of Interest in Intel IPP 2-22
Tiled Image Processing 2-26

Chapter 3 Support Functions

Version Information Function 3-2
GetLibVersion 3-2
Status Information Function 3-4
ippGetStatusString 3-4
Memory Allocation Functions 3-5
Malloc 3-5
Free 3-6

Chapter 4 Image Data Exchange and Initialization Functions

Convert 4-2
Scale 4-6
Set 4-9
Copy 4-11
CopyManaged 4-15
CopyConstBorder 4-16
CopyReplicateBorder 4-19
CopyWrapBorder 4-22
CopySubpix 4-25
CopySubpixIntersect 4-27
Dup 4-31
Transpose 4-32
SwapChannels 4-34
AddRandUniform_Direct 4-37

AddRandGauss_Direct 4-39
ImageJaehne 4-40
ImageRamp 4-42
SampleLine 4-44
ZigzagFwd8x8 4-45
ZigzagInv8x8 4-47
Row Oriented Resampling 4-48
ResampleRowGetSize 4-49
ResampleRowInit 4-50
ResampleRowGetBorderWidth 4-51
ResampleRow, ResampleRowReplicateBorder 4-52

Chapter 5 Image Arithmetic and Logical Operations

Arithmetic Operations 5-4
Add 5-4
AddC 5-8
AddSquare 5-11
AddProduct 5-13
AddWeighted 5-14
Mul 5-16
MulC 5-18
MulScale 5-22
MulCScale 5-24
Sub 5-26
SubC 5-29
Div 5-32
Div_Round 5-35
DivC 5-37
Abs 5-40
AbsDiff 5-41
AbsDiffC 5-43
Sqr 5-44
Sqrt 5-46
Ln 5-48

Exp 5-51
Complement 5-53
DotProd 5-54
DotProdCol 5-56
Logical Operations 5-57
And 5-57
AndC 5-58
Not 5-61
Or 5-62
OrC 5-64
Xor 5-67
XorC 5-68
LShiftC 5-71
RShiftC 5-73
Alpha Composition 5-76
AlphaComp 5-77
AlphaCompC 5-79
AlphaPremul 5-82
AlphaPremulC 5-84

Chapter 6 Image Color Conversion

Gamma Correction 6-8
CIE Chromaticity Diagram and Color Gamut 6-8
Color Models 6-10
RGB Color Model 6-10
CMYK Color Model 6-12
YUV Color Model 6-12
YCbCr and YCCK Color Models 6-14
PhotoYCC Color Model 6-15
YCoCg Color Models 6-17
HSV, and HLS Color Models 6-18
CIE XYZ Color Model 6-21
CIE LUV and CIE Lab Color Models 6-22
Image Downsampling 6-25

RGB Image Formats 6-27
Pixel and Planar Image Formats 6-27
Color Model Conversion 6-31
RGBToYUV 6-31
YUVToRGB 6-34
RGBToYUV422 6-35
YUV422ToRGB 6-37
RGB565ToYUV422 6-38
RGBToYUV420 6-39
YUV420ToRGB 6-41
BGRToYUV420 6-43
YUV420ToBGR..... 6-44
RGB565ToYUV420 6-45
YUV420ToRGB565, YUV420ToRGB555, YUV420ToRGB444 6-46
YUV420ToRGB565Dither, YUV420ToRGB555Dither, YUV420ToRGB444Dither 6-48
BGR565ToYUV420, BGR555ToYUV420 6-49
YUV420ToBGR565, YUV420ToBGR555, YUV420ToBGR444 6-50
YUV420ToBGR565Dither, YUV420ToBGR555Dither YUV420ToBGR444Dither 6-51
RGBToYCbCr 6-53
YCbCrToRGB 6-54
YCbCrToBGR 6-56
YCbCrToRGB565, YCbCrToRGB555, YCbCrToRGB444..... 6-57
YCbCrToRGB565Dither, YCbCrToRGB555Dither, YCbCrToRGB444Dither 6-58
YCbCrToBGR565, YCbCrToBGR555, YCbCrToBGR444..... 6-60
YCbCrToBGR565Dither, YCbCrToBGR555Dither, YCbCrToBGR444Dither 6-61
RGBToYCbCr422 6-63
YCbCr422ToRGB 6-64
RGBToYCrCb422 6-65
YCrCb422ToRGB 6-66
BGRToYCbCr422 6-68

YCbCr422ToBGR 6-69
BGR565ToYCbCr422, BGR555ToYCbCr422 6-70
RGBToCbYCr422, RGBToCbYCr422Gamma 6-71
CbYCr422ToRGB 6-73
BGRToCbYCr422 6-74
BGRToCbYCr422_709HDTV 6-75
CbYCr422ToBGR 6-76
CbYCr422ToBGR_709HDTV 6-77
YCbCr422ToRGB565, YCbCr422ToRGB555, YCbCr422ToRGB444 .. 6-79
YCbCr422ToRGB565Dither, YCbCr422ToRGB555Dither,
YCbCr422ToRGB444Dither 6-80
YCbCr422ToBGR565, YCbCr422ToBGR555, YCbCr422ToBGR444 .. 6-82
YCbC422ToBGR565Dither, YCbC422ToBGR555Dither,
YCbCr422ToBGR444Dither 6-83
RGBToYCbCr420 6-85
YCbCr420ToRGB 6-86
YCbCr420ToRGB565, YCbCr420ToRGB555, YCbCr420ToRGB444 .. 6-87
YCbCr420ToRGB565Dither, YCbCr420ToRGB555Dither,
YCbCr420ToRGB444Dither 6-89
RGBToYCrCb420 6-90
YCrCb420ToRGB 6-91
BGRToYCbCr420 6-92
BGRToYCbCr420_709CSC 6-93
BGRToYCbCr420_709HDTV 6-95
BGRToYCrCb420_709CSC 6-96
YCbCr420ToBGR 6-97
YCbCr420ToBGR_709CSC 6-99
YCbCr420ToBGR_709HDTV 6-100
YCbCrToBGR_709CSC 6-101
BGR565ToYCbCr420, BGR555ToYCbCr420 6-102
YCbCr420ToBGR565, YCbCr420ToBGR555, YCbCr420ToBGR444 . 6-104
YCbCr420ToBGR565Dither, YCbCr420ToBGR555Dither,
YCbCr420ToBGR444Dither 6-105
BGRToYCrCb420 6-106

BGR565ToYCrCb420, BGR555ToYCrCb420 6-107
BGRToYCbCr411 6-108
YCbCr411ToBGR 6-109
BGR565ToYCbCr411, BGR555ToYCbCr411 6-111
YCbCr411ToBGR565, YCbCr411ToBGR555 6-112
RGBToXYZ 6-113
XYZToRGB 6-114
RGBToLUV 6-115
LUVToRGB 6-118
BGRToLab 6-120
LabToBGR 6-121
RGBToYCC 6-123
YCCToRGB 6-124
RGBToHLS 6-126
HLSToRGB 6-127
BGRToHLS 6-129
HLSToBGR 6-130
RGBToHSV 6-132
HSVToRGB 6-133
BGRToYCoCg 6-135
SBGRToYCoCg 6-136
YCoCgToBGR 6-137
YCoCgToSBGR 6-138
BGRToYCoCg_Rev 6-139
SBGRToYCoCg_Rev 6-140
YCoCgToBGR_Rev 6-141
YCoCgToSBGR_Rev 6-142
Color - Gray Scale Conversions 6-143
RGBToGray 6-143
ColorToGray 6-145
CFAToRGB 6-146
DemosaicAHD 6-149
Format Conversion 6-151
RGBToRGB565, BGRToBGR565 6-151

RGB565ToRGB, BGR565ToBGR 6-152
YCbCr422 6-153
YCbCr422ToYCrCb422 6-154
YCbCr422ToCbYCr422 6-155
YCbCr422ToYCbCr420 6-156
YCbCr422To420_Interlace 6-158
YCbCr422ToYCrCb420 6-160
YCbCr422ToYCbCr411 6-161
YCrCb422ToYCbCr422 6-162
YCrCb422ToYCbCr420 6-163
YCrCb422ToYCbCr411 6-164
CbYCr422ToYCbCr422 6-165
CbYCr422ToYCbCr420 6-166
CbYCr422ToYCbCr420_Interlace 6-167
CbYCr422ToYCrCb420 6-168
CbYCr422ToYCbCr411 6-169
YCbCr420 6-170
YCbCr420ToYCbCr422 6-172
YCbCr420ToYCbCr422_Filter 6-173
YCbCr420To422_Interlace 6-175
YCbCr420ToCbYCr422 6-176
YCbCr420ToCbYCr422_Interlace 6-177
YCbCr420ToYCrCb420 6-179
YCbCr420ToYCrCb420_Filter 6-180
YCbCr420ToYCbCr411 6-181
YCrCb420ToYCbCr422 6-183
YCrCb420ToYCbCr422_Filter 6-184
YCrCb420ToCbYCr422 6-185
YCrCb420ToYCbCr420 6-186
YCrCb420ToYCbCr411 6-187
YCbCr411 6-188
YCbCr411ToYCbCr422 6-189
YCbCr411ToYCrCb422 6-191
YCbCr411ToYCbCr420 6-192

YCbCr411ToYCrCb420 6-193
Color Twist 6-194
ColorTwist 6-195
ColorTwist32f 6-197
CompColorKey 6-199
AlphaCompColorKey 6-201
Gamma Correction 6-203
GammaFwd 6-203
GammaInv 6-206
Intensity Transformation 6-209
ReduceBits 6-209
LUT 6-211
LUT_Linear 6-215
LUT_Cubic 6-219
LUTPalette, LUTPaletteSwap 6-224
Tone MapLinear, ToneMapMean 6-227

Chapter 7 Threshold and Compare Operations

Thresholding 7-2
Threshold 7-2
Threshold_GT 7-5
Threshold_LT 7-7
Threshold_Val 7-9
Threshold_GTVal 7-11
Threshold_LTVal 7-13
Threshold_LTValGTVal 7-16
ComputeThreshold_Otsu 7-19
Compare Operations 7-21
Compare 7-21
CompareC 7-22
CompareEqualEps 7-25
CompareEqualEpsC 7-26

Chapter 8 Morphological Operation

Flat Structuring Elements for Grayscale Image 8-6

Dilate3x3 8-7
Erode3x3 8-9
Dilate 8-11
Erode 8-13
MorphologyInitAlloc 8-15
MorphologyFree 8-16
MorphologyInit 8-17
MorphologyGetSize 8-18
DilateBorderReplicate 8-19
ErodeBorderReplicate 8-21
MorphAdvInitAlloc 8-23
MorphAdvFree 8-24
MorphAdvInit 8-25
MorphAdvGetSize 8-26
MorphOpenBorder 8-27
MorphCloseBorder 8-29
MorphTophatBorder 8-30
MorphBlackhatBorder 8-32
MorphGradientBorder 8-33
MorphGrayInitAlloc 8-35
MorphGrayFree 8-36
MorphGrayInit 8-37
MorphGrayGetSize 8-38
GrayDilateBorder 8-39
GrayErodeBorder 8-40
MorphReconstructGetBufferSize 8-42
MorphReconstructDilate..... 8-43
MorphReconstructErode 8-46

Chapter 9 Filtering Functions

Borders..... 9-5
FilterBox 9-8
SumWindowRow 9-10
SumWindowColumn 9-11

FilterMin 9-12
FilterMax 9-14
FilterMinGetBufferSize 9-16
FilterMaxGetBufferSize 9-17
FilterMinBorderReplicate 9-18
FilterMaxBorderReplicate 9-19
FilterBilateralGetBufSize 9-21
FilterBilateralInit 9-22
FilterBilateral 9-24
DecimateFilterRow, DecimateFilterColumn 9-28
Median Filters 9-30
FilterMedian 9-31
FilterMedianHoriz 9-33
FilterMedianVert 9-34
FilterMedianCross 9-35
FilterMedianWeightedCenter3x3 9-37
FilterMedianColor 9-38
General Linear Filters 9-40
Filter 9-40
FilterGetBufSize 9-43
Filter32f 9-44
Filter_Round16s, Filter_Round32s, Filter_Round32f 9-45
FilterRoundGetBufSize16s, FilterRoundGetBufSize32s, FilterRoundGetBufSize32f 9-48
Separable Filters 9-51
FilterColumn 9-51
FilterColumn32f 9-54
FilterRow 9-55
FilterRow32f 9-57
FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize_Low 9-58
FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize_Low 9-60
FilterRowBorderPipeline, FilterRowBorderPipeline_Low 9-61

FilterColumnPipeline, FilterColumnPipeline_Low 9-64
DotProdCol 9-67
Wiener Filters 9-68
FilterWienerGetBufferSize 9-68
FilterWiener 9-69
Convolution 9-74
ConvFull 9-74
ConvValid 9-77
Deconvolution 9-81
DeconvFFTInitAlloc 9-81
DeconvFFTFree 9-82
DeconvFFT 9-83
DeconvLRInitAlloc 9-84
DeconvLRFree 9-85
DeconvLR 9-85
Fixed Filters 9-87
FilterPrewittHoriz 9-88
FilterPrewittVert 9-89
FilterScharrHoriz 9-91
FilterScharrVert 9-92
FilterSobelHoriz, FilterSobelHorizMask 9-94
FilterSobelVert, FilterSobelVertMask 9-95
FilterSobelHorizSecond 9-97
FilterSobelVertSecond 9-99
FilterSobelCross 9-101
FilterRobertsDown 9-103
FilterRobertsUp 9-104
FilterLaplace 9-105
FilterGauss 9-107
FilterHipass 9-110
FilterLowpass 9-112
FilterSharpen 9-113
Fixed Filters with Border 9-115
FilterScharrHorizGetBufferSize 9-115

FilterScharrVertGetBufferSize 9-116
FilterSobelHorizGetBufferSize 9-117
FilterSobelVertGetBufferSize, FilterSobelNegVertGetBufferSize . 9-118
FilterSobelHorizSecondGetBufferSize 9-119
FilterSobelVertSecondGetBufferSize 9-120
FilterSobelCrossGetBufferSize 9-121
FilterLaplacianGetBufferSize 9-122
FilterGaussGetBufferSize 9-123
FilterLowpassGetBufferSize 9-123
GenSobelKernel 9-124
FilterScharrHorizBorder 9-125
FilterScharrVertBorder 9-127
FilterSobelHorizBorder 9-128
FilterSobelVertBorder, FilterSobelNegVertBorder, ... 9-130
FilterSobelHorizSecondBorder 9-133
FilterSobelVertSecondBorder 9-135
FilterSobelCrossBorder 9-138
FilterLaplacianBorder 9-140
FilterGaussBorder 9-142
FilterLowpassBorder 9-143

Chapter 10 Image Linear Transforms

Fourier Transforms 10-4
Real - Complex Packed (RCPack2D) Format 10-5
FFTInitAlloc 10-6
FFTFree 10-7
FFTGetBufSize 10-8
FFTFwd 10-9
FFTInv 10-13
DFTInitAlloc 10-15
DFTFree 10-16
DFTGetBufSize 10-17
DFTFwd 10-18
DFTInv 10-21

MulPack 10-23
MulPackConj 10-27
Magnitude 10-29
MagnitudePack 10-30
Phase 10-31
PhasePack 10-32
PolarToCart 10-34
PackToCplxExtend 10-35
CplxExtendToPack 10-37
Windowing Functions 10-38
WinBartlett, WinBartlettSep 10-38
WinHamming, WinHammingSep 10-40
Discrete Cosine Transforms 10-43
DCTFwdInitAlloc 10-43
DCTInvInitAlloc 10-44
DCTFwdFree 10-45
DCTInvFree 10-46
DCTFwdGetBufSize 10-46
DCTInvGetBufSize 10-47
DCTFwd 10-48
DCTInv 10-50
DCT8x8Fwd 10-51
DCT8x8Inv, DCT8x8Inv_A10 10-53
DCT8x8FwdLS 10-55
DCT8x8InvLSClip 10-56
DCT8x8Inv_2x2, DCT8x8Inv_4x4 10-57
DCT8x8To2x2Inv, DCT8x8To4x4Inv 10-58

Chapter 11 Image Statistics Functions

Sum 11-5
Integral 11-7
SqrIntegral 11-9
TiltedIntegral 11-11
TiltedSqrIntegral 11-13

Mean 11-15
Mean_StdDev 11-18
RectStdDev 11-20
TiltedRectStdDev 11-22
HistogramRange 11-24
HistogramEven 11-27
CountInRange 11-29
Min 11-30
MinIndx 11-32
Max 11-34
MaxIndx 11-35
MinMax 11-37
MinMaxIndx 11-38
MaxEvery 11-40
MinEvery 11-42
FindPeaks3x3GetBufferSize 11-43
FindPeaks3x3 11-44
Image Moments 11-47
MomentInitAlloc 11-48
MomentFree 11-49
MomentGetStateSize 11-50
MomentInit 11-50
Moments 11-51
GetSpatialMoment 11-53
GetNormalizedSpatialMoment 11-54
GetCentralMoment 11-55
GetNormalizedCentralMoment 11-56
GetHuMoments 11-58
Image Norms 11-60
Norm_Inf 11-60
Norm_L1 11-62
Norm_L2 11-65
NormDiff_Inf 11-67
NormDiff_L1 11-70

NormDiff_L2 11-73
NormRel_Inf 11-76
NormRel_L1 11-78
NormRel_L2 11-81
Image Quality Index 11-85
QualityIndex 11-86
Image Proximity Measures 11-89
SqrDistanceFull_Norm 11-91
SqrDistanceSame_Norm 11-94
SqrDistanceValid_Norm 11-96
CrossCorrFull_Norm 11-98
CrossCorrSame_Norm 11-100
CrossCorrValid_Norm 11-103
CrossCorrValid 11-105
CrossCorrFull_NormLevel 11-106
CrossCorrSame_NormLevel 11-108
CrossCorrValid_NormLevel 11-111

Chapter 12 Image Geometry Transforms

ROI Processing in Geometric Transforms 12-4
Geometric Transform Functions 12-5
Resize 12-5
ResizeCenter 12-8
ResizeSqrPixel 12-12
Example of Using the Function ippiResizeSqrPixel 12-14
ResizeGetBufSize 12-17
ResizeSqrPixelGetBufSize 12-19
GetResizeFract 12-20
ResizeShift 12-21
ResizeYUV422 12-23
ResizeFilterGetSize 12-25
ResizeFilterInit 12-26
ResizeFilter 12-27
Mirror 12-28

Remap 12-30
Rotate 12-34
GetRotateShift 12-37
AddRotateShift 12-38
GetRotateQuad 12-39
GetRotateBound 12-40
RotateCenter 12-42
Shear 12-45
GetShearQuad 12-49
GetShearBound 12-50
WarpAffine 12-51
WarpAffineBack 12-54
WarpAffineQuad 12-57
GetAffineQuad 12-60
GetAffineBound 12-61
GetAffineTransform 12-62
WarpPerspective 12-64
WarpPerspectiveBack 12-67
WarpPerspectiveQuad 12-70
GetPerspectiveQuad 12-72
GetPerspectiveBound 12-74
GetPerspectiveTransform 12-75
WarpBilinear 12-77
WarpBilinearBack 12-80
WarpBilinearQuad 12-83
GetBilinearQuad 12-86
GetBilinearBound 12-87
GetBilinearTransform 12-88

Chapter 13 Wavelet Transforms

WTFwdInitAlloc 13-5
WTFwdFree 13-7
WTFwdGetBufSize 13-8
WTFwd 13-9

WTInvInitAlloc 13-13
WTInvFree 13-15
WTInvGetBufSize 13-16
WTInv 13-17

Chapter 14 Computer Vision

Using ippiAdd for Background Differencing 14-5
Feature Detection Functions 14-6
Corner Detection 14-7
Canny Edge Detector 14-7
Stage 1: Differentiation 14-7
Stage 2: Non-Maximum Suppression 14-8
Stage 3: Edge Thresholding 14-9
CannyGetSize 14-9
Canny 14-10
EigenValsVecsGetBufferSize 14-12
EigenValsVecs 14-13
MinEigenValGetBufferSize 14-16
MinEigenVal 14-17
Hough Transform 14-19
HoughLineGetSize 14-19
HoughLine 14-20
HoughLine_Region 14-21
Distance Transform Functions 14-24
DistanceTransform 14-24
GetDistanceTransformMask 14-27
TrueDistanceTransformGetBufferSize 14-29
TrueDistanceTransform 14-30
FastMarchingGetBufferSize 14-31
FastMarching 14-32
Image Gradients 14-34
GradientColorToGray 14-34
Flood Fill Functions 14-36
FloodFillGetSize 14-36

FloodFillGetSize_Grad 14-37
FloodFill 14-38
FloodFill_Grad 14-40
FloodFill_Range 14-43
Motion Analysis and Object Tracking 14-45
Motion Template Functions 14-45
Motion Representation 14-45
Updating MHI Images 14-46
UpdateMotionHistory 14-47
Optical Flow 14-48
OpticalFlowPyrLKInitAlloc 14-48
OpticalFlowPyrLKFree 14-49
OpticalFlowPyrLK 14-50
Pyramids Functions..... 14-54
PyrDownGetBufSize 14-55
PyrUpGetBufSize 14-56
PyrDown 14-57
PyrUp 14-58
Universal Pyramids 14-61
PyramidInitAlloc 14-61
PyramidFree 14-62
PyramidLayerDownInitAlloc 14-63
PyramidLayerDownFree 14-65
PyramidLayerUpInitAlloc 14-65
PyramidLayerUpFree 14-67
GetPyramidDownROI 14-68
GetPyramidUpROI 14-69
PyramidLayerDown 14-70
PyramidLayerUp 14-72
Example of Using General Pyramid Functions 14-74
Image Inpainting 14-76
InpaintInitAlloc 14-76
InpaintFree 14-78
Inpaint 14-78

Image Segmentation 14-81
LabelMarkersGetBufferSize 14-81
LabelMarkers 14-82
SegmentWatershedGetBufferSize 14-83
SegmentWatershed 14-84
SegmentGradientGetBufferSize 14-88
SegmentGradient 14-89
BoundSegments 14-91
ForegroundHistogramInitAlloc 14-92
ForegroundHistogramFree 14-94
ForegroundHistogram 14-95
ForegroundHistogramUpdate 14-97
ForegroundGaussianInitAlloc 14-98
ForegroundGaussianFree 14-100
ForegroundGaussian 14-101
Pattern Recognition 14-104
Object Detection Using Haar-like Features 14-104
HaarClassifierInitAlloc 14-106
TiltedHaarClassifierInitAlloc 14-107
HaarClassifierFree 14-108
GetHaarClassifierSize 14-109
TiltHaarFeatures 14-110
ApplyHaarClassifier 14-111
ApplyMixedHaarClassifier 14-113
Camera Calibration and 3D Reconstruction..... 14-116
Correction of Camera Lens Distortion 14-116
UndistortGetSize 14-116
UndistortRadial 14-117
CreateMapCameraUndistort 14-119
Image Enchancement Functions 14-121
Algorithm Overview 14-121
Point Spread Function 14-123
Error Functions 14-124
SRHNInitAlloc_PSF3x3, SRHNInitAlloc_PSF2x2 14-124

SRHNFree_PSF3x3, SRHNFree_PSF3x3 14-125
SRHNCalcResidual_PSF3x3, SRHNCalcResidual_PSF2x2 14-126
SRHNUpdateGradient_PSF3x3, SRHNUpdateGradient_PSF2x2 . 14-128

Chapter 15 Image Compression Functions

Support Functions 15-2
ippjGetLibVersion 15-2
Color Conversion Functions 15-3
RGBToY_JPEG 15-4
BGRToY_JPEG 15-6
RGBToYCbCr_JPEG 15-7
YCbCrToRGB_JPEG 15-8
RGB565ToYCbCr_JPEG, RGB555ToYCbCr_JPEG 15-9
YCbCrToRGB565_JPEG, YCbCrToRGB555_JPEG 15-10
BGRToYCbCr_JPEG 15-11
YCbCrToBGR_JPEG 15-12
BGR565ToYCbCr_JPEG, BGR555ToYCbCr_JPEG 15-13
YCbCrToBGR565_JPEG, YCbCrToBGR555_JPEG 15-14
YCbCr422ToRGB_JPEG 15-15
CMYKToYCCK_JPEG 15-16
YCCKToCMYK_JPEG 15-18
Combined Color Conversion Functions 15-20
RGBToYCbCr444LS_MCU 15-22
RGBToYCbCr422LS_MCU 15-23
RGBToYCbCr411LS_MCU 15-25
BGRToYCbCr444LS_MCU 15-26
BGR565ToYCbCr444LS_MCU, BGR555ToYCbCr444LS_MCU 15-27
BGRToYCbCr422LS_MCU 15-28
BGR565ToYCbCr422LS_MCU, BGR555ToYCbCr422LS_MCU 15-29
BGRToYCbCr411LS_MCU 15-30
BGR565ToYCbCr411LS_MCU, BGR555ToYCbCr411LS_MCU 15-31
CMYKToYCCK444LS_MCU 15-32
CMYKToYCCK422LS_MCU 15-33
CMYKToYCCK411LS_MCU 15-34

YCbCr444ToRGBLS_MCU 15-35
YCbCr422ToRGBLS_MCU 15-36
YCbCr411ToRGBLS_MCU 15-37
YCbCr444ToBGRLS_MCU 15-38
YCbCr444ToBGR565LS_MCU, YCbCr444ToBGR555LS_MCU 15-39
YCbCr422ToBGRLS_MCU 15-40
YCbCr422ToBGR565LS_MCU, YCbCr422ToBGR555LS_MCU 15-41
YCbCr411ToBGRLS_MCU 15-42
YCbCr411ToBGR565LS_MCU, YCbCr411ToBGR555LS_MCU 15-43
YCCK444ToCMYKLS_MCU 15-44
YCCK422ToCMYKLS_MCU 15-45
YCCK411ToCMYKLS_MCU 15-46
Quantization Functions 15-47
QuantFwdRawTableInit_JPEG 15-47
QuantFwdTableInit_JPEG 15-48
QuantFwd8x8_JPEG 15-49
QuantInvTableInit_JPEG 15-51
QuantInv8x8_JPEG 15-52
Combined DCT Functions 15-53
DCTQuantFwd8x8_JPEG 15-54
DCTQuantFwd8x8LS_JPEG 15-55
DCTQuantInv8x8_JPEG 15-57
DCTQuantInv8x8LS_JPEG 15-58
DCTQuantInv8x8LS_1x1_JPEG, DCTQuantInv8x8LS_2x2_JPEG, DCTQuantInv8x8LS_4x4_JPEG 15-59
DCTQuantInv8x8To4x4LS_JPEG, DCTQuantInv8x8To2x2LS_JPEG 15-60
Level Shift Functions 15-62
Sub128_JPEG 15-62
Add128_JPEG 15-63
Sampling Functions 15-64
SampleDownH2V1_JPEG 15-65
SampleDownH2V2_JPEG 15-66
SampleDownRowH2V1_Box_JPEG 15-67

SampleDownRowH2V2_Box_JPEG 15-68
SampleUpH2V1_JPEG 15-69
SampleUpH2V2_JPEG 15-71
SampleUpRowH2V1_Triangle_JPEG 15-72
SampleUpRowH2V2_Triangle_JPEG 15-73
SampleDown444LS_MCU 15-74
SampleDown422LS_MCU 15-75
SampleDown411LS_MCU 15-76
SampleUp444LS_MCU 15-77
SampleUp422LS_MCU 15-78
SampleUp411LS_MCU 15-79
Planar-to-Pixel and Pixel-to-Planar Conversion Functions 15-80
Split422LS_MCU 15-80
Join422LS_MCU 15-81
Huffman Codec Functions 15-82
EncodeHuffmanRawTableInit_JPEG 15-84
EncodeHuffmanSpecGetBufSize_JPEG 15-85
EncodeHuffmanSpecInit_JPEG 15-86
EncodeHuffmanSpecInitAlloc_JPEG 15-87
EncodeHuffmanSpecFree_JPEG 15-88
EncodeHuffmanStateGetBufSize_JPEG 15-88
EncodeHuffmanStateInit_JPEG 15-89
EncodeHuffmanStateInitAlloc_JPEG 15-90
EncodeHuffmanStateFree_JPEG 15-90
EncodeHuffman8x8_JPEG 15-91
EncodeHuffman8x8_Direct_JPEG 15-92
GetHuffmanStatistics8x8_JPEG 15-93
GetHuffmanStatistics8x8_DCFirst_JPEG 15-94
GetHuffmanStatistics8x8_ACFirst_JPEG 15-95
GetHuffmanStatistics8x8_ACRefine_JPEG 15-96
EncodeHuffman8x8_DCFirst_JPEG 15-97
EncodeHuffman8x8_DCRefine_JPEG 15-98
EncodeHuffman8x8_ACFirst_JPEG 15-99
EncodeHuffman8x8_ACRefine_JPEG 15-101

DecodeHuffmanSpecGetBufSize_JPEG 15-102
DecodeHuffmanSpecInit_JPEG 15-103
DecodeHuffmanSpecInitAlloc_JPEG 15-104
DecodeHuffmanSpecFree_JPEG 15-105
DecodeHuffmanStateGetBufSize_JPEG 15-105
DecodeHuffmanStateInit_JPEG 15-106
DecodeHuffmanStateInitAlloc_JPEG 15-107
DecodeHuffmanStateFree_JPEG 15-107
DecodeHuffman8x8_JPEG 15-108
DecodeHuffman8x8_Direct_JPEG 15-109
DecodeHuffman8x8_DCFirst_JPEG 15-111
DecodeHuffman8x8_DCRefine_JPEG 15-112
DecodeHuffman8x8_ACFirst_JPEG 15-113
DecodeHuffman8x8_ACRefine_JPEG 15-114
Functions for Lossless JPEG Coding 15-116
DiffPredFirstRow_JPEG 15-116
DiffPredRow_JPEG 15-118
ReconstructPredFirstRow_JPEG 15-119
ReconstructPredRow_JPEG 15-120
GetHuffmanStatisticsOne_JPEG 15-121
EncodeHuffmanOne_JPEG 15-121
DecodeHuffmanOne_JPEG 15-123
EncodeHuffmanRow_JPEG 15-124
DecodeHuffmanRow_JPEG 15-125
Wavelet Transform Functions..... 15-127
Low-Level Operations..... 15-128
WTFwdRow_B53_JPEG2K 15-129
WTInvRow_B53_JPEG2K 15-131
WTFwdCol_B53_JPEG2K 15-132
WTFwdColLift_B53_JPEG2K 15-134
WTInvCol_B53_JPEG2K 15-135
WTInvColLift_B53_JPEG2K 15-136
WTFwdRow_D97_JPEG2K 15-138
WTInvRow_D97_JPEG2K 15-139

WTFwdCol_D97_JPEG2K 15-141
WTFwdColLift_D97_JPEG2K 15-142
WTInvCol_D97_JPEG2K 15-144
WTInvColLift_D97_JPEG2K 15-145
Tile-Oriented Transforms 15-146
WTGetBufSize_B53_JPEG2K 15-147
WTFwd_B53_JPEG2K 15-148
WTInv_B53_JPEG2K 15-150
WTGetBufSize_D97_JPEG2K 15-151
WTFwd_D97_JPEG2K 15-152
WTInv_D97_JPEG2K 15-154
JPEG2000 Entropy Coding and Decoding Functions 15-155
EncodeInitAlloc_JPEG2K 15-157
EncodeFree_JPEG2K 15-157
EncodeLoadCodeBlock_JPEG2K 15-158
EncodeStoreBits_JPEG2K 15-161
EncodeGetTermPassLen_JPEG2K 15-163
EncodeGetRate_JPEG2K 15-164
EncodeGetDist_JPEG2K 15-165
DecodeGetBufSize_JPEG2K 15-166
DecodeCodeBlock_JPEG2K 15-166
DecodeCBProgrGetStateSize_JPEG2K 15-168
DecodeCBProgrInit_JPEG2K 15-169
DecodeCBProgrInitAlloc_JPEG2K 15-169
DecodeCBProgrFree_JPEG2K 15-170
DecodeCBProgrAttach_JPEG2K 15-171
DecodeCBProgrSetPassCounter_JPEG2K 15-172
DecodeCBProgrGetPassCounter_JPEG2K 15-173
DecodeCBProgrGetCurBitPlane_JPEG2K 15-174
DecodeCBProgrStep_JPEG2K 15-174
Component Transform Functions..... 15-176
RCTFwd_JPEG2K 15-176
RCTInv_JPEG2K 15-178
ICTFwd_JPEG2K 15-180

ICTInv_JPEG2K 15-182
Functions for RLE Coding 15-183
RLE Coding Model 15-183
PackBitsRow_TIFF 15-184
UnpackBitsRow_TIFF 15-185
SplitRow_TIFF 15-186
JoinRow_TIFF 15-187
Texture Compression Functions 15-188
EncodeTextureBlockFromRGBA 15-189
DecodeTextureBlockToRGBA 15-190

Chapter 16 Video Coding

General Functions 16-2
Structures and Enumerators 16-3
Variable Length Decoding 16-13
HuffmanTableInitAlloc 16-20
HuffmanRunLevelTableInitAlloc 16-21
DecodeHuffmanOne 16-22
DecodeHuffmanPair 16-23
HuffmanTableFree 16-24
Motion Compensation 16-25
MC16x16 16-26
MC16x8 16-27
MC8x16 16-28
MC8x8 16-29
MC8x4 16-30
MC4x8 16-31
MC4x4 16-32
MC2x4 16-33
MC4x2 16-34
MC2x2 16-35
MC16x4 16-36
MC16x8UV 16-37
MC16x16B 16-39

MC16x8B 16-40
MC8x16B 16-41
MC8x8B 16-42
MC8x4B 16-43
MC4x8B 16-44
MC4x4B 16-46
MC2x4B 16-47
MC4x2B 16-48
MC2x2B 16-49
MC16x4B 16-50
MC16x8UVB 16-52
Copy8x8,
Copy16x16 16-53
Copy8x4HP, Copy8x8HP, Copy16x8HP, Copy16x16HP 16-54
InterpolateAverage8x4, InterpolateAverage8x8,
InterpolateAverage16x8, InterpolateAverage16x16 16-55
Add8x8 16-56
Add8x8HP 16-57
AddC8x8 16-58
Average8x8, Average16x16 16-59
Motion Estimation..... 16-60
Difference Evaluation 16-62
GetDiff16x16 16-63
GetDiff16x8 16-64
GetDiff8x8 16-65
GetDiff8x16 16-66
GetDiff8x4 16-68
GetDiff4x4 16-69
GetDiff16x16B 16-70
GetDiff16x8B 16-71
GetDiff8x8B 16-73
GetDiff8x16B 16-74
GetDiff8x4B 16-75
Sub4x4, Sub8x8, Sub16x16 16-76

SubSAD8x8 16-78
Sum of Squares of Differences Evaluation 16-79
SqrDiff16x16 16-79
SqrDiff16x16B 16-80
SSD8x8 16-81
SSD4x4 16-82
Block Variance and Mean Evaluation 16-83
VarMean8x8 16-83
Evaluation of Variances and Means of Blocks of Difference Between Two Blocks 16-85
VarMeanDiff16x16 16-85
VarMeanDiff16x8 16-86
Block Variance Evaluation 16-88
Variance16x16 16-88
Evaluation of Block Deviation 16-89
MeanAbsDev8x8 16-89
MeanAbsDev16x16 16-90
Edges Detection 16-91
EdgesDetect16x16 16-91
SAD Functions 16-92
SAD16x16 16-92
SAD16x8 16-93
SAD8x16 16-94
SAD8x8 16-95
SAD8x4 16-96
SAD4x8 16-97
SAD4x4 16-98
SAD16x16Blocks8x8 16-99
SAD16x16Blocks4x4 16-100
SATD16x16 16-101
SATD16x8 16-103
SATD8x16 16-104
SATD8x8 16-105
SATD8x4 16-106

SATD4x8 16-108
SATD4x4 16-109
SAT8x8D 16-110
FrameFieldSAD16x16 16-112
Sum of Differences Evaluation 16-113
SumsDiff16x16Blocks4x4 16-113
SumsDiff8x8Blocks4x4 16-115
Scanning Functions 16-117
ScanInv 16-117
ScanFwd 16-118
Color Conversion 16-119
CbYCr422ToYCbCr420_Rotate 16-119
ResizeCCRotate 16-122
Video Processing 16-124
Deinterlacing Functions 16-125
DeinterlaceFilterTriangle 16-125
DeinterlaceFilterCAVT 16-126
Median 16-127
DeinterlaceMedianThreshold 16-128
DeinterlaceEdgeDetect 16-129
DeinterlaceMotionAdaptive 16-130
DeinterlaceBlendInitAlloc 16-132
DeinterlaceBlendFree 16-133
DeinterlaceBlend 16-134
Denoising Functions 16-136
FilterDenoiseCASTInit 16-136
FilterDenoiseCAST 16-137
FilterDenoiseSmooth 16-139
FilterDenoiseAdaptiveInitAlloc 16-140
FilterDenoiseAdaptiveFree 16-141
FilterDenoiseAdaptive 16-141
FilterDenoiseMosquitoInitAlloc 16-144
FilterDenoiseMosquitoFree 16-145
FilterDenoiseMosquito 16-145

MPEG-1 and MPEG-2 16-147
Structures and Enumerations 16-147
Video Data Decoding 16-152
Variable Length Decoding 16-156
ReconstructDCTBlock_MPEG1 16-157
ReconstructDCTBlockIntra_MPEG1 16-158
ReconstructDCTBlock_MPEG2 16-160
ReconstructDCTBlockIntra_MPEG2 16-163
Inverse Quantization 16-165
QuantInvIntra_MPEG2 16-165
QuantInv_MPEG2 16-166
Inverse Discrete Cosine Transformation 16-166
DCT8x8Inv_AANTransposed_16s_C1R 16-166
DCT8x8Inv_AANTransposed_16s8u_C1R 16-167
DCT8x8Inv_AANTransposed_16s_P2C2R 16-168
DCT8x8Inv_AANTransposed_16s8u_P2C2R 16-169
Motion Compensation 16-171
Video Data Encoding 16-172
Motion Estimation 16-173
Quantization 16-175
QuantIntra_MPEG2 16-176
Quant_MPEG2 16-177
Huffman Encoding Functions 16-178
CreateRLEncodeTable 16-178
PutIntraBlock 16-179
PutNonIntraBlock 16-180
DV 16-182
DV Decoding Functions 16-189
Variable Length Decoding 16-190
InitAllocHuffmanTable_DV 16-192
HuffmanDecodeSegment_DV 16-196
HuffmanDecodeSegment_DV100 16-201
HuffmanDecodeSegmentOnePass_DV 16-203
FreeHuffmanTable_DV 16-204

Inverse Quantization 16-204
QuantInv_DV 16-204
QuantWeightBlockInv_DV 16-205
QuantWeightBlockInv_DV100 16-211
Inverse Discrete Cosine Transformation 16-215
DCT2x4x8Inv 16-215
DCT8x4x2To4x4Inv_DV 16-217
DV Encoding Functions 16-220
Discrete Cosine Transformation 16-221
DCT2x4x8Frw 16-221
CountZeros8x8 16-222
DV Color Conversion Functions 16-223
YCrCb411ToYCbCr422_5MBDV,
YCrCb411ToYCbCr422_ZoomOut2_5MBDV,
YCrCb411ToYCbCr422_ZoomOut4_5MBDV,
YCrCb411ToYCbCr422_ZoomOut8_5MBDV, ... 16-224
YCrCb411ToYCbCr422_16x4x5MB_DV
YCrCb411ToYCbCr422_8x8MB_DV 16-225
YCrCb411ToYCbCr422_EdgeDV,
YCrCb411ToYCbCr422_ZoomOut2_EdgeDV,
YCrCb411ToYCbCr422_ZoomOut4_EdgeDV,
YCrCb411ToYCbCr422_ZoomOut8_EdgeDV 16-227
YCrCb420ToYCbCr422_5MBDV,
YCrCb420ToYCbCr422_ZoomOut2_5MBDV,
YCrCb420ToYCbCr422_ZoomOut4_5MBDV,
YCrCb420ToYCbCr422_ZoomOut8_5MBDV 16-228
YCrCb420ToYCbCr422_8x8x5MB_DV 16-229
YCrCb422ToYCbCr422_5MBDV,
YCrCb422ToYCbCr422_ZoomOut2_5MBDV,
YCrCb422ToYCbCr422_ZoomOut4_5MBDV,
YCrCb422ToYCbCr422_ZoomOut8_5MBDV 16-230
YCrCb422ToYCbCr422_8x4x5MB_DV 16-232
YCrCb422ToYCbCr422_10HalvesMB16x8_DV100 16-233
MPEG-4 16-236
MPEG-4 Video Decoder Functions 16-239
High Level Description 16-239

Data Types and Structures 16-241
Motion Compensation 16-242
Copy8x8QP_MPEG4, Copy16x8QP_MPEG4, Copy16x16QP_MPEG4 16-242
OBMC8x8HP_MPEG4, OBMC16x16HP_MPEG4, OBMC8x8QP_MPEG4, ... 16-243
Sprite and Global Motion Compensation 16-245
WarpInit_MPEG4 16-245
WarpGetSize_MPEG4 16-246
WarpLuma_MPEG4 16-247
WarpChroma_MPEG4 16-248
CalcGlobalMV_MPEG4 16-249
ChangeSpriteBrightness_MPEG4 16-250
Inverse Quantization 16-251
QuantInvIntraInit_MPEG4, QuantInvInterInit_MPEG4 16-251
QuantInvIntraGetSize_MPEG4, QuantInvInterGetSize_MPEG4 16-252
QuantInvIntra_MPEG4, QuantInvInter_MPEG4 16-253
VLC Decoding 16-254
DecodeDCIntra_MPEG4 16-254
DecodeCoeffsIntra_MPEG4 16-255
DecodeCoeffsIntraRVLCBack_MPEG4 16-257
DecodeCoeffsInter_MPEG4 16-258
DecodeCoeffsInterRVLCBack_MPEG4 16-259
ReconstructCoeffsInter_MPEG4 16-260
Postprocessing 16-261
FilterDeblocking8x8HorEdge_MPEG4, FilterDeblocking8x8VerEdge_MPEG4 16-261
FilterDeringingThreshold_MPEG4 16-262
FilterDeringingSmooth8x8_MPEG4 16-263
MPEG-4 Video Encoder Functions 16-264
Data Types and Structures 16-265
Quantization 16-266

QuantIntraInit_MPEG4, QuantInterInit_MPEG4 16-266
QuantIntraGetSize_MPEG4, QuantInterGetSize_MPEG4 16-267
QuantIntra_MPEG4, QuantInter_MPEG4 16-267
VLC Encoding 16-269
EncodeDCIntra_MPEG4 16-269
EncodeCoeffsIntra_MPEG4 16-270
EncodeCoeffsInter_MPEG4 16-271
H.261 16-273
H.261 Decoder Functions 16-274
Decoding INTRA and INTER Macroblocks 16-275
DecodeCoeffsIntra_H261 16-277
DecodeCoeffsInter_H261 16-278
ReconstructCoeffsIntra_H261 16-279
ReconstructCoeffsInter_H261 16-281
H.261 Encoder Functions 16-283
EncodeCoeffsIntra_H261 16-284
EncodeCoeffsInter_H261 16-285
Filter8x8_H261 16-286
H.263 16-288
H.263 Decoder Functions 16-291
INTRA and INTER Macroblocks Decoding 16-292
VLC Decoding 16-294
DecodeDCIntra_H263 16-294
DecodeCoeffsIntra_H263 16-295
DecodeCoeffsInter_H263 16-296
Inverse Quantization 16-298
QuantInvIntra_H263 16-298
QuantInvInter_H263 16-300
Prediction 16-301
AddBackPredPB_H263 16-301
Resampling 16-303
Resample_H263 16-303
UpsampleFour_H263 16-305
DownsampleFour_H263 16-306

UpsampleFour8x8_H263 16-307
SpatialInterpolation_H263 16-307
Boundary Filtering 16-309
FilterBlockBoundaryHorEdge_H263, FilterBlockBoundaryVerEdge_H263 16-309
FilterDeblocking8x8HorEdge_H263, FilterDeblocking8x8VerEdge_H263 16-310
FilterDeblocking16x16HorEdge_H263, FilterDeblocking16x16VerEdge_H263 16-312
Middle Level Functions 16-313
ReconstructCoeffsIntra_H263 16-313
ReconstructCoeffsInter_H263 16-315
H.263 Encoder Functions 16-317
VLC Encoding 16-318
EncodeDCIntra_H263 16-318
EncodeCoeffsIntra_H263 16-319
EncodeCoeffsInter_H263 16-320
Quantization 16-322
QuantIntra_H263 16-322
QuantInter_H263 16-323
Resampling 16-324
DownsampleFour16x16_H263 16-324
H.264 16-325
H.264 Decoder Functions 16-336
CAVLC Parsing 16-341
DecodeCAVLCCoeffs_H264 16-341
DecodeCAVLCChromaDcCoeffs_H264, DecodeCAVLCChroma422DcCoeffs_H264 16-343
DecodeExpGolombOne_H264 16-345
Inverse Quantization and Inverse Transform 16-346
TransformDequantLumaDC_H264 16-346
TransformDequantChromaDC_H264 16-347
TransformResidual4x4Inv_H264 16-348
DequantTransformResidual_H264 16-349

DequantTransformResidualAndAdd_H264 16-350
TransformPrediction_H264 16-352
DequantTransformResidual_SISP_H264 16-353
TransformDequantChromaDC_SISP_H264 16-354
Intra Prediction 16-355
PredictIntra_4x4_H264 16-355
PredictIntra_16x16_H264 16-359
PredictIntraChroma8x8_H264 16-360
Inter Prediction 16-362
ExpandPlane_H264 16-366
InterpolateLuma_H264 16-367
InterpolateLumaTop_H264 16-371
InterpolateLumaBottom_H264 16-374
InterpolateLumaBlock_H264 16-377
InterpolateChroma_H264 16-379
InterpolateChromaTop_H264 16-384
InterpolateChromaBottom_H264 16-386
InterpolateChromaBlock_H264 16-388
InterpolateBlock_H264, Bidir_H264 16-390
WeightedAverage_H264 16-392
UniDirWeightBlock_H264, UnidirWeight_H264 16-393
BiDirWeightBlock_H264, BidirWeight_H264 16-395
BiDirWeightBlockImplicit_H264, BidirWeightImplicit_H264 . 16-397
Macroblock Reconstruction..... 16-398
ReconstructChromaInterMB_H264 16-400
ReconstructChromaIntraHalvesMB_H264 16-401
ReconstructChromaIntraMB_H264 16-403
ReconstructChromaInter4x4MB_H264,
ReconstructChromaInter4x4_H264High 16-405
ReconstructChroma422Inter4x4_H264High 16-409
ReconstructChromaIntraHalves4x4MB_H264,
ReconstructChromaIntraHalf4x4_H264High 16-411
ReconstructChromaIntra4x4MB_H264,
ReconstructChromaIntra4x4_H264High 16-413

ReconstructChroma422IntraHalf4x4_H264High 16-416
ReconstructChroma422Intra4x4_H264High 16-418
ReconstructLumaInterMB_H264 16-421
ReconstructLumaIntraHalfMB_H264 16-422
ReconstructLumaIntraMB_H264 16-423
ReconstructLumaInter4x4MB_H264,
ReconstructLumaInter4x4_H264High 16-425
ReconstructLumaIntraHalf4x4MB_H264,
ReconstructLumaIntraHalf4x4_H264High 16-428
ReconstructLumaIntra4x4MB_H264,
ReconstructLumaIntra4x4_H264High 16-430
ReconstructLumaInter8x8MB_H264,
ReconstructLumaInter8x8_H264High 16-432
ReconstructLumaIntraHalf8x8MB_H264,
ReconstructLumaIntraHalf8x8_H264High 16-433
ReconstructLumaIntra8x8MB_H264,
ReconstructLumaIntra8x8_H264High 16-435
ReconstructLumaIntra16x16MB_H264 16-437
ReconstructLumaIntra_16x16MB_H264,
ReconstructLumaIntra16x16_H264High 16-438
Deblocking Filtering 16-440
FilterDeblockingLuma_VerEdge_H264 16-441
FilterDeblockingLuma_VerEdge_MBAFF_H264 16-443
FilterDeblockingLuma_HorEdge_H264 16-445
FilterDeblockingChroma_VerEdge_H264,
FilterDeblockingChroma422VerEdge_H264 16-447
FilterDeblockingChroma_VerEdge_MBAFF_H264,
FilterDeblockingChroma422VerEdgeMBAFF_H264 16-451
FilterDeblockingChroma_HorEdge_H264,
FilterDeblockingChroma422HorEdge_H264 16-453
H.264 Encoder Functions 16-458
Forward Transform and Quantization 16-459
TransformQuantChromaDC_H264 16-459
TransformQuantLumaDC_H264 16-461
TransformResidual4x4Fwd_H264 16-462

QuantizeResidual4x4Fwd_H264 16-463
TransformQuantResidual_H264 16-465
TransformLuma8x8Fwd_H264 16-466
QuantLuma8x8_H264 16-467
GenScaleLevel8x8_H264 16-468
CAVLC Functions 16-469
EncodeCoeffsCAVLC_H264 16-469
EncodeChromaDcCoeffsCAVLC_H264 16-472
Inverse Quantization and Transform 16-473
QuantLuma8x8Inv_H264 16-473
TransformLuma8x8InvAddPred_H264 16-474
AVS 16-475
AVS Decoder Functions 16-477
Entropy Decoding 16-479
DecodeLumaBlockIntra_AVS 16-479
DecodeLumaBlockInter_AVS 16-480
DecodeChromaBlock_AVS 16-481
Inter Prediction 16-482
InterpolateLumaBlock_AVS 16-482
WeightPrediction_AVS 16-484
Macroblock Reconstruction 16-485
ReconstructLumaIntra_AVS 16-485
ReconstructLumaInter_AVS 16-486
ReconstructChromaIntra_AVS 16-488
ReconstructChromaInter_AVS 16-489
Deblocking Filtering 16-490
FilterDeblockingLuma_VerEdge_AVS 16-490
FilterDeblockingLuma_HorEdge_AVS 16-492
FilterDeblockingChroma_VerEdge_AVS 16-494
FilterDeblockingChroma_HorEdge_AVS 16-496
AVS Encoder Functions 16-498
Forward Transform and Quantization 16-499
TransformQuant8x8Fwd_AVS 16-499
Macroblock Disassembling 16-500

DisassembleLumaIntra_AVS 16-500
DisassembleChroma420Intra_AVS 16-502
VC-1 16-504
VC-1 Decoder 16-508
Inverse Transform 16-510
Transform8x8Inv_VC1, Transform4x8Inv_VC1, Transform8x4Inv_VC1, Transform4x4Inv_VC1 16-510
Interpolation 16-520
InterpolateQPBilinear_VC1 16-520
InterpolateQPBicubic_VC1 16-522
Smoothing 16-523
SmoothingLuma_VerEdge_VC1 16-523
SmoothingLuma_HorEdge_VC1 16-526
SmoothingChroma_VerEdge_VC1 16-528
SmoothingChroma_HorEdge_VC1 16-530
Deblocking 16-532
FilterDeblockingLuma_VerEdge_VC1 16-532
FilterDeblockingLuma_HorEdge_VC1 16-533
FilterDeblockingChroma_VerEdge_VC1 16-534
FilterDeblockingChroma_HorEdge_VC1 16-535
Dequantization 16-536
QuantInvIntraUniform_VC1, QuantInvIntraNonuniform_VC1 16-536
QuantInvInterUniform_VC1, QuantInvInterNonuniform_VC1 16-540
Range Reduction 16-544
RangeMapping_VC1 16-544
VC-1 Encoder 16-546
Forward Transform 16-547
Transform8x8Fwd_VC1, Transform4x8Fwd_VC1, Transform8x4Fwd_VC1, Transform4x4Fwd_VC1 16-547

Quantization 16-555
QuantIntraUniform_VC1, QuantIntraNonuniform_VC1 16-555
QuantInterUniform_VC1, QuantInterNonuniform_VC1 16-557

Appendix A Handling of Special Cases

Appendix B Interpolation in Image Geometric Transform Functions

Overview of Interpolation Modes B-1
Mathematical Notation B-3
Nearest Neighbor Interpolation B-3
Linear Interpolation B-3
Cubic Interpolation B-4
Super Sampling B-6
Lanczos Interpolation B-7
Interpolation with Two-Parameter Cubic Filters B-8

Appendix C Removed Functions

Bibliography

Glossary

Index

Overview

1

This manual describes the structure, operation, and functions of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® architecture that operate on two-dimensional signals and are used for image and video processing. This is the second volume of the Intel IPP Reference Manual, which also comprises descriptions of Intel IPP for signal processing (volume 1), operations on small matrices (volume 3), and cryptography functions (volume 4).
The Intel IPP software package supports many functions whose performance can be significantly enhanced on Intel architecture, particularly using the MMX™ technology, Streaming SIMD Extensions (SSE), Streaming SIMD Extensions 2 (SSE2), Streaming SIMD Extensions 3 (SSE3), as well as Intel® Itanium® architecture.

The Intel IPP for image and video processing software is a collection of low-overhead, high-performance operations performed on two-dimensional (2D) arrays of pixels.

This manual explains the concepts of Intel IPP as well as specific data type definitions and operation models used in the image and video processing domain and provides detailed descriptions of the Intel IPP image and video processing functions.

This chapter introduces the Intel IPP software and explains the organization of this manual.

About This Software

The Intel IPP software enables taking advantage of the parallelism of the single-instruction, multiple data (SIMD) instructions that comprise the core of the MMX technology and Streaming SIMD Extensions. These technologies improve the performance of computation-intensive signal, image, and video processing applications. Use of the Intel IPP primitive functions can help to drastically reduce development costs and accelerate time-to-market by eliminating the need of writing processor-specific code for computation intensive routines.

Hardware and Software Requirements

The Intel IPP for Intel architecture software runs on personal computers that are based on processors using IA-32, Intel® 64 or IA-64 architecture and running Microsoft Windows* OS, Linux* OS, or Apple Mac OS* X. Intel IPP can be integrated into the customer's application or library written in C or C++.

Platforms Supported

The Intel IPP for Intel architecture software runs on Windows* OS, Linux* OS, and Mac OS* X platforms. The code and syntax used in this manual for function and variable declarations are written in the ANSI C style. However, versions of Intel IPP for different processors or operating systems may, of necessity, vary slightly.

Cross Architecture Alignment

Intel IPP has been designed to support application development on various Intel® architectures.

By providing a single cross-architecture application programming interface (API), Intel IPP allows software application repurposing and enables developers to port to unique features across Intel® processor-based desktop, server, and mobile platforms. Developers can write their code once in order to realize the application performance over many processor generations.

API Changes in Version 5.0

Starting from the version 5.0 Intel IPP has a limited compatibility with the previous versions. To improve the library in several aspects, the following changes have been made:

- Some functions have been replaced by new functions with extended functionality.
- Some functions have been changed to make API more consistent and handy.
- Odd and unusable functions have been discarded.
- Several complicated functions have been discarded from Intel IPP, but their functionalities have been moved to the codec/sample level.

All these changes affect existent applications. Table C-1 in the Appendix C lists such functions for the image processing domains and specifies the corresponding Intel IPP 5.0 functions to replace them. If an application calls functions listed in this table, then the source code should be modified.

Several functions, for example, all color conversion functions, have been moved to the newly created functional domains. These changes do not affect existent applications.

Additionally, certain modifications breaking binary compatibility have been made in Intel IPP 5.0, for example, directory tree structure has been modified, the `ipp20` folder has been discarded. These changes also affect existent applications that should be at least rebuilt.

Version 5.1, as well as the subsequent versions, has full backward compatibility with version 5.0.

Technical Support

Intel IPP provides a product web site that offers timely and comprehensive product information, including product features, white papers, and technical articles. For the latest information, see: http://developer.intel.com/software/products/.

Intel also provides a support web site that contains a rich repository of self-help information, including getting started tips, known product issues, product errata, license information, and more (visit http://support.intel.com/support/).

Registering your product entitles you to one-year technical support and product updates through Intel® Premier Support. Intel Premier Support is an interactive issue management and communication web site providing the following services:

- Submit issues and review their status.
- Download product updates anytime of the day.

To register your product, or contact Intel, or seek product support, please visit: http://www.intel.com/software/products/support

Intel® IPP Code Samples

An extensive library of code samples and codecs has been implemented using the Intel IPP functions to demonstrate the use of Intel IPP and to help accelerate the development of your application, components, and codecs. The samples can be downloaded from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

About This Manual

This manual provides a background for the image and video processing concepts used in the Intel IPP software as well as detailed description of the respective Intel IPP functions. The Intel IPP functions are combined in groups by their functionality. Each group of functions is described in a separate chapter (chapters 3 through 16).

Manual Organization

This manual contains the following chapters:

Chapter 1	"Overview" (this chapter). Introduces the Intel IPP software for image and video processing, provides information on manual organization, and explains notational conventions.
Chapter 2	"Intel® Integrated Performance Primitives Concepts". Explains the basic concepts underlying image processing in Intel IPP and describes the supported data formats and operation modes.
Chapter 3	"Support Functions". Describes functions that are used to support the operation of the Intel IPP software, such as memory allocation functions and status information function.
Chapter 4	"Image Data Exchange and Initialization Functions". Describes image processing primitive functions that are used to set, copy, convert, and scale images, as well as initialize specific images.
Chapter 5	"Image Arithmetic and Logical Operations". Describes Intel IPP image processing functions that modify pixel values using arithmetic, logical, and alpha composition operations.
Chapter 6	"Image Color Conversion". Describes Intel IPP color space conversion operations.
Chapter 7	"Threshold and Compare Operations". Describes functions that perform thresholding and image comparison operations.

Chapter 8	"Morphological Operation". Describes common and advanced morphological operations as well as operations with gray kernel and morphological reconstruction.
Chapter 9	"Filtering Functions". Describes Intel IPP filtering operations using linear and non-linear filters.
Chapter 10	"Image Linear Transforms". Describes the Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), and Discrete Cosine Transform (DCT) functions implemented in Intel IPP for image processing.
Chapter 11	"Image Statistics Functions". Describes Intel IPP functions that compute image statistical properties such as image norms and moments.
Chapter 12	"Image Geometry Transforms". Describes the supported geometric transformations of images.
Chapter 13	"Wavelet Transforms". Describes the supported functions for wavelet decomposition and reconstruction of images.
Chapter 14	"Computer Vision". Describes Intel IPP functions that are specific for computer vision applications.
Chapter 15	"Image Compression Functions". Describes Intel IPP functions that perform still image compression and coding in accordance with JPEG and JPEG2000 standards.
Chapter 16	"Video Coding". Describes Intel IPP functions that support encoding and decoding of video data according to MPEG-1, MPEG-2 and MPEG-4 standards, as well as functions that support H.261, H.263, H.264, DV, and VC-1 specifications.

The manual also includes a Glossary of terms, a Bibliography, and an Index, as well as Appendix A, "Handling of Special Cases" that describes handling of special cases by Intel IPP functions, Appendix B, "Interpolation in Image Geometric Transform Functions" that describes the interpolation algorithms used in the geometric transformation functions of the Intel IPP, and Appendix C, "Removed Functions" that lists functions removed from Intel IPP version 5.0.

Function Descriptions

In Chapters 3 through 16, each function is introduced by its short name (without the `ippi` prefix and descriptors) and a brief description of its purpose. This is followed by the function call sequence, definitions of all function arguments, and a more detailed explanation of the function purpose. The following sections are included in the function descriptions:

Syntax	Lists function prototypes.

Parameters	Describes all function parameters.
Description	Defines the function and details the operation performed by the function. Code examples and equations that the function implements may be included in the description.
Return Values	Describes values indicating status codes set as the result of the function execution.

Audience for This Manual

The manual is intended for the developers of image and video processing applications and libraries, as well as cross-domain applications. The audience must have experience in using C and working knowledge of the vocabulary and principles of image and video processing.

Online Version

This manual is available in an electronic format (Portable Document Format, or PDF). To obtain a hard copy of the manual, print the file using the printing capability of Adobe Acrobat*, the tool used for the online presentation of the document.

Related Publications

For more information about image and video processing concepts and algorithms, refer to the books and materials listed in the Bibliography.

Notational Conventions

In this manual, notational conventions include:

- Fonts used for distinction between the text and the code
- Naming conventions for different items.

Font Conventions

The following font conventions are used in this manual:

`THIS TYPE STYLE`	Used in the text for Intel IPP constant identifiers; for example, `IPPI_INTER_LINEAR`.
`This type style`	Mixed with the uppercase in structure names as in `IppiSize`; also used in function names, code examples and call statements; for example, `ippiMomentInitAlloc()`.

This type style	Parameters in function prototypes and parameters description; for example: *value, srcStep*.

Naming Conventions

The following naming conventions for different items are used by the Intel IPP software:

- Constant identifiers are in uppercase; for example, `IPPI_INTER_CUBIC`.
- All structures and enumerators, specific for the image and video processing domain have the `Ippi` prefix, while those common for entire Intel IPP software have the `Ipp` prefix; for example, `IppiPoint, IppDitherType`.
- All names of the functions used for image and video processing have the `ippi` prefix. In code examples, you can distinguish the Intel IPP interface functions from the application functions by this prefix.

NOTE. In this manual, the *ippi* prefix in function names is always used in code examples and function prototypes. In the text, this prefix is usually omitted when referring to the function group.

- Each new part of a function name starts with an uppercase character, without underscore; for example, `ippiGetSpatialMoment`.

For the detailed description of function name structure in Intel IPP, see Function Naming in Chapter 2.

Intel® Integrated Performance Primitives Concepts

2

This chapter explains the purpose and structure of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® Architecture software and looks over some of the basic concepts used in the image and video processing part of Intel IPP. It also describes the supported data formats and operation modes, and defines function naming conventions in the manual.

Basic Features

The Intel Integrated Performance Primitives, like other members of the Intel® Performance Libraries, is a collection of high-performance code that performs domain-specific operations. It is distinguished by providing a low-level, stateless interface.

Based on experience in developing and using Intel Performance Libraries, Intel IPP has the following major distinctive features:

- The Intel IPP provides basic low-level functions for creating applications in several different domains, such as signal processing, image and video processing, and operations on small matrices;
- Intel IPP functions follow the same interface conventions including uniform naming rules and similar composition of prototypes for primitives that refer to different application domains;
- Intel IPP functions use abstraction level which is best suited to achieve superior performance figures by the application programs.

To speed up performance, Intel IPP functions are optimized to use all benefits of Intel® architecture processors. Besides that, most of Intel IPP functions do not use complicated data structures, which helps reduce overall execution overhead.

The Intel IPP software work with two-dimensional arrays of data, not an image abstraction, thus providing lower-level image processing functions than in most other image processing libraries. This serves the following major goals:

- To free developers from the necessity of filling specific structures that describe image data and architecture;
- To achieve a maximum function performance by avoiding the structures' field parsing and code branching;
- To provide a collection of simple and obvious lower-level functions to facilitate incorporation within an existing application or architecture.

Intel IPP is well-suited for cross-platform applications. For example, the functions developed for IA-32 platform can be readily ported to Intel® Itanium®-based platforms.

For more information on platform compatibility, see Cross Architecture Alignment section in chapter 1.

Function Naming

Naming conventions for the Intel IPP functions are similar for all covered domains. You can distinguish signal processing functions by the `ipps` prefix, while image and video processing functions have `ippi` prefix, and functions that are specific for operations on small matrices have `ippm` prefix in their names.

Function names in Intel IPP have the following general format:

```
ipp<data-domain><name>_<datatype>[_<descriptor>](<parameters>);
```

The elements of this format are explained in the sections that follow.

Data-Domain

The *data-domain* is a single character that expresses the subset of functionality to which a given function belongs. The current version of Intel IPP supports the following data-domains:

`s`	for signals (expected data type is a 1D signal);
`i`	for images and video (expected data type is a 2D image);
`m`	for matrices (expected data type is a matrix).
`r`	for realistic rendering functionality and 3D data processing (expected data type depends on supported rendering techniques).

`g` for signals of the fixed length.

For example, function names that begin with `ippi` signify that respective functions are used for image or video processing.

Name

The *name* is an abbreviation for the core operation that the function really does, for example `Add`, `Sqrt`, followed in some cases by a function-specific modifier:

<name> = <operation>[*_modifier*]

This modifier, if present, denotes a slight modification or variation of the given function.

Data Types

The *datatype* field indicates data types used by the function, in the following format:

```
<bit depth><bit interpretation> ,
```

where

bit depth = <1|8|16|32|64>

and

bit interpretation = <*u*|*s*|*f*>[*c*]

Here *u* indicates “unsigned integer”, *s* indicates “signed integer”, *f* indicates “floating point”, and *c* indicates “complex”.

The Intel IPP supports the following data types for image and video processing functions:

- `8u` 8-bit, unsigned data
- `8s` 8-bit, signed data
- `16u` 16-bit, unsigned data
- `16uc` 16-bit, complex unsigned short data †
- `16s` 16-bit, signed data
- `16sc` 16-bit, complex short data
- `32u` 32-bit, unsigned data
- `32s` 32-bit, signed data
- `32sc` 32-bit, complex int data

- `32f` 32-bit, single-precision real floating point data
- `32fc` 32-bit, single-precision complex floating point data
- `64s` 64-bit, quadword signed data
- `64f` 64-bit, double-precision real floating point data

† - only partial support for intermediate result after transforms (in so called “time” domain).

NOTE. Intel IPP does not support `1u` data type, therefore bitonal images should be converted to `8u` gray scale images for further processing by the library functions. There is a special function for such conversion `ippiConvert_1u8u_C1R`..

The formats for complex data are represented in Intel IPP by structures defined as follows:

```
typedef struct {
    Ipp16s re;
    Ipp16s im;
} Ipp16sc;
typedef struct {
    Ipp32s re;
    Ipp32s im;
} Ipp32sc;
typedef struct {
    Ipp32f re;
    Ipp32f im;
} Ipp32fc;
```

where `re,im` denote the real and imaginary part, respectively.

Complex data formats are used by several arithmetic image processing functions. The `32fc` format is also used to store input/output data in some Fourier transform functions. The 64-bit formats, `64s` and `64f`, are used for storing data computed by some image statistics functions.

For functions that operate on a single data type, the *`datatype`* field contains only one of the values listed above.

If a function operates on source and destination images that have different data types, the respective data type identifiers are listed in the function name in order of source and destination as follows:

`<datatype> = <src1Depth>[src2Depth] [dstDepth]`

For example, the function that converts 8-bit unsigned source image data to 32-bit floating point destination image data has the `8u32f` value for the *datatype* field.

NOTE. In the lists of function parameters (arguments), the `Ipp` prefix is written in the data type. For example, the 8-bit unsigned data is denoted as `Ipp8u` type. These Intel IPP -specific data types are defined in the respective library header files.

Descriptor

The *descriptor* field further describes the data associated with the operation. It may contain implied parameters and/or indicate additional required parameters.

To minimize the number of code branches in the function and thus reduce potentially unnecessary execution overhead, most of the general functions are split into separate primitive functions, with some of their parameters entering the primitive function name as descriptors.

However, where the number of permutations of the function becomes large and unreasonable, some functions may still have parameters that determine internal operation (for example, `ippiThreshold`).

The following descriptors are used in image and video processing functions:

- `A` Data contains an alpha channel as the last channel, requires `C4`, alpha channel is not processed
- `A0` Data contains an alpha channel as the first channel, requires `C4`, alpha channel is not processed
- `Cn` Data is made up of `n` discrete interleaved channels (1, 2, 3, 4)
- `C` Channel of interest (COI) is used in the operation
- `D2` Image is two-dimensional
- `I` Operation is in-place

- `M` Uses mask ROI for source and destination images
- `Pn` Data is made up of `n` discrete planar (non-interleaved) channels, with a separate pointer to each plane
- `R` Uses region of interest (ROI)
- `Sfs` Saturation and fixed scaling mode is used

The abbreviations of descriptors in function names are always presented in alphabetical order.

Not all functions have every abbreviation listed above. For example, in-place mode makes no sense for a copy operation.

Also, some data descriptors are implied when dealing with some operations.
The default for image processing functions is to operate on a two-dimensional image and to saturate the results without scaling them. In these cases, the implied abbreviations `D2` and `Sns` (saturation and no scaling) are not contained in the function name.

Parameters

The parameters in functions are in the following order: all source operands; all destination operands; all other, operation-specific parameters.

Source parameters are named "*Src*" or "*SrcN*," if there is more than one input image. Destination parameters are named "*Dst*". For in-place operations, the input/output parameter contains the name "*SrcDst*." All parameters defined as pointers start with lowercase *p*, for example, *pSrc*, *pMean*, *pSpec*.

Function Prototypes in Intel IPP

Function names in Intel IPP contain *datatype* and *descriptor* fields after the *name* field (see Function Naming in this chapter). Most Intel IPP functions for image processing have a number of flavors that differ in data types associated with the operation, and in some additional parameters.

Each function flavor has its unique prototype used in function definition and for calling the function from the application program. For many flavors of a given function, these prototypes look quite similar.

To avoid listing all the similar prototypes in function description sections of some chapters in this manual, only different templates for such prototypes followed by the table of applicable data types and descriptors for each function may be given. For simplicity, in such cases the data type and descriptor fields in the function name are denoted as *mod*:

```
<mod> = <datatype>_<descriptor>
```

For example, the template for the prototype of the image dilation function that performs not-in-place operation, looks like this:

```
IppStatus ippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
      Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

where the supported values for *mod* are:

```
8u_C1R
8u_C3R
8u_AC4R
```

This notation means that the `ippiDilate` function has three flavors for a not-in-place operation, which process 8-bit unsigned data (of `Ipp8u` type) and differ in the number of channels in processed images. These flavors have the following prototypes:

```
IppStatus ippiDilate_8u_C1R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate_8u_C3R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate_8u_AC4R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Thus, to obtain the full name and parameters list for the specific function flavor, not listed directly, do the following:

- choose the function operation mode (denoted in this manual as **Case 1**, **2**,...) and look in the table for the supported data types and descriptors;
- set the *mod* field in the function name as the concatenation of chosen data type and descriptor, delimited by the underscore;
- use the respective template, substituting all the *datatype* fields in the parameters list with the chosen data type. Note that `Ipp` prefix is written before the *datatype* in the parameters list (see Data Types in this chapter for details).

Example.
To get the prototype for the ippiSet function flavor that sets each channel of a 3-channel destination image to 16-bit signed values, choose **Case 2: Setting each color channel to a specified value** and use *datatype* = 16s, *descriptor* = C3R.
After substituting the *mod* field with 16s_C3R, obtain the required prototype as

```
ippiSet_16s_C3R(const Ipp16s value[3], Ipp16s* pDst, int dstStep,
    IppiSize roiSize);
```

Integer Result Scaling

Some image processing functions operating on integer data use scaling of the internally computed output results by the integer *scaleFactor*, which is specified as one of the function parameters. These functions have the Sfs descriptor in their names.

The scale factor can be negative, positive, or zero. Scaling is applied because internal computations are generally performed with a higher precision than the data types used for input and output images.

NOTE. The result of integer operations is always saturated to the destination data type range even when scaling is used.

The scaling of an integer result is done by multiplying the output pixel values by $2^{-scaleFactor}$ before the function returns. This helps retain either the output data range or its precision. Usually the scaling with a positive factor is performed by the shift operation. The result is rounded off to the nearest integer number.
For example, the integer Ipp16s result of the square operation ippiSqr for the input value 200 is equal to 32767 instead of 40000, that is, the result is saturated and the exact value can not be restored. The scaling of the output value with the factor *scaleFactor* = 1 yields the result 20000 which is not saturated, and the exact value can be restored as 20000*2. Thus, the output data range is retained.
The following example shows how the precision can be partially retained by means of scaling.
The integer square root operation ippiSqrt (without scaling) for the input value 2 gives the result equal to 1 instead of 1.414. Scaling of the internally computed output value with the factor *scaleFactor* = -3 gives the result 11, and permits to restore the more precise value as $11*2^{-3} = 1.375$.

Error Reporting

Intel IPP functions return status codes of the performed operation to report errors and warnings to the calling program. Thus, it is up to the application to perform error-related actions and/or recover from the error. The last value of the error status is not stored, and the user is to decide whether to check it or not as the function returns.
The status codes are of `IppStatus` type and are global constant integers.

The status codes and corresponding messages reported by the Intel IPP for image and video processing are listed in Table 2-1.

Table 2-1 Status Codes and Messages

Status Code	Message
`ippStsNotSupportedModeErr`	The requested mode is currently not supported
`ippStsDecimateFractionErr`	Unsupported fraction in decimate
`ippStsWeightErr`	Wrong value of weight
`ippStsQualityIndexErr`	Quality Index can't be calculated for image filled with constant
`ippStsResizeNoOperationErr`	One of the output image dimensions is less than 1 pixel
`ippStsBlockStepErr`	Step for Block less than 8
`ippStsMBStepErr`	Step for MB less than 16
`ippStsNoiseRangeErr`	Noise value for Wiener Filter is out of range
`ippStsLPCCalcErr`	Linear prediction could not be evaluated
`ippStsJPEG2KBadPassNumber`	Pass number exceeds allowed limits [0,nOfPasses-1]"
`ippStsJPEG2KDamagedCodeBlock`	Codeblock for decoding is damaged
`ippStsH263CBPYCodeErr`	Illegal Huffman code during CBPY stream processing
`ippStsH263MCBPCInterCodeErr`	Illegal Huffman code while MCBPC Inter stream processing
`ippStsH263MCBPCIntraCodeErr`	Illegal Huffman code while MCBPC Intra stream processing
`ippStsNotEvenStepErr`	Step value is not pixel multiple
`ippStsHistoNofLevelsErr`	Number of levels for histogram is less than 2
`ippStsLUTNofLevelsErr`	Number of levels for LUT is less than 2
`ippStsMP4BitOffsetErr`	Incorrect bit offset value
`ippStsMP4QPErr`	Incorrect quantization parameter
`ippStsMP4BlockIdxErr`	Incorrect block index
`ippStsMP4BlockTypeErr`	Incorrect block type

Table 2-1 Status Codes and Messages (continued)

ippStsMP4MVCodeErr	Illegal Huffman code while MV stream processing
ippStsMP4VLCCodeErr	Illegal Huffman code while VLC stream processing
ippStsMP4DCCodeErr	Illegal code while DC stream processing
ippStsMP4FcodeErr	Incorrect fcode value
ippStsMP4AlignErr	Incorrect buffer alignment
ippStsMP4TempDiffErr	Incorrect temporal difference
ippStsMP4BlockSizeErr	Incorrect size of block or macroblock
ippStsMP4ZeroBABErr	All BAB values are zero
ippStsMP4PredDirErr	Incorrect prediction direction
ippStsMP4BitsPerPixelErr	Incorrect number of bits per pixel
ippStsMP4VideoCompModeErr	Incorrect video component mode
ippStsMP4LinearModeErr	Incorrect DC linear mode
ippStsH263PredModeErr	Prediction Mode value error
ippStsH263BlockStepErr	The step value is less than 8
ippStsH263MBStepErr	The step value is less than 16
ippStsH263FrameWidthErr	The frame width is less than 8
ippStsH263FrameHeightErr	The frame height is less than or equal to zero
ippStsH263ExpandPelsErr	The expand pixels number is less than 8
ippStsH263PlaneStepErr	Step value is less than the plane width
ippStsH263QuantErr	Quantizer value is less than or equal to zero, or greater than 31
ippStsH263MVCodeErr	Illegal Huffman code while MV stream processing
ippStsH263VLCCodeErr	Illegal Huffman code while VLC stream processing
ippStsH263DCCodeErr	Illegal code while DC stream processing
ippStsH263ZigzagLenErr	Zigzag compact length is more than 64
ippStsJPEGHuffTableErr	JPEG Huffman table is destroyed
ippStsJPEGDCTRangeErr	JPEG DCT coefficient is out of the range
ippStsJPEGOutOfBufErr	Attempt to access out of the buffer
ippStsChannelOrderErr	Wrong order of the destination channels
ippStsZeroMaskValueErr	All values of the mask are zero
ippStsRangeErr	Bad values of bounds: the lower bound is greater than the upper bound
ippStsQPErr	Incorrect value of quantizer parameter

Table 2-1 Status Codes and Messages (continued)

ippStsQuadErr	The quadrangle is nonconvex or degenerates into triangle, line or point
ippStsRectErr	Size of the rectangular region is less than or equal to 1
ippStsCoeffErr	Unallowable values of the transformation coefficients
ippStsNoiseValErr	Bad value of noise amplitude for dithering
ippStsDitherLevelsErr	Number of dithering levels is out of range
ippStsNumChannelsErr	Bad or unsupported number of channels
ippStsDataTypeErr	Bad or unsupported data type
ippStsCOIErr	COI is out of range
ippStsOutOfRangeErr	Argument is out of range or point is outside the image
ippStsDivisorErr	Divisor is equal to zero, function is aborted
ippStsAlphaTypeErr	Illegal type of image compositing operation
ippStsGammaRangeErr	Gamma range bound is less than or equal to zero
ippStsGrayCoefSumErr	Sum of the conversion coefficients must be less than or equal to 1
ippStsChannelErr	Illegal channel number
ippStsJaehneErr	Magnitude value is negative
ippStsStepErr	Step value is less than or equal to zero
ippStsStrideErr	Stride value is less than the row length
ippStsEpsValErr	Negative epsilon value error
ippStsScaleRangeErr	Scale bounds are out of the range
ippStsThresholdErr	Invalid threshold bounds
ippStsWtOffsetErr	Invalid offset value of wavelet filter
ippStsAnchorErr	Anchor point is outside the mask
ippStsMaskSizeErr	Invalid mask size
ippStsShiftErr	Shift value is less than zero
ippStsSampleFactorErr	Sampling factor is less than or equal to zero
ippStsResizeFactorErr	Resize factor(s) is less or equal to zero
ippStsDivByZeroErr	An attempt to divide by zero
ippStsInterpolationErr	Invalid interpolation mode
ippStsMirrorFlipErr	Invalid flip mode
ippStsMoment00ZeroErr	Moment value M(0,0) is too small to continue calculations

Table 2-1 Status Codes and Messages (continued)

ippStsThreshNegLevelErr	Negative value of the level in the threshold operation
ippStsContextMatchErr	Context parameter doesn't match the operation
ippStsFftFlagErr	Invalid value of the FFT flag parameter
ippStsFftOrderErr	Invalid value of the FFT order parameter
ippStsMemAllocErr	Not enough memory for the operation
ippStsNullPtrErr	Null pointer error
ippStsSizeErr	Wrong value of data size
ippStsBadArgErr	Invalid or bad argument
ippStsNoErr	No error, it's OK
ippStsNoOperation	No operation has been executed
ippStsMisalignedBuf	Misaligned pointer in operation in which it must be aligned
ippStsSqrtNegArg	Negative value(s) of the argument in the function Sqrt
ippStsInvZero	INF result. Zero value was met by InvThresh with zero level
ippStsEvenMedianMaskSize	Even size of the Median Filter mask was replaced by the odd number
ippStsDivByZero	Zero value(s) of the divisor in the function Div
ippStsLnZeroArg	Zero value(s) of the argument in the function Ln
ippStsLnNegArg	Negative value(s) of the argument in the function Ln
ippStsNanArg	Not a Number argument value warning
ippStsDoubleSize	Sizes of image are not multiples of 2
ippStsJPEGMarker	JPEG marker was met in the bitstream
ippStsResFloor	All result values are floored
ippStsAffineQuadChanged	4th vertex of destination quad is not equal to customer's one
ippStsWrongIntersectROI	Wrong ROI that has no intersection with the source or destination image. No operation
ippStsWrongIntersectQuad	Wrong quadrangle that has no intersection with the source or destination ROI. No operation
ippStsSymKernelExpected	The kernel is not symmetric
ippStsEvenMedianWeight	Even weight of the Weighted Median Filter was replaced by the odd one

The status codes ending with `Err` (except for the `ippStsNoErr` status) indicate an error; the integer values of these codes are negative. When an error occurs, the function execution is interrupted.

The status code `ippStsNoErr` indicates no error.

All other status codes indicate warnings. When a specific case is encountered, the function execution will be completed and the corresponding warning status code will be returned. For example, if the function `ippiDiv` meets an attempt to divide a positive value by zero, the function execution is not aborted. The result of the operation is set to the maximum value that can be represented by the source data type, and the user is warned by the output status `ippStsDivByZero`. See appendix A "Handling of Special Cases" for more information.

Structures and Enumerators

This section describes the structures and enumerators used by the Intel Integrated Performance Primitives for image and video processing.

The `IppStatus` constant enumerates the status code values returned by Intel IPP functions, indicating whether the operation was error-free or not.

See section Error Reporting in this chapter for more information on the set of valid status codes and corresponding error messages for image and video processing functions.

The structure `IppiPoint` for storing the geometric position of a point is defined as

```
typedef struct {
    int x;
    int y;
} IppiPoint;
```

where `x,y` denote the coordinates of the point.

The structure `IppPointPolar` for storing the geometric position of a point in polar coordinates is defined as

```
typedef struct {
    Ipp32f rho;
    Ipp32f theta;
} IppPointPolar;
```

where `rho` - radial coordinate (radial distance from the origin), `theta` - angular coordinate (counterclockwise angle from the x-axis).

The structure `IppiSize` for storing the size of a rectangle is defined as

```
typedef struct {
    int width;
    int height;
} IppiSize;
```

where `width` and `height` denote the dimensions of the rectangle in the `x-` and `y-` directions, respectively.

The structure `IppiRect` for storing the geometric position and size of a rectangle is defined as

```
typedef struct {
    int x;
    int y;
    int width;
    int height;
} IppiRect;
```

where `x,y` denote the coordinates of the top left corner of the rectangle that has dimensions `width` in the `x`-direction by `height` in the `y`-direction.

The `ippiConnectedComp` structure used in computer vision functions defines the connected component as follows:

```
typedef struct _IppiConnectedComp {
    Ipp64f area;
    Ipp64f value[3];
    IppiRect rect;
} IppiConnectedComp;
```

where `area` - area of the segmented component;
`value[3]` - gray scale value of the segmented component;
`rect` - bounding rectangle of the segmented component.

The `IppiMaskSize` enumeration defines the neighborhood area for some morphological and filtering functions:

```
typedef enum {
    ippMskSize1x3 = 13,
    ippMskSize1x5 = 15,
    ippMskSize3x1 = 31,
    ippMskSize3x3 = 33,
    ippMskSize5x1 = 51,
    ippMskSize5x5 = 55
} IppiMaskSize;
```

The IppCmpOp enumeration defines the type of compare operation to be used in image comparison functions:

```
typedef enum {
    ippCmpLess,
    ippCmpLessEq,
    ippCmpEq,
    ippCmpGreaterEq,
    ippCmpGreater
} IppCmpOp;
```

The IppRoundMode enumeration defines the rounding mode to be used in conversion functions:

```
typedef enum {
    ippRndZero,
    ippRndNear,
    ippRndFinancial
} IppRoundMode;
```

The IppHintAlgorithm enumeration defines the type of code to be used in some image transform and statistics functions, that is, faster but less accurate, or vice-versa, more accurate but slower. For more information on using this enumerator, see Table 10-2 or Table 11-3 .

```
typedef enum {
    ippAlgHintNone,
    ippAlgHintFast,
    ippAlgHintAccurate
} IppHintAlgorithm;
```

The types of interpolation used by geometric transform functions are defined as follows:

```
enum {
    IPPI_INTER_NN        = 1,
    IPPI_INTER_LINEAR    = 2,
    IPPI_INTER_CUBIC     = 4,
    IPPI_INTER_CUBIC2P_BSPLINE,
    IPPI_INTER_CUBIC2P_CATMULLROM,
    IPPI_INTER_CUBIC2P_B05C03,
    IPPI_INTER_SUPER     = 8,
```

```
    IPPI_INTER_LANCZOS  = 16,
    IPPI_SMOOTH_EDGE    =(1 << 31)
};
```

The `IppiAlphaType` enumeration defines the type of the compositing operation to be used in the alpha composition functions:

```
typedef enum {
    ippAlphaOver,
    ippAlphaIn,
    ippAlphaOut,
    ippAlphaATop,
    ippAlphaXor,
    ippAlphaPlus,
    ippAlphaOverPremul,
    ippAlphaInPremul,
    ippAlphaOutPremul,
    ippAlphaATopPremul,
    ippAlphaXorPremul,
    ippAlphaPlusPremul
} IppiAlphaType;
```

The `IppiDitherType` enumeration defines the type of dithering to be used by the `ippiReduceBits` function:

```
typedef enum {
    ippDitherNone,
    ippDitherFS,
    ippDitherJJN,
    ippDitherStucki,
    ippDitherBayer
} IppiDitherType;
```

The layout of the image slices used in some image format conversion functions is defined as follows:

```
enum {
     IPP_UPPER        = 1,
     IPP_LEFT         = 2,
     IPP_CENTER       = 4,
     IPP_RIGHT        = 8,
     IPP_LOWER        = 16,
     IPP_UPPER_LEFT   = 32,
     IPP_UPPER_RIGHT  = 64,
     IPP_LOWER_LEFT   = 128,
     IPP_LOWER_RIGHT  = 256
};
```

The `IppiShape` enumeration defines shapes of the structuring element used in some morphological functions:

```
typedef enum {
    ippiShapeRect    = 0,
    ippiShapeCross   = 1,
    ippiShapeEllipse = 2,
    ippiShapeCustom  = 100
} IppiShape;
```

The `IppiAxis` enumeration defines the flip axes for the `ippiMirror` functions or direction of the image intensity ramp for the `ippiImageRamp` functions:

```
typedef enum {
    ippAxsHorizontal,
    ippAxsVertical,
    ippAxsBoth
} IppiAxis;
```

The `IppiBorderType` enumeration defines the border type that is used by some Separable Filters and Fixed Filters with Border functions:

```
 typedef enum _IppiBorderType {
    ippBorderConst     =  0,
    ippBorderRepl      =  1,
    ippBorderWrap      =  2,
    ippBorderMirror    =  3,
    ippBorderMirrorR   =  4,
    ippBorderMirror2   =  5,
    ippBorderInMem     =  6,
```

```
    ippBorderInMemTop      =  0x0010,
    ippBorderInMemBottom   =  0x0020
} IppiBorderType;
```

The `IppiFraction` enumeration defines the fraction that is used by decimate filter functions:

```
 typedef enum {
    ippPolyphase_1_2,
    ippPolyphase_3_5,
    ippPolyphase_2_3,
    ippPolyphase_7_10,
    ippPolyphase_3_4,
} IppiFraction;
```

The `IppiWTSubband` enumeration defines the appropriate wavelet transform subband used in the JPEG2000 coding functions:

```
typedef enum {
    ippWTSubbandLxLy,
    ippWTSubbandLxHy,
    ippWTSubbandHxLy,
    ippWTSubbandHxHy
} IppiWTSubband;
```

The `IppiMQTermination` enumeration defines how the MQ-coder will be terminated during the operation of the JPEG2000 coding functions:

```
typedef enum {
    ippMQTermSimple,
    ippMQTermNearOptimal,
    ippMQTermPredictable
} IppiMQTermination;
```

The `IppiMQRateAppr` enumeration defines the padding-approximation model in the estimation of rate and distortion procedures performed by the JPEG2000 coding functions:

```
typedef enum {
    ippMQRateApprGood
} IppiMQRateAppr;
```

The code flags used by the JPEG2000 coding functions are defined as follows:

```
enum
{
  IPP_JPEG2K_VERTICALLY_CAUSAL_CONTEXT,
  IPP_JPEG2K_SELECTIVE_MQ_BYPASS,
  IPP_JPEG2K_TERMINATE_ON_EVERY_PASS,
  IPP_JPEG2K_RESETCTX_ON_EVERY_PASS,
  IPP_JPEG2K_USE_SEGMENTATION_SYMBOLS,
  IPP_JPEG2K_LOSSLESS_MODE,
  IPP_JPEG2K_DEC_CONCEAL_ERRORS,
  IPP_JPEG2K_DEC_DO_NOT_CLEAR_CB,
  IPP_JPEG2K_DEC_DO_NOT_RESET_LOW_BITS,
  IPP_JPEG2K_DEC_DO_NOT_CLEAR_SFBUFFER
  IPP_JPEG2K_DEC_CHECK_PRED_TERM
};
```

Function Context Structures

Some Intel IPP functions use special structures to store function-specific (context) information. For example, the `IppiFFTSpec` structure stores twiddle factors and bit reverse indexes needed in computing the fast Fourier transform.

Two different kinds of structures are used: structures that are not modified during function operation - they have the suffix `Spec` in their names, and structures that are modified during operation - they have the suffix `State` in their names.

These context-related structures are not defined in public headers, and their fields are not accessible. It was done because the function context interpretation is processor dependent. Thus, the user has no possibilities to modify these structures or to create a function context as an automatic variable.

Image Data Types and Ranges

The Intel IPP image processing functions support only absolute color images in which each pixel is represented by its channel intensities. The data storage for an image can be either pixel-oriented or plane-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively, for example, RGBRGBRGB in case of an RGB image. The number of channels in a pixel-order image can be 1, 2, 3, or 4.

For images in planar order, all image data for each channel is stored contiguously followed by the next channel, for example, RRR...GGG...BBB.

Functions that operate on planar images are identified by the presence of `Pn` descriptor in their names. In this case, `n` pointers (one for each plane) may be specified.

The image data type is determined by the pixel depth in bits per channel, or bit depth. Bit depth for each channel can be 8, 16 or 32 and is included in the function name as one of these numbers (see Function Naming in this chapter for details). Some functions operate with images in 16-bit packed RGB format (see RGB Image Formats in Chapter 6 for more details). In this case data of all 3 channels are represented as 16u data type. The data may be signed (`s`), unsigned (`u`), or floating-point real (`f`). For some arithmetic and FFT/DFT functions, data in complex format (`sc` or `fc`) can be used, where each channel value is represented by two numbers: real and imaginary part. All channels in an image must have the same data type.

For example, in an absolute color 24-bit RGB image, three consecutive bytes (24 bits) per pixel represent the three channel intensities in pixel mode. This data type is identified in function names as `8u_C3` descriptor, where `8u` represents 8-bit unsigned data for each channel and `C3` represents three channels.

For another example, in an absolute color 16-bit packed RGB image, two consecutive bytes (16 bits) per pixel represent the three channel intensities in pixel mode. This data type is identified in function names as `16u_C3` descriptor, where `16u` represents 16-bit unsigned data (not a bit depth) for all packed channels together and `C3` stands for three channels.

If an alpha (opacity) channel is present in image data, the image must have four channels, with alpha channel being the last or the first one. This data type is indicated by the `AC4` or `A0C4` descriptors respectively. The presence of alpha channel can modify the function's behavior. For such functions, Intel IPP provides versions with and without alpha. If an alpha channel is specified, the operation usually does not take place on that channel.

The range of values that can be represented by each data type lies between the lower and upper bounds. The following table lists data ranges and constant identifiers used in Intel IPP to denote the respective range bounds:

Table 2-2 Image Data Types and Ranges

Data Type	Lower Bound		Upper Bound	
	Identifier	Value	Identifier	Value
8s	IPP_MIN_8S	-128	IPP_MAX_8S	127
8u		0	IPP_MAX_8U	255
16s	IPP_MIN_16S	-32768	IPP_MAX_16S	32767
16u		0	IPP_MAX_16U	65535
32s	IPP_MIN_32S	-2^{31}	IPP_MAX_32S	$2^{31}-1$
32u		0	IPP_MAX_32U	$2^{32}-1$
32f †	IPP_MINABS_32F	$1.175494351e^{-38}$	IPP_MAXABS_32F	$3.402823466e^{38}$

† The range for absolute values

Major Operation Models

Most Intel IPP image processing functions perform identical and independent operations on all channels of the processed image (except alpha channel). It means that the same operation is applied to each channel, and the computed results do not depend upon values of other channels. Some exceptions include the `ippiFilterMedianColor` function and color conversion functions, which process three channels together.

The Intel IPP image processing functions can be broken into two major models of operation: the functions that operate on one pixel to compute the result (also known as point operations), for example, `ippiAdd`, and the functions that operate on a group of pixels (also referred to as a neighborhood), for example, `ippiFilterBox`.

Neighborhood Operations

The result of a neighborhood operation is based on values of a certain group of pixels, located near a given input pixel. The set of neighboring pixels is typically defined by a rectangular mask (or kernel) and anchor cell, specifying the mask alignment with respect to the position of the input pixel.

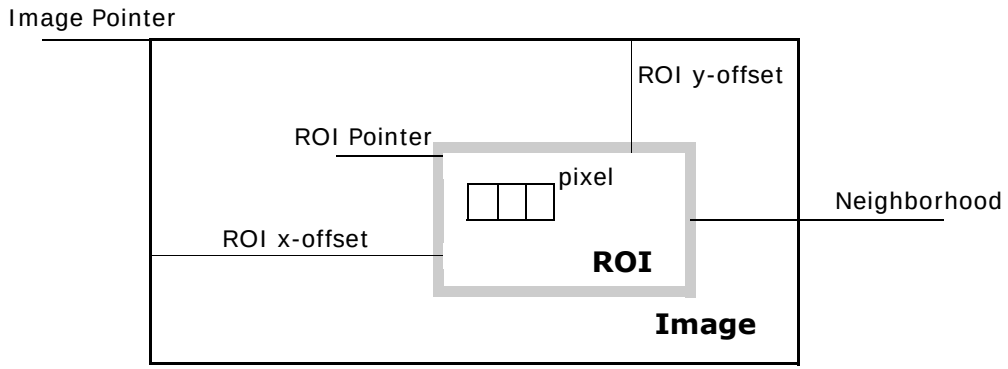
The Intel IPP functions that process a neighborhood operate on the assumption that all referred points of the image are available. To support this mode, the application must check that ROI parameters passed to the function have such values that all processed neighborhood pixels actually exist in the image.

Regions of Interest in Intel IPP

Most Intel IPP image processing functions can operate not only on entire images but also on image areas. Image region of interest (ROI) is a rectangular area that may be either some part of the image or the whole image.

Intel IPP functions with ROI support are distinguished by the presence of an `R` descriptor in their names. ROI of an image is defined by the size and offset from the image origin as shown in Figure 2-1. The origin of an image is implied to be in the top left corner, with `x` values increasing from left to right and `y` values increasing downwards.

Figure 2-1 Image, ROI, and Offsets



Both the source and destination images can have a region of interest. In such cases the sizes of ROIs are assumed to be the same while offsets may differ. The image processing is then performed on data of the source ROI, and the results are written to the destination ROI.

In function call sequences, an ROI is specified by:

- *roiSize* parameter of the IppiSize type
- *pSrc* and *pDst* pointers to the starts of source and destination ROI buffers
- *srcStep* and *dstStep* parameters which are equal to distances in bytes between the starts of consecutive lines in source and destination images, respectively.

Thus, the parameters *srcStep*, *dstStep* set steps in bytes through image buffers to start processing a new line in the ROI of an image.

The following code example illustrates the use of the *dstStep* parameter in function calls:

Example 2-1 Using Buffer Step Parameter

```
IppStatus alignedLine( void ) {
   Ipp8u x[8*3] = {0};
   IppiSize imgSize = {5,3};
   /// The image is of size 5x3. Width 8 has been
   /// chosen by the user to align every line of the image
   return ippiSet_8u_C1R( 7, x, 8, imgSize);
}
```

The resultant image *x* contains the following data:

```
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
```

If ROI is present,

- source and destination images can have different sizes
- lines may have padding at the end for aligning the line sizes
- application must correctly define the *pSrc*, *pDst* and *roiSize* parameters

The *pSrc* and *pDst* parameters are the shifted pointers to the image data. For example, in case of ROI operations on 3-bytes-per-pixel image data (8u_C3R), *pSrc* points to the start of the source ROI buffer and can be interpreted as follows:

pSrc = *pSrcImg* + 3*(*srcImgSize.width* * *srcRoiOffset.y* + *srcRoiOffset.x*),

where

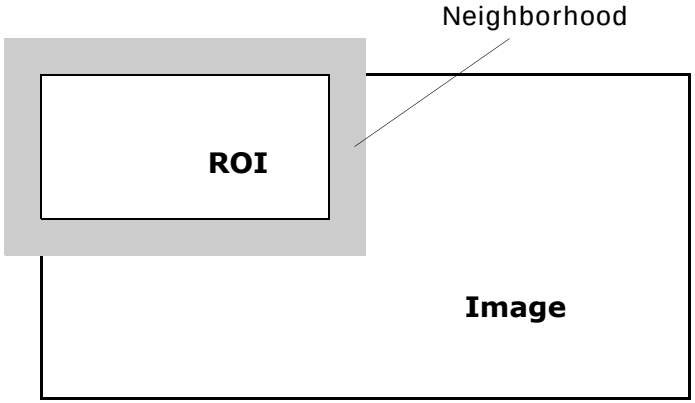
pSrcImg points to the start of the source image buffer;

srcImgSize is the image size in pixels (of the IppiSize type);

srcRoiOffset determines an offset of ROI relative to the start of the image as shown in Figure 2-1.

For functions using ROI with a neighborhood, you should correctly use values of the *pSrc* and *roiSize* parameters. These functions assume that the points in the neighborhood exist and that therefore *pSrc* is almost never equal to *pSrcImg*. Figure 2-2 illustrates the case when neighborhood pixels can fall outside the source image.

Figure 2-2 Using ROI with Neighborhood



To ensure valid operation when image pixels are processed, the application should correctly define additional border pixels (see Borders).

WARNING. If the required border pixels are not defined prior to calling neighborhood functions that attempt to process such pixels, you may get memory violation errors.

The following code example shows how to process an image with ROI:

Example 2-2 ROI Processing

```
IppStatus roi( void ) {
   Ipp8u x[8*3] = {0};
   IppiSize roiSize = {3,2};
   IppiPoint roiPoint = {2,1};
   /// place the pointer to the ROI start position
   return ippiSet_8u_C1R( 7, x+8*roiPoint.y+roiPoint.x, 8,
roiSize );
}
```

The destination image x will contain the following data

```
00 00 00 00 00 00 00 00
00 00 07 07 07 00 00 00
00 00 07 07 07 00 00 00
```

Tiled Image Processing

The Intel IPP can process images composed from tiles, or tiled images. The specially created code sample “Tiled Image processing” demonstrates how to do it. See *Intel IPP Image Processing Samples* downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

Support Functions

3

This chapter describes Intel® IPP support functions that are used to:

- retrieve information about the current Intel IPP software version
- get a brief explanation of the returned status codes
- allocate and free memory that is needed for the operation of other Intel IPP image and video processing functions.

The list of support functions is given in Table 3-1.

Table 3-1 Intel IPP Support Functions

Function Base Name	Operation
Version Information Functions	
GetLibVersion	Returns information about the active library version.
Status Information Function	
ippGetStatusString	Translates a status code into a message.
Memory Allocation Functions	
Malloc	Allocates memory aligned to 32-byte boundary.
Free	Frees memory allocated by the function ippiMalloc.

Version Information Function

This function returns the version number and other information about the active Intel IPP image processing software.

GetLibVersion

Returns information about the used version of Intel IPP software for image processing.

Syntax

```
const IppLibraryVersion* ippiGetLibVersion(void);
```

Description

The function `ippiGetLibVersion` is declared in the `ippi.h` file. This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of the Intel IPP for image processing. There is no need for you to release memory referenced by the returned pointer, as it points to a static variable. The following fields of the `IppLibraryVersion` structure are available:

major	the major number of the current library version.
minor	the minor number of the current library version.
majorBuild	the number of builds of the major version.
build	current build number.
Name	the name of the current library version.
Version	the library version string.
BuildDate	the library version actual build date.

For example, if the library version is “v1.2 Beta”, library name is “ippim6”, and build date is “Jul 20 99”, then the fields in this structure are set as:

major = 1, *minor* = 2, *Name* = “ippim6”,
Version = “v1.2 Beta”, *BuildDate* = “Jul 20 99”

Example 3-1 shows how to use the function `ippiGetLibVersion`.

Example 3-1 Using the ippiGetLibVersion Function

```
void libinfo(void) {
        const IppLibraryVersion* lib = ippiGetLibVersion();
        printf("%s %s %d.%d.%d.%d\n", lib->Name, lib->Version,
            lib->major, lib->minor, lib->majorBuild, lib->build);
}

Output:
    ippia6 v0.0 Alpha 0.0.5.5
```

NOTE. Each sub-library that is used in the image processing domain has its own similar function to retrieve information about the active library version. These functions are: `ippGetLibVersion`, `ippjGetLibVersion`, `ippcvGetLibVersion`, `ippvcGetLibVersion`, and `ippccGetLibVersion`. They are declared in the following header files: `ippcore.h`, `ippj.h`, `ippcv.h`, `ippvc.h`, `ippcc.h`, respectively, and have the same interface as the above described function.

Status Information Function

Use this function to get a brief description of the status code returned by the current Intel IPP software.

ippGetStatusString

Translates a status code into a message.

Syntax

```
const char* ippGetStatusString(IppStatus StsCode);
```

Parameters

StsCode	Code that indicates the status type (see Table 2-1)

Description

The function `ippGetStatusString` is declared in the `ippcore.h` file. This function returns a pointer to the text string associated with a status code *StsCode*. Use this function to produce error and warning messages for users. The returned pointer is a pointer to an internal static buffer and need not be released.

A code example below shows how to use the function `ippGetStatusString`.
If you call an Intel IPP function, in this example `ippiSet_8u_C1R`, with a `NULL` pointer, it returns an error code `-8`.

The status information function translates this code into the corresponding message "`Null Pointer Error`":

Example 3-2 Using the Status Information Function

```
void statusInfo( void ) {
   IppiSize roi = {0};
   IppStatus st = ippiSet_8u_C1R( 3, 0, 0, roi );
   printf( " %d : %s\n", st, ippGetStatusString( st ));
}
Output:
    -8, Null Pointer Error
```

Memory Allocation Functions

This section describes the Intel IPP functions that allocate aligned memory blocks for data of required type, or free the previously allocated memory.

NOTE. The only function to free the memory allocated by any of these functions is `ippiFree()`.

Malloc

Allocates memory aligned to 32-byte boundary.

Syntax

```
Ipp<datatype>* ippiMalloc_<mod>(int widthPixels, int heightPixels,
    int* pStepBytes);
```

Supported values for *mod* :

8u_C1	16u_C1	16s_C1	32s_C1	32f_C1	32sc_C1	32fc_C1
8u_C2	16u_C2	16s_C2	32s_C2	32f_C2	32sc_C2	32fc_C2
8u_C3	16u_C3	16s_C3	32s_C3	32f_C3	32sc_C3	32fc_C3
8u_C4	16u_C4	16s_C4	32s_C4	32f_C4	32sc_C4	32fc_C4
8u_AC4	16u_AC4	16s_AC4	32s_AC4	32f_AC4	32sc_AC4	32fc_AC4

Parameters

widthPixels	Width of an image in pixels.
heightPixels	Height of an image in pixels.
pStepBytes	Pointer to the distance in bytes between starts of consecutive lines in the image.

Description

The function `ippiMalloc` is declared in the `ippi.h` file. This function allocates a memory block aligned to a 32-byte boundary for elements of different data types. Every line of the image is aligned by padding with zeros in accordance with the *pStepBytes* parameter, which is calculated by the function `ippiMalloc` and returned for further use.

The function ippiMalloc allocates one continuous memory block, whereas functions that operate on planar images require an array of separate pointers (IppType* plane[3]) to each plane as input. The function ippiMalloc should be called three times in this case. The code Example 3-3 demonstrates how to construct such an array and set correct values to the pointers in order to use the allocated memory block with Intel IPP functions operating on planar images. Note that *pStepBytes* should be specified for each plane. The example is given for 8u data type.

Example 3-3 Setting Values to the Pointers to Image Planes

```
int stepBytes[3];
Ipp8u* plane[3];
    plane[0] = ippiMalloc_8u_C1(widthPixels, heightPixels,
                                &(stepBytes [0]));
    plane[1] = ippiMalloc_8u_C1(widthPixels/2, heightPixels/2,
                                &(stepBytes [1]));
    plane[2] = ippiMalloc_8u_C1(widthPixels/2, heightPixels/2,
                                &(stepBytes [2]));
```

Return Values

The return value of ippiMalloc function is a pointer to an aligned memory block.
If no memory is available in the system, the NULL value is returned.
To free the allocated memory block, the function ippiFree must be used.

Free

Frees memory allocated by the function ippiMalloc.

Syntax

```
void ippiFree(void* ptr);
```

Parameters

ptr	Pointer to a memory block to be freed. This block must have been previously allocated by the function ippiMalloc.

Description

The function `ippiFree` is declared in the `ippi.h` file. This function frees the aligned memory block allocated by the function `ippiMalloc`.

NOTE. The function `ippiFree` cannot be used to free memory allocated by standard functions like `malloc` or `calloc`, nor can the memory allocated by `ippiMalloc` be freed by `free`.

Image Data Exchange and Initialization Functions

4

This chapter describes Intel® IPP image processing functions that perform image data manipulation, exchange and initialization operations.

Table 4-1 lists functions described in more detail later in this chapter:

Table 4-1 Image Data Exchange and Initialization Functions

Function Base Name	Operation
Convert	Converts pixel values in an image from one bit depth to another
Scale	Scales pixel values of a image and converts them to another bit depth
Set	Sets an array of pixels to a value
Copy	Copies pixel values between two images
CopyManaged	Copies pixel values between two images in accordance with the specified type of copying.
CopyConstBorder	Copies pixels values between two images and adds the border pixels with a constant value
CopyReplicateBorder	Copies pixel values between two images and adds the replicated border pixels
CopyWrapBorder	Copies pixel values between two images and adds the border pixels
CopySubpix	Copies pixel values between two images with subpixel precision.
CopySubpixIntersect	Copies pixel values of the intersection with specified window with subpixel precision.
Dup	Copies gray scale image to all channels of the color image.
Transpose	Transpose a source image.

Table 4-1 Image Data Exchange and Initialization Functions (continued)

Function Base Name	Operation
SwapChannels	Changes the order of channels of the image
AddRandUniform_Direct	Generates random samples with uniform distribution and adds them to an image data
AddRandGauss_Direct	Generates random samples with Gaussian distribution and adds them to an image data
ImageJaehne	Creates the Jaehneís test image
ImageRamp	Creates the test image that has an intensity ramp
SampleLine	Reads all pixels on the raster line into the buffer
ZigzagFwd8x8	Converts a conventional order to the zigzag sequence.
ZigzagInv8x8	Converts a zigzag sequence to the conventional order.
ResampleRowGetSize	Computes the sizes of the resample structure and the work buffer
ResampleRowInit	Initializes the row resampling structure
ResampleRowGetBorderWidth	Computes the width of the left and right borders
ResampleRow, ResampleRowReplicateBorder	Perfroms row-oriented resampling of the image.

Many solutions and hints for use of these functions can be found in Intel IPP Samples. See *Intel IPP Image Processing and Media Processing Samples* downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

Convert

Converts pixel values of an image from one bit depth to another.

Syntax

Case 1: Conversion to increased bit depth

```
IppStatus ippiConvert_1u8u_C1R(const Ipp8u* pSrc, int srcStep,
    int srcBitOffset, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u16u_C1R	8u16s_C1R	8u32s_C1R	8u32f_C1R	8s32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R	8u32f_C3R	8s32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R	8u32f_C4R	8s32s_C4R
8u16u_AC4R	8u16s_AC4R	8u32s_AC4R	8u32f_AC4R	8s32s_AC4R

8s32f_C1R	16u32s_C1R	16u32f_C1R	16s32s_C1R	16s32f_C1R
8s32f_C3R	16u32s_C3R	16u32f_C3R	16s32s_C3R	16s32f_C3R
8s32f_C4R	16u32s_C4R	16u32f_C4R	16s32s_C4R	16s32f_C4R
8s32f_AC4R	16u32s_AC4R	16u32f_AC4R	16s32s_AC4R	16s32f_AC4R

Case 2: Conversion to reduced bit depth:

integer to integer type

```
IppStatus ippiConvert_8u1u_C1R( const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize,
    Ipp8u threshold);
```

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

16u8u_C1R	16s8u_C1R	32s8u_C1R	32s8s_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R	32s8s_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R	32s8s_C4R
16u8u_AC4R	16s8u_AC4R	32s8u_AC4R	32s8s_AC4R

floating point to integer type

```
IppStatus ippiConvert_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    IppRoundMode roundMode);
```

Supported values for *mod*:

32f8u_C1R	32f8s_C1R	32f16u_C1R	32f16s_C1R
32f8u_C3R	32f8s_C3R	32f16u_C3R	32f16s_C3R
32f8u_C4R	32f8s_C4R	32f16u_C4R	32f16s_C4R
32f8u_AC4R	32f8s_AC4R	32f16u_AC4R	32f16s_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcBitOffset	Offset (in bits) in the first byte of the source image row.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstBitOffset	Offset (in bits) in the first byte of the destination image row.
roiSize	Size of the source and destination ROI in pixels.
threshold	Threshold level for Stucki's dithering.
roundMode	Rounding mode, the following values are possible:
	`ippRndZero` — Specifies that floating-point values must be truncated toward zero.
	`ippRndNear` — Specifies that floating-point values must be rounded to the nearest even integer.
	`ippRndFinancial` — Specifies that floating-point values must be rounded down to the nearest integer if decimal value is less than 0.5, or rounded up to the nearest integer if decimal value is equal or greater than 0.5.

Description

The function `ippiConvert` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts pixel values in the source image ROI *pSrc* to a different bit depth and writes them to the destination image ROI *pDst*. No scaling is done. When converting from floating-point to integer type, rounding defined by *roundMode* is performed, and the result is saturated to the destination data type range.

1u to 8u conversion. The source image has a `8u` data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter *srcBitOffset* is required to specify the start position of the source ROI buffer. Each source bit is transformed into the byte of the destination image in the following way: if an input pixel is 0, the corresponding output pixel is set to 0, if an input pixel is 1, the

corresponding output pixel is set to 255. Note that in the source image the bit order in each byte is inverse relative to the pixel order, that is, the first pixel in a row is represented by the last (7th) bit of the first byte in the row.

8u to 1u conversion. Source image is converted to a bitonal image using the Stucki's error diffusion dithering algorithm.
The destination image has a `8u` data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter *dstBitOffset* is required to specify the start position of the destination ROI buffer.

Example 4-1 illustrates data conversion without scaling.

Example 4-1 Straightforward Data Conversion

```
IppStatus convert( void ) {
   IppiSize roi = {5,4};
   Ipp32f x[5*4];

   Ipp8u y[5*4];
   ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
   x[1] = 300; x[2] = 150;
   return ippiConvert_32f8u_C1R(x, 5*sizeof(Ipp32f), y, 5, roi,
                                                    ippRndNear);
}
The destination image y contains:

     00 FF 96 00 00
     00 00 00 00 00
     00 00 00 00 00
     00 00 00 00 00
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value, or *srcBitOffset*/*dstBitOffset* is less than zero.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

Scale

Scales pixel values of an image and converts them to another bit depth.

Syntax

Case 1: Scaling with conversion to integer data of increased bit depth

```
IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u16u_C1R	8u16s_C1R	8u32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R
8u16u_AC4R	8u16s_AC4R	8u32s_AC4R

Case 2: Scaling with conversion to floating-point data

```
IppStatus ippiScale_<mod>(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

8u32f_C1R
8u32f_C3R
8u32f_C4R
8u32f_AC4R

Case 3: Scaling of integer data with conversion to reduced bit depth

```
IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    IppHintAlgorithm hint);
```

Supported values for *mod*:

16u8u_C1R	16s8u_C1R	32s8u_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R
16u8u_AC4R	16s8u_AC4R	32s8u_AC4R

Case 4: Scaling of floating-point data with conversion to integer data type

```
IppStatus ippiScale_<mod>(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

```
32f8u_C1R
32f8u_C3R
32f8u_C4R
32f8u_AC4R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
vMin, vMax	Minimum and maximum values of the input data.
hint	Option to select the algorithmic implementation of the function.

Description

The function `ippiScale` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts pixel values of a source image ROI *pSrc* to the destination data type, using a linear mapping. Computation algorithm is specified by the *hint* argument (see Table 11-3). For conversions between integer data types, the whole range [*src_Min..src_Max*] of the input data type is mapped onto the range [*dst_Min..dst_Max*] of the output data type (see chapter 2 for more information on data types and ranges).
The following scaling formula to map the source pixel *p* to the destination pixel *p'* is used:

$$p' = dst_Min + k*(p - src_Min)$$

where $k = (dst_Max - dst_Min)/(src_Max - src_Min)$

For conversions to and from floating-point data type, the user-defined floating-point data range [*vMin..vMax*] is mapped onto the source or destination data type range.

If the conversion is from `Ipp32f` type and some of the input floating-point values are outside the specified input data range [*vMin..vMax*], the corresponding output values will saturate. To determine the actual floating-point data range in your image, use the function `ippiMinMax`.

Example 4-2 shows how to use scaling to preserve the data range.

Example 4-2 Data Conversion with Scaling

```
IppStatus scale( void ) {
   IppiSize roi = {5,4};
   Ipp32f x[5*4];
   Ipp8u y[5*4];
   ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
   x[1] = 300; x[2] = 150;
   return ippiScale_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, -1, 300 );
}
The destination image y contains:
         00 FF 80 00 00
         00 00 00 00 00
         00 00 00 00 00
         00 00 00 00 00
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsScaleRangeErr`	Indicates an error condition if the input data bounds are incorrect, that is *vMax* is less than or equal to *vMin*.

Set

Sets an array of pixels to a value.

Syntax

Case 1: Setting one-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R

Case 2: Setting each color channel to a specified value

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

Case 3: Setting color channels and alpha channel to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[4],
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R

Case 4: Setting masked one-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for *mod*:

8u_C1MR	16u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR

Case 5: Setting color channels of masked multi-channel data to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for *mod*:

8u_C3MR	16u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_AC4MR	16u_AC4MR	16s_AC4MR	32s_AC4MR	32f_AC4MR

Case 6: Setting all channels of masked multi-channel data to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for *mod*:

8u_C4MR	16u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR

Case 7: Setting selected channel of multi-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3CR	16u_C3CR	16s_C3CR	32s_C3CR	32f_C3CR
8u_C4CR	16u_C4CR	16s_C4CR	32s_C4CR	32f_C4CR

Parameters

value	Constant value assigned to each pixel in the destination image ROI.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the image ROI in pixels.
pMask	Pointer to the mask image buffer.
maskStep	Distance in bytes between starts of consecutive lines in the mask image buffer.

Description

The function `ippiSet` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets pixels in the destination image ROI *pDst* to a constant *value*. Either all pixels in a rectangular ROI, or only those selected by the specified mask *pMask*, can be set to a value. In case of masked operation, the function sets pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero.
When a channel of interest is selected, that is only one channel of a multi-channel image must be set (see **Case 7**), the *pDst* pointer points to the start of ROI buffer in the required channel.
If alpha channel is present in the source image data, the alpha components may be either skipped, or set to a value, depending on the chosen `ippiSet` function flavor.

Example 4-3 shows how to use the function `ippiSet_8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *dstStep* or *maskStep* has a zero or negative value.

Example 4-3 Using the Function `ippiSet`

```
void func_set()
{
    IppiSize roi = {5,4};
    Ipp8u x[8*4] = {0};

    ippiSet_8u_C1R(1, x,8, roi);
}

Result:
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
```

Copy

Copies pixel values between two images.

Syntax

Case 1: Copying all pixels of all color channels

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32s_C1R`	`32f_C1R`
`8u_C3R`	`16u_C3R`	`16s_C3R`	`32s_C3R`	`32f_C3R`
`8u_C4R`	`16u_C4R`	`16s_C4R`	`32s_C4R`	`32f_C4R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`	`32s_AC4R`	`32f_AC4R`

8u_C3AC4R	16u_C3AC4R	16s_C3AC4R	32s_C3AC4R	32f_C3AC4R
8u_AC4C3R	16u_AC4C3R	16s_AC4C3R	32s_AC4C3R	32f_AC4C3R

Case 2: Operation on masked pixels only

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp8u* pMask, int maskStep);
```

Supported values for *mod*:

8u_C1MR	16u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR
8u_C3MR	16u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_C4MR	16u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR
8u_AC4MR	16u_AC4MR	16s_AC4MR	32s_AC4MR	32f_AC4MR

Case 3: Copying of a selected channel in a multi-channel image

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3CR	16u_C3CR	16s_C3CR	32s_C3CR	32f_C3CR
8u_C4CR	16u_C4CR	16s_C4CR	32s_C4CR	32f_C4CR

Case 4: Copying of a selected channel to a one-channel image

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32s_C3C1R	32f_C3C1R
8u_C4C1R	16u_C4C1R	16s_C4C1R	32s_C4C1R	32f_C4C1R

Case 5: Copying a one-channel image to a multi-channel image

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1C3R	16u_C1C3R	16s_C1C3R	32s_C1C3R	32f_C1C3R
8u_C1C4R	16u_C1C4R	16s_C1C4R	32s_C1C4R	32f_C1C4R

Case 6: Splitting color image into separate planes

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* const pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3P3R      16u_C3P3R      16s_C3P3R      32s_C3P3R      32f_C3P3R
```

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C4P4R      16u_C4P4R      16s_C4P4R      32s_C4P4R      32f_C4P4R
```

Case 7: Composing color image from separate planes

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_P3C3R      16u_P3C3R      16s_P3C3R      32s_P3C3R      32f_P3C3R
```

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_P4C4R      16u_P4C4R      16s_P4C4R      32s_P4C4R      32f_P4C4R
```

Parameters

pSrc	Pointer to the source image ROI. The array storing pointers to the ROIs in the separate color planes of the source image for **Case 7.**
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI. The array storing pointers to the ROIs in the resulting color planes for **Case 6.**
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
pMask	Pointer to the mask image buffer.
maskStep	Sep in bytes through the mask image buffer.

Description

The function `ippiCopy` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function copies data from a source image ROI *pSrc* to the destination image ROI *pDst*. Copying of pixels selected by a mask *pMask* is supported as well. In case of masked operation, the function writes pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero (as illustrated in the code example below).

Function flavors with channel of interest descriptor (C) copy only one specified channel of a source multi-channel image to another multi-channel image (see **Case 3**). For these functions, the *pSrc* and *pDst* pointers point to the starts of ROI buffers in the specified channels of source and destination images, respectively.

Function flavors exist that can add alpha channel to the 3-channel source image (flavors with the _C3AC4R descriptor), or discard aplha channel from the source image (flavors with the _AC4C3R descriptor) - see **Case 1**.

Function flavors exist that can be used to copy data from only one specified channel *pSrc* of a multi-channel image to a one-channel image *pDst* (see **Case 4**), as well as to copy data from a one-channel image *pSrc* to only one specified channel of a multi-channel image *pDst* (see **Case 5**).

The function ippiCopy can also be used to split the color image into separate planes (see **Case 6**).

Example 4-4 shows how to copy masked data.

Example 4-4 Copying Masked Data

```
IppStatus  copyWithMask( void ) {
   Ipp8u mask[3*3], x[5*4], y[5*4]={0};
   IppiSize imgroi={5,4}, mskroi={3,3};
   ippiSet_8u_C1R( 3, x, 5, imgroi );
   /// set mask with a hole in upper left corner
   ippiSet_8u_C1R( 1, mask, 3, mskroi );
   mask[0] = 0;
   /// copy roi with mask
   return ippiCopy_8u_C1MR( x, 5, y, 5, mskroi, mask, 3 );
}
The destination image y contains:
00 03 03 00 00
03 03 03 00 00
03 03 03 00 00
00 00 00 00 00
```

The function ippiCopy_8u_C1R is used in H.264 decoder and encoder included into Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL, with the exception of second mode in **Case 4**.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <pixelSize> for **Cases 4** and **5**.

CopyManaged

Copies pixel values between two images in accordance with the specified type of copying.

Syntax

```
IppStatus ippiCopyManaged_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, IppiSize roiSize, int flags);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
flags	Specifies the type of copying. Possible values are: IPP_TEMPORAL_COPY - standard copying; IPP_NONTEMPORAL_STORE - copying without caching the destination image.

Description

The function ippiCopyManaged is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP). This function copies data from a source image ROI *pSrc* to the destination image ROI *pDst*. The parameter *flags* specifies the type of copying that

the function performs. When the *flags* is set to IPP_TEMPORAL_COPY, the function is identical to the function ippiCopy_8u_C1R. When the *flags* is set to IPP_NONTEMPORAL_STORE, the processor uses non-temporal store instructions, and operation is performed without caching the data of the destination image.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

CopyConstBorder

Copies pixels values between two images and adds the border pixels with a constant value.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth,
    Ipp<datatype> value);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const
    Ipp<datatype> value[3]);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth,
    const Ipp<datatype> value[4]);
```

Supported values for *mod*:

```
8u_C4R      16u_C4R     16s_C4R     32s_C4R     32f_C4R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
srcRoiSize	Size of the source ROI in pixels.
dstRoiSize	Size of the destination ROI in pixels.
topBorderHeight	Height of the top border in pixels.
leftBorderWidth	Width of the left border in pixels.
value	The constant value to assign to the border pixels (constant vector in case of multi-channel images).

Description

The function `ippiCopyConstBorder` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image *pSrc* to the destination image *pDst* and creates border outside the copied area; pixel values of the border are set to the specified constant value that is passed by the *value* argument. The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI (see Figure 4-1.) Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

Example 4-5 shows how to use the function `ippiCopyConstBorder_8u_C1R`.

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

Figure 4-1 Creating a Border of Pixels with Constant Value

v	v	v	v	v	v	v	v	v
v	v	v	v	v	v	v	v	v
v	v	1	2	3	4	5	v	v
v	v	6	7	8	9	10	v	v
v	v	11	12	13	14	15	v	v
v	v	16	17	18	19	20	v	v
v	v	v	v	v	v	v	v	v

topBorderHeight=2
leftBorderWidth=2

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value, or *topBorderHeight* or *leftBorderWidth* is less than zero, or *dstRoiSize.width* < *srcRoiSize.width* + *leftBorderWidth*, or *dstRoiSize.height* < *srcRoiSize.height* + *topBorderHeight*.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

Example 4-5 Using the Function `ippiCopyConstBorder`

```
Ipp8u src[8*4] = {3, 3, 3, 3, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 3, 3, 3, 3, 8, 8, 8};
Ipp8u dst[8*6];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 7, 6 };
int borderWidth  = 1;
int borderHeight = 1;
int borderVal = 0;

ippiCopyConstBorder_8u_C1R(src, 8, srcRoi, dst, 8, dstRoi, borsderHeight,
borderWidth, borderVal);
```

Example 4-5 Using the Function `ippiCopyConstBorder`

Results
source image:

```
3 3 3 3 3 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8        src
3 3 3 3 3 8 8 8
```

destination image:

```
0 0 0 0 0 0 0
0 3 3 3 3 3 0
0 3 2 1 2 3 0
0 3 2 1 2 3 0     dst
0 3 3 3 3 3 0
0 0 0 0 0 0 0
```

CopyReplicateBorder

Copies pixel values between two images and adds the replicated border pixels.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

Case 2: In-place operation

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc,
    int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
```

Supported values for *mod* :

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
srcRoiSize	Size of the source ROI in pixels.
dstRoiSize	Size of the destination ROI in pixels.
topBorderHeight	Height of the top border in pixels.
leftBorderWidth	Width of the left border in pixels.

Description

The function `ippiCopyReplicateBorder` is declared in the `ippi.h` file.It operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image ROI *pSrc* to the destination image ROI *pDst* and fills pixels (“border”) outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in the Figure 4-2. Squares marked in red correspond to pixels copied from the source image.
Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI.

Example 4-6 shows how to use the function `ippiCopyReplicateBorder_8u_C1R`.

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

Figure 4-2 Creating a Replicated Border

1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
6	6	6	7	8	9	10	10	10
11	11	11	12	13	14	15	15	15
16	16	16	17	18	19	20	20	20
16	16	16	17	18	19	20	20	20

topBorderHeight=2
leftBorderWidth=2

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value, or *topBorderHeight* or *leftBorderWidth* is less than zero, or *dstRoiSize.width* < *srcRoiSize.width* + *leftBorderWidth*, or *dstRoiSize.height* < *srcRoiSize.height* + *topBorderHeight*.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

Example 4-6 Using the Function `ippiCopyReplicateBorder`

```
Ipp8u src[8*4] = { 5, 4, 3, 4, 5, 8, 8, 8,
       3, 2, 1, 2, 3, 8, 8, 8,
       3, 2, 1, 2, 3, 8, 8, 8,
       5, 4, 3, 4, 5, 8, 8, 8};
Ipp8u dst[9*8];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 9, 8 };
int topBorderHeight = 2;
int leftBorderWidth = 2;
```

Example 4-6 Using the Function `ippiCopyReplicateBorder` (continued)

```
ippiCopyReplicateBorder_8u_C1R (src, 8, srcRoi, dst, 9, dstRoi,
topBorderHeight, leftBorderWidth);
```

Results

source image:

```
5 4 3 4 5 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8
5 4 3 4 5 8 8 8
```

destination image:

```
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
3 3 3 2 1 2 3 3 3
3 3 3 2 1 2 3 3 3
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
```

CopyWrapBorder

Copies pixel values between two images and adds the border pixels.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiCopyWrapBorder_32s_C1R(const Ipp32s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep,
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyWrapBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32f* pDst, int dstStep,
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Case 2: In-place operation

```
IppStatus ippiCopyWrapBorder_32s_C1IR(const Ipp32s* pSrc,
    int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize,
    int topBorderHeight, int leftBorderWidth);
```

```
IppStatus ippiCopyWrapBorder_32f_C1IR(const Ipp32f* pSrc,
    int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize,
    int topBorderHeight, int leftBorderWidth);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for in-place flavor.
topBorderHeight	Height of the top border in pixels.
leftBorderWidth	Width of the left border in pixels.

Description

The function `ippiCopyWrapBorder` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image ROI *pSrc* to the destination image *pDst* and fills pixels (“border”) outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in the Figure 4-3. Squares marked in red correspond to pixels copied from the source image.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI.

Example 4-7 shows how to use the function `ippiCopywrapBorder_32s_C1R`.

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

Figure 4-3 Creating a Border of Pixels by ippiCopyWrapBorder Function

14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6
14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6
14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6

topBorderHeight=2
leftBorderWidth=2

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value, or *topBorderHeight* or *leftBorderWidth* is less than zero, or *dstRoiSize.width* < *srcRoiSize.width* + *leftBorderWidth*, or *dstRoiSize.height* < *srcRoiSize.height* + *topBorderHeight*.
ippStsStepErr	Indicates an error condition if one of *srcStep*, *dstStep*, *srcDstStep* has a zero or negative value.

Example 4-7 Using the Function ippiCopyWrapBorder

```
Ipp32s src[8*4] = {
       5, 4, 3, 4, 5, 8, 8, 8,
       3, 2, 1, 2, 3, 8, 8, 8,
       3, 2, 1, 2, 3, 8, 8, 8,
       5, 4, 3, 4, 5, 8, 8, 8
```

Example 4-7 Using the Function `ippiCopyWrapBorder` (continued)

```
     };
Ipp32s dst[9*8];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 9, 8 };
int topBorderHeight = 2;
int leftBorderWidth = 2;

ippiCopyWrapBorder_32s_C1R (src , 8*sizeof(Ipp32s), srcRoi, dst,
            9*sizeof(Ipp32s), dstRoi, topBorderHeight, leftBorderWidth );
```

Results

source image:

```
5 4 3 4 5 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8
5 4 3 4 5 8 8 8
```

destination image:

```
2 3 3 2 1 2 3 3 2
4 5 5 4 3 4 5 5 4
4 5 5 4 3 4 5 5 4
2 3 3 2 1 2 3 3 2
2 3 3 2 1 2 3 3 2
4 5 5 4 3 4 5 5 4
4 5 5 4 3 4 5 5 4
2 3 3 2 1 2 3 3 2
```

CopySubpix

Copies pixel values between two images with subpixel precision.

Syntax

Case 1: Copying without conversion or with conversion to floating point data

```
IppStatus ippiCopySubpix_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp32f dx, Ipp32f dy);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u32f_C1R	16u32f_C1R	

Case 2: Copying with conversion to integer data

```
IppStatus ippiCopySubpix_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy,
    int scaleFactor);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
dx	Fractional part of the x-coordinate in the source image.
dy	Fractional part of the y-coordinate in the source image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiCopySubpix` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the pixel value of the destination image using linear interpolation (see "Linear Interpolation" on page B-3) in accordance with the following formulas:
$pDst_{j,\ i} = pSrc_{j+dx,\ i+dy}$, where i=0, ... *roiSize.height*-1, j=0, ..., *roiSize.width*-1.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width**<*pixelSize*>.
ippStsNotEvenStepErr	Indicates an error condition if one of *srcStep* or *dstStep* is not divisible by 4 for floating point images, or by 2 for short integer images.

CopySubpixIntersect

Copies pixel values of the intersection with specified window with subpixel precision

Syntax

Case 1: Copying without conversion or with conversion to floating point data

```
IppStatus ippiCopySubpixIntersect_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, Ipp<dstDatatype>* pDst, int
    dstStep, IppiSize dstRoiSize, IppiPoint_32f point, IppiPoint* pMin,
    IppiPoint* pMax);
```

Supported values for *mod*:

```
8u_C1R      16u_C1R     32f_C1R
8u32f_C1R   16u32f_C1R
```

Case 2: Copying with conversion to integer data

```
IppStatus ippiCopySubpixIntersect_8u16u_C1R_Sfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, IppiPoint_32f point, IppiPoint* pMin, IppiPoint* pMax,
    int scaleFactor);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source image ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination image ROI in pixels.
point	Center point of the window.
pMin	Pointer to coordinates of the top left pixel of the intersection in the destination image.
pMax	Pointer to coordinates of the bottom right pixel of the intersection in the destination image.

scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiCopySubpixIntersect` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function determines the intersection of the source image and the window of size *dstRoiSize* centered in point *point*. Then, corresponding pixels of the destination image are calculated using linear interpolation (see "Linear Interpolation" on page B-3):

$pDst_{j,\ i} = pSrc_{Xsubpix(j),\ Ysubpix(i)}$, where

$Xsubpix(j) = min(max(point.x + j - 0.5*(dstRoiSize.width - 1), 0), srcRoiSize.width - 1)$,

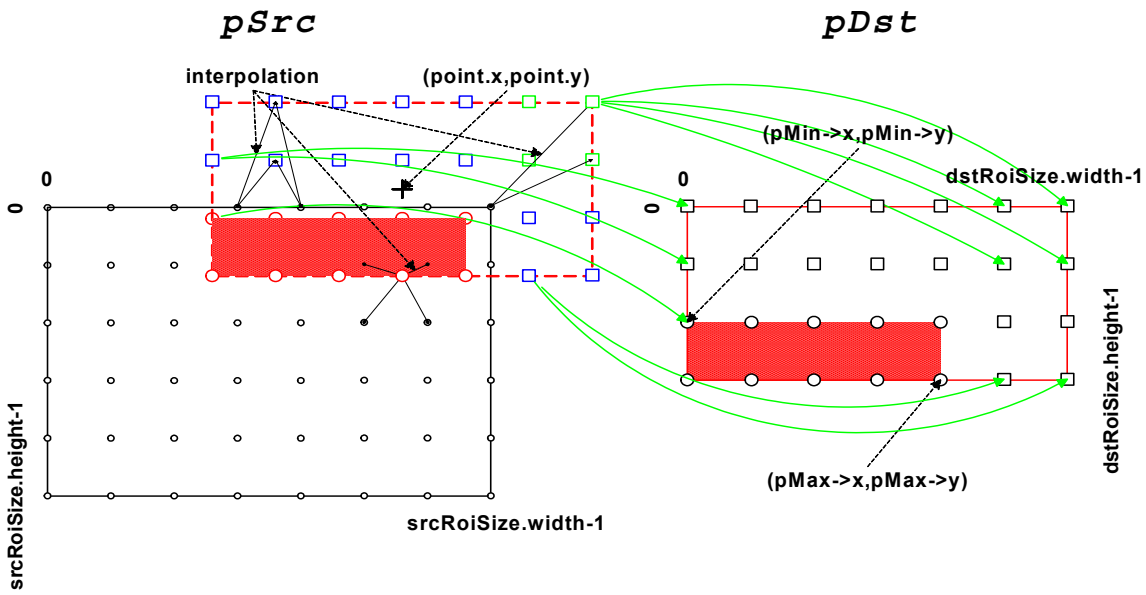
$Ysubpix(i) = min(max(point.y + j - 0.5*(dstRoiSize.heigth - 1), 0), srcRoiSize.heigth - 1)$,

$j = 0, ..., dstRoiSize.width - 1$, $i = 0, ... dstRoiSize.height - 1$.

Minimal values of *j* and *i* (coordinates of top left calculated destination pixel) are assigned to *pMin.x* and *pMin.y*, maximal values (coordinates of top left calculated destination pixel) - to *pMax.x* and *pMax.y*. (See Figure 4-4.)

Example 4-8 shows how to use the function `ippiCopySubpixIntersect_8u_C1R`.

Figure 4-4 Image Copying with Subpixel Precision Using `ippiCopySubpixIntersect` Function



Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* is less than *srcRoiSize.width* * *<pixelSize>* , or *dstStep* is less than *dstRoiSize.width* * *<pixelSize>*.

`ippStsNotEvenStepErr` Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

Example 4-8 Using the Function `ippiCopySubpixIntersect`

```
Ipp8u src[8*6] = {
      7, 7, 6, 6, 6, 6, 7, 7,
      6, 5, 5, 5, 5, 5, 5, 6,
      6, 5, 4, 3, 3, 4, 5, 6,
      6, 5, 4, 3, 3, 4, 5, 6,
      6, 5, 5, 5, 5, 5, 5, 6,
      6, 6, 6, 6, 6, 6, 6, 6
    };
Ipp8u dst[7*4];
IppiSize srcRoi = { 8, 6 };
IppiSize dstRoi = { 7, 4 };
IppiPoint_32f point = { 4, 1 };
IppiPoint min;
IppiPoint max;

ippiCopySubpixIntersect_8u_C1R (src, 8, srcRoi, dst, 7, dstRoi, point,
&min, &max );
```

Results

source image:

```
7 7 6 6 6 6 7 7
6 5 5 5 5 5 5 6
6 5 4 3 3 4 5 6
6 5 4 3 3 4 5 6
6 5 5 5 5 5 5 6
6 6 6 6 6 6 6 6
```

destination image:

```
7 6 6 6 6 7 7
6 6 6 6 6 6 7
5 5 4 4 5 5 6
5 4 3 3 4 5 6

min = { 0, 1 }
max = { 5, 3 }
```

Dup

Copies gray scale image to all channels of the color image.

Syntax

```
IppStatus, ippiDup_8u_C1C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiDup` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function copies one-channel (gray scale) image *pSrc* to every channel of the three-channel (color) image *pDst*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

Transpose

Transposes a source image.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiTranspose_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R

Case 2: In-place operation

```
IppStatus ippiTranspose_<mod>(const Ipp<datatype>* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination ROI for in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source ROI in pixels.

Description

The function `ippiTranspose` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function transposes the source image *pSrc* (*pSrcDst* for in-place flavors) and stores the result in the *pDst* (*pSRcDst*). The destination image is obtained from the source image by transforming the columns to the rows, that is
pDst(*x*,*y*) = *pSrc*(*y*,*x*) for each pixel.

The parameter *roiSize* is specified for the source image; therefore the *roiSize.width* for the destination image is equal to the *roiSize.height* for the source image, and the *roiSize.height* for the destination image is equal to the *roiSize.width* for the source image.

CAUTION. For in-place opeartions *roiSize.width* must be equal to the *roiSize.height*.

Example 4-9 shows how to use the function `ippiTranspose_8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value; or *roiSize.width* is not equal to *roiSize.height* for in-place flavors.

Example 4-9 Using the Function `ippiTranspose`

```
Ipp8u src[8*4] = {1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8};
Ipp8u dst[4*4];
IppiSize srcRoi = { 4, 4 };

ippiTranspose_8u_C1R ( src, 8, dst, 4, srcRoi );

Result:
1 2 3 4 8 8 8 8
1 2 3 4 8 8 8 8    src
1 2 3 4 8 8 8 8
```

Example 4-9 Using the Function `ippiTranspose`

```
1 1 1 1
2 2 2 2
3 3 3 3        dst
4 4 4 4
```

SwapChannels

Copies channels of the source image to the destination image.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const int dstOrder[3]);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

Case 2: In-place operation

```
IppStatus ippiSwapChannels_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const int dstOrder[3]);
IppStatus ippiSwapChannels_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const int dstOrder[4]);
```

Case 3: Operation with converting 3-channel image to the 4-channel image

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const int
    dstOrder[4], Ipp<datatype> val);
```

Supported values for *mod*:

8u_C3C4R	16u_C3C4R	16s_C3C4R	32s_C3C4R	32f_C3C4R

Case 4: Operation with converting 4-channel image to the 3-channel image

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const int
    dstOrder[3]);
```

Supported values for *mod*:

`8u_C4C3R` `16u_C4C3R` `16s_C4C3R` `32s_C4C3R` `32f_C4C3R`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
dstOrder	The order of channels in the destination image.
val	Constant value.

Description

The function `ippiSwapChannels` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function copies the data from specified channels of the source image ROI *pSrc* to the specified channels of the destination image ROI *pDst*.

First channel in the destination image is determined by the first component of *dstOrder*. Its value lies in the range [0..2] for 3-channel image, and [0..3] for 4-channel image, and indicates the corresponding channel number of the source image. Other channels are specified in the similar way. For example, if sequence of channels in source 3-channel image is *A, B, C*, and *dstOrder*[0]=2, *dstOrder*[1]=0, *dstOrder*[2]=1, then the order of channels in the 3-channel destination image is *C, A, B*. Some or all components of *dstOrder* may have the same values, it means that data from a certain channel of the source image may be copied to several channels of the destination image.

Some functions flavors convert a 3-channel source image to the 4-channel destination image (see **Case 3**). In this case additional channel contains data from the any specified source channel, or its pixel values are set to the specified constant value *val* (corresponding component *dstOrder*[n] should be set to 3), or its pixel values are not set at all (corresponding component *dstOrder*[n] should be set to the arbitrary value greater than 3). For example, the sequence of channels in source 3-channel image is *A, B, C*, if *dstOrder*[0]=1, *dstOrder*[1]=0, *dstOrder*[2]=1, *dstOrder*[3]=2, then the order of channels in the 4-channel destination image will be *B*, *A*, *B*, *C*; if *dstOrder*[0]=4, *dstOrder*[1]=0, *dstOrder*[2]=1, *dstOrder*[3]=2, then the order of channels in the 4-channel destination image will be *D*, *A*, *B*, *C*, where *D* is a channel whose pixel values are not set.

The function flavors that support image with the alpha channel do not perform operation on it.

Example 4-10 shows how to use the function ippiSwapChannels_8u_C3R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of steps values is less than or equal to 0.
ippStsChannelOrderErr	Indicates an error condition if *dstOrder* is out of the range.

Example 4-10 Using the Function ippiSwapChannels

```
Ipp8u src[12*3] = { 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
                    0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0,
                    0,0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255};
Ipp8u dst[12*3];
IppiSize srcRoi = { 12, 3 };
int order = { 2, 1, 0 }

ippiSwapChannels_8u_C3R ( src, 12, dst, 12, srcRoi, order );

Result:
src     [rgb]

255 0   0 255 0   0 255 0   0 255 0   0
0   255 0   0 255 0   0 255 0   0 255 0
0    0 255 0   0 255 0   0 255 0   0 255

dst     [bgr]
```

Example 4-10 Using the Function `ippiSwapChannels`

```
0    0 255 0   0 255 0   0 255 0   0 255
0   255 0   0 255 0   0 255 0   0 255 0
255 0   0 255 0   0 255 0   0 255 0   0
```

AddRandUniform_Direct

Generates random samples with uniform distribution and adds them to an image data.

Syntax

```
IppStatus ippiAddRandUniform_Direct_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp<datatype> low,
    Ipp<datatype> high, unsigned int* pSeed);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Parameters

pSrcDst	Pointer to the source and destination image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer.
roiSize	Size of the image ROI in pixels.
low	The lower bound for the range of uniformly distributed values.
high	The upper bound for the range of uniformly distributed values.
pSeed	The initial seed value for the pseudo-random number generator.

Description

The function `ippiAddRandUniform_Direct` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function generates samples with uniform distribution over the range [*low, high*] and adds them to a source image pointed to by *pSrcDst*.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image which contains pure noise with uniform distribution, call ippiAddRandUniform_Direct using a source image with zero data as input.

This function operates with ROI (see Regions of Interest in Intel IPP).

The code Example 4-11 illustrates the generation of random samples.

Example 4-11 Random Samples Generatiing

```
IppStatus randUniform( void ) {
   unsigned int seed = 123456;
   Ipp8u img[2048], mn, mx;
   IppiSize roi = { 2048, 1 };
   Ipp64f mean;
   IppStatus st;
   ippiSet_8u_C1R( 0, img, 2048, roi );
   st = ippiAddRandUniform_Direct_8u_C1IR(img, 2048, roi, 0, 255, &seed);
   ippiMean_8u_C1R( img, 2048, roi, &mean );
   ippiMinMax_8u_C1R( img, 2048, roi, &mn, &mx );
   printf( "[%d..%d], mean=%.3f\n", mn, mx, mean );
   return st;
}
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcDstStep* has a zero or negative value.

AddRandGauss_Direct

Generates random samples with Gaussian distribution and adds them to an image data.

Syntax

```
IppStatus ippiAddRandGauss_Direct_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp<datatype> mean,
    Ipp<datatype> stDev, unsigned int* pSeed);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Parameters

pSrcDst	Pointer to the source and destination image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image.
roiSize	Size of the image ROI in pixels.
mean	The mean of the Gaussian distribution.
stDev	The standard deviation of the Gaussian distribution.
pSeed	The initial seed value for the pseudo-random number generator.

Description

The function `ippiAddRandGauss_Direct` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

The function generates samples with Gaussian distribution that have the mean value *mean* and standard deviation *stdev* and adds them to a source image ROI pointed to by *pSrcDst*.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image which contains pure noise with Gaussian distribution, call `ippiAddRandGauss_Direct` using a source image with zero data as input.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrcDst* or *pSeed* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcDstStep* has a zero or negative value.

ImageJaehne

Creates Jaehne test image.

Syntax

```
IppStatus ippiImageJaehne_<mod>(Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	8s_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	8s_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	8s_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

Parameters

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
roiSize	Size of the destination image ROI in pixels.

Description

The function ippiImageJaehne is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates a specific one- or three-channel test image that has been first introduced to digital image processing by B.Jaehne (see [Jae95]).

The destination image pixel values are computed according to the following formula:

$$Dst(x,y) = A*\sin(0.5*IPP_PI* (x_2^2 + y_2^2) / roiSize.height),$$

where

`x,y` are pixel coordinates varying in the range
$0 \le x \le roiSize.width-1, \quad 0 \le y \le roiSize.height-1;$

`IPPI_PI` is the library constant that stands for π value.

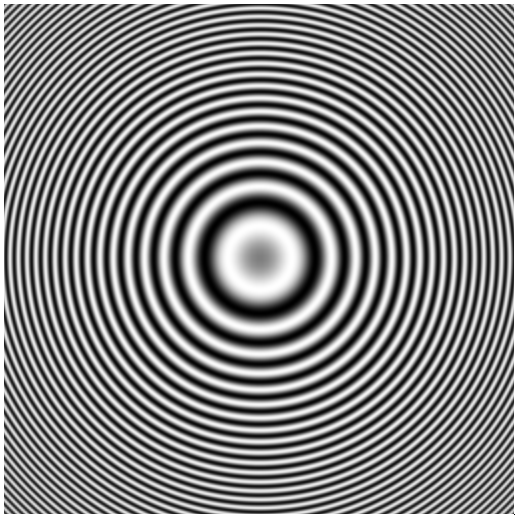
$x_2 = (x-roiSize.width+1)/2.0$,
$y_2 = (y-roiSize.height+1)/2.0;$

`A` is the constant value that depends upon the image type being created.

For the `32f` floating point data the pixel values in the created image can vary in the range between 0 (inclusive) and 1 (exclusive).

Figure 4-5 illustrates an example of a test image generated by the `ippiImageJaehne` function.

Figure 4-5 Example of a Generated Jaehne's Test Image



These test images can be effectively used when you need to visualize and interpret the results of applying filtering functions, similarly to what is proposed in [Jae95].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value, or if *DstStep* is less than or equal to zero.

ImageRamp

Creates a test image that has an intensity ramp.

Syntax

```
IppStatus ippiImageRamp_<mod>(Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, float offset, float slope, IppiAxis axis);
```

Supported values for *mod*:

8u_C1R	8s_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	8s_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	8s_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

Parameters

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
roiSize	Size of the destination image ROI in pixels.
offset	Offset value.
slope	Slope coefficient.
axis	Specifies the direction of the image intensity ramp; can be one of the following: ippAxsHorizontal for the ramp in X-direction, ippAxsVertical for the ramp in Y-direction, ippAxsBoth for ramp in both X and Y-directions.

Description

The function `ippiImageRamp` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

The function creates a one- or three-channel image that can be used as a test image to examine the effect of applying different image processing functions.
The destination image pixel values are computed according to one of the following formulas:

```
  dst(x,y) = offset + slope * x      , if axis = ippAxsHorizontal;
dst(x,y) = offset + slope * y      , if axis = ippAxsVertical;
dst(x,y) = offset + slope * x * y , if axis = ippAxsBoth;
```

where

`x,y` are pixel coordinates varying in the range
$0 \le x \le$ `roiSize.width-1`, $0 \le y \le$ `roiSize.height-1`

Note that linear transform coefficients *offset* and *slope* have floating-point values for all function flavors. The computed pixel values that exceed the image data range are saturated to the respective data-range limits.
The code Example 4-12 illustrates how to use the `ippiImageRamp` function.

Example 4-12 Creating the Test Image with `ippiImageRamp` Function

```
IppStatus ramp( void ) {
   Ipp8u dst[8*4];
   IppiSize roiSize = { 8, 4 };
   return ippiImageRamp_8u_C1R( dst, 8, roiSize, 0.0f, 256.0f/7,
                                                    ippAxsHorizontal );
}
```

```
 The destination image contains the following data:

 00 25 49 6E 92 B7 DB FF
 00 25 49 6E 92 B7 DB FF
 00 25 49 6E 92 B7 DB FF
 00 25 49 6E 92 B7 DB FF
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value, or if *DstStep* is less than or equal to zero.

SampleLine

Stores raster line into buffer.

Syntax

```
IppStatus ippiSampleLine_<mod>(const Ipp<DataType>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<DataType>* pDst, IppiPoint pt1, IppiPoint pt2);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

Parameters

pSrc	Pointer to the ROI in the source raster image.
srcStep	Distance in bytes between starts of consecutive lines in in the raster image.
roiSize	Size of the image ROI.
pDst	Pointer to the destination buffer. The buffer is to store at least max(|*pt2.x* - *pt1.x*| + 1, |*pt2.y* - *pt1.y*| + 1) points.
pt1	Starting point of the line.
pt2	Ending point of the line.

Description

The function `SampleLine` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function iterates through the points that belong to the raster line using the 8-point connected Bresenham algorithm, and stores the resulting pixels in the destination buffer.

The code Example 4-13 illustrates how to use the `ippiSampleLine_8u_C1R` function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when *roiSize.width* or *roiSize.height* is less than or equal to zero.

ippStsStepErr	Indicates an error when *srcStep* is less than *roiSize.width * <pixelSize>*.
ippStsNotEvenStepErr	Indicates an error when the step for the floating-point image cannot be divided by 4.
ippStsOutOfRangeErr	Indicates an error when any of the line ending points is outside the image.

Example 4-13 Using the Function ippiSampleLine

```
void func_sampleline()
{
    Ipp8u pSrc[5*4] = { 0, 1, 2, 3, 4,
                        5, 6, 7, 8, 9,
                        0, 9, 8, 7, 6,
                        5, 4, 3, 2, 1};
    IppiSize roiSize = { 5, 4};
    IppiPoint pt1 = {1, 1};
    IppiPoint pt2 = {2, 3};
    Ipp8u pDst[3];
    int srcStep = 8*sizeof(Ipp8u);

    ippiSampleLine_8u_C1R( pSrc, srcStep, roiSize, pDst, pt1, pt2);
}
Result:   9  3  18
```

ZigzagFwd8x8

Converts a conventional order to the zigzag order.

Syntax

```
IppStatus ippiZigzagFwd8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

Parameters

pSrc	Pointer to the source data.
pDst	Pointer to the destination data.

Description

The function ippiZigzagFwd8x8 is declared in the ippi.h file. This function rearranges data in an 8x8 block from a conventional order (left-to-right, top-to-bottom) to the zigzag

sequence. The zigzag sequence is specified in Figure 4-6.

Figure 4-6 Zigzag Sequence

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

The code Example 4-14 illustrates how to use the `ippiZigzagFwd8x8_16s_C1` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers are NULL.

Example 4-14 Using the Function `ippiZigzagFwd8x8`

```
Ipp16s src[8*8] = {
                    0, 1, 5, 7, 9, 2, 4, 1,
                    5, 4, 8, 6, 3, 8, 0, 3,
                    6, 2, 6, 8, 1, 4, 2, 8,
                    4, 3, 2, 9, 3, 0, 6, 6,
                    7, 7, 3, 0, 4, 1, 0, 9,
                    5, 1, 9, 2, 5, 7, 1, 7,
                    0, 3, 5, 0, 7, 5, 9, 8,
                    2, 9, 1, 4, 6, 8, 2, 3
                  };
Ipp16s dst[8*8];

ippiZigzagFwd8x8_16s_C1 ( src, dst );

Result:
0 1 5 7 9 2 4 1
5 4 8 6 3 8 0 3
6 2 6 8 1 4 2 8
4 3 2 9 3 0 6 6      src   //conventional order
7 7 3 0 4 1 0 9
5 1 9 2 5 7 1 7
```

Example 4-14 Using the Function `ippiZigzagFwd8x8` (continued)

```
0 3 5 0 7 5 9 8
2 9 1 4 6 8 2 3


0 1 5 6 4 5 7 8
2 4 7 3 6 6 9 2
3 8 2 7 5 0 1 3
9 1 8 4 1 0 4 3      dst  //zigzag order
0 9 3 2 9 5 2 4
0 2 3 8 6 1 5 0
1 4 7 7 0 6 9 1
5 6 8 9 7 8 2 3
```

ZigzagInv8x8

Converts a zigzag order to the conventional order.

Syntax

```
IppStatus ippiZigzagInv8x8_16s_C1(const Ipp16s* pSrc,  Ipp16s* pDst);
```

Parameters

pSrc	Pointer to the source data.
pDst	Pointer to the destination data.

Description

The function `ippiZigzagInv8x8` is declared in the `ippi.h` file. This function rearranges data in an 8x8 block from a zigzag order to the conventional order (left-to-right, top-to-bottom). The zigzag sequence is specified in Figure 4-6.

Return Values

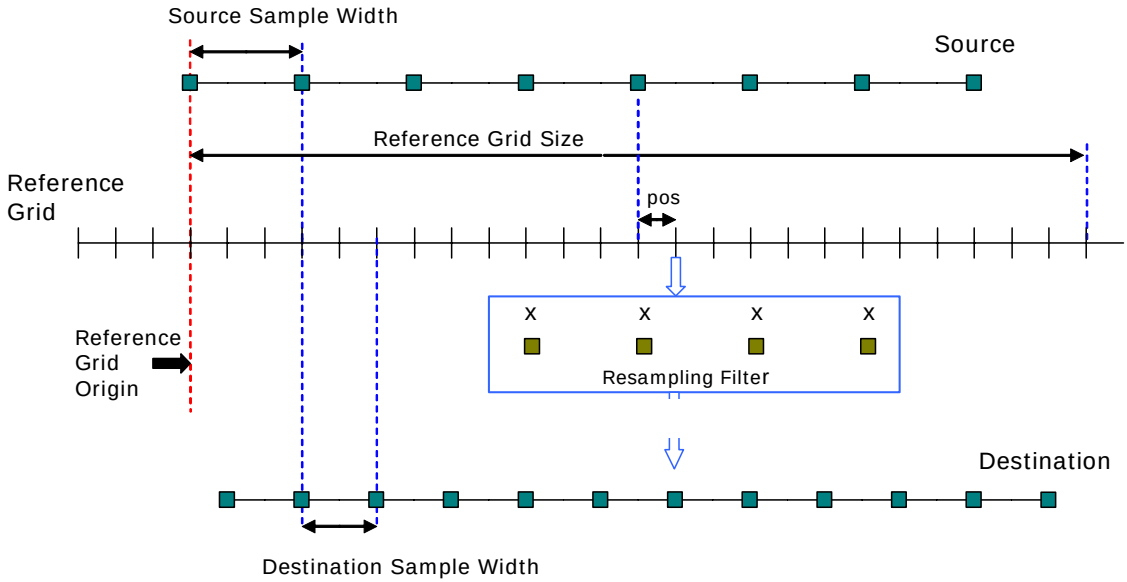
`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers are `NULL`.

Row Oriented Resampling

This section describes the low-level high-optimized functions for image resampling. They can be used for resizing of images with arbitrary rational resizing factor. The functions use anti-aliasing filters in effective polyphase implementation.(The similar implementation is used in signal-processing multi-rate filters.)

For the perfect unshifted block-by-block or parallelized image data processing, the source and destination images are defined with respect to the high-resolution reference grid (see Figure 4-7). Required borders around the source image may either be created manually by the user or automatically by the special flavor of the resampling function.

Figure 4-7 Row Oriented Resampling of an Image



ResampleRowGetSize

Computes the sizes of the resample structure and the work buffer.

Syntax

```
IppStatus ippiResampleRowGetSize_32f(int  srcSampleWidth, int
    dstSampleWidth, int* pSpecSize, int* pBufSize);
```

Parameters

srcSampleWidth	Distance between neighbor pixels of a source row on the reference grid.
dstSampleWidth	Distance between neighbor pixels of a destination row on the reference grid.
pSpecSize	Pointer to the computed size of the structure.
pBufSize	Pointer to the computed size of the buffer.

Description

The function `ippiResampleRowGetSize` is declared in the `ippi.h` file.

This function calculates sizes of the resampling parameters structure `IppiResampleRowSpec` and the work buffer utilized by the functions ippiResampleRow and ippiResampleRowReplicateBorder. These values are stored in the *pSpecSize* and *pBufSize* respectively. Calculations are performed in accordance with the parameters of resampling: widths of source and destination samples. The structure and the buffer must be created by user. Use the function ippiResampleRowInit to initialize the `IppiResampleRowSpec` structure.

This function computes the sizes of the row resampling structure `IppiResampleRowSpec` and of the work buffer. T

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsOutOfRangeErr`	Indicates an error condition if one of the *srcSampleWidth* or *dstSampleWidth* is not valid.

ResampleRowInit

Initializes the row resampling structure.

Syntax

```
IppStatus ippiResampleRowInit_32f(IppiResampleRowSpec_32f* pRowSpec,
    IppiResampleType filterType, int srcSampleWidth, int dstSampleWidth);
```

Parameters

filterType	Specifies the type of the interpolation filter that is used for resampling. Possible value is `ippResampleCatmullRom` for Catmull-Rom interpolation.
srcSampleWidth	Distance between neighbor pixels of a source row on the reference grid.
dstSampleWidth	Distance between neighbor pixels of a destination row on the reference grid.
pRowSpec	Pointer to the initialized structure.

Description

The function `ippiResampleRowInit` is declared in the `ippi.h` file.

This function initializes in the external buffer the resampling specification structure `IppiResampleRowSpec` created by the user. The buffer's size should be computed previously by the function `ippiResampleRowGetSize`. This structure is necessary for the functions `ippiResampleRow` and `ippiResampleRowReplicateBorder`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsBadArgErr`	Indicates an error condition if specified filter type is incorrect or unsupported.
`ippStsOutOfRangeErr`	Indicates an error condition if one of the *srcSampleWidth* or *dstSampleWidth* si not valid.

ResampleRowGetBorderWidth

Computes the width of the left and right borders.

Syntax

```
IppStatus ippiResampleRowGetBorderWidth_32f(const
    IppiResampleRowSpec_32f* pRowSpec, int* pLeftWidth, int*
    pRightWidth);
```

Parameters

pRowSpec	Pointer to the initialized row resampling structure.
pLeftWidth	Pointer to the size (in pixels) of the left border width.
pRightWidth	Pointer to the size (in pixels) of the right border width.

Description

The function `ippiResampleRowGetBorderWidth` is declared in the `ippi.h` file.

This function computes the width of the right and left borders (in pixels) that must be added to the source buffer before resampling with the function `ippiResampleRowReplicateBorder`. The results are stored in the *pRightWidth* and *pLeftWidth* respectively. The width of each border depends on parameters of resampling determined in the `IppiResampleRowSpec` structure.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the *pRowSpec* pointer is `NULL`.
`ippStsBadArgErr`	Indicates an error condition if the structure is corrupted.

ResampleRow, ResampleRowReplicateBorder

Performs row-oriented resampling of the image.

Syntax

```
IppStatus ippiResampleRow_32f_C1(const Ipp32f* pSrc, int srcLineStep,
    Ipp32f* pDst, int dstLineStep, int xOriginRefGrid, IppiSize
    sizeRefGrid, const IppiResampleRowSpec_32f* pRowSpec, Ipp8u*
    pBuffer);
```

```
IppStatus ippiResampleRowReplicateBorder_32f_C1R(const Ipp32f* pSrc, int
    srcLineStep, Ipp32f* pDst, int dstLineStep, int xOriginRefGrid,
    IppiSize sizeRefGrid, const IppiResampleRowSpec_32f* pRowSpec, Ipp8u*
    pBuffer);
```

Parameters

pSrc	Pointer to the source row origin.
srcLineStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination row origin.
dstLineStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
xOriginRefGrid	*x*-coordinate of the image origin in the reference grid.
sizeRefGrid	Size of the image in terms of the reference grid.
pRowSpec	Pointer to the initialized row resampling structure.
pBuffer	Pointer to the work buffer.

Description

The functions `ippiResampleRow` and `ippiResampleRowReplicateBorder` are declared in the `ippi.h` file.

These functions resample one or several rows of the source image *pSrc*. Parameters of resampling (including filter taps) are defined by the `IppiResampleRowSpec` structure. Reference grid is determined by its origin (the *xOriginRefGrid* parameter) and size (the *sizeRefGrid* parameter). Height of the *sizeRefGrid* structure defines number of rows to be resampled. The result is written to the destination image *pDst*.

Function `ippiResampleRow` automatically creates replicated borders around the source image.

Function `ippiResampleRowReplicateBorder` requires left and right replicate borders created by the user around the source image for each row. Use the function `ippiResampleRowGetBorderWidth` to compute width of the borders.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsBadArgErr`	Indicates an error condition if the structure is corrupted.
`ippStsOutOfRangeErr`	Indicates an error condition if the *`xOriginRefGrid`* is out of the range.
`ippStsSizeErr`	Indicates an error condition if the *`sizeRefGrid`* is not valid.

Image Arithmetic and Logical Operations

5

This chapter describes Intel® IPP image processing functions that modify pixel values of an image buffer using arithmetic or logical operations. It also includes functions that perform image compositing based on opacity (alpha-blending).

Table 5-1 lists functions described in more detail later in this chapter:

Table 5-1 Arithmetic and Logical Functions

Function Base Name	Operation
Arithmetic Functions	
Add	Adds pixel values of two image buffers
AddC	Adds a constant to pixel values of an image buffer
AddSquare	Adds squared pixel values of a source image to floating-point pixel values of an accumulator image
AddProduct	Adds product of pixel values of two source images to floating-point pixel values of an accumulator image
AddWeighted	Adds pixel values of a source image multiplied by factor α to floating-point pixel values of an accumulator image multiplied by factor $(1-\alpha)$
Mul	Multiplies pixel values of two image buffers
MulC	Multiplies pixel values of an image buffer by a constant
MulScale	Multiplies pixel values of two image buffers and scales the products
MulCScale	Multiplies pixel values of an image buffer by a constant and scales the products
Sub	Subtracts pixel values of two image buffers

Table 5-1 Arithmetic and Logical Functions (continued)

Function Base Name	Operation
SubC	Subtracts a constant from pixel values of an image buffer
Div	Divides pixel values of two image buffers
Div_Round	Divides pixel values of an image by pixel values of another image with different rounding modes
DivC	Divides pixel values of an image buffer by a constant
Abs	Computes absolute pixel values of a source image and places them into the destination image
AbsDiff	Finds the absolute difference between two images.
AbsDiffC	Finds the absolute difference between an image and a scalar value.
Sqr	Squares pixel values of an image and writes them into the destination image
Sqrt	Computes square roots of pixel values of a source image and writes them into the destination image
Ln	Computes the natural logarithm of pixel values in a source image and writes the results into the destination image
Exp	Computes the exponential of pixel values in a source image and writes the results into the destination image
Complement	Converts negative number from the complement to direct code.
DotProd	Computes the dot product of pixel values of two source images.
DotProdCol	Calculates the dot product of taps vector and columns of the specified set of rows.
Logical Functions	
And	Combines corresponding pixels of two image buffers by a bitwise AND operation.
AndC	Performs a bitwise AND operation on each pixel with a constant.
Or	Combines corresponding pixels of two image buffers by a bitwise OR operation.

Table 5-1 Arithmetic and Logical Functions (continued)

Function Base Name	Operation
OrC	Performs a bitwise OR operation on each pixel with a constant.
Xor	Combines corresponding pixels of two image buffers by a bitwise XOR operation.
XorC	Performs a bitwise OR operation on each pixel with a constant.
Not	Performs a bitwise NOT operation on each pixel
RShiftC	Divides pixel values by a constant power of 2 by shifting bits logically or arithmetically to the right.
LShiftC	Shifts bits in pixel values to the left.
Alpha-Blending Functions	
AlphaComp	Combines two image buffers using alpha (opacity) values.
AlphaCompC	Combines two image buffers using constant alpha values
AlphaPremul	Pre-multiplies pixel values of an image buffer by its alpha values.
AlphaPremulC	Pre-multiplies pixel values of an image buffer using constant alpha values.

An additional suffix C, if present in the function name, indicates operation with a constant. Arithmetic functions that operate on integer data perform fixed scaling of the internally computed results. In case of overflow the result value is saturated to the destination data type range.

NOTE. Most arithmetic and logical functions support data with 1-,3-, or 4-channel pixel values. In the alpha channel case (AC4), the alpha channels are not processed.

Many solutions and hints for use of these functions can be found in Intel IPP Samples. See *Intel IPP Image Processing Samples* downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

Arithmetic Operations

Functions described in this section perform arithmetic operations on pixel values. Arithmetic operations include addition, multiplication, subtraction, and division of pixel values of two images as well as similar operations on a single image and a constant. Computation of absolute value, square, square root, exponential, and natural logarithm of pixels in an image buffer is also supported. Functions of this group perform operations on each pixel in the source buffer(s), and write the results into the destination buffer. Some functions also support processing of images with complex data.

Add

Adds pixel values of two images.

Syntax

Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs		

Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R	32fc_C1R
32f_C3R	32fc_C3R
32f_AC4R	32fc_AC4R
32f_C4R	

Case 3: In-place operation on integer or complex data

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
    int scaleFactor);
```

Supported values for *mod*:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs
8u_C4IRsfs	16u_C4IRSfs	16s_C4IRSfs		

Case 4: In-place operation on floating-point or complex data

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	

Case 5: In-place operation using a floating-point accumulator image

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u32f_C1IR	8s32f_C1IR	16u32f_C1IR

Case 6: Masked in-place operation using a floating-point accumulator image

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u32f_C1IMR	8s32f_C1IMR	16u32f_C1IMR	32f_C1IMR

Parameters

pSrc1, *pSrc2*	Pointer to the ROI in the source images.
src1Step, *src2Step*	Distance in bytes between starts of consecutive lines in the source images.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrc	Pointer to the first source image ROI for the in-place operation.
srcStep	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
pSrcDst	Pointer to the second source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
pMask	Pointer to the mask image ROI for the masked operation.
maskStep	Distance in bytes between starts of consecutive lines in the mask image for the masked operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiAdd` is declared in the `ippi.h` file with the exception of the function flavors in **Case 5** and **Case 6**, which are declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function adds corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by *scaleFactor*. For complex data, the function processes both real and imaginary parts of pixel values.
Some function flavors add `8u`, `8s`, `16u`, or `32f` source image pixel values to a floating point accumulator image in-place. Addition of pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

WARNING. Step values must be positive for functions that operate on complex data, and no less than *roiSize.width**<pixelSize> for functions using an accumulator image.

Note that the functions with AC4 descriptor do not process alpha channel.

Example 5-1 shows how to use the function `ippiAdd_8u_C1RSfs`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition in the following cases: for functions that operate on complex data, if any of the specified step values is zero or negative; for functions using an accumulator image, if any of the specified step values is less than *roiSize.width* * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

Example 5-1 Using the Function `ippiAdd`.

```
Ipp8u src1[8*4] = {8, 4, 2, 1, 0, 0, 0, 0,
                   8, 4, 2, 1, 0, 0, 0, 0,
                   8, 4, 2, 1, 0, 0, 0, 0,
                   8, 4, 2, 1, 0, 0, 0, 0};
Ipp8u src2[8*4] = {4, 3, 2, 1, 0, 0, 0, 0,
                   4, 3, 2, 1, 0, 0, 0, 0,
                   4, 3, 2, 1, 0, 0, 0, 0,
                   4, 3, 2, 1, 0, 0, 0, 0};
Ipp8u dst[8*4];
IppiSize srcRoi = { 4, 4 };
Int scaleFactor = 1;    // later examples for 2 and -2 values

ippiAdd_8u_C1RSfs (src1, 8, src2, 8, dst, 4, srcRoi, scaleFactor );

Result:
    src1                        src2
8 4 2 1 0 0 0 0         4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0         4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0         4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0         4 3 2 1 0 0 0 0

dst >>    scaleFactor = 1        scaleFactor = 2    scaleFactor = -2
            6 4 2 1                3 2 1 0            48 28 16 8
            6 4 2 1                3 2 1 0            48 28 16 8
            6 4 2 1                3 2 1 0            48 28 16 8
            6 4 2 1                3 2 1 0            48 28 16 8
```

AddC

Adds a constant to pixel values of an image.

Syntax

Case 1: Not-in-place operation on 1-channel integer or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs     16u_C1RSfs     16s_C1RSfs     16sc_C1RSfs    32sc_C1RSfs
```

Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C3RSfs     16u_C3RSfs     16s_C3RSfs     16sc_C3RSfs    32sc_C3RSfs
8u_AC4RSfs    16u_AC4RSfs    16s_AC4RSfs    16sc_AC4RSfs   32sc_AC4RSfs
```

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C4RSfs     16u_C4RSfs     16s_C4RSfs
```

Case 3: Not-in-place operation on 1-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1R        32fc_C1R
```

Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3R        32fc_C3R
32f_AC4R       32fc_AC4R
```

```
IppStatus ippiAddC_32f_C4R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 5: In-place operation on 1-channel integer or complex data

```
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs    16u_C1IRSfs   16s_C1IRSfs   16sc_C1IRSfs  32sc_C1IRSfs
```

Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C3IRSfs    16u_C3RSfs    16s_C3IRSfs   16sc_C3IRSfs  32sc_C3IRSfs
8u_AC4IRSfs   16u_AC4RSfs   16s_AC4IRSfs  16sc_AC4IRSfs 32sc_AC4IRSfs
```

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C4IRSfs     16u_C4IRSfs    16s_C4IRSfs
```

Case 7: In-place operation on 1-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR       32fc_C1IR
```

Case 8: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3IR       32fc_C3IR
32f_AC4IR      32fc_AC4IR
```

```
IppStatus ippiAddC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
value	The constant value to add to image pixel values (constant vector in case of multi-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiAddC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function changes the image intensity by adding the *value* to image pixel values. For one-channel images, a positive *value* brightens the image (increases the intensity); a negative *value* darkens the image (decreases the intensity). For multi-channel images, the components of a constant vector *value* are added to pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.

WARNING. Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

AddSquare

Adds squared pixel values of a source image to floating-point pixel values of an accumulator image.

Syntax

Case 1: In-place operation

```
IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u32f_C1IR     8s32f_C1IR     16u32f_C1IR    32f_C1IR
```

Case 2: Masked in-place operation

```
IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u32f_C1IMR    8s32f_C1IMR    16u32f_C1IMR   32f_C1IMR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.

pSrcDst	Pointer to the destination (accumulator) image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the accumulator image.
roiSize	Size of the image ROI in pixels.

Description

The function `ippiAddSquare` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function adds squared pixel values of the source image *pSrc* to floating-point pixel values of the accumulator image *pSrcDst* as follows:

$$pSrcDst(x,y) = pSrcDst(x,y) + pSrc(x,y)^2$$

Addition of the squared pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when *roiSize.width* or *roiSize.height* is negative.
`ippStsStepErr`	Indicates an error when *srcStep*, *maskStep*, or *srcDstStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

AddProduct

Adds product of pixel values of two source images to floating-point pixel values of an accumulator image.

Syntax

Case 1: In-place operation

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int
    src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u32f_C1IR     8s32f_C1IR     16u32f_C1IR    32f_C1IR
```

Case 2: Masked in-place operation

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1,
    int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u32f_C1IMR    8s32f_C1IMR    16u32f_C1IMR   32f_C1IMR
```

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
pSrcDst	Pointer to the destination (accumulator) image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the accumulator image.
roiSize	Size of the image ROI in pixels.

Description

The function `ippiAddProduct` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function adds the product of pixel values of two source images *pSrc1* and *pSrc2* to floating-point pixel values of the accumulator image *pSrcDst* as given by:

*pSrcDst(x,y) = pSrcDst(x,y) + pSrc1(x,y) * pSrc2(x,y)*

Addition of the products of pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when *roiSize.width* or *roiSize.height* is negative.
`ippStsStepErr`	Indicates an error if *src1Step*, *src2Step*, *maskStep*, or *srcDstStep* is less than *roiSize.width * <pixelSize>*.
`ippStsNotEvenStepErr`	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

AddWeighted

Adds weighted pixel values of a source image to floating-point pixel values of an accumulator image.

Syntax

Case 1: In-place operation

```
IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
    Ipp32f alpha);
```

Supported values for *mod*:

```
8u32f_C1IR     8s32f_C1IR     16u32f_C1IR     32f_C1IR
```

Case 2: Masked in-place operation

```
IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

Supported values for *mod*:

```
8u32f_C1IMR    8s32f_C1IMR    16u32f_C1IMR    32f_C1IMR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
pSrcDst	Pointer to the destination (accumulator) image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the accumulator image.
roiSize	Size of the image ROI in pixels.
alpha	Weight α of the source image.

Description

The function `ippiAddWeighted` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function adds pixel values of the source image *pSrc* multiplied by a weight factor α to floating-point pixel values of the accumulator image *pSrcDst* multiplied by $(1-\alpha)$ as follows:

$$pSrcDst(x,y) = pSrcDst(x,y) * (1-\alpha) + pSrc(x,y) * \alpha$$

Addition of the weighted pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error when *roiSize.width* or *roiSize.height* is negative.
ippStsStepErr	Indicates an error when *srcStep, maskStep,* or *srcDstStep* is less than *roiSize.width * <pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

Mul

Multiplies pixel values of two images.

Syntax

Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs		

Case 2: Not-in-place operation on floating-point or complex data.

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int
    dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R	32fc_C1R
32f_C3R	32fc_C3R
32f_AC4R	32fc_AC4R
32f_C3R	

Case 3: In-place operation on integer or complex data

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
    int scaleFactor);
```

Supported values for *mod* :

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs		

Case 4: In-place operation on floating-point or complex data

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source images ROI.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiMul` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

For complex data, the function processes both real and imaginary parts of pixel values.

WARNING. Step values must be positive for functions that operate on complex data.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

MulC

Multiplies pixel values of an image by a constant.

Syntax

Case 1: Not-in-place operation on 1-channel integer or complex data

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

`8u_C1RSfs` `16u_C1RSfs` `16s_C1RSfs` `16sc_C1RSfs` `32sc_C1RSfs`

Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C3RSfs     16u_C3RSfs     16s_C3RSfs     16sc_C3RSfs    32sc_C3RSfs
8u_AC4RSfs    16u_AC4RSfs    16s_AC4RSfs    16sc_AC4RSfs   32sc_AC4RSfs
```

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C4RSfs      16u_C4RSfs      16s_C4RSfs
```

Case 3: Not-in-place operation on 1-channel floating-point or complex data

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod* :

```
32f_C1R        32fc_C1R
```

Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
    Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3R        32fc_C3R
32f_AC4R       32fc_AC4R
```

```
IppStatus ippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 5: In-place operation on 1-channel integer or complex data

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs    16u_C1IRSfs    16s_C1IRSfs    16sc_C1IRSfs   32sc_C1IRSfs
```

Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C3IRSfs     16u_C3IRSfs    16s_C3IRSfs    16sc_C3IRSfs   32sc_C3IRSfs
8u_AC4IRSfs    16u_AC4IRSfs   16s_AC4IRSfs   16sc_AC4IRSfs  32sc_AC4IRSfs
```

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C4IRSfs     16u_C4IRSfs     16s_C4IRSfs
```

Case 7: In-place operation on 1-channel floating-point or complex data

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR       32fc_C1IR
```

Case 8: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3IR       32fc_C3IR
32f_AC4IR      32fc_AC4IR
```

```
IppStatus ippiMulC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
value	The constant value to add to image pixel values (constant vector in case of multi-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.

pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiMulC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies pixel values of an image by a constant *value*. For multi-channel images, pixel channel values are multiplied by the components of a constant vector *value*. For complex data, the function processes both real and imaginary parts of pixel values.

WARNING. Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

MulScale

Multiplies pixel values of two images and scales the products.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

Case 2: In-place operation

```
IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR
8u_C3IR	16u_C3IR
8u_C4IR	16u_C4IR
8u_AC4IR	16u_AC4IR

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source images ROI.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.

srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiMulScale` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies corresponding pixel values of two input buffers and scales the products using the following formula:

```
dst_pixel = src1_pixel * src2_pixel / max_val,
```

where `src1_pixel` and `src2_pixel` are pixel values of the source buffers, `dst_pixel` is the resultant pixel value, and `max_val` is the maximum value of the pixel data range (see Table 2-2 in Chapter 2 for details).
The function is implemented for 8-bit and 16-bit unsigned data types only.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

MulCScale

Multiplies pixel values of an image by a constant and scales the products.

Syntax

Case 1: Not-in-place operation on 1-channel data

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize
    roiSize);
```

Supported values for *mod*:

```
8u_C1R          16u_C1R
```

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3],  Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3R          16u_C3R
8u_AC4R         16u_AC4R
```

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4],  Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C4R          16u_C4R
```

Case 3: In-place operation on 1-channel data

```
IppStatus ippiMulCScale_<mod>(Ipp<datatype> value,
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1IR         16u_C1IR
```

Case 4: In-place operation on multi-channel data

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[3], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3IR        16u_C3IR
8u_AC4IR       16u_AC4IR
```

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[4], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C4IR        16u_C4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
value	The constant value to multiply each pixel value in a source buffer (constant vector in case of 3- or 4-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiMulCScale` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies pixel values in the input buffer by a constant *value* and scales the products using the following formula:

```
dst_pixel = src_pixel * value / max_val,
```

where `src_pixel` is a pixel value in the source buffer, `dst_pixel` is the resultant pixel value, and `max_val` is the maximum value of the pixel data range (see Table 2-2 in Chapter 2 for details).
The function is implemented for 8-bit and 16-bit unsigned data types only. It can be used to multiply pixel values by a number between 0 and 1.
Note that the functions with AC4 descriptor do not process alpha channel.

Example 5-2 shows how to use the function `ippiMulCScale_8u_C1R`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

Example 5-2 Using the Function ippiMulCScale.

```
void func_mulcscale()
{
    IppiSize ROI = {8,4};
    IppiSize ROI2 = {5,4};
    Ipp8u src[8*4];
    Ipp8u dst[8*4];
    Ipp8u v = 100;

    ippiSet_8u_C1R(100,src,8,ROI);
    ippiSet_8u_C1R(0,dst,8,ROI);
    ippiMulCScale_8u_C1R(src,8,v,dst,8,ROI2);
}
Result:
            src1                                dst
100 100 100 100 100 100 100 100          39 39 39 39 39 0 0 0
100 100 100 100 100 100 100 100          39 39 39 39 39 0 0 0
100 100 100 100 100 100 100 100          39 39 39 39 39 0 0 0
100 100 100 100 100 100 100 100          39 39 39 39 39 0 0 0
```

Sub

Subtracts pixel values of two images.

Syntax

Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs

```
8u_AC4RSfs    16u_AC4RSfs    16s_AC4RSfs    16sc_AC4RSfs   32sc_AC4RSfs
8u_C4RSfs     16u_C4RSfs     16s_C4RSfs
```

Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1R    32fc_C1R
32f_C3R    32fc_C3R
32f_AC4R   32fc_AC4R
32f_C4R
```

Case 3: In-place operation on integer or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
    int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs     16u_C1IRSfs    16s_C1IRSfs    16sc_C1IRSfs   32sc_C1IRSfs
8u_C3IRSfs     16u_C3IRSfs    16s_C3IRSfs    16sc_C3IRSfs   32sc_C3IRSfs
8u_AC4IRSfs    16u_AC4IRSfs   16s_AC4IRSfs   16sc_AC4IRSfs  32sc_AC4IRSfs
8u_C4IRSfs     16u_C4IRSfs    16s_C4IRSfs
```

Case 4: In-place operation on floating-point or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR       32fc_C1IR
32f_C3IR       32fc_C3IR
32f_AC4IR      32fc_AC4IR
32f_C4IR
```

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source images ROI.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiSub` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function subtracts pixel values of the source buffer *pSrc1* from the corresponding pixel values of the buffer *pSrc2* and places the result in a destination buffer *pDst*. For in-place operations, the values in *pSrc* are subtracted from the values in *pSrcDst* and the results are placed into *pSrcDst*. For complex data, the function processes both real and imaginary parts of pixel values.

WARNING. Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by *scaleFactor*. Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

SubC

Subtracts a constant from pixel values of an image.

Syntax

Case 1: Not-in-place operation on 1-channel integer or complex data

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs

Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

Case 3: Not-in-place operation on 1-channel floating-point or complex data

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R	32fc_C1R

Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
    Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize
    roiSize);
```

Supported values for *mod*:

```
32f_C3R        32fc_C3R
32f_AC4R       32fc_AC4R
```

```
IppStatus ippiSubC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 5: In-place operation on 1-channel integer or complex data

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs    16u_C1IRSfs   16s_C1IRSfs   16sc_C1IRSfs  32sc_C1IRSfs
```

Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C3IRSfs    16u_C3IRSfs   16s_C3IRSfs   16sc_C3IRSfs  32sc_C3IRSfs
8u_AC4IRSfs   16u_AC4IRSfs  16s_AC4IRSfs  16sc_AC4IRSfs 32sc_AC4IRSfs
```

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C4IRSfs     16u_C4IRSfs    16s_C4IRSfs
```

Case 7: In-place operation on 1-channel floating-point or complex data

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR       32fc_C1IR
```

Case 8: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3IR       32fc_C3IR
32f_AC4IR      32fc_AC4IR
```

```
IppStatus ippiSubC_32f_C4IR(const Ipp32f> value[4], Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
value	The constant value to subtract from each pixel value in a source buffer (constant vector in case of multi-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiSubC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function changes image intensity by subtracting the constant *value* from pixel values of an image buffer. For multi-channel images, the components of a constant vector *value* are subtracted from pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.

WARNING. Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

Div

Divides pixel values of an image by pixel values of another image.

Syntax

Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs		

Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int
    dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R	32fc_C1R
32f_C3R	32fc_C3R
32f_AC4R	32fc_AC4R
32f_C4R	

Case 3: In-place operation on integer or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
    int scaleFactor);
```

Supported values for *mod*:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs		

Case 4: In-place operation on floating-point or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source images ROI.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function ippiDiv is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function divides pixel values of the source buffer *pSrc2* by the corresponding pixel values of the buffer *pSrc1* and places the result in a destination buffer *pDst*. For in-place operations, the values in *pSrcDst* are divided by the values in *pSrc* and placed into *pSrcDst*. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by *scaleFactor* and rounded (not truncated).
When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix A "Handling of Special Cases" for more information).

Note that the functions with AC4 descriptor do not process alpha channel.

The code Example 5-3 illustrates how the function ippiDiv can be used.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.
ippStsDivByZero	Indicates a warning that a divisor value is zero. The function execution is continued.

Example 5-3 Division of Pixel Values

```
IppStatus div32f( void ) {
   Ipp32f a[4*3], b[4*3];
   IppiSize roi = {2,2};
   int i;
   for( i=0; i<4*3; ++i ) a[i] = b[i] = (float)i;
   return ippiDiv_32f_C1IR( a, 4*sizeof(Ipp32f), b,
                            4*sizeof(Ipp32f), roi );
}
The destination image b contains
-1.#IND  +1.000  +2.000  +3.000
+1.000   +1.000  +6.000  +7.000
+8.000   +9.000  +10.00  +11.00
Console output:
-- warning in div32f: (6) Zero value(s) of divisor in the function Div
```

Div_Round

Divides pixel values of an image by pixel values of another image with different rounding modes.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiDiv_Round_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int
    dstStep, IppiSize roiSize, IppRoundMode rndMode, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

Case 2: In-place operation on integer data

```
IppStatus ippiDiv_Round_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
    IppRoundMode rndMode, int scaleFactor);
```

Supported values for *mod*:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source images ROI.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.

srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.	
roiSize	Size of the source and destination ROI in pixels.	
roundMode	Rounding mode, the following values are possible:	
	ippRndZero	Specifies that floating-point values must be truncated toward zero.
	ippRndNear	Specifies that floating-point values must be rounded to the nearest even integer.
	ippRndFinancial	Specifies that floating-point values must be rounded down to the nearest integer if decimal value is less than 0.5, or rounded up to the nearest integer if decimal value is equal or greater than 0.5.
scaleFactor	Scale factor (see Integer Result Scaling).	

Description

The function ippiDiv_Round is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function divides pixel values of the source buffer *pSrc2* by the corresponding pixel values of the buffer *pSrc1* and places the result in a destination buffer *pDst*. For in-place operations, the values in *pSrcDst* are divided by the values in *pSrc* and placed into *pSrcDst*. The resulting values are scaled by *scaleFactor* and rounded using the rounding method specified by the parameter *roundMode*.
When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix A "Handling of Special Cases" for more information).

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

ippStsDivByZero	Indicates a warning that a divisor value is zero. The function execution is continued.
ippStsStsRoundModeNotSupportedErr	Indicates an error condition if the *roundMode* has an illegal value.

DivC

Divides pixel values of an image by a constant.

Syntax

Case 1: Not-in-place operation on 1-channel integer or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs

Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

Case 3: Not-in-place operation on 1-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1R        32fc_C1R
```

Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
    Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize
    roiSize);
```

Supported values for *mod*:

```
32f_C3R        32fc_C3R
32f_AC4R       32fc_AC4R
```

```
IppStatus ippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    val[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 5: In-place operation on 1-channel integer or complex data

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs     16u_C1IRSfs    16s_C1IRSfs    16sc_C1IRSfs   32sc_C1IRSfs
```

Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C3IRSfs     16u_C3IRSfs    16s_C3IRSfs    16sc_C3IRSfs   32sc_C3IRSfs
8u_AC4IRSfs    16u_AC4IRSfs   16s_AC4IRSfs   16sc_AC4IRSfs  32sc_AC4IRSfs
```

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C4IRSfs     16u_C4IRSfs    16s_C4IRSfs
```

Case 7: In-place operation on 1-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR       32fc_C1IR
```

Case 8: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3IR        32fc_C3IR
32f_AC4IR       32fc_AC4IR
```

```
IppStatus ippiDivC_32f_C4IR(const Ipp32f val[4], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
value	The constant value to divide each pixel value in a source buffer (constant vector in case of 3- or 4-channel images).
pDst	Pointer to the destination buffer.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiDivC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function changes image intensity by dividing pixel values of an image buffer by the constant *value*. For multi-channel images, pixel channel values are divided by the components of a constant vector *value*. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by *scaleFactor* and rounded (not truncated).

When the divisor value is zero, the function execution is aborted and the `ippStsDivByZeroErr` error status is set.
Note that in the alpha channel case (AC4), the alpha channels are not processed.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.
`ippStsDivByZeroErr`	Indicates an error condition if the divisor value is zero.

Abs

Computes absolute pixel values of a source image and places them into the destination image.

Case 1: Not-in-place operation

```
IppStatus ippiAbs_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
16s_C1R     32f_C1R
16s_C3R     32f_C3R
16s_C4R     32f_C4R
16s_AC4R    32f_AC4R
```

Case 2: In-place operation

```
IppStatus ippiAbs_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
16s_C1IR    32f_C1IR
16s_C3IR    32f_C3IR
16s_C4IR    32f_C4IR
16s_AC4IR   32f_AC4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiAbs` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function takes the absolute value of each channel in each pixel of the source image ROI and places the result into a destination image ROI. It operates on signed data only. Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *roiSize* has a field with zero or negative value.

AbsDiff

Calculates absolute difference between two images.

Syntax

```
IppStatus ippiAbsDiff_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R      16u_C1R     32f_C1R
8u_C3R
```

Parameters

pSrc1	Pointer to the first source image.
src1Step	Step in bytes in the first source image.
pSrc2	Pointer to second source image.
src2Step	Step in bytes in the second source image.
pDst	Pointer to the destination image.
dstStep	Step in bytes in the destination image.
roiSize	Size of the image ROI in pixels.

Description

The function `ippiAbsDiff` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the absolute pixel-wise difference between two images by the formula:

$$pDst(x, y) = abs(pSrc1(x, y) - pSrc2(x, y)) \quad .$$

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error when *src1Step*, *src2Step* or *dstStep* is less than *roiSize.width * <pixelSize>*.
`ippStsNotEvenStepErr`	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

AbsDiffC

Calculates absolute difference between image and scalar value.

Syntax

```
IppStatus ippiAbsDiffC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int value);
```

Supported values for *mod*:

```
8u_C1R      16u_C1R
```

```
IppStatus ippiAbsDiffC_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f value);
```

Parameters

pSrc	Pointer to the source image.
srcStep	Step in bytes in the source image.
pDst	Pointer to the destination image.
dstStep	Step in bytes in the destination image.
roiSize	Size of the image ROI in pixels.
value	Scalar value used to decrement each element of the source image.

Description

The function `ippiAbsDiffC` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the absolute pixel-wise difference between the source image *pSrc* and the scalar *value* by the formula:

$$pDst(x, y) = abs(pSrc(x, y) - value) \quad .$$

The function clips the *value* to the range [0, 255] for `8u` data type, and to the range [0, 65535] for `16u` data type.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if the *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error when *srcStep* or *dstStep* is less than *roiSize.width * <pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

Sqr

Squares pixel values of an image and writes them into the destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiSqr_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiSqr_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Case 3: In-place operation on integer data

```
IppStatus ippiSqr_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs      16u_C1IRSfs     16s_C1IRSfs
8u_C3IRSfs      16u_C3IRSfs     16s_C3IRSfs
8u_C4IRSfs      16u_C4IRSfs     16s_C4RISfs
8u_AC4IRSfs     16u_AC4IRSfs    16s_AC4IRSfs
```

Case 4: In-place operation on floating-point data

```
IppStatus ippiSqr_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiSqr` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function squares pixel values of the source image ROI and writes them to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *roiSize* has a field with zero or negative value.

Sqrt

Computes square roots of pixel values of a source image and writes them into the destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiSqrt_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiSqrt_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R

32f_C3R

32f_AC4R

Case 3: In-place operation on integer data

```
IppStatus ippiSqrt_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs      16u_C1IRSfs     16s_C1IRSfs
8u_C3IRSfs      16u_C3IRSfs     16s_C3IRSfs
8u_AC4IRSfs     16u_AC4IRSfs    16s_AC4IRSfs
```

Case 4: In-place operation on floating-point data

```
IppStatus ippiSqrt_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiSqrt` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes square roots of pixel values of the source image ROI and writes them into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

If a source pixel value is negative, the function issues a warning and continues execution with the corresponding result value (see appendix A "Handling of Special Cases" for more information).

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *roiSize* has a field with zero or negative value.
ippStsSqrtNegArg	Indicates a warning that a source pixel has a negative value.

Ln

Computes the natural logarithm of pixel values in a source image and writes the results into the destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiLn_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs      16u_C1RSfs     16s_C1RSfs
8u_C3RSfs      16u_C3RSfs     16s_C3RSfs
```

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiLn_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
```

Case 3: In-place operation on integer data

```
IppStatus ippiLn_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs      16u_C1IRSfs     16s_C1IRSfs
8u_C3IRSfs      16u_C3IRSfs     16s_C3IRSfs
```

Case 4: In-place operation on floating-point data

```
IppStatus ippiLn_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiLn` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes natural logarithms of pixel values of the source image ROI and writes the resultant values to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

If a source pixel value is zero or negative, the function issues a corresponding warning and continues execution with the corresponding result value (see appendix A "Handling of Special Cases" for more information).

When several inputs have zero or negative value, the status code returned by the function corresponds to the first encountered case as illustrated in the code Example 5-4.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.
ippStsLnZeroArg	Indicates a warning that a source pixel has a zero value.
ippStsLnNegArg	Indicates a warning that a source pixel has a negative value.

Example 5-4 Using the Logarithm Function

```
IppStatus ln( void ) {
   Ipp32f img[8*8];
   IppiSize roi = { 8, 8 };
   IppStatus st;
   ippiSet_32f_C1R( (float)IPP_E, img, 8*4, roi );
   img[0] = -0;
   img[1] = -1;
   st = ippiLn_32f_C1IR( img, 8*sizeof(Ipp32f), roi );
   printf( "%f %f %f\n", img[0], img[1], img[2] );
   return st;
}
Output values:
-1.#INF00 -1.#IND00 1.000000

Status value and message:
(7) Zero value(s) of argument in the Ln function
```

Exp

Computes the exponential of pixel values in a source image and writes the results into the destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiExp_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs      16u_C1RSfs     16s_C1RSfs
8u_C3RSfs      16u_C3RSfs     16s_C3RSfs
```

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiExp_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
```

Case 3: In-place operation on integer data

```
IppStatus ippiExp_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs     16u_C1IRSfs    16s_C1IRSfs
8u_C3IRSfs     16u_C3IRSfs    16s_C3IRSfs
```

Case 4: In-place operation on floating-point data

```
IppStatus ippiExp_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiExp` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes *e* to the power of pixel values of the source image ROI and writes the resultant values into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI. When the overflow occurs, the resultant value is determined in accordance with the data type (see appendix A "Handling of Special Cases" for more information).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.

Complement

Converts negative number from the complement to direct code.

Syntax

```
IppStatus ippiComplement_32s_C1IR(Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Parameters

pSrcDst	Pointer to the source and destination image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiComplement` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts negative integer number from the complement to direct code reserving the sign in the most significant bit.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrcDst* pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcDstStep* has zero or negative value.
`ippStsStrideErr`	Indicates an error condition if *srcDstStep* is less than the image width.

DotProd

Computes the dot product of pixel values of two source images.

Syntax

Case 1: Operation on 1-channel integer data

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step,
    const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f* pDp);
```

Supported values for *mod*:

```
8u64f_C1R   8s64f_C1R   16u64f_C1R  16s64f_C1R  32u64f_C1R  32s64f_C1R
```

Case 2: Operation on 3-channel integer data

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step,
    const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f* pDp[3]);
```

Supported values for *mod*:

```
8u64f_C3R   8s64f_C3R   16u64f_C3R  16s64f_C3R  32u64f_C3R  32s64f_C3R
```

Case 3: Operation on 4-channel integer data

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step,
    const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f* pDp[4]);
```

Supported values for *mod*:

```
8u64f_C4R   8s64f_C4R   16u64f_C4R  16s64f_C4R  32u64f_C4R  32s64f_C4R
8u64f_AC4R  8s64f_AC4R  16u64f_AC4R 16s64f_AC4R 32u64f_AC4R 32s64f_AC4R
```

Case 4: Operation on floating-point data

```
IppStatus ippiDotProd_32f64f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp,
    IppHintAlgorithm hint);
IppStatus ippiDotProd_32f64f_C3R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp[3],
    IppHintAlgorithm hint);
```

```
IppStatus ippiDotProd_32f64f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp[4],
    IppHintAlgorithm hint);
IppStatus ippiDotProd_32f64f_AC4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp[4],
    IppHintAlgorithm hint);
```

Parameters

pSrc1, *pSrc2*	Pointer to the ROI in the source images.
src1Step, *src2Step*	Distance in bytes between starts of consecutive lines in the source images.
pDp	Pointer to the dot product or to the array containing the computed dot products for multi-channel images.
hint	Option to select the algorithmic implementation of the function, see Table 11-3.

Description

The function `ippiDotProd` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the dot product of pixel values of two source images *pSrc1* and *pSrc2* using algorithm indicated by the *hint* argument (see Table 11-3) and stores the result in *pDp*. In case of multi-channel images, the dot product is computed for each channel separately and stored in the array *pDp*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of the step values has zero or negative value.

DotProdCol

Calculates the dot product of taps vector and columns of the specified set of rows.

Syntax

```
IppStatus ippiDotProdCol_32f_L2(const Ipp32f* const ppSrcRow[], const
    Ipp32f* pTaps, int tapsLen, Ipp32f* pDst, int width);
```

Parameters

ppSrcRow	Pointer to the set of rows.
pTaps	Pointer to the taps vector.
tapsLen	Length of taps vector, is equal to the number of rows.
pDst	Pointer to the destination row.
width	Width of the source and destination rows.

Description

The function `ippiDotProdCol` is declared in the `ippi.h` file.

This function calculates the dot product of taps vector *pTaps* and columns of the specified set of the rows *ppSrcRow*. It is useful for external vertical filtering pipeline implementation.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *width* is less than or equal to 0.

Logical Operations

Functions described in this section perform bitwise operations on pixel values. The operations include logical AND, NOT, inclusive OR, exclusive OR, and bit shifts.

And

Performs a bitwise AND operation between corresponding pixels of two images.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiAnd_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

Case 2: In-place operation

```
IppStatus ippiAnd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source images ROI.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI

dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiAnd` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise AND operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

AndC

Performs a bitwise AND operation of each pixel with a constant.

Syntax

Case 1: Not-in-place operation on 1-channel data

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R          16u_C1R         32s_C1R
```

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3R          16u_C3R         32s_C3R
8u_AC4R         16u_AC4R        32s_AC4R
```

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C4R          16u_C4R         32s_C4R
```

Case 3: In-place operation on 1-channel data

```
IppStatus ippiAndC_<mod>(Ipp<datatype> value, const Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1IR         16u_C1IR        32s_C1IR
```

Case 4: In-place operation on multi-channel data

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[3], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3IR         16u_C3IR        32s_C3IR
8u_AC4IR        16u_AC4IR       32s_AC4IR
```

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[4], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_AC4IR        16u_AC4IR       32s_AC4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
value	The constant value to perform the bitwise AND operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiAndC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise AND operation between each pixel value of a source image ROI and constant *value*.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.

Not

Performs a bitwise NOT operation on each pixel of a source buffer.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiNot_<mod>(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R
8u_C3R
8u_C4R
8u_AC4R
```

Case 2: In-place operation

```
IppStatus ippiNot_<mod>(Ipp8u* pSrcDst, int srcDstStep, IppiSize
    roiSize);
```

Supported values for *mod*:

```
8u_C1IR
8u_C3IR
8u_C4IR
8u_AC4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.

roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiNot` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise NOT operation on each pixel value of a source image ROI.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.

Or

Performs bitwise inclusive OR operation between pixels of two source buffers.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

Case 2: In-place operation

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source images ROI.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiOr` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise inclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.
Note that the functions with AC4 descriptor do not process alpha channel.

Example 5-5 show how to use the function `ippiOr_8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

Example 5-5 Using the Function ippiOr.

```
void func_or()
{
    IppiSize Src1ROI = {8,4};
    IppiSize Src2ROI = {8,4};
    IppiSize DstROI = {5,4};
    Ipp8u src1[8*4];
    Ipp8u src2[8*4];
    Ipp8u dst[8*4];

    ippiSet_8u_C1R(0,dest,8,Src1ROI);
    ippiSet_8u_C1R(5,src1,8,Src1ROI);
    ippiSet_8u_C1R(6,src2,8,Src2ROI);
    ippiOr_8u_C1R(src1,8,src2,8,dst,8,DstROI);

}
Result:
          src1                          src2                      dst
05 05 05 05 05 05 05 05   06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00
05 05 05 05 05 05 05 05   06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00
05 05 05 05 05 05 05 05   06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00
05 05 05 05 05 05 05 05   06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00
```

OrC

Performs a bitwise inclusive OR operation between each pixel of a buffer and a constant.

Syntax

Case 1: Not-in-place operation on 1-channel data

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R          16u_C1R         32s_C1R
```

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C4R	16u_C4R	32s_C4R

Case 3: In-place operation on 1-channel data

```
IppStatus ippiOrC_<mod>(Ipp<datatype> value,
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[3],
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[4],
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C4IR	16u_C4IR	32s_C4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.

value	The constant value to perform the bitwise OR operation on each pixel of the source buffer (constant vector in case of multi-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiOrC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise inclusive OR operation between each pixel value of a source image ROI and constant *value*.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.

Xor

Performs bitwise exclusive OR operation between pixels of two source buffers.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiXor_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

Case 2: In-place operation

```
IppStatus ippiXor_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source image ROIs.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.

srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiXor` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise exclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

XorC

Performs a bitwise exclusive OR operation between each pixel of a buffer and a constant.

Syntax

Case 1: Not-in-place operation on 1-channel data

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R          16u_C1R         32s_C1R
```

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C4R	16u_C4R	32s_C4R

Case 3: In-place operation on 1-channel data

```
IppStatus ippiXorC_<mod>(Ipp<datatype> value,
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[3], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[4], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C4IR	16u_C4IR	32s_C4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.

value	The constant value to perform the bitwise XOR operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiXorC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise exclusive OR operation between each pixel value of a source image ROI and constant *value*.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.

LShiftC

Shifts bits in pixel values to the left.

Syntax

Case 1: Not-in-place operation on 1-channel data

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp32u value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R      16u_C1R     32s_C1R
```

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp32u value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3R      16u_C3R     32s_C3R
8u_AC4R     16u_AC4R    32s_AC4R
```

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp32u value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C4R      16u_C4R     32s_C4R
```

Case 3: In-place operation on 1-channel data

```
IppStatus ippiLShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1IR     16u_C1IR    32s_C1IR
```

Case 4: In-place operation on multi-channel data

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3IR     16u_C3IR     32s_C3IR
8u_AC4IR    16u_AC4IR    32s_AC4IR
```

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C4IR     16u_C4IR     32s_C4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
value	The number of bits to shift (constant vector in case of multi-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiLShiftC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function changes the intensity of pixels in the source image ROI by shifting the bits in each pixel value by *value* bits to the left. In case of multi-channel data, each color channel can have its own shift value. The positions vacated after shifting the bits are filled with zeros. Values obtained as a result of left shift operations are not saturated. To get saturated values, use multiplication functions instead.
Note that the functions with AC4 descriptor do not process alpha channel.

The code example below illustrates the use of left shift function.

Example 5-6 Shifting Bits to the Left

```
IppStatus lshift( void ) {
   Ipp8u img[8*8] = { 1, 0x7F, 0xFE };
   IppiSize roi = { 8, 8 };
   IppStatus st = ippiLShiftC_8u_C1IR( 1, img, 8, roi );
   printf( "%02x %02x %02x\n", img[0], img[1], img[2] );
   return st;
}
```

Output values:

```
02 fe fc
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.

RShiftC

Shifts bits in pixel values to the right.

Case 1: Not-in-place operation on 1-channel data

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
   Ipp32u value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R 8s_C1R 16u_C1R 16s_C1R 32s_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp32u value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	8s_C3R	16u_C3R	16s_C3R	32s_C3R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R	32s_AC4R

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp32u value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C4R	8s_C4R	16u_C4R	16s_C4R	32s_C4R

Case 3: In-place operation on 1-channel data

```
IppStatus ippiRShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	8s_C1IR	16u_C1IR	16s_C1IR	32s_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3IR	8s_C3IR	16u_C3IR	16s_C3IR	32s_C3IR
8u_AC4IR	8s_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C4IR	8s_C4IR	16u_C4IR	16s_C4IR	32s_C4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
value	The number of bits to shift (constant vector in case of multi-channel images).

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRShiftC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function decreases the intensity of pixels in the source image ROI by shifting the bits in each pixel value by *value* bits to the right. The positions vacated after shifting the bits are filled with the sign bit. In case of multi-channel data, each color channel can have its own shift value. This operation is equivalent to dividing the pixel values by a constant power of 2.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.

Alpha Composition

The Intel IPP provides functions that composite two image buffers using either the opacity (alpha) channel in the images or provided alpha values.

These functions operate on image buffers with 8-bit or 16-bit data in RGB or RGBA format. In all compositing operations a resultant pixel in destination buffer *pDst* is created by overlaying a pixel from the foreground image buffer *pSrc1* over a pixel from the background image buffer *pSrc2*. The supported types of images' combining by using alpha values are listed in Table 5-2.

Table 5-2 Types of Image Compositing Operations

Type	Output Pixel		Description in Imaging Terms
	Color Components	Alpha value	
OVER	$\alpha_A*A+(1-\alpha_A)*\alpha_B*B$	$\alpha_A+(1-\alpha_A)*\alpha_B$	A occludes B
IN	$\alpha_A*A*\alpha_B$	$\alpha_A*\alpha_B$	A within B. A acts as a matte for B. A shows only where B is visible.
OUT	$\alpha_A*A*(1-\alpha_B)$	$\alpha_A*(1-\alpha_B)$	A outside B. NOT-B acts as a matte for A. A shows only where B is not visi-ble.
ATOP	$\alpha_A*A*\alpha_B+(1-\alpha_A)*\alpha_B*B$	$\alpha_A*\alpha_B+(1-\alpha_A)*\alpha_B$	Combination of (A IN B) and (B OUT A). B is both back-ground and matte for A.
XOR	$\alpha_A*A*(1-\alpha_B)+(1-\alpha_A)*\alpha_B*B$	$\alpha_A*(1-\alpha_B)+(1-\alpha_A)*\alpha_B$	Combination of (A OUT B) and (B OUT A). A and B mutually exclude each other.
PLUS	$\alpha_A*A+\alpha_B*B$	$\alpha_A+\alpha_B$	Blend without precedence

In the formulas above, we denote for simplicity the input image buffers as A and B. The Greek letter α with subscripts denotes the normalized (scaled) alpha value in the range 0 to 1. It is related to the integer alpha value *alpha* as:

$$\alpha = alpha / max_val$$

where *max_val* is 255 for 8-bit or 65535 for 16-bit unsigned pixel data.

For the ippiAlphaComp function that operates on 4-channel RGBA buffers only, α_A and α_B are the normalized alpha values of the two input image buffers, respectively.
For the ippiAlphaCompC function, α_A and α_B are the normalized constant alpha values that are passed as parameters to the function.
Note that in formulas for computing the resultant color channel values, A and B stand for the pixel color components of the respective input image buffers.

To save a significant amount of computation for some of the alpha compositing operations, use functions ippiAlphaPremul, ippiAlphaPremulC for pre-multiplying color channel values by the alpha values. This reduces the number of multiplications required in the compositing operations, which is especially efficient for a repeated compositing of an image.

The type of composition operation that is performed by the function ippiAlphaComp and ippiAlphaCompC is specified by the parameter *alphaType*, Table 5-3 lists its possible values.

Table 5-3 Possible Values of the Parameter *alphaType*

Operation Type	Parameter Value	
OVER	ippAlphaOver	ippAlphaOverPremul
IN	ippAlphaIn	ippAlphaInPremul
OUT	ippAlphaOut	ippAlphaOutPremul
ATOP	ippAlphaATop	ippAlphaATopPremul
XOR	ippAlphaXor	ippAlphaXorPremul
PLUS	ippAlphaPlus	ippAlphaPlusPremul

AlphaComp

Combines two images using alpha (opacity) values of both images.

Syntax

```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for *mod*:

```
8u_AC1R      8u_AC4R
16u_AC1R     16u_AC4R
```

```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* const pSrc1[4],
    int src1Step, const Ipp<datatype>* const pSrc2[4], int src2Step,
    Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize,
    IppiAlphaType alphaType);
```

Supported values for *mod*:

```
8u_AP4R      16u_AP4R
```

Parameters

pSrc1, pSrc2	Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
roiSize	Size of the source and destination ROI in pixels.
alphaType	The composition type to perform. See Table 5-3 for the type value and description.

Description

The function `ippiAlphaComp` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs an image compositing operation on RGBA images using alpha values of both images. The compositing is done by overlaying pixels (r_A ,g_A ,b_A , α_A) from the foreground image *pSrc1* with pixels (r_B ,g_B ,b_B , α_B) from the background image *pSrc2* to produce pixels (r_C ,g_C ,b_C , α_C) in the resultant image *pDst*.
The alpha values are assumed to be normalized to the range [0..1].
The type of the compositing operation is indicated by the *alphaType* parameter.
Use Table 5-3 to choose a valid *alphaType* value depending on the required composition

type.
For example, the resulting pixel color components for the OVER operation (see Table 5-2) are computed as follows:

$r_C = \alpha_A * r_A + (1 - \alpha_A) * \alpha_B * r_B$

$g_C = \alpha_A * g_A + (1 - \alpha_A) * \alpha_B * g_B$

$b_C = \alpha_A * b_A + (1 - \alpha_A) * \alpha_B * b_B$

The resulting (normalized) alpha value is computed as

$\alpha_C = \alpha_A + (1 - \alpha_A) * \alpha_B$

This function can be used for unsigned pixel data only.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *roiSize* has a field with zero or negative value.

AlphaCompC

Combines two images using constant alpha values.

Syntax

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    Ipp<datatype> alpha1, const Ipp<datatype>* pSrc2, int src2Step,
    Ipp<datatype> alpha2, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for *mod*:

`8u_C1R`	`16u_C1R`
`8u_C3R`	`16u_C3R`
`8u_C4R`	`16u_C4R`
`8u_AC4R`	`16u_AC4R`

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* const pSrc1[4],
    int src1Step, Ipp<datatype> alpha1, const Ipp<datatype>* const
    pSrc2[4], int src2Step, Ipp<datatype> alpha2, Ipp<datatype>* const
    pDst[4],
    int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for *mod*:

```
8u_AP4R     16u_AP4R
```

Parameters

pSrc1, pSrc2	Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source image buffers.
pDst	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
roiSize	Size of the source and destination ROI in pixels.
alpha1, alpha2	Constant alpha values to use for compositing operation.
alphaType	The composition type to perform. See Table 5-3 for the type value and description.

Description

The function `ippiAlphaCompC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs an image compositing operation on 1-channel image buffers, 3-channel RGB and 4-channel RGBA image buffers and on planar images, using constant alpha values *alpha1* and *alpha2*. These values are passed to the function as parameters.
The compositing is done by overlaying pixels from the foreground image ROI *pSrc1* with pixels from the background image ROI *pSrc2* to produce pixels in the resultant image ROI *pDst*.
The alpha values are normalized to the range [0..1].
The type of the compositing operation is indicated by the *alphaType* argument. Use

Table 5-3 to choose a valid *alphaType* value depending on the required composition type. For example, the resulting pixel color components for the OVER operation (see Table 5-2) are computed as follows:

$r_C = \alpha_1 * r_A + (1 - \alpha_1) * \alpha_2 * r_B$

$g_C = \alpha_1 * g_A + (1 - \alpha_1) * \alpha_2 * g_B$

$b_C = \alpha_1 * b_A + (1 - \alpha_1) * \alpha_2 * b_B$

where α_1 , α_2 are the normalized alpha values *alpha1, alpha2*.

This function can be used for unsigned pixel data only.

The following code example shows how to use alpha composition function:

Example 5-7 Using Alpha Composition Function

```
IppStatus acomp( void ) {
   Ipp8u imga[8*8], imgb[8*8], imgc[8*8];
   IppiSize roi = { 8, 8 };
   IppStatus st;
   ippiImageRamp_8u_C1R( imga, 8, roi, 0, 1, ippAxsHorizontal );
   ippiImageRamp_8u_C1R( imgb, 8, roi, 0, 2, ippAxsHorizontal );
   st = ippiAlphaCompC_8u_C1R( imga, 8, 255/3, imgb, 8, 255, imgc, 8,
roi, ippAlphaOver );
   printf( "over: a=%d,A=255/3; b=%d,B=255; c=%d //
c=a*A+b*(1-A)*B\n",imga[6],imgb[6],imgc[6] );
   return st;
}
```

Output

```
over: a=6,A=255/3; b=12,B=255; c=10 // c=a*A+b*B*(1-A)
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *roiSize* has a field with zero or negative value.

AlphaPremul

Pre-multiplies pixel values of an image by its alpha values.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_AC4R     16u_AC4R
```

```
IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* const pSrc[4],
    int srcStep, Ipp<datatype>* const pDst[4], int dstStep,
    IppiSize roiSize);
```

Supported values for *mod* :

```
8u_AP4R     16u_AP4R
```

Case 2: In-place operation

```
IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod* :

```
8u_AC4IR    16u_AC4IR
```

```
IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* const pSrcDst[4],
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod* :

```
8u_AP4IR    16u_AP4IR
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.

pDst	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiAlphaPremul` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a RGBA source image (pixel order or planar) to the pre-multiplied alpha form. If (r,g,b,a) are the red, green, blue, and alpha values of a pixel, then new pixel values are ($r*\alpha$, $g*\alpha$, $b*\alpha$, a) after execution of this function. Here α is the pixel's normalized alpha value in the range 0 to 1.

The function `ippiAlphaPremul` can be used for unsigned pixel data only.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *roiSize* has a field with zero or negative value.

AlphaPremulC

Pre-multiplies pixel values of an image using constant alpha (opacity) values.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> alpha, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod* :

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

```
IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* const pSrc[4],
    int srcStep, Ipp<datatype> alpha, Ipp<datatype>* const pDst[4],
    int dstStep, IppiSize roiSize);
```

Supported values for *mod* :

8u_AP4R	16u_AP4R

Case 2: In-place operation

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod* :

8u_C1IR	16u_C1IR
8u_C3IR	16u_C3IR
8u_C4IR	16u_C4IR
8u_AC4IR	16u_AC4IR

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha,
    Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppiSize roiSize);
```

Supported values for *mod* :

8u_AP4IR	16u_AP4IR

Parameters

pSrc	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI for pixel-order data. An array of pointers to separate ROI in the destination color planes in case of planar data.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination image ROI or an array of pointers to ROI in the separate source and destination color planes for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
alpha	Global alpha value used for pre-multiplying pixel values.

Description

The function `ippiAlphaPremulC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts either a 1-, 3-, 4-channel image or planar RGBA image to the pre-multiplied alpha form, using global alpha value *alpha*.
For 1-, 3-, 4-channel image buffers, pixel values in each channel are multiplied by α; for RGBA (pixel order and planar) images with (r,g,b,a) pixel values, new pixel values are ($r*\alpha$, $g*\alpha$, $b*\alpha$, *alpha*) after execution of this function.
Here α is the normalized *alpha* value in the range 0 to 1.

The function `ippiAlphaPremulC` can be used for unsigned pixel data only.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *roiSize* has a field with zero or negative value.

Image Color Conversion

6

This chapter describes the Intel® IPP image processing functions that perform different type of image color conversion. The Intel IPP software supports the following image color conversions:

- Color models conversion
- Conversion from color to gray scale and vice versa
- Different types of format conversion:
    - from pixel-order to planar format and vice versa
    - changing number of channels or planes
    - changing sampling format
    - altering order of samples or planes
- Gamma correction
- Reduction from high bit resolution color to low bit resolution color
- Intensity transformation using lookup tables
- Color twist
- Color keying

All Intel IPP color conversion functions perform point operations on pixels of the source image. For a given destination pixel, the resultant channel values are computed using channel values of the corresponding source pixel only, and not involving any neighborhood pixels. Thus, the rectangular region of interest (ROI, see Regions of Interest in Intel IPP) used in function operations may extend to the size of the whole image.

Table 6-1 lists Intel IPP color space conversion functions.

Table 6-1 Color Conversion Functions

Function Base Name	Description
Color Model Conversion	
RGBToYUV YUVToRGB	Convert RGB image to and from YUV color model.
RGBToYUV422 YUV422ToRGB	Convert RGB image to and from 4:2:2 YUV image.
RGB565ToYUV422	Converts a 16-bit RGB image to the 4:2:2 YUV image
RGBToYUV420 YUV420ToRGB	Convert RGB images to and from 4:0:0 YUV image.
BGRToYUV420	Converts a BGR image to the 4:2:0 YUV image.
YUV420ToBGR	Converts a 4:2:0 YUV image to BGR image.
RGB565ToYUV420	Converts a 16-bit RGB image to the 4:2:0 YUV image
YUV420ToRGB565 YUV420ToRGB555 YUV420ToRGB444	Convert 4:2:0 YUV image to the 16-bit RGB image.
YUV420ToRGB565Dither YUV420ToRGB555Dither YUV420ToRGB444Dither	Convert 4:2:0 YUV image to the 16-bit RGB image with dithering.
BGR565ToYUV420 BGR555ToYUV420	Convert 16-bit BGR image to the 4:2:0 YUV image.
YUV420ToBGR565 YUV420ToBGR555 YUV420ToBGR444	Convert 4:2:0 YUV image to the 16-bit BGR image.
YUV420ToBGR565Dither YUV420ToBGR555Dither YUV420ToBGR444Dither	Convert 4:2:0 YUV image to the 16-bit BGR image with dithering.
RGBToYCbCr YCbCrToRGB	Convert RGB images to and from YCbCr color model
YCbCrToBGR	Converts a YCbCr image to the BGR color model.
YCbCrToBGR_709CSC	Converts a YCbCr image to the BGR image for ITU-R BT.709 signal
YCbCrToRGB565 YCbCrToRGB555 YCbCrToRGB444	Convert YCbCr image to the 16-bit per pixel RGB image.
YCbCrToRGB565Dither YCbCrToRGB555Dither YCbCrToRGB444Dither	Convert YCbCr image to the 16-bit per pixel RGB image with dithering

Table 6-1 Color Conversion Functions (continued)

Function Base Name	Description
YCbCrToBGR565 YCbCrToBGR555 YCbCrToBGR444	Convert YCbCr image to the 16-bit per pixel BGR image.
YCbCrToBGR565Dither YCbCrToBGR555Dither YCbCrToBGR444Dither	Convert YCbCr image to the 16-bit per pixel BGR image with dithering.
RGBToYCbCr422 YCbCr422ToRGB	Convert RGB image to and from 4:2:2 YCbCr image.
RGBToYCrCb422 YCrCb422ToRGB	Convert RGB image to and from 4:2:2 YCrCb image.
BGRToYCbCr422	Converts 24-bit per pixel BGR image to 16-bit per pixel YCbCr image.
YCbCr422ToBGR	Converts 16-bit per pixel YCbCr image to 24-bit per pixel BGR image.
BGR565ToYCbCr422 BGR555ToYCbCr422	Converts 16-bit per pixel BGR image to 16-bit per pixel YCbCr image.
RGBToCbYCr422 RGBToCbYCr422Gamma CbYCr422ToRGB	Convert RGB images to and from 4:2:2 CbYCr image.
BGRToCbYCr422	Converts 32-bit per pixel BGR image to 16-bit per pixel CbYCr image.
BGRToCbYCr422_709HDTV	Converts BGR image to 16-bit per pixel CbYCr image for ITU-R BT.709 HDTV signal.
CbYCr422ToBGR	Converts 16-bit per pixel CbYCr image to four channel BGR image.
CbYCr422ToBGR_709HDTV	Converts 16-bit per pixel CbYCr image to the BGR image for ITU-R BT.709 HDTV signal.
YCbCr422ToRGB565 YCbCr422ToRGB555 YCbCr422ToRGB444	Convert 16-bit per pixel YCbCr image that has 4:2:2 sampling format to the 16-bit per pixel RGB image.
YCbCr422ToRGB565Dither YCbCr422ToRGB555Dither YCbCr422ToRGB444Dither	Convert 16-bit per pixel YCbCr image that has 4:2:2 sampling format to the 16-bit per pixel RGB image with dithering
YCbCr422ToBGR565 YCbCr422ToBGR555 YCbCr422ToBGR444	Convert 16-bit per pixel YCbCr image that has 4:2:2 sampling format to the 16-bit per pixel BGR image.
YCbC422ToBGR565Dither YCbC422ToBGR555Dither YCbCr422ToBGR444Dither	Convert 16-bit per pixel YCbCr image that has 4:2:2 sampling format to the 16-bit per pixel BGR image with dithering.

Table 6-1 Color Conversion Functions (continued)

Function Base Name	Description
RGBToYCbCr420 YCbCr420ToRGB	Convert RGB images to and from YCbCr color model with 4:2:0 sampling format.
YCbCr420ToRGB565 YCbCr420ToRGB555 YCbCr420ToRGB444	Convert YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.
YCbCr420ToRGB565Dither YCbCr420ToRGB555Dither YCbCr420ToRGB444Dither	Convert YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.
RGBToYCrCb420	Converts an RGB image to the YCrCb image with the 4:2:0 sampling.
YCrCb420ToRGB	Converts a YCrCb image with the 4:2:0 sampling to the RGB image.
BGRToYCbCr420	Converts a BGR image to the YCbCr image with 4:2:0 sampling format.
BGRToYCbCr420_709CSC	Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 CSC signal.
BGRToYCbCr420_709HDTV	Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 HDTV signal.
BGRToYCrCb420_709CSC	Converts a BGR image to the YCrCb image with 4:2:0 sampling for ITU-R BT.709 signal.
YCbCr420ToBGR	Convert YCbCr image with 4:2:0 sampling format to the RGB color model.
YCbCr420ToBGR_709CSC	Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 CSC signal.
YCbCr420ToBGR_709HDTV	Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 HDTV signal.
BGR565ToYCbCr420, BGR555ToYCbCr420	Convert a 16-bit per pixel BGR image to the YCbCr image that has 4:2:0 sampling format.
YCbCr420ToBGR565, YCbCr420ToBGR555, YCbCr420ToBGR444	Convert YCbCr image with 4:2:0 sampling format to the 16-bit per pixel BGR image.
YCbCr420ToBGR565Dither YCbCr420ToBGR555Dither YCbCr420ToBGR444Dither	Convert YCbCr image with 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.
BGRToYCrCb420	Converts a BGR image to the YCrCb image with 4:2:0 sampling format.
BGR565ToYCrCb420, BGR555ToYCrCb420	Converts a 16-bit per pixel BGR image to the YCrCb image with 4:2:0 sampling format.

Table 6-1 Color Conversion Functions (continued)

Function Base Name	Description
BGRToYCbCr411	Converts a BGR image to the YCbCr planar image that has a 4:1:1 sampling format.
YCbCr411ToBGR	Converts a YCbCr image with 4:1:1 sampling format to the RGB color model.
BGR565ToYCbCr411, BGR555ToYCbCr411	Converts a 16-bit per pixel BGR image to the YCbCr image that has 4:1:1 sampling format.
YCbCr411ToBGR565, YCbCr411ToBGR555	Convert a YCbCr image that has 4:1:1 sampling format to the 16-bit per pixel BGR image.
RGBToXYZ XYZToRGB	Convert RGB images to and from XYZ color model
RGBToLUV LUVToRGB	Convert RGB images to and from CIE LUV color model
BGRToLab LabToBGR	Convert BGR images to and from CIE Lab color model
RGBToYCC YCCToRGB	Convert RGB images to and from YCC color model
RGBToHLS HLSToRGB	Convert RGB images to and from HLS color model
BGRToHLS	Converts BGR images to HLS color model
HLSToBGR	Converts HLS images to RGB color model
RGBToHSV HSVToRGB	Convert RGB images to and from HSV color model
BGRToYCoCg	Converts a 24-bit BGR image to the YCoCg color model.
SBGRToYCoCg	Converts a 48-bit BGR image to the YCoCg color model.
YCoCgToBGR	Converts a YCoCg image to the 24-bit BGR image.
YCoCgToSBGR	Converts a YCoCg image to the 48-bit BGR image.
BGRToYCoCg_Rev	Converts a 24-bit BGR image to the YCoCg-R color model.
SBGRToYCoCg_Rev	Converts a 48-bit BGR image to the YCoCg-R color model.
YCoCgToBGR_Rev	Converts a YCoCg-R image to the 24-bit BGR image.
YCoCgToSBGR_Rev	Converts a YCoCg-R image to the 48-bit BGR image.
Color - Gray Scale Conversion	
RGBToGray	Converts an RGB image to gray scale using fixed transform coefficients
ColorToGray	Converts an RGB image to gray scale using custom transform coefficients
CFAToRGB	Restores the RGB image from the gray-scale CFA image

Table 6-1 Color Conversion Functions (continued)

Function Base Name	Description
DemosaicAHD	Restores the RGB image from the gray-scale CFA image using AHD algorithm.
Format Conversion	
RGBToRGB565, BGRToBGR565	Convert 24-bit per pixel image to the 16-bit image.
RGB565ToRGB, BGR565ToBGR	Convert 16-bit per pixel image to the 24-bit image.
YCbCr422	Converts 4:2:2 YCbCr image
YCbCr422ToYCrCb422	Converts 4:2:2 YCbCr image to 4:2:2 YCrCb image.
YCbCr422ToCbYCr422	Converts 4:2:2 YCbCr image to 4:2:2 CbYCr image.
YCbCr422ToYCbCr420	Converts 4:2:2 YCbCr image to 4:2:0 YCbCr image.
YCbCr422To420_Interlace	Converts interlaced YCbCr image from 4:2:2 sampling format to 4:2:0 format.
YCbCr422ToYCrCb420	Converts 4:2:2 YCbCr image to 4:2:0 YCrCb image.
YCbCr422ToYCbCr411	Converts 4:2:2 YCbCr image to 4:1:1 YCbCr image.
YCrCb422ToYCbCr420	Converts 4:2:2 YCrCb image to 4:2:2 YCbCr image.
YCrCb422ToYCbCr420	Converts 4:2:2 YCrCb image to 4:2:0 YCbCr image.
YCrCb422ToYCbCr411	Converts 4:2:2 YCrCb image to 4:1:1 YCbCr image.
CbYCr422ToYCbCr422	Converts 4:2:2 CbYCr image to 4:2:2 YCbCr image.
CbYCr422ToYCbCr420	Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image.
CbYCr422ToYCbCr420_Interlace	Converts interlaced 4:2:2 CbYCr image to 4:2:0 YCbCr image.
CbYCr422ToYCrCb420	Converts 4:2:2 CbYCr image to 4:2:0 YCrCb image.
CbYCr422ToYCbCr411	Converts 4:2:2 CbYCr image to 4:1:1 YCbCr image.
YCbCr420	Converts 4:2:0 YCbCr image
YCbCr420ToYCbCr422	Converts 4:2:0 YCbCr image to 4:2:2 YCbCr image.
YCbCr420ToYCbCr422_Filter	Converts 4:2:0 YCbCr image to 4:2:2 YCbCr image with additional filtering.
YCbCr420To422_Interlace	Converts interlaced 4:2:0 YCbCr image to 4:2:2 YCbCr image.
YCbCr420ToCbYCr422	Converts 4:2:0 YCbCr image to 4:2:2 CbYCr image.
YCbCr420ToCbYCr422_Interlace	Converts interlaced 4:2:0 YCbCr image to 4:2:2 CbYCr image.
YCbCr420ToYCrCb420	Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image
YCbCr420ToYCrCb420_Filter	Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image with deinterlace filtering.
YCbCr420ToYCbCr411	Converts 4:2:0 YCbCr image to 4:1:1 YCbCr image

Table 6-1 Color Conversion Functions (continued)

Function Base Name	Description
YCrCb420ToYCbCr422	Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image
YCrCb420ToYCbCr422_Filter	Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image with additional filtering
YCrCb420ToCbYCr422	Converts 4:2:0 YCrCb image to 4:2:2 CbYCr image
YCrCb420ToYCbCr420	Converts 4:2:0 YCrCb image to 4:2:0 YCbCr image
YCrCb420ToYCbCr411	Converts 4:2:0 YCrCb image to 4:1:1 YCbCr image
YCbCr411	Converts 4:1:1 YCbCr image
YCbCr411ToYCbCr422	Converts 4:1:1 YCbCr image to 4:2:2 YCbCr image
YCbCr411ToYCrCb422	Converts 4:1:1 YCbCr image to 4:2:2 YCrCb image
YCbCr411ToYCbCr420	Converts 4:1:1 YCbCr image to 4:2:0 YCbCr image
YCbCr411ToYCrCb420	Converts 4:1:1 YCbCr image to 4:2:0 YCrCb image
Color Twist	
ColorTwist	Applies a color-twist matrix to an image with integer pixel values
ColorTwist32f	Applies a color-twist matrix to an image with floating-point pixel values
Color Keying	
CompColorKey	Performs color keying of two images
AlphaCompColorKey	Performs color keying and alpha composition of two images
Gamma Correction	
GammaFwd	Performs gamma-correction of the source RGB image
GammaInv	Converts a gamma-corrected R´G´B´ image back to the original RGB image
Intensity Transformation	
ReduceBits	Reduces the bit resolution of an image
LUT	Maps an image by applying intensity transformation
LUT_Linear	Maps an image by applying intensity transformation with linear interpolation
LUT_Cubic	Maps an image by applying intensity transformation with cubic interpolation
LUTPalette, LUTPaletteSwap	Maps an image by applying intensity transformation in accordance with a palette table.
Tone MapLinear, ToneMapMean	Maps an HDRI image to the LDRI image

This chapter starts with introductory material that discusses color space models essential for understanding the Intel IPP color conversion functions.

For more information about color spaces and color conversion techniques, see [Jack01], [Rogers85], and [Foley90].

Gamma Correction

The luminance intensity generated by most displays is not a linear function of the applied signal but is proportional to some power (referred to as *gamma*) of the signal voltage. As a result, high intensity ranges are expanded and low intensity ranges are compressed. This nonlinearity must be compensated for in order to achieve correct color reproduction. To do this, luminance of each of the linear red, green, and blue components is reduced to a nonlinear form using an inverse transformation. This process is called "gamma correction".

The Intel IPP functions use the following basic equations to convert an RGB image to the gamma-corrected R'G'B' image:

for $R,G,B < 0.018$

$$R' = 4.5R$$
$$G' = 4.5G$$
$$B' = 4.5B$$

for $R,G,B \geq 0.018$

$$R' = 1.099R^{0.45} - 0.099$$
$$G' = 1.099G^{0.45} - 0.099$$
$$B' = 1.099B^{0.45} - 0.099$$

Note that the channel intensity values are normalized to fit in the range [0..1]. The gamma value is equal to 1/0.45 = 2.22 in conformity with ITU Rec.709 specification (see [ITU709]).

CIE Chromaticity Diagram and Color Gamut

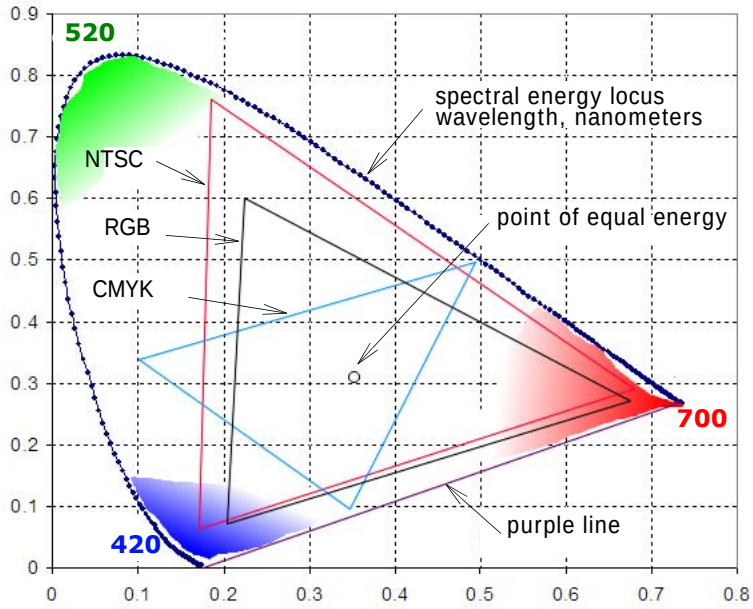
Figure 6-1 presents a diagram of all visible colors. It is called a chromaticity diagram and was developed as a result of the experimental investigations performed by CIE (International Commission on Illumination, http://members.eunet.at/cie). The diagram presents visible colors as a function of x (red) and y (green) components called *chromaticity coordinates*. Positions of various spectrum colors (from violet to red) are indicated as the points of tongue-shaped curve called *spectrum locus*. The straight line

connecting the ends of the curve is called the *purple line*. The point of equal energy represents the CIE standard for white light. Any point within the diagram represents some mixture of spectrum colors. The pure or fully saturated colors lie on the spectrum locus. Straight-line segment joining any two points in the diagram defines all color variations than can be obtained by additively combining these two colors. A triangle with vertices at any three points determine the gamut of colors that can be obtained by combining corresponding three colors.

The structure of the human eye that distinguishes three different stimuli, establishes the three-dimensional nature of color. The color may be described with a set of three parameters called tristimulus values, or components. These values may, for example, be dominant wavelength, purity, and luminance, or so-called primary colors: red, green, and blue.

The chromaticity diagram exhibits that the gamut of any three fixed colors can not enclose all visible colors. For example, Figure 6-1 shows schematically the gamut of reproducible colors for the RGB primaries of a typical color CRT monitor, CMYK color printing, and for the NTSC television.

Figure 6-1 CIE *xyY* Chromaticity Diagram and Color Gamut



Color Models

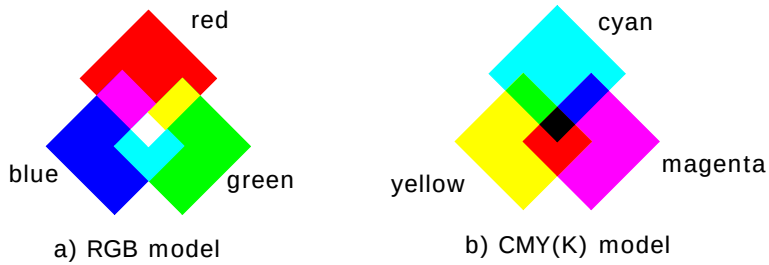
The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point.

Each industry that uses color employs the most suitable color model. For example, RGB color model is used in computer graphics, and YUV or YCbCr are used in video systems, PhotoYCC* is used in PhotoCD* production and so on. Transferring color information from one industry to another requires transformation from one set of values to another. The Intel IPP provides a wide number of functions to convert different color spaces to RGB and vice versa.

RGB Color Model

In the RGB model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the secondary colors of light (see Figure 6-2) - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white.

Figure 6-2 Primary and Secondary Colors for RGB and CMYK Models



The color subspace of interest is a cube shown in Figure 6-3 (RGB values are normalized to 0..1), in which RGB values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.

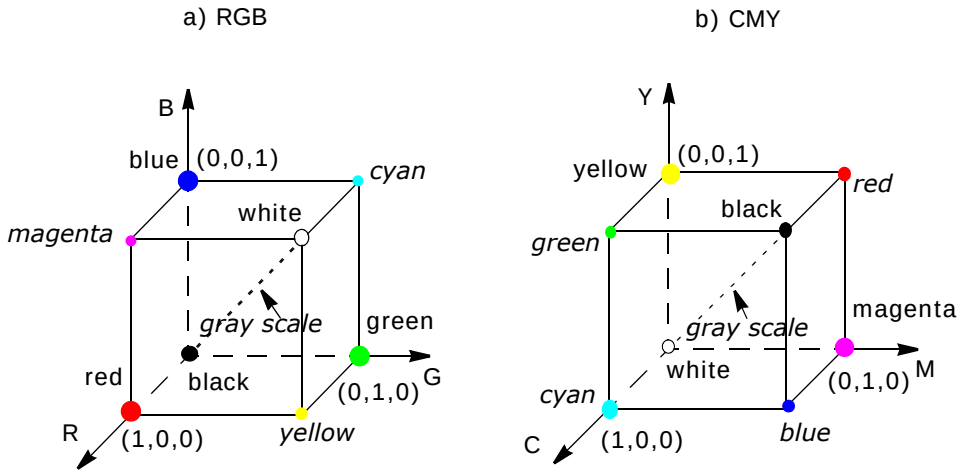
The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin.

Thus, images in the RGB color model consist of three independent image planes, one for each primary color.

As a rule, Intel IPP color conversion functions operate with non-linear gamma-corrected images R'G'B'.

The importance of the RGB color model is that it relates very closely to the way that the human eye perceives color. RGB is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Also, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, since this color space has been around for a number of years.

Figure 6-3 RGB and CMY Color Models



However, RGB is not very efficient when dealing with real-world images. To generate any color within the RGB color cube, all three RGB components need to be of equal pixel depth and display resolution. Also, any modification of the image requires modification of all three planes.

CMYK Color Model

The CMYK color model is a subset of the RGB model and is primarily used in color print production. CMYK is an acronym for cyan, magenta, and yellow along with black (noted as K). The CMYK color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. For example (see Figure 6-2), cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the CMY at full saturation should, in theory, render black.However, impurities in the existing CMY inks make full and equal saturation impossible, and some RGB light does filter through, rendering a muddy brown color. Hence, the addition of black ink to CMY. The CMY cube is shown in Figure 6-3, in which CMY values are at three corners; red, green, and blue are the three other corners, white is at the origin; and black is at the corner farthest from the origin.

YUV Color Model

The YUV color model is the basic color model used in analogue color TV broadcasting. Initially YUV is the re-coding of RGB for transmission efficiency (minimizing bandwidth) and for downward compatibility with black-and white television. The YUV color space is “derived” from the RGB space. It comprises the *luminance* (Y) and two color difference (U, V) components. The luminance can be computed as a weighted sum of red, green and blue components; the color difference, or *chrominance,* components are formed by subtracting luminance from blue and from red.

The principal advantage of the YUV model in image processing is decoupling of luminance and color information. The importance of this decoupling is that the luminance component of an image can be processed without affecting its color component. For example, the histogram equalization of the color image in YUV format may be performed simply by applying histogram equalization to its Y component.

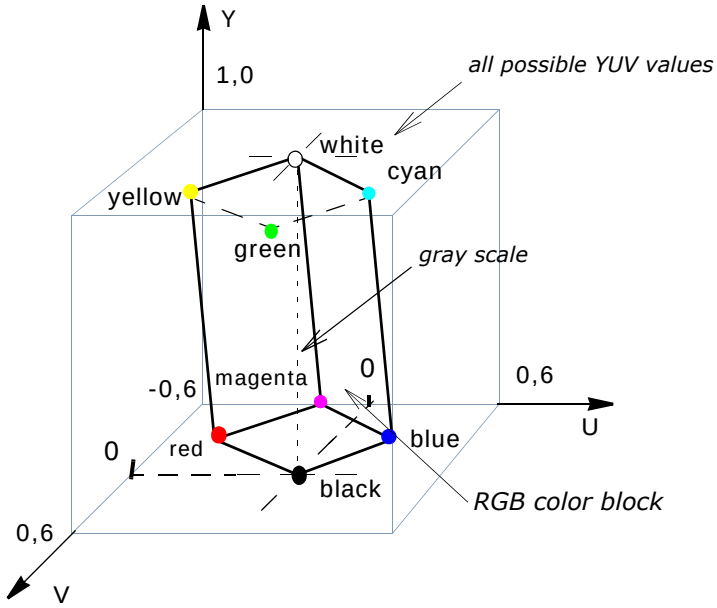
There are many combinations of YUV values from nominal ranges that result in invalid RGB values, since the possible RGB colors occupy only part of the YUV space limited by these ranges. Figure 6-4 shows the valid color block in the YUV space that corresponds to the RGB color cube RGB values are normalized to [0..1]).

The Y’U’V’ notation means that the components are derived from gamma-corrected R’G’B’. Weighted sum of these non-linear components forms a signal representative of luminance that is called *luma* Y’. (*Luma* is often loosely referred to as *luminance*, so you need be careful to determine whether a particular author assigns a linear or non-linear interpretation to the term *luminance*).

The Intel IPP functions use the following basic equation [Jack01] to convert between gamma-corrected R'G'B' and Y'U'V' models:

```
Y'=  0.299*R' + 0.587*G' + 0.114*B'
U'= -0.147*R' - 0.289*G' + 0.436*B' = 0.492*(B'- Y')
V'=  0.615*R' - 0.515*G' - 0.100*B' = 0.877*(R'- Y')

R' = Y' + 1.140*V'
G' = Y' - 0.394*U' - 0.581*V'
B' = Y' + 2.032*U'
```

Figure 6-4 RGB Colors Cube in the YUV Color Space



There are several YUV sampling formats such as 4:4:4, 4:2:2, and 4:2:0 that are supported by the Intel IPP color conversion functions and are described later in this chapter (see "Image Downsampling").

YCbCr and YCCK Color Models

The YCbCr color space is used for component digital video and was developed as part of the ITU-R BT.601 Recommendation. YCbCr is a scaled and offset version of the YUV color space.

The Intel IPP functions use the following basic equations [Jack01] to convert between `R'G'B'` in the range 0-255 and `Y'Cb'Cr'` (this notation means that all components are derived from gamma-corrected `R'G'B'`):

```
Y'  =   0.257*R' + 0.504*G' + 0.098*B' + 16
Cb' = -0.148*R' - 0.291*G' + 0.439*B' + 128
Cr' =  0.439*R' - 0.368*G' - 0.071*B' + 128

R' = 1.164*(Y'-16) + 1.596*(Cr'-128)
G' = 1.164*(Y'-16) - 0.813*(Cr'-128) - 0.392*(Cb'-128)
B' = 1.164*(Y'-16) + 2.017*(Cb'-128)
```

Intel IPP color conversion functions specific for JPEG codec used different equations:

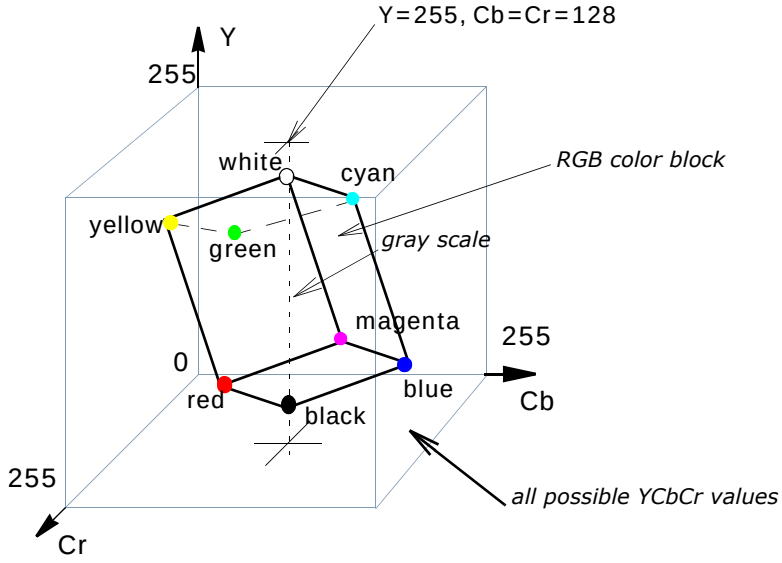
```
Y  =   0.299*R + 0.587*G + 0.114*B
Cb = -0.16874*R - 0.33126*G + 0.5*B + 128
Cr =  0.5*R - 0.41869*G - 0.08131*B + 128


R =  Y + 1.402*Cr - 179,456
G =  Y - 0.34414*Cb - 0.71414*Cr + 135.45984
B =  Y  + 1.772*Cb - 226.816
```

YCCK model is specific for the JPEG image compression. It is a variant of YCbCr model containing an additional *K* channel (black). The fact is that JPEG codec performs more effectively if the luminance and color information are decoupled. Therefore a CMYK image should be converted to YCCK before JPEG compression (see description of the function `ippiCMYKToYCCK_JPEG` in Chapter 15 for more details).

The possible RGB colors occupy only part of the YCbCr color space (see Figure 6-5) limited by the nominal ranges, therefore there are many YCbCr combinations that result in the invalid RGB values.

There are several YCbCr sampling formats such as 4:4:4, 4:2:2, 4:1:1, and 4:2:0, which are supported by the Intel IPP color conversion functions and are described later in this chapter (see "Image Downsampling").

Figure 6-5 RGB Colors Cube in the YCbCr Space



PhotoYCC Color Model

The Kodak* PhotoYCC* was developed for encoding Photo CD* image data. It is based on both the ITU Recommendations 601 and 709, using luminance-chrominance representation of color like in BT.601 YCbCr and BT.709 ([ITU709]). This model comprises luminance (Y) and two color difference, or chrominance (C1, C2) components.The PhotoYCC is optimized for the color photographic material, and provides a color gamut that is greater than that which can currently be displayed.

The Intel IPP functions use the following basic equations [Jack01] to convert non-linear gamma-corrected `R'G'B'` to `Y'C'C'`:

```
Y' =  0.213*R' + 0.419*G' + 0.081*B'
C1' = -0.131*R' - 0.256*G' + 0.387*B'+ 0.612
C2' =  0.373*R' - 0.312*R' - 0.061*B' + 0.537
```

The equations above are given on the assumption that `R',G'`, and `B'` values are normalized to the range `[0..1]`.

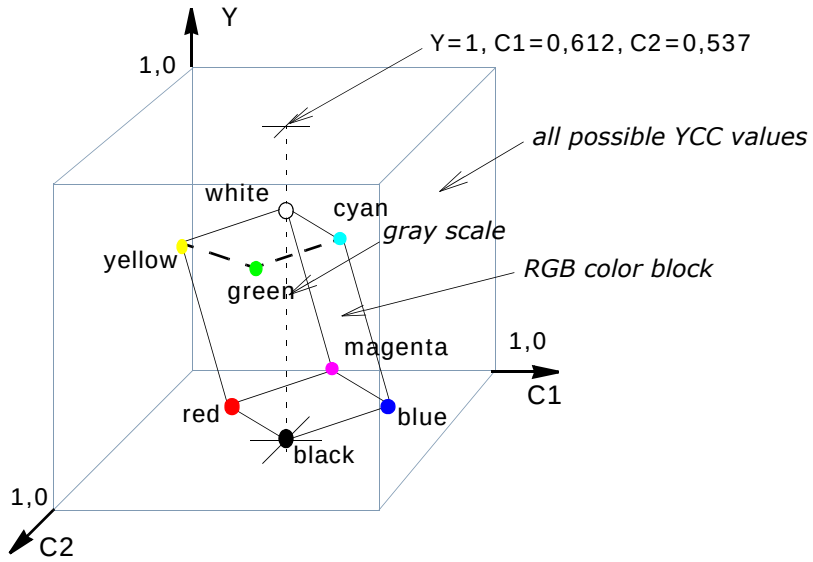
Since PhotoYCC model attempts to preserve the dynamic range of film, decoding PhotoYCC images requires the selection of a color space and range appropriate for the output device. Thus, the decoding equations are not always the exact inverse of the encoding equations. The following equations [Jack01] are used in Intel IPP to generate `R'G'B'` values for driving a CRT display and require a unity relationship between the luma in the encoded image and the displayed image:

```
R' = 0.981 * Y + 1.315 * (C2 - 0.537)
G' = 0.981 * Y - 0.311 * (C1 - 0.612) - 0.669 * (C2 - 0.537)
B' = 0.981 * Y + 1.601 * (C1 - 0.612)
```

The equations above are given on the assumption that source `Y`, `C1`, and `C2` values are normalized to the range `[0..1]`, and the display primaries have the chromaticity values in accordance with [ITU709] specifications.

The possible RGB colors occupy only part of the YCC color space (see Figure 6-6) limited by the nominal ranges, hence there are many YCbCr combinations that result in the invalid RGB values.

Figure 6-6 RGB Colors in the YCC Color Space



YCoCg Color Models

The YCoCg color model was developed to increase the effectiveness of the image compression [Malvar03]. This color model comprises the luminance (Y) and two color difference components (Co - offset orange, Cg - offset green).

The Intel IPP functions use the following simple basic equations [Malvar03] to convert between `RGB` and `YCoCg`:

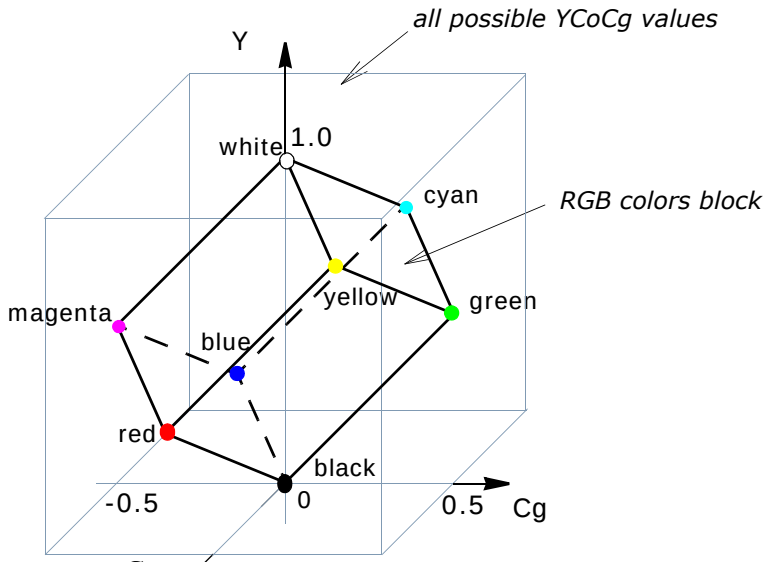
```
Y  =  R/4 + G/2 + B/4
Co =  R/2 - B/2
Cg = -R/4 + G/2 - B/4
```

```
R = Y + Co - Cg
G = Y + Cg
B = Y - Co - Cg
```

A variation of this color space which is called YCoCg -R, enables transformation reversibility with a smaller dynamic range requirements than does YCoCg [Malvar03-1].

The possible RGB colors occupy only part of the YCoCg color space (see Figure 6-7) limited by the nominal ranges, hence there are many YCoCg combinations that result in the invalid RGB values.

Figure 6-7 RGB Color Cube in the YCoCg Color Space



HSV, and HLS Color Models

The HLS (hue, lightness, saturation) and HSV (hue, saturation, value) color models were developed to be more “intuitive” in manipulating with color and were designed to approximate the way humans perceive and interpret color.

Hue defines the color itself. The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors.

Saturation indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination.

Intensity component - *lightness* (HLS) or *value* (HSV), indicates the illumination level. Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue (S=1) is at *value* V=1 (full illumination) in the HSV color model, and at *lightness* L=0.5 in the HLS color model.

The HSV color space is essentially a cylinder, but usually it is represented as a cone or hexagonal cone (hexcone) as shown in the Figure 6-8, because the hexcone defines the subset of the HSV space with valid RGB values. The *value* V is the vertical axis, and the vertex V=0 corresponds to black color. Similarly, a color solid, or 3D-representation, of the HLS model is a double hexcone (Figure 6-9) with *lightness* as the axis, and the vertex of the second hexcone corresponding to white.

Both color models have intensity component decoupled from the color information. The HSV color space yields a greater dynamic range of saturation.
Conversions from RGB to HSV/HLS and vice-versa in Intel IPP are performed in accordance with the respective pseudocode algorithms [Rogers85], given in the descriptions of corresponding conversion functions.

Figure 6-8 HSV Solid

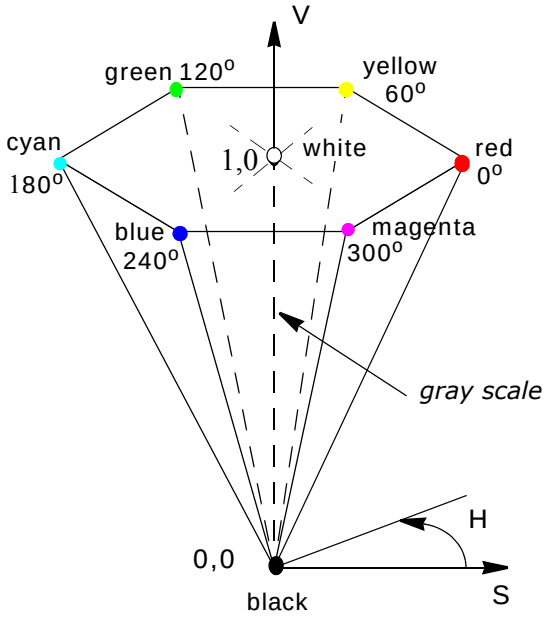
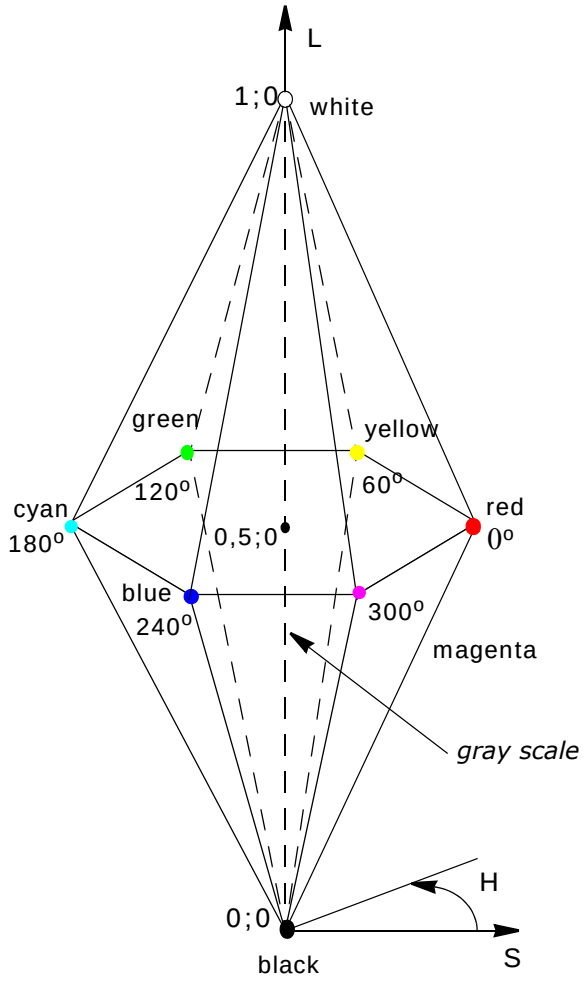


Figure 6-9 HLS Solid

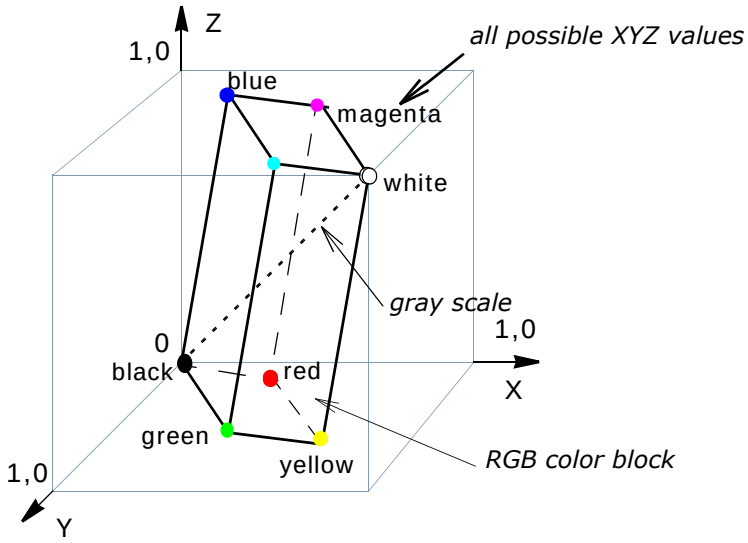


CIE XYZ Color Model

The XYZ color space is an international standard developed by the CIE (Commission Internationale de l'Eclairage). This model is based on three hypothetical primaries, XYZ, and all visible colors can be represented by using only positive values of X, Y, and Z. The CIE XYZ primaries are hypothetical because they do not correspond to any real light wavelengths. The Y primary is intentionally defined to match closely to luminance, while X and Z primaries give color information.
The main advantage of CIE XYZ space (and any color space based on it) is that this space is completely device independent. The chromaticity diagram on Figure 6-1 is in fact a two-dimensional projection of the CIE XYZ sub-space.
Note that arbitrarily combining X, Y, and Z values within nominal ranges can easily lead to a "color" outside of the visible color spectrum.

The position of the block of RGB-representable colors in the XYZ space is shown in the Figure 6-10 .

Figure 6-10 RGB Colors Cube in the XYZ Color Space



Intel IPP functions use the following basic equations [Rogers85], to convert between gamma-corrected R'G'B' and CIE XYZ models:

```
X =  0.412453*R' + 0.35758 *G' + 0.180423*B'
Y =  0.212671*R' + 0.71516 *G' + 0.072169*B'
Z =  0.019334*R' + 0.119193*G' + 0.950227*B'
```

The equations for `X,Y,Z` calculation are given on the assumption that `R',G'`, and `B'` values are normalized to the range `[0..1]`.

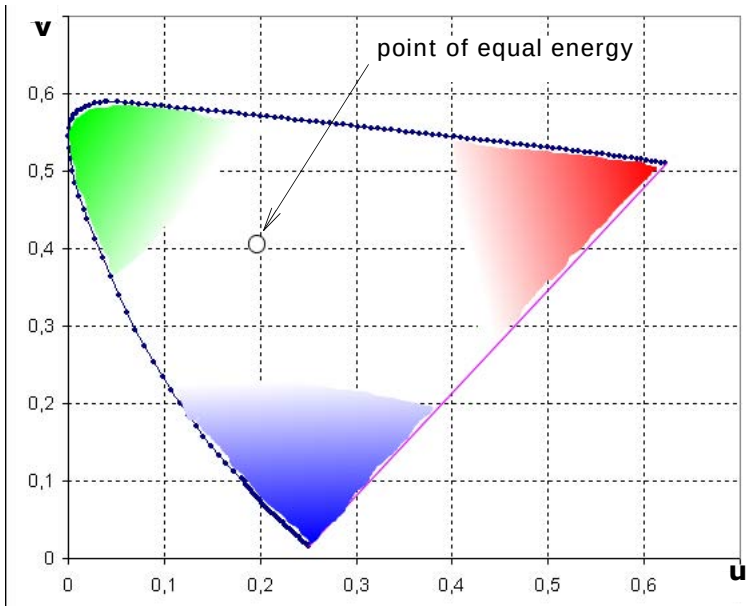
```
R' =  3.240479 * X - 1.53715  * Y  - 0.498535 * Z
G' = -0.969256 * X + 1.875991 * Y  + 0.041556 * Z
B' =  0.055648 * X - 0.204043 * Y  + 1.057311 * Z
```

The equations for `R',G', B'` calculation are given on the assumption that `X,Y`, and `Z` values are in the range `[0..1]`.

CIE LUV and CIE Lab Color Models

The CIE LUV and CIE Lab color models are considered to be perceptually uniform and are referred to as uniform color models. Both are uniform derivations from the standard CIE XYZ space. “Perceptually uniform” means that two colors that are equally distant in the color space are equally distant perceptually. To accomplish this approach, a uniform chromaticity scale (UCS) diagram was proposed by CIE (Figure 6-11). The UCS diagram uses a mathematical formula to transform the XYZ values or x, y coordinates (Figure 6-1), to a new set of values that present a visually more accurate two-dimensional model.
The Y lightness scale is replaced with a new scale called L that is approximately uniformly spaced but is more indicative of the actual visual differences. Chrominance components are U and V for CIE LUV, and *a* and *b* (referred also respectively as red/blue and yellow/blue chrominances) in CIE Lab. Both color spaces are derived from the CIE XYZ color space.

Figure 6-11 CIE u',v' Uniform Chromaticity Scale Diagram



The CIE LUV color space is derived from CIE XYZ as follows ([Rogers85]),

$$L = 116. * (Y/Y_n)^{1/3.} - 16.$$
$$U = 13. * L * (u - u_n)$$
$$V = 13. * L * (v - v_n)$$

where

$$u = 4.*X / (X + 15.*Y + 3.*Z)$$
$$v = 9.*Y / (X + 15.*Y + 3.*Z)$$
$$u_n = 4.*x_n / (-2.*x_n + 12.*y_n + 3.)$$
$$v_n = 9.*y_n / (-2.*x_n + 12.*y_n + 3.)$$

Inverse conversion is performed in accordance with equations:

$$Y = Y_n * ((L + 16.) / 116.)^{3.}$$
$$X = -9.* Y * u / ((u - 4.)* v - u * v)$$
$$Z = (9.* Y - 15*v*Y - v*X) / 3. * v$$

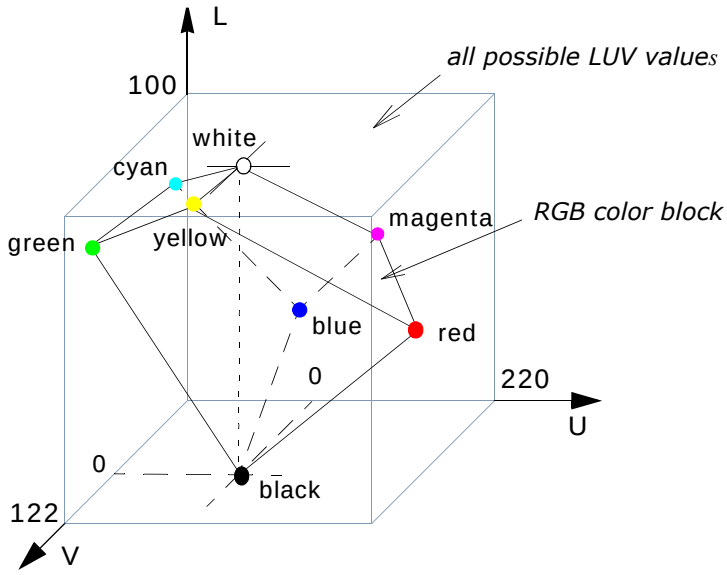
where

$$u = U / (13.* L) + u_n$$
$$v = V / (13.* L) + v_n$$

and u_n, v_n are defined above.
Here $x_n = 0.312713$, $y_n = 0.329016$ are the CIE chromaticity coordinates of the D65 white point ([ITU709]), and $Y_n = 1.0$ is the luminance of the D65 white point.
The values of the L component are in the range [0..100], U component in the range [-134..220], and V component in the range [-140..122].

The RGB-representable colors occupy only part of the LUV color space (see Figure 6-12) limited by the nominal ranges, therefore there are many LUV combinations that result in the invalid RGB value.

Figure 6-12 RGB Color Cube in the CIE LUV Color Space



The CIE Lab color space is derived from CIE XYZ as follows:

$L = 116. * (Y/Y_n)^{1/3.} - 16$ for $Y/Y_n > 0.008856$
$L = 903.3 * (Y/Y_n)^{1/3.}$ for $Y/Y_n \leq 0.008856$

$a = 500. * [f(X/X_n)-f(Y/Y_n)]$

$b = 200. * [f(Y/Y_n)-f(Z/Z_n)]$

where

$f(t) = t^{1/3.} - 16$ for $t > 0.008856$

$f(t) = 7.787*t + 16/116$ for $t \leq 0.008856$

Here $Y_n = 1.0$ is the luminance, and $X_n = 0.950455$, $Z_n = 1.088753$ are the chrominances for the D65 white point.

The values of the L component are in the range [0..100], a and b component values are in the range [-128..127].

Inverse conversion is performed in accordance with equations:

$$Y = Y_n * P^{3.}$$
$$X = X_n * (P + a/500.)^{3.}$$
$$Z = Z_n * (P - b/200.)^{3.}$$

where

$$P = (L + 16)/116.$$

Image Downsampling

Conventionally, digital color images are represented by setting specific values of the color space coordinates for each pixel. Color spaces with decoupled luminance and chrominance coordinates (YUV type) allow to reduce the number of bits required for acceptable color description of an image. This reduction is based on greater sensitivity of the human eye to changes in luminance than to changes in chrominance.
The idea behind this approach is to set individual value of luminance component to each pixel, while assigning the same color (chrominance components) to certain groups of pixels (sometimes called *macropixels*) in accordance with some specific rules. This process is called *downsampling*, and there are different sampling formats depending on the underlying scheme.

The following sampling formats are supported by Intel IPP image processing functions (excluding JPEG functions):

4:4:4 YUV (YCbCr) - conventional format, no downsampling, Y, U(Cb), V(Cr) components are sampled at every pixel. If each component takes 8 bits, than every pixel requires 24 bits. This format is often denoted as YUV (YCbCr) with “4:4:4” descriptor omitted.

4:2:2 YUV (YCbCr) - uses 2:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while U(Cb) and V(Cr) components are sampled every 2 pixels in horizontal direction. If each component takes 8 bits, the pair of pixels requires 32 bits.

4:1:1 YCbCr - uses 4:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while Cb and Cr components are sampled every 4 pixels horizontally. If each component takes 8 bits, each four horizontal pixels require 48 bits.

4:2:0 YUV (YCbCr) - uses 2:1 horizontal downsampling and 2:1 vertical downsampling. Y is sampled at each pixel, U(Cb) and V(Cr) are sampled at every block of 2x2 pixels. If each component takes 8 bits, each four-pixel block requires 48 bits.

In JPEG compression, downsampling has specific distinctive features and is denoted in a slightly different way. In JPEG domain, sampling formats determine the structure of minimal coded units, or MCUs. Hence, Intel IPP functions specific for JPEG codec, support the following sampling formats:

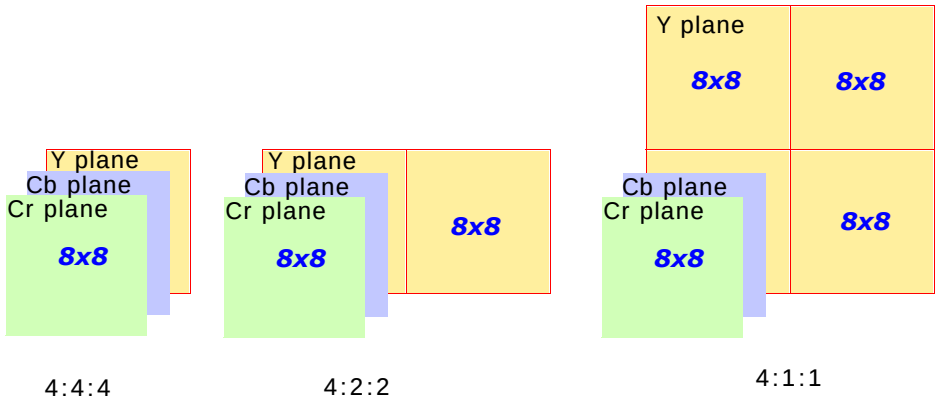
4:4:4 YCbCr - for every 8x8 block of Y samples, there is one 8x8 block of each Cb and Cr samples.

4:2:2 YCbCr - for every two horizontal 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

4:1:1 YCbCr - for every four (two in horizontal and two in vertical direction) 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

Structure of the corresponding MCU for each of these sampling formats is shown in Figure 6-13.

Figure 6-13 MCU Structure for Different JPEG Sampling Formats



RGB Image Formats

In addition to the 24-bit-per-pixel RGB/BGR image formats, Intel IPP color conversion functions support 32-bit-per-pixel RGB/BGR formats (which include three RGB channels plus alpha channel). For 24-bit formats, each color is one byte, every pixel is three bytes. For 32-bit formats, each color is one byte and alpha component is one byte, which yields four bytes per pixel. Memory layout for these formats is given in Table 6-2.

For 16-bit formats, every pixel is two bytes and each color occupies a specified number of bits. Figure 6-14 shows all supported 16-bit-per-pixel formats and their memory layout (bit order):

Figure 6-14 16-bit pixel formats

16-bit pixel formats

	High-order byte	Low-order byte
RGB565	B4 B3 B2 B1 B0 G5 G4 G3	G2 G1 G0 R4 R3 R2 R1 R0
RGB555	X B4 B3 B2 B1 B0 G4 G3	G2 G1 G0 R4 R3 R2 R1 R0
RGB444	X X X X B3 B2 B1 B0	G3 G2 G1 G0 R3 R2 R1 R0
BGR565	R4 R3 R2 R1 R0 G5 G4 G3	G2 G1 G0 B4 B3 B2 B1 B0
BGR555	X R4 R3 R2 R1 R0 G4 G3	G2 G1 G0 B4 B3 B2 B1 B0
BGR444	X X X X R3 R2 R1 R0	G3 G2 G1 G0 B3 B2 B1 B0

R - Red, G - Green, B - Blue, X - free bit

Pixel and Planar Image Formats

Data storage for an image can be pixel-oriented or planar-oriented (planar).
For images in pixel order, all channel values for each pixel are clustered and stored consecutively. Their layout depends on the color model and downsampling scheme.

Table 6-2 lists all pixel-order image formats that are supported by Intel IPP color conversion functions and shows the corresponding channel values order (here, group of underlined symbols represents one pixel and symbol *A* denotes alpha-channel value). Last column of this table gives an example of Intel IPP color conversion function that uses the respective image format.

Table 6-2 Pixel-Order Image Formats

Image Format	Number of Channels	Channel Values Order	Example
RGB	3	R0 G0 B0 R1 G1 B1 R2 G2 B2	ippiRGBToYUV_8u_C3R
RGB444			ippiYCbCrToRGB444_8u16u_C3R
RGB555			ippiYCbCrToRGB555_8u16u_C3R
RGB565			ippiYCbCrToRGB565_8u16u_C3R
RGB	4	R0 G0 B0 A0 R1 G1 B1 A1	ippiRGBToYUV_8u_AC4R
BGR	3	B0 G0 R0 B1 G1 R1 B2 G2 R2	ippiYCbCrToBGR_8u_P3C3R
BGR444			ippiYCbCrToBGR444_8u16u_C3R
BGR555			ippiYCbCrToBGR555_8u16u_C3R
BGR565			ippiYCbCrToBGR565_8u16u_C3R
BGR	4	B0 G0 R0 A0 B1 G1 R1 A1	ippiBGRToHLS_8u_AC4R
YUV	3	Y0 U0 V0 Y1 U1 V1 Y2 U2 V2	ippiYUVToRGB_8u_C3R
YUV	4	Y0 U0 V0 A0 Y1 U1 V1 A1	ippiYUVToRGB_8u_AC4R
4:2:2 YUV	2	Y0 U0 Y1 V0 Y2 U1 Y3 V1	ippiYUV422T0RGB_8u_C2C3R
YCbCr	3	Y0 Cb0 Cr0 Y1 Cb1 Cr1	ippiYCbCrToRGB_8u_C3R
YCbCr	4	Y0 Cb0 Cr0 A0 Y1 Cb1 Cr1 A1	ippiYCbCrToRGB_8u_AC4R
4:2:2 YCbCr	2	Y0 Cb0 Y1 Cr0 Y2 Cb1 Y3 Cr1	ippiYCbCr422ToRGB_8u_C2C3R
4:2:2 YCrCb	2	Y0 Cr0 Y1 Cb0 Y2 Cr1 Y3 Cb1	ippiYCrCb422ToYCbCr422_8u_C2P3R
4:2:2 CbYCr	2	Cb0 Y0 Cr0 Y1 Cb1 Y2 Cr1 Y3	ippiCbYCr422ToRGB_8u_C2C3R
XYZ	3	X0 Y0 Z0 X1 Y1 Z1 X2 Y2 Z2	ippiXYZToRGB_8u_C3R
XYZ	4	X0 Y0 Z0 X1 Y1 Z1 X2 Y2 Z2	ippiXYZToRGB_16u_AC4R
LUV	3	L0 U0 V0 L1 U1 V1 L2 U2 V2	ippiLUVToRGB_16s_C3R
LUV	4	L0 U0 V0 A0 L1 U1 V1 A1	ippiLUVToRGB_32f_AC4R
YCC	3	Y0 C0 C0 Y1 C1 C1 Y2 C2 C2	ippiYCCToRGB_8u_C3R
YCC	4	Y0 C0 C0 A0 Y1 C1 C1 A1	ippiYCCToRGB_8u_AC4R
HLS	3	H0 L0 S0 H1 L1 S1 H2 L2 S2	ippiHLSToRGB_16u_C3R

Table 6-2 Pixel-Order Image Formats (continued)

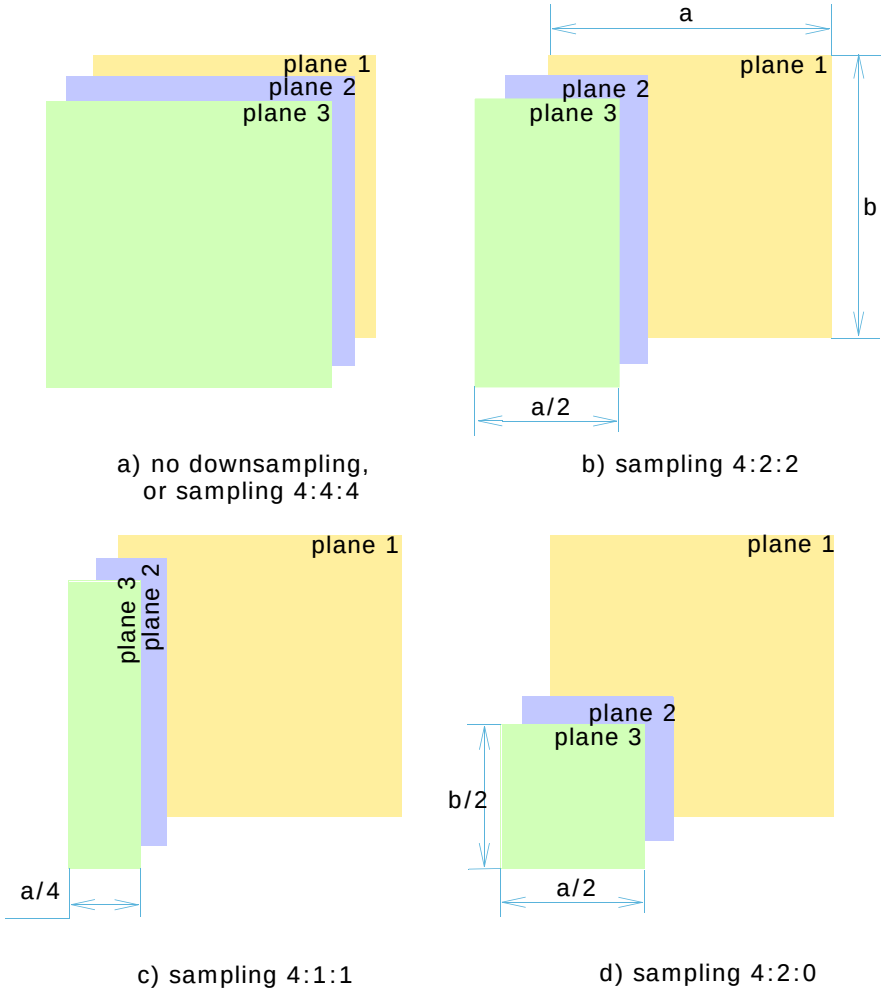
Image Format	Number of Channels	Channel Values Order	Example
HLS	4	H0 L0 S0 A0 H1 L1 S1 A0	ippiHLSToRGB_16u_AC4R
HSV	3	H0 S0 V0 H1 S1 V1 H2 S2 V2	ippiHSVToRGB_16s_C3R
HSV	4	H0 S0 V0 A0 H1 S1 C1 A1	ippiHSVToRGB_16s_AC4R

Planar image formats supported by the Intel IPP color conversion functions are listed in Table 6-3 along with examples of the Intel IPP functions using that format. Planes layout and their relative sizes are shown in Figure 6-15 and Figure 6-16.

Table 6-3 Planar Image Formats

Image Format	Number of Planes	Planes Layout				Example
		pl.1	pl.2	pl.3		
RGB	3	R	G	B	see Figure 6-15 a	ippiRGBToYUV_8u_P3R
YUV	3	Y	U	V	see Figure 6-15 a	ippiYUVToRGB_8u_P3R
4:2:2 YUV	3	Y	U	V	see Figure 6-15, b	ippiYUV422ToRGB_8u_P3
4:2:0 YUV	3	Y	U	V	see Figure 6-15, d	ippiYUV420ToRGB_8uP3C3R
YCbCr	3	Y	Cb	Cr	see Figure 6-15, a	ippiYCbCrToRGB_8u_P3R
4:2:2 YCbCr	3	Y	Cb	Cr	see Figure 6-15, b	ippiYCbCr422_8u_P3C2R
4:1:1 YCbCr	3	Y	Cb	Cr	see Figure 6-15, c	ippiYCbCr411_8u_P3P2R
4:1:1 YCbCr	2	Y	CbCr	-	see Figure 6-16, a	ippiYCbCr411_8u_P2P3R
4:2:0 YCbCr	3	Y	Cb	Cr	see Figure 6-15, d	ippiRGBToYCbCr420_8u_C3P3R
4:2:0 YCbCr	2	Y	CbCr	-	see Figure 6-16, b	ippiYCbCr420_8u_P2P3R
4:2:0 YCrCb	3	Y	Cr	Cb	see Figure 6-15, d	ippiYCrCb420ToYCbCr422_8u_P3R

Figure 6-15 Plane Size and Layout - 3-planes Images



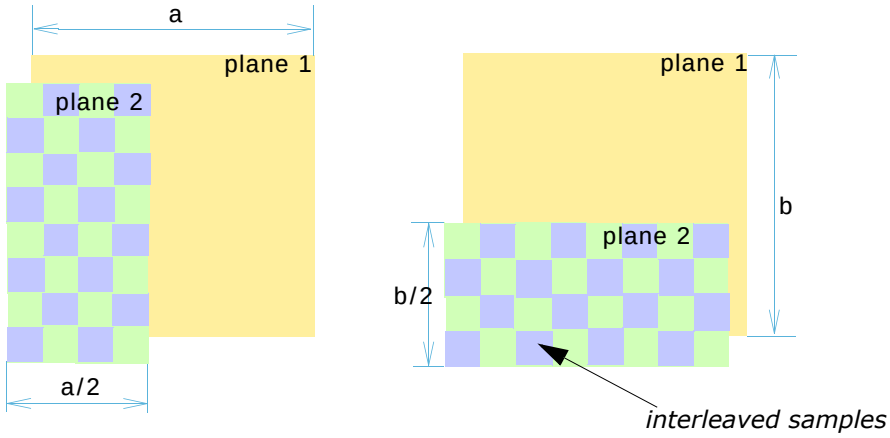
a) no downsampling, or sampling 4:4:4

b) sampling 4:2:2

c) sampling 4:1:1

d) sampling 4:2:0

Figure 6-16 Plane Size and Layout - 2-planes Images



a) sampling 4:1:1 b) sampling 4:2:0

Color Model Conversion

RGBToYUV

Converts an RGB image to the YUV color model.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYUV_<mod>(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R 8u_AC4R

Case 2: Operation on planar data

```
IppStatus ippiRGBToYUV_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 3: Conversion from pixel-order to planar data

```
IppStatus ippiRGBToYUV_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

```
IppStatus ippiRGBToYUV_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[4], int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order data. An array of pointers to the source image ROI in separate color planes in case of planar data.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination ROI for pixel-order data. An array of pointers to destination buffers in separate color planes in case of planar data.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGBToYUV` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

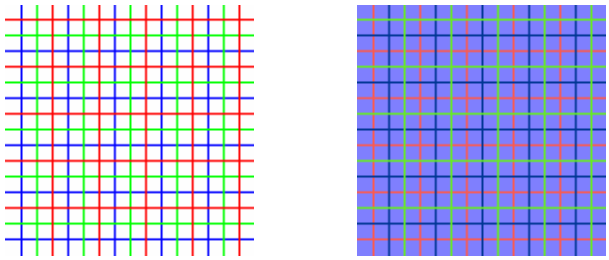
This function converts the gamma-corrected `R'G'B'` image *pSrc* to the Y'U'V' image *pDst* (see Figure 6-17) according to the following formulas:

```
Y' =  0.299*R' + 0.587*G' + 0.114*B'
U' = -0.147*R' - 0.289*G' + 0.436*B' = 0.492*(B'-Y')
V' =  0.615*R' - 0.515*G' - 0.100*B' = 0.877*(R'-Y')
```

For digital RGB values in the range `[0..255]`, `Y'` has the range `[0..255]`, `U` varies in the range `[-112..+112]`, and `V` in the range `[-157..+157]`.
To fit in the range of `[0..255]`, a constant value `128` is added to computed `U` and `V` values, and `V` is then saturated.

Example 6-1 show how to use the function `ippiRGBToYUV_8u_C3R`.

Figure 6-17 Converting an RGB image to YUV



source image destination image

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

Example 6-1 Using the Function `ippiRGBToYUV`

```
Ipp8u src[9*3] = {255, 0, 0, 255, 0, 0, 255, 0, 0,
                  0, 255, 0, 0, 255, 0, 0, 255, 0,
                  0, 0, 255, 0, 0, 255, 0, 0, 255};
Ipp8u dst[9*3];
IppiSize roiSize = { 9, 3 };

ippiRGBToYUV_8u_C3R ( src, 9, dst, 9, roiSize );

Result:
255 0 0 255 0 0 255 0 0
0 255 0 0 255 0 0 255 0     src
0 0 255 0 0 255 0 0 255

77 122 130 0 127 127 17 194 113
 0 255   0 0 255   0  0 255   0      "gamma-corrected"  src
 0   0 255 0   0 255  0   0 255


 76  90 255  76  90 255  76  90 255
149  54   0 149  54   0 149  54   0         dst
 29 239 102  29 239 102  29 239 102
```

YUVToRGB

Converts a YUV image to the RGB color model.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYUVToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R
8u_AC4R

Case 2: Operation on planar data.

```
IppStatus ippiYUVToRGB_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 3: Conversion from planar to pixel-order data.

```
IppStatus ippiYUVToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source buffer for pixel-order data. An array of pointers to separate source color planes in case of planar data.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination buffer for pixel-order data. An array of pointers to separate destination color planes in case of planar data.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYUVToRGB` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'U'V' image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the following formulas:

```
R' = Y' + 1.140*V'
G' = Y' - 0.394*U' - 0.581*V'
B' = Y' + 2.032*U'
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToYUV422

Converts an RGB image to the YUV color model; uses 4:2:2 sampling.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiRGBToYUV422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Operation on planar data with ROI.

```
IppStatus ippiRGBToYUV422_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 3: Operation on planar data without ROI.

```
IppStatus ippiRGBToYUV422_8u_P3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst[3], IppiSize imgSize);
```

Case 4: Conversion from pixel-order to planar data with ROI.

```
IppStatus ippiRGBToYUV422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 5: Conversion from pixel-order to planar data without ROI.

```
IppStatus ippiRGBToYUV422_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3],
    IppiSize imgSize);
```

Parameters

pSrc	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
srcStep	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.
pDst	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffer in each color plane for planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI. An array of three values for planar image.
roiSize	Size of the source and destination ROI in pixels.
imgSize	Size of the source and destination images in pixels for operations without ROI.

Description

The function `ippiRGBToYUV422` is declared in the `ippcc.h` file. This function converts the gamma-corrected `R'G'B'` image *pSrc* to the `Y'U'V'` image *pDst* with 4:2:2 sampling format, according to the same formulas as the function `ippiRGBToYUV` does. For more details on this sampling format, see Table 6-2 and Table 6-3.

For digital RGB values in the range `[0..255]`, `Y'` has the range `[0..255]`, `U` varies in the range `[-112..+112]`, and `V` in the range `[-157..+157]`.
To fit in the range of `[0..255]`, a constant value `128` is added to computed `U` and `V` values, and `V` is then saturated.

Some function flavors operates with ROI (see Regions of Interest in Intel IPP).
The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* or *imgSize* has a field with zero or negative value.

YUV422ToRGB

Converts a YUV image with the 4:2:2 sampling to the RGB color model.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYUV422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Operation on planar data with ROI.

```
IppStatus ippiYUV422ToRGB_8u_P3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 3: Operation on planar data without ROI.

```
IppStatus ippiYUV422ToRGB_8u_P3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst[3], IppiSize imgSize);
```

Case 4: Conversion from planar to pixel-order data with ROI.

```
IppStatus ippiYUV422ToRGB_<mod>(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_P3C3R 8u_P3AC4R

Case 5: Conversion from planar to pixel-order data without ROI.

```
IppStatus ippiYUV422ToRGB_8u_P3C3(const Ipp8u* const pSrc[3],
Ipp8u* pDst, IppiSize imgSize);
```

Parameters

pSrc	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
srcStep	Distance in bytes between starts of consecutive lines in the source image for operations with ROI. An array of three values in case of planar image.

pDst	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffers in each color plane for planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
roiSize	Size of the source and destination ROI in pixels.
imgSize	Size of the source and destination images in pixels for operations without ROI.

Description

The function `ippiYUV422ToRGB` is declared in the `ippcc.h` file. This function converts the Y'U'V' image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that `ippiYUV422ToRGB` 4:2:0 sampling the input data to be in 4:2:2 sampling format (see Table 6-2 and Table 6-3 for more details).

The function `ppiYUV422ToRGB_P3AC4R` additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI (see Regions of Interest in Intel IPP).
The function flavors that do not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step arguments are not needed.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* or *imgSize* has a field with zero or negative value.

RGB565ToYUV422

Convert 16-bit RGB image to the 4:2:2 YUV image.

Syntax

```
IppStatus ippiRGB565ToYUV422_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to ROI in the separate color planes in the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGB565ToYUV422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected `R'G'B'` image *pSrc* to the Y'U'V' image *pDst* according to the same formulas as the function `ippiRGBToYUV` does.

The source image *pSrc* is a packed 16-bit RGB565 image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type) in the following order: 5 bits for red, 6 bits for green, 5 bits for blue (see Figure 6-14).

The destination image *pDst* has a 4:2:2 sampling format (see Table 6-3 for more details).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsDoubleSize`	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

RGBToYUV420

Converts an RGB image to the 4:2:0 YUV image.

Syntax

Case 1: Operation on planar data with ROI

```
IppStatus ippiRGBToYUV420_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 2: Operation on planar data without ROI

```
IppStatus ippiRGBToYUV420_8u_P3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst[3], IppiSize imgSize);
```

Case 3: Conversion from pixel-order to planar data with ROI

```
IppStatus ippiRGBToYUV420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 4: Conversion from pixel-order to planar data without ROI

```
IppStatus ippiRGBToYUV420_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3],
    IppiSize imgSize);
```

Parameters

pSrc	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffers in each color plane for planar image.
srcStep	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.
pDst	An array of pointers to the destination image buffers in each color plane.
dstStep	An array of distances in bytes between starts of consecutive lines in each plane of the destination image for operations with ROI.
roiSize	Size of the source and destination ROI in pixels.
imgSize	Size of the source and destination images in pixels for operations without ROI.

Description

The function `ippiRGBToYUV420` is declared in the `ippcc.h` file. This function converts the gamma-corrected `R'G'B'` image *pSrc* to the Y'U'V' image *pDst* with the 4:2:0 sampling (see Table 6-3 for more details). The conversion is performed in the accordance with the same formulas as the function `ippiRGBToYUV` does.

For digital RGB values in the range `[0..255]`, `Y'` has the range `[0..255]`, `U` varies in the range `[-112..+112]`, and `V` in the range `[-157..+157]`.
To fit in the range of `[0..255]`, a constant value `128` is added to computed `U` and `V` values, and `V` is then saturated.
Some function flavors operates with ROI see Regions of Interest in Intel IPP).
The function flavors that does not use ROI operate on the assumption that both the source

and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

roiSize.width (*imgSize.width*) and *roiSize.height* (*imgSize.height*) should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* or *imgSize* has a field with zero or negative value.
ippStsDoubleSize	Indicates a warning if *roiSize* or *imgSize* has a field that is not a multiple of 2.

YUV420ToRGB

Converts a YUV image that has 4:2:0 sampling format to the RGB image.

Syntax

Case 1: Operation on planar data with ROI.

```
IppStatus ippiYUV420ToRGB_8u_P3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 2: Operation on planar data without ROI.

```
IppStatus ippiYUV420ToRGB_8u_P3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst[3], IppiSize imgSize);
```

Case 3: Conversion from planar to pixel-order data with ROI.

```
IppStatus ippiYUV420ToRGB_<mod>(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_P3C3R 8u_P3AC4R

Case 4: Conversion from planar to pixel-order data without ROI.

```
IppStatus ippiYUV420ToRGB_8u_P3C3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst, IppiSize imgSize);
```

Parameters

pSrc	An array of pointers to the source image buffers in each color plane.
srcStep	An array of distances in bytes between starts of consecutive lines in each source image planes for operations with ROI.
pDst	Pointer to the destination image buffer for pixel-order images. An array of pointers to the destination image buffers in each color plane for planar images.
dstStep	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
roiSize	Size of the source and destination ROI in pixels.
imgSize	Size of the source and destination images in pixels for operations without ROI.

Description

The function `ippiYUV420ToRGB` is declared in the `ippcc.h` file. This function converts the Y'U'V' image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that `ippiYUV420ToRGB` 4:2:0 sampling the input data to be in 4:2:0 sampling format (see Table 6-3 for more details).

The function `ippiYUV420ToRGB_P3AC4R` additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI see Regions of Interest in Intel IPP).
The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

roiSize.width (*imgSize.width*) and *roiSize.height* (*imgSize.height*) should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* or *imgSize* has a field with zero or negative value.
ippStsDoubleSize	Indicates a warning if *roiSize* or *imgSize* has a field that is not a multiple of 2.

BGRToYUV420

Converts an BGR image to the YUV color model; uses 4:2:0 sampling

Syntax

```
IppStatus ippiBGRToYUV420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to the destination image buffers in each color plane.
dstStep	An array of distances in bytes between starts of consecutive lines in each plane of the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiBGRToYUV420 is declared in the ippcc.h file. It operates with ROI see Regions of Interest in Intel IPP).

This function converts the gamma-corrected B'G'R' image *pSrc* to the Y'U'V' image *pDst* with the 4:2:0 sampling (see Table 6-3 for more details). The function uses the same formulas as the function ippiRGBToYUV does.

For digital BGR values in the range [0..255], Y' varies in the range [0..255], U - in the range [-112..+112], and V - in the range [-157..+157].
To fit in the range of [0..255], a constant value 128 is added to the computed U and V values, and V is then saturated.

roiSize.width and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* or *imgSize* has a field with zero or negative value.
ippStsDoubleSize	Indicates a warning if *roiSize* or *imgSize* has a field that is not a multiple of 2.

YUV420ToBGR

Converts a YUV image that has 4:2:0 sampling to the BGR image.

Syntax

```
IppStatus ippiYUV420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in each color plane in the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiYUV420ToBGR is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'U'V' image *pSrc* to the gamma-corrected B'G'R' image *pDst* according to the same formulas as the function ippiYUVToRGB does. The input data must be presented in the 4:2:0 sampling format (see Table 6-3 for more details).

roiSize.width and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsDoubleSize	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

RGB565ToYUV420

Converts 16-bit RGB image to the 4:2:0 YUV image.

Syntax

```
IppStatus ippiRGB565ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to ROI in the separate color planes in the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGB565ToYUV420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected `R'G'B'` image *pSrc* to the Y'U'V' image *pDst* according to the same formulas as the function `ippiRGBToYUV` does.

The source image *pSrc* is a packed 16-bit RGB565 image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type) in the following order: 5 bits for red, 6 bits for green, 5 bits for blue (see Figure 6-14).

The destination image *pDst* has a 4:2:0 sampling format (see Table 6-3 for more details).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsDoubleSize`	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

YUV420ToRGB565
YUV420ToRGB555
YUV420ToRGB444

Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.

Syntax

```
IppStatus ippiYUV420ToRGB565_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB555_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB444_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in each color plane in the source image.

srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYUV420ToRGB444`, `ippiYUV420ToRGB555`, `ippiYUV420ToRGB565` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'U'V' image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that the input data are in 4:2:0 sampling format (see Table 6-3 for more details). The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), `RGB555` (5 bits for red, green, blue), or `RGB444` (4 bits for red, green, blue). Bit reduction is performed by discarding the least significant bits in the image after color conversion.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

YUV420ToRGB565Dither
YUV420ToRGB555Dither
YUV420ToRGB444Dither

Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.

Syntax

```
IppStatus ippiYUV420ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in each color plane in the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYUV420ToRGB565Dither`, `ippiYUV420ToRGB555Dither`, `ippiYUV420ToRGB444Dither` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'U'V' image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that the input data are in 4:2:0 sampling format (see Table 6-3 for more details). The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `RGB565` (5 bits for red, 6 bits for green, 5

bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [Thomas].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGR565ToYUV420
BGR555ToYUV420

Convert 16-bit BGR image to the 4:2:0 YUV image.

Syntax

```
IppStatus ippiBGR565ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGR555ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to ROI in the separate color planes in the destination image.
dstStep	Array of istances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions ippiBGR565ToYUV420, ippiBGR555ToYUV420 are declared in the ippcc.h file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the gamma-corrected B'G'R' image *pSrc* to the Y'U'V' image *pDst* according to the same formulas as the function ippiRGBToYUV does. The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see Figure 6-14): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red). The destination image *pDst* has a 4:2:0 sampling format (see Table 6-3 for more details).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 0.

YUV420ToBGR565
YUV420ToBGR555
YUV420ToBGR444

Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel BGR image.

Syntax

```
IppStatus ippiYUV420ToBGR565_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToBGR555_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToBGR444_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in each separate color plane in the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYUV420ToBGR565`, `ippiYUV420ToBGR555`, `ippiYUV420ToBGR444` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'U'V' image *pSrc* to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that the input data are in 4:2:0 sampling format (see Table 6-3 for more details). The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), `BGR555` (5 bits for blue, green, red), or `BGR444` (4 bits for blue, green, red). Bit reduction is performed by discarding the least significant bits in the image after color conversion.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

YUV420ToBGR565Dither
YUV420ToBGR555Dither
YUV420ToBGR444Dither

Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.

Syntax

```
IppStatus ippiYUV420ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYUV420ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV420ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in each separate color plane in the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYUV420ToBGR565Dither`, `ippiYUV420ToBGR555Dither`, `ippiYUV420ToBGR444Dither` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'U'V' image *pSrc* to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYUVToRGB` does. The difference is that the input data are in 4:2:0 sampling format (see Table 6-3 for more details). The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), `BGR555` (5 bits for blue, green, red), or `BGR444` (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [Thomas].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToYCbCr

Converts an RGB image to the YCbCr color model.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R 8u_AC4R

Case 2: Operation on planar data.

```
IppStatus ippiRGBToYCbCr_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 3: Conversion from pixel-order to planar data

```
IppStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3P3R 8u_AC4P3R

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination imnage ROI. Array of pointers to ROI in the separate destination color planes for planar images.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGBToYCbCr` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected `R'G'B'` image *pSrc* with values in the range `[0..255]` to the Y'Cb'Cr' image *pDst* according to the following formulas:

```
Y'  =   0.257*R' + 0.504*G' + 0.098*B' + 16
Cb' = -0.148*R' - 0.291*G' + 0.439*B' + 128
Cr' =  0.439*R' - 0.368*G' - 0.071*B' + 128
```

In `YCbCr` model, `Y` is defined to have a nominal range `[16..235]`, while `Cb` and `Cr` are defined to have a range `[16..240]`, with `128` value as corresponding to zero.

Both the source and destination images have the same bit depth.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCrToRGB

Converts a YCbCr image to the RGB color model.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCrToRGB_<mod>(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

`8u_C3R` `8u_AC4R`

Case 2: Operation on planar data

```
IppStatus ippiYCbCrToRGB_<mod>(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_P3R 8u_P3C3R

Case 3: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCrToRGB_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep,
    Ipp8u* pDst , int dstStep, IppiSize roiSize, Ipp8u aval);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. Array of pointers to the ROI in each separate source color planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI. Array of pointers to the ROI in the separate destination color planes for planar images.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

The function ippiYCbCrToRGB is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image *pSrc* to the 24-bit gamma-corrected R'G'B' image *pDst*. The following formulas are used for conversion:

```
R' = 1.164*(Y' - 16) + 1.596*(Cr' - 128)
G' = 1.164*(Y' - 16) - 0.813*(Cr' - 128) - 0.392*(Cb' - 128)
B' = 1.164*(Y' - 16) + 2.017*(Cb' - 128)
```

The output R'G'B' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCrToBGR

Converts a YCbCr image to the BGR color model.

Syntax

```
IppStatus ippiYCbCrToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Parameters

pSrc	An array of pointers to ROI in each separate source color planes.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

The function `ippiYCbCrToBGR` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image *pSrc* to the 24-bit gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does. The output `B'G'R'` values are saturated to the range `[0..255]`.

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 4 or *roiSize.height* is less than 1.

YCbCrToRGB565
YCbCrToRGB555
YCbCrToRGB444

Convert a YCbCr image to the 16-bit per pixel RGB image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCrToRGB565_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB444_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCrToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

roiSize	Size of the source and destination ROI in pixels.

Description

The functions ippiYCbCrToRGB565, ippiYCbCrToRGB555, ippiYCbCrToRGB444 are declared in the ippcc.h file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the same formulas as the function ippiYCbCrToRGB does.
The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see Figure 6-14 for more details): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue).
Bit reduction is performed by discarding the least significant bits in the image after color conversion.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCrToRGB565Dither
YCbCrToRGB555Dither
YCbCrToRGB444Dither

Convert a YCbCr image to the 16-bit per pixel RGB image with dithering.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCrToRGB565Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToRGB444Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data.

```
IppStatus ippiYCbCrToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCrToRGB565Dither`, `ippiYCbCrToRGB555Dither`, `ippiYCbCrToRGB444Dither` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14 for more details): `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), `RGB555` (5 bits for red, green, blue), or `RGB444` (4 bits for red, green, blue). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [Thomas].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCrToBGR565
YCbCrToBGR555
YCbCrToBGR444

Convert a YCbCr image to the 16-bit per pixel BGR image.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCrToBGR565_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR444_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data.

```
IppStatus ippiYCbCrToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

roiSize	Size of the source and destination ROI in pixels.

Description

The functions ippiYCbCrToBGR565, ippiYCbCrToBGR555, ippiYCbCrToBGR444 are declared in the ippcc.h file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc* to the gamma-corrected B'G'R' image *pDst* according to the same formulas as the function ippiYCbCrToRGB does.
The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see Figure 6-14): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), or BGR444 (4 bits for blue, green, red).
Bit reduction is performed by discarding the least significant bits in the image after color conversion.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCrToBGR565Dither
YCbCrToBGR555Dither
YCbCrToBGR444Dither

Convert a YCbCr image to the 16-bit per pixel BGR image with dithering.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR444Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data.

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCrToBGR565Dither`, `ippiYCbCrToBGR555Dither`, `ippiYCbCrToBGR444Dither` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc* to the gamma-corrected B'G'R' image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14 for more details): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), `BGR555` (5 bits for blue, green, red), or `BGR444` (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [Thomas].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToYCbCr422

Converts an RGB image to the YCbCr image with 4:2:2 sampling.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2; Conversion from pixel-order to planar data

```
IppStatus, ippiRGBToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiRGBToYCbCr422_8u_P3C2R(const Ipp8u* const pSrc[3],
    int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI for pixel-order image. An array of pointer to ROI in each separate planes for the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGBToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function converts the gamma-corrected `R'G'B'` image *pSrc* to the Y'Cb'Cr' image *pDst* with 4:2:2 sampling (see Table 6-2 and Table 6-3 for more details). The conversion is performed according to the same formulas as the function `ippiRGBToYCbCr` does.

The converted buffer for pixel-order image has the reduced bit depth of 16 bits per pixel, whereas the source buffer has 24 bit depth.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCr422ToRGB

Converts an YCbCr image with the 4:2:2 sampling to the RGB image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiYCbCr422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 3: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCr422ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source planes for planar images.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYCbCr422ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image *pSrc* with the 4:2:2 sampling (see Table 6-2 and Table 6-3 for more details) to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The output `R'G'B'` values are saturated to the range `[0..255]`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToYCrCb422

Converts 24-bit per pixel RGB image to 16-bit per pixel YCrCb image

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYCrCb422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiRGBToYCrCb422_8u_P3C2R(const Ipp8u* const pSrc[3],
    int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGBToYCrCb422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function converts the gamma-corrected `R'G'B'` image *pSrc* to the Y'Cr'Cb' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiRGBToYCrCb422` uses 4:2:2 sampling format for the converted image (see Table 6-2 and Table 6-3 for more details).

The converted buffer has the reduced bit depth of 16 bits per pixel, whereas the source buffer has 24 bit depth.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCrCb422ToRGB

Converts 16-bit per pixel YCrCb image to 24-bit per pixel RGB image.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYCrCb422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from pixel-order to planar data.

```
IppStatus ippiYCrCb422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYCrCb422ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cr'Cb' image *pSrc*, packed in 4:2:2 sampling format (see Table 6-2 and Table 6-3 for more details) to the 24-bit gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The output `R'G'B'` values are saturated to the range `[0..255]`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGRToYCbCr422

Converts 24-bit per pixel BGR image to 16-bit per pixel YCbCr image.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiBGRToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToYCbCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from pixel-order to planar data.

```
IppStatus ippiBGRToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr422_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source mage ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination pixel-order image. An array of distances in bytes for each plane of the destination planar image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected `B'G'R'` image *pSrc* to the two-channel or three-planes Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr422` uses 4:2:2 sampling format (see Table 6-2 and Table 6-3 for more details).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 1.

YCbCr422ToBGR

Converts a YCbCr image with 4:2:2 sampling to the BGR image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCr422ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI. An array of pointers to the ROI in each separate plane of the source planar image.
srcStep	Distance in bytes between starts of consecutive lines in the source image. An array of such distances in bytes for each plane of the source planar image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

The function ippiYCbCr422ToBGR is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image *pSrc* with 4:2:2 sampling (see Table 6-2 and Table 6-3 for more details) to the gamma-corrected B'G'R' image *pDst* according to the same formulas as the function ippiYCbCrToRGB does.

The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 1.

BGR565ToYCbCr422
BGR555ToYCbCr422

Converts 16-bit per pixel BGR image to 16-bit per pixel YCbCr image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiBGR565ToYCbCr422_16u8u_C3C2R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGR555ToYCbCr422_16u8u_C3C2R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiBGR565ToYCbCr422_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGR555ToYCbCr422_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination pixel-order image. An array of distances in bytes for each plane of the destination planar image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiBGR565ToYCbCr422` and `ippiBGR555ToYCbCr422` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the two-channel or three-planes Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that these functions use 4:2:2 sampling format (see Table 6-2 and Table 6-3 for more details).
The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of two possible formats (see Figure 6-14 for more details): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), and `BGR555` (5 bits for blue, green, red).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 1.

RGBToCbYCr422
RGBToCbYCr422Gamma

Convert 24-bit per pixel RGB image to 16-bit per pixel CbYCr image.

Syntax

```
IppStatus ippiRGBToCbYCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToCbYCr422Gamma_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiRGBToCbYCr422` and `ippiRGBToCbYCr422Gamma` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

The function `ippiRGBToCbYCr422` converts the gamma-corrected `R'G'B'` image *pSrc* to the Cb'Y'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does.
The function `ippiRGBToCbYCr422Gamma` performs gamma-correction of the source `RGB` image *pSrc* according to the same formula as the function `ippiGammaFwd` does, and then converts it to the Cb'Y'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does.
The functions `ippiRGBToCbYCr422` and `ippiRGBToCbYCr422Gamma` use 4:2:2 sampling format for the converted image.

An `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1`, `Cb1Y2Cr1Y3`, ...

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

CbYCr422ToRGB

Converts 16-bit per pixel CbYCr image to 24-bit per pixel RGB image.

Syntax

```
IppStatus ippiCbYCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels

Description

The function `ippiCbYCr422ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Cb'Y'Cr' image *pSrc*, packed in 4:2:2 sampling format, to the 24-bit gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
A `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1`, `Cb1Y2Cr1Y3`, ...

The output `R'G'B'` values are saturated to the range `[0..255]`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGRToCbYCr422

Converts 32-bit per pixel BGR image to 16-bit per pixel CbYCr image.

Syntax

```
IppStatus ippiBGRToCbYCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the four-channel gamma-corrected `B'G'R'` image *pSrc* to the two-channel Cb'Y'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The function `ippiBGRToCbYCr422` uses 4:2:2 sampling format for the converted image. An alpha-channel information is lost.

An `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1`, `Cb1Y2Cr1Y3`, ...

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGRToCbYCr422_709HDTV

Converts BGR image to 16-bit per pixel CbYCr image for ITU-R BT.709 HDTV signal.

Syntax

```
IppStatus ippiBGRToCbYCr422_709HDTV_8u_C3C2R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToCbYCr422_709HDTV_8u_AC4C2R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToCbYCr422_709HDTV` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the three- or four-channel gamma-corrected `B'G'R'` image *pSrc* to the two-channel Cb'Y'Cr' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The source image pixel values are in the range `[16..235]`. The conversion is performed according to the following formulas [Jack01]:

```
Y'  =   0.213*R' + 0.715*G' + 0.072*B'
Cb' = -0.117*R' - 0.394*G' + 0.511*B' + 128
Cr' =  0.511*R' - 0.464*G' - 0.047*B' + 128
```

The values of `Y'` of the destination image are in the range `[16..235]`, the values of `Cb'`, `Cr'` are in the range `[16..240]`. They should be saturated at the 1 and 254 levels.

The function ippiBGRToCbYCr422_709HDTV uses 4:2:2 sampling format for the converted image. The alpha-channel information is lost.

An CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

CbYCr422ToBGR

Converts 16-bit per pixel CbYCr image to four channel BGR image.

Syntax

```
IppStatus ippiCbYCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

The function ippiCbYCr422ToBGR is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Cb'Y'Cr' image *pSrc*, packed in 4:2:2 sampling format, to the four channel gamma-corrected B'G'R' image *pDst* according to the same formulas as the function ippiYCbCrToRGB does.
A CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...

The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

CbYCr422ToBGR_709HDTV

Converts 16-bit per pixel CbYCr image to the BGR image for ITU-R BT.709 HDTV signal.

Syntax

```
IppStatus ippiCbYCr422ToBGR_709HDTV_8u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCbYCr422ToBGR_709HDTV_8u_C2C4R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

The function ippiCbYCr422ToBGR_709HDTV is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Cb'Y'Cr' image *pSrc*, packed in 4:2:2 sampling format, to the three- or four-channel gamma-corrected B'G'R' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). A source CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... The values of Y' are in the range [16..235], the values of Cb', Cr' are in the range [16..240]. The conversion is performed according to the following formulas [Jack01]:

```
R' = Y' + 1.540*(Cr' - 128)
G' = Y' - 0.459*(Cr' - 128) - 0.183*(Cb' - 128)
B' = Y' + 1.816*(Cb' - 128)
```

The destination image pixel values have a nominal range [16..235]. The resulting R'G'B' values should be saturated at the 0 and 255 levels.

The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCr422ToRGB565
YCbCr422ToRGB555
YCbCr422ToRGB444

Convert 16-bit per pixel YCbCr image to 16-bit per pixel RGB image.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCr422ToRGB565_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data .

```
IppStatus ippiYCbCr422ToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize));
IppStatus ippiYCbCr422ToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCr422ToRGB565`, `ippiYCbCr422ToRGB565`, `ippiYCbCr422ToRGB565` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc*, packed in 4:2:2 sampling format (see Table 6-2 for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), `RGB555` (5 bits for red, green, blue), `RGB444` (4 bits for red, green, blue).
Bit reduction is performed by discarding the least significant bits in the image after color conversion.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCr422ToRGB565Dither
YCbCr422ToRGB555Dither
YCbCr422ToRGB444Dither

Convert 16-bit per pixel YCbCr image to 16-bit per pixel RGB image with dithering.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCr422ToRGB565Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToRGB444Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data .

```
IppStatus ippiYCbCr422ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCr422ToRGB565Dither`, `ippiYCbCr422ToRGB565Dither`, `ippiYCbCr422ToRGB565Dither` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc*, packed in 4:2:2 sampling format (see Table 6-2 for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), `RGB555` (5 bits for red, green, blue), `RGB444` (4 bits for red, green, blue).Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [Thomas].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCr422ToBGR565
YCbCr422ToBGR555
YCbCr422ToBGR444

Convert 16-bit per pixel YCbCr image to 16-bit per pixel BGR image.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCr422ToBGR565_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR555_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data .

```
IppStatus ippiYCbCr422ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCr422ToBGR565, ippiYCbCr422ToBGR565, ippiYCbCr422ToBGR565` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc*, packed in 4:2:2 sampling format (see Table 6-2 for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), `BGR555` (5 bits for blue, green, red), `BGR444` (4 bits for blue, green, red).
Bit reduction is performed by discarding the least significant bits in the image after color conversion.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbC422ToBGR565Dither
YCbC422ToBGR555Dither
YCbCr422ToBGR444Dither

Convert 16-bit per pixel YCbCr image to 16-bit per pixel BGR image with dithering.

Syntax

Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCr422ToBGR565Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToBGR555Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data.

```
IppStatus ippiYCbCr422ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCr422ToBGR565Dither`, `ippiYCbCr422ToBGR565Dither`, `ippiYCbCr422ToBGR565Dither` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc*, packed in 4:2:2 sampling format (see Table 6-2 for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), `BGR555` (5 bits for blue, green, red), `BGR444` (4 bits for blue, green, red).Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [Thomas].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToYCbCr420

Converts an RGB image to the YCbCr color model; uses 4:2:0 sampling.

Syntax

```
IppStatus ippiRGBToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in the separate planes of the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiRGBToYCbCr420 is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'Cb'Cr' image *pDst* according to the same formulas as the function ippiRGBToYCbCr does. The difference is that ippiRGBToYCbCr420 uses 4:2:0 sampling format for the converted image (see Table 6-3 for more details).

roiSize.width and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsDoubleSize	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

YCbCr420ToRGB

Converts a YCbCr image that has 4:2:0 sampling format to the RGB color model.

Syntax

```
IppStatus ippiYCbCr420ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiYCbCr420ToRGB is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the same formulas as the function ippiYCbCrToRGB does. The difference is that ippiYCbCr420ToRGB 4:2:0 sampling the input data to be in 4:2:0 sampling format, in which the number of Cb and Cr samples is reduced by half in both vertical and horizontal directions (see Table 6-3 for more details).

roiSize.width and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsDoubleSize	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

YCbCr420ToRGB565
YCbCr420ToRGB555
YCbCr420ToRGB444

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.

Syntax

```
IppStatus ippiYCbCr420ToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.

srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

The functions `ippiYCbCr420ToRGB565`, `ippiYCbCr422ToRGB565`, `ippiYCbCr422ToRGB565` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc*, packed in 4:2:0 sampling format (see Table 6-2 for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), `RGB555` (5 bits for red, green, blue), `RGB444` (4 bits for red, green, blue).
Bit reduction is performed by discarding the least significant bits in the image after color conversion.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCr420ToRGB565Dither
YCbCr420ToRGB555Dither
YCbCr420ToRGB444Dither

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.

Syntax

```
IppStatus ippiYCbCr420ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCr420ToRGB565Dither`, `ippiYCbCr420ToRGB555Dither`, `ippiYCbCr420ToRGB444Dither` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc*, packed in 4:2:0 sampling format (see Table 6-2 for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14 for more details): `RGB565` (5 bits for

red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [Thomas].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToYCrCb420

Converts an RGB image to the YCrCb image with 4:2:0 sampling format.

Syntax

```
IppStatus ippiRGBToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to ROI in separate planes of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiRGBToYCrCb420 is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a four-channel gamma-corrected R'G'B' image *pSrc* to the planar Y'Cr'Cb' image *pDst* with the 4:2:0 sampling (see Table 6-3 for more details). The conversion is performed according to the same formulas as the function `ippiRGBToYCbCr` does.

roiSize.width and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsDoubleSize`	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

YCrCb420ToRGB

Converts a YCrCb image with the 4:2:0 sampling to the RGB image.

Syntax

```
IppStatus, ippiYCrCb420ToRGB_8u_P3C4R,(const Ipp8u* pSrc[3],int
    srcStep[3],Ipp8u* pDst, int dstStep, IppiSize roiSize,  Ipp8u aval);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create fourth channel.

Description

The function `ippiYCrCb420ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cr'Cb' image *pSrc* with the 4:2:0 sampling (see Table 6-3 for more details) to the four-channel gamma-corrected `R'G'B'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.

Fourth channel is created by setting channel values to the constant value *aval*.

roiSize.width and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsDoubleSize`	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

BGRToYCbCr420

Converts a BGR image to the YCbCr image with 4:2:0 sampling format.

Syntax

```
IppStatus ippiBGRToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr420` uses 4:2:0 sampling format (see Table 6-3 for more details).

roiSize.width and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 2.
`ippStsDoubleSize`	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

BGRToYCbCr420_709CSC

Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 CSC signal.

Syntax

```
IppStatus ippiBGRToYCbCr420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToYCrCb420_709CSC` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for computer systems consideration (CSC). The source image pixel values are in the range `[0..255]`. The conversion is performed according to the following formulas [Jack01]:

```
Y'  =   0.183*R' + 0.614*G' + 0.062*B' + 16
Cb' = -0.101*R' - 0.338*G' + 0.439*B' + 128
Cr' =  0.439*R' - 0.399*G' - 0.040*B' + 128
```

The destination image *pDst* has 4:2:0 sampling format (see Table 6-3 for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 2.
`ippStsDoubleSize`	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

BGRToYCbCr420_709HDTV

Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 HDTV signal.

Syntax

```
IppStatus ippiBGRToYCbCr420_709HDTV_8u_AC4P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToYCrCb420_709HDTV` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The source image pixel values are in the range `[16..235]`. The conversion is performed according to the following formulas [Jack01]:

```
Y'  =   0.213*R' + 0.715*G' + 0.072*B'
Cb' = -0.117*R' - 0.394*G' + 0.511*B' + 128
Cr' =  0.511*R' - 0.464*G' - 0.047*B' + 128
```

The values of `Y'` of the destination image are in the range `[16..235]`, the values of `Cb'`, `Cr'` are in the range `[16..240]`. They should be saturated at the 1 and 254 levels.

The destination image *pDst* has 4:2:0 sampling format (see Table 6-3 for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 2.
ippStsDoubleSize	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

BGRToYCrCb420_709CSC

Converts a BGR image to the YCrCb image with 4:2:0 sampling for ITU-R BT.709 CSC signal.

Syntax

```
IppStatus ippiBGRToYCrCb420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCrCb420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiBGRToYCrCb420_709CSC is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cr'Cb' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for computer systems consideration (CSC). The source image pixel values are in the range [0..255]. The conversion is performed according to the following formulas [Jack01]:

```
Y'  =   0.183*R' + 0.614*G' + 0.062*B' + 16
Cb' = -0.101*R' - 0.338*G' + 0.439*B' + 128
Cr' =  0.439*R' - 0.399*G' - 0.040*B' + 128
```

The destination image *pDst* has 4:2:0 sampling format (see Table 6-3 for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 2.
ippStsDoubleSize	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

YCbCr420ToBGR

Converts a YCbCr image with the 4:2:0 sampling to the BGR image.

Syntax

```
IppStatus ippiYCbCr420ToBGR_8u_P3C3R(const Ipp8u* const pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int
    srcStep[3],Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.

srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create fourth channel.

Description

The function `ippiYCbCr420ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image *pSrc* with the 4:2:0 sampling (see Table 6-3 for more details) to the gamma-corrected three- or four-channel `B'G'R'` image *pDst*. The conversion is performed according to the same formulas as the function `ippiYCbCrToRGB` does.

Fourth channel is created by setting channel values to the constant value *aval*.

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsDoubleSize`	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

YCbCr420ToBGR_709CSC

Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 CSC signal.

Syntax

```
IppStatus ippiYCbCr420ToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYCbCr420ToBGR_709CSC` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a planar Y'Cb'Cr' image *pSrc* to the three-channel gamma-corrected B'G'R' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for computer systems consideration (CSC). The conversion is performed according to the following formulas [Jack01]:

```
R' = 1.164*(Y' - 16) + 1.793*(Cr' - 128)
G' = 1.164*(Y' - 16) - 0.534*(Cr' - 128) - 0.213*(Cb' - 128)
B' = 1.164*(Y' - 16) + 2.115*(Cb' - 128)
```

The output `R'G'B'` values are saturated to the range `[0..255]`.

The source image *pDst* has 4:2:0 sampling format (see Table 6-3 for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsDoubleSize	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

YCbCr420ToBGR_709HDTV

Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 HDTV signal.

Syntax

```
ippiYCbCr420ToBGR_709HDTV_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create fourth channel.

Description

The function ippiYCbCr420ToBGR_709HDTV is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a planar Y'Cb'Cr' image *pSrc* to the four-channel gamma-corrected B'G'R' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The values of Y' are in the range [16..235], the values of Cb', Cr' are in the range [16..240]. The conversion is performed according to the following formulas [Jack01]:

```
R' = Y' + 1.540*(Cr' - 128)
G' = Y' - 0.459*(Cr' - 128) - 0.183*(Cb' - 128)
B' = Y' + 1.816*(Cb' - 128)
```

The destination image pixel values have a nominal range `[16..235]`. The resulting `R'G'B'` values should be saturated at the 0 and 255 levels.

The source image *pDst* has 4:2:0 sampling format (see Table 6-3 for more details).

The values of *roiSize.width* and *roiSize.height* should be multiples of 2. Otherwise the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsDoubleSize`	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

YCbCrToBGR_709CSC

Converts a YCbCr image to the BGR image for ITU-R BT.709 CSC signal.

Syntax

```
IppStatus ippiYCbCrToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR_709CSC_8u_P3C4R(const Ipp8u* pSrc[3], int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create fourth channel.

Description

The function `ippiYCbCrToBGR_709CSC` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a planar Y'Cb'Cr' image *pSrc* to the three- or four-channel gamma-corrected B'G'R' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for computer systems consideration (CSC). The conversion is performed according to the following formulas [Jack01]:

```
R' = 1.164*(Y' - 16) + 1.793*(Cr' - 128)
G' = 1.164*(Y' - 16) - 0.534*(Cr' - 128) - 0.213*(Cb' - 128)
B' = 1.164*(Y' - 16) + 2.115*(Cb' - 128)
```

The output `R'G'B'` values are saturated to the range `[0..255]`.

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGR565ToYCbCr420, BGR555ToYCbCr420

Converts a 16-bit per pixel BGR image to the YCbCr image that has 4:2:0 sampling format.

Syntax

```
IppStatus ippiBGR565ToYCbCr420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGR555ToYCbCr420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiBGR565ToYCbCr420` and `ippiBGR555ToYCbCr420` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert a three-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The destination image has a 4:2:0 sampling format (see Table 6-3 for more details). The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of two possible formats (see Figure 6-14 for more details): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red).

roiSize.width and *roiSize.height* should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 2.
`ippStsDoubleSize`	Indicates a warning if *roiSize* has a field that is not a multiple of 2.

YCbCr420ToBGR565, YCbCr420ToBGR555, YCbCr420ToBGR444

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel BGR image.

Syntax

```
IppStatus ippiYCbCr420ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCr420ToBGR565`, `ippiYCbCr420ToBGR565`, `ippiYCbCr420ToBGR565` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc*, packed in 4:2:0 sampling format (see Table 6-2 for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), `BGR555` (5 bits for blue, green, red), `BGR444` (4 bits for blue, green, red).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCbCr420ToBGR565Dither
YCbCr420ToBGR555Dither
YCbCr420ToBGR444Dither

Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.

Syntax

```
IppStatus ippiYCbCr420ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCr420ToBGR565Dither`, `ippiYCbCr420ToBGR555Dither`, `ippiYCbCr420ToBGR444Dither` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the Y'Cb'Cr' image *pSrc*, packed in 4:2:0 sampling format (see Table 6-2 for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB` does.
The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see Figure 6-14 for more details): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), `BGR555` (5 bits for blue, green, red), `BGR444` (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [Thomas].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGRToYCrCb420

Converts a BGR image to the YCrCb image with 4:2:0 sampling format.

Syntax

```
IppStatus ippiBGRToYCrCb420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected `B'G'R'` image *pSrc* to the planar Y'Cr'Cb' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The destination image *pDst* has 4:2:0 sampling format and the following order of pointers: `Y`-plane, `Cr`-plane, `Cb`-plane (see Table 6-3 for more details).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 2.

BGR565ToYCrCb420, BGR555ToYCrCb420

Converts a 16-bit per pixel BGR image to the YCrCb image with 4:2:0 sampling format.

Syntax

```
IppStatus ippiBGR565ToYCrCb420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGR555ToYCrCb420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiBGR565ToYCrCb420` and `ippiBGR555ToYCrCb420` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert a three-channel gamma-corrected `B'G'R'` image *pSrc* to the planar Y'Cr'Cb' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The destination image *pDst* has 4:2:0 sampling format and the following order of pointers: `Y`-plane, `Cr`-plane, `Cb`-plane (see Table 6-3 for more details).

The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of two possible formats (see Figure 6-14 for more details): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), and `BGR555` (5 bits for blue, green, red).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 2.

BGRToYCbCr411

Converts a BGR image to the YCbCr planar image that has a 4:1:1 sampling format.

Syntax

```
IppStatus ippiBGRToYCbCr411_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr411_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr411` uses 4:1:1 sampling format (see Table 6-3 for more details).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 4 or *roiSize.height* is less than 1.

YCbCr411ToBGR

Converts a YCbCr image that has 4:1:1 sampling format to the RGB color model.

Syntax

```
IppStatus ippiYCbCr411ToBGR_8u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToBGR_8u_P3C4R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    Ipp8u aval);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create fourth channel.

Description

The function `ippiYCbCr411ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the planar Y'Cb'Cr' image *pSrc* to the three- or four-channel image *pDst*.To compute gamma-corrected `R'G'B'` (`B'G'R'`) channel values the above formulas are used. The difference is that `ippiYCbCr411ToBGR` 4:2:0 sampling the input data to be in 4:1:1 sampling format (see Table 6-3 for more details). Fourth channel is created by setting channel values to the constant value *aval*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGR565ToYCbCr411, BGR555ToYCbCr411

Converts a 16-bit per pixel BGR image to the YCbCr image that has 4:1:1 sampling format.

Syntax

```
IppStatus ippiBGR565ToYCbCr411_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGR555ToYCbCr411_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiBGR565ToYCbCr411` and `ippiBGR555ToYCbCr411` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert a three-channel gamma-corrected `B'G'R'` image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that these functions use 4:1:1 sampling format (see Table 6-3 for more details).

The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of two possible formats (see Figure 6-14 for more details): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), and `BGR555` (5 bits for blue, green, red).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 4 or *roiSize.height* is less than 1.

YCbCr411ToBGR565, YCbCr411ToBGR555

Convert a YCbCr image that has 4:1:1 sampling format to the 16-bit per pixel BGR image.

Syntax

```
IppStatus ippiYCbCr411ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr411ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions ippiYCbCr411ToBGR565 and ippiYCbCr411ToBGR555 are declared in the ippcc.h file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the planar Y'Cb'Cr' image *pSrc*, packed in 4:1:1 sampling format (see Table 6-3 for more details), to the gamma-corrected B'G'R' image *pDst* according to the same formulas as the function ippiYCbCrToRGB does.
The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3

channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of two possible formats (see Figure 6-14 for more details): `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red) and `BGR555` (5 bits for blue, green, red).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToXYZ

Converts an RGB image to the XYZ color model.

Syntax

```
IppStatus ippiRGBToXYZ_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`	`32f_AC4R`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGBToXYZ` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the RGB image *pSrc* to the CIE XYZ image *pDst* according to the following basic equations:

```
X =  0.412453*R + 0.35758 *G + 0.180423*B
Y =  0.212671*R + 0.71516 *G + 0.072169*B
Z =  0.019334*R + 0.119193*G + 0.950227*B
```

The equations above are given on the assumption that R,G, and B values are normalized to the range [0..1]. In case of the floating-point data type, the input RGB values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.
The computed XYZ values are saturated if they fall out of range [0..1].
In case of integer function flavors, these values are then scaled to the full range of the destination data type (see Table 2-2 in Chapter 2).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

XYZToRGB

Converts an XYZ image to the RGB color model.

Syntax

```
IppStatus ippiXYZToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiXYZToRGB is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the CIE XYZ image *pSrc* to the RGB image *pDst* according to the following basic equations:

```
R =  3.240479 * X - 1.53715  * Y  - 0.498535 * Z

G = -0.969256 * X + 1.875991 * Y  + 0.041556 * Z

B =  0.055648 * X - 0.204043 * Y  + 1.057311 * Z
```

The equations above are given on the assumption that X,Y, and Z values are in the range [0..1]. In case of the floating-point data type, the input XYZ values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.
The computed RGB values are saturated if they fall out of range [0..1].
In case of integer function flavors, these values are then scaled to the full range of the destination data type (see Table 2-2 in Chapter 2).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToLUV

Converts an RGB image to the LUV color model.

Syntax

```
IppStatus ippiRGBToLUV_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGBToLUV` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the `RGB` image *pSrc* to the CIE LUV image *pDst* in two steps. First, the conversion is done into CIE XYZ format, using equations defined for the `ippiRGBToXYZ` function. After that, conversion to `LUV` image is performed in accordance with the following equations:

$$L = 116. * (Y/Y_n)^{1/3.} - 16.$$

$$U = 13. * L * (u - u_n)$$

$$V = 13. * L * (v - v_n)$$

where

$$u = 4.*X / (X + 15.*Y + 3.*Z)$$

$$v = 9.*Y / (X + 15.*Y + 3.*Z)$$

$$u_n = 4.*x_n / (-2.*x_n + 12.*y_n + 3.)$$

$$v_n = 9.*y_n / (-2.*x_n + 12.*y_n + 3.)$$

Here $x_n = 0.312713$, $y_n = 0.329016$ are the CIE chromaticity coordinates of the D65 white point, and $Y_n = 1.0$ is the luminance of the D65 white point.
The computed values of the `L` component are in the range `[0..100]`, `U` component in the range `[-134..220]`, and `V` component in the range `[-140..122]`.

The equations above are given on the assumption that R,G, and B values are normalized to the range [0..1]. In case of the floating-point data type, the input RGB values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.
In case of 8u data type, the computed L,U, and V values are quantized and converted to fit in the range [0..IPP_MAX_8U] as follows:

```
L =   L * IPP_MAX_8U / 100.

U = (U + 134.) * IPP_MAX_8U / 354.

V = (V + 140.) * IPP_MAX_8U / 262.
```

In case of 16u data type, the computed L,U, and V values are quantized and converted to fit in the range [0..IPP_MAX_16U] as follows:

```
L =   L * IPP_MAX_16U / 100.

U = (U + 134.) * IPP_MAX_16U / 354.

V = (V + 140.) * IPP_MAX_16U / 262.
```

In case of 16s data type, the computed L,U, and V values are quantized and converted to fit in the range [IPP_MIN_16S..IPP_MAX_16S] as follows:

```
L =   L * IPP_MAX_16U / 100. + IPP_MIN_16S

U = (U + 134.) * IPP_MAX_16U / 354. + IPP_MIN_16S

V = (V + 140.) * IPP_MAX_16U / 262. + IPP_MIN_16S
```

For 32f data type, no further conversion is done and L,U, and V components remain in the ranges [0..100], [-134..220], and [-140..122], respectively.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

LUVToRGB

Converts a LUV image to the RGB color model.

Syntax

```
IppStatus ippiLUVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiLUVToRGB` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the CIE LUV image *pSrc* to the `RGB` image *pDst* in two steps. First, the conversion is carried out into CIE XYZ format.

To accomplish it, `LUV` components are transformed back into their original range. This is done for different data types in the following way.

For `8u` data type:

```
L  =   L * 100./ IPP_MAX_8U
U  = (U * 354./ IPP_MAX_8U) - 134.
V  = (V * 262./ IPP_MAX_8U) - 140.
```

For `16u` data type:

```
L  =   L * 100./ IPP_MAX_16U
U  = (U * 354./ IPP_MAX_16U) - 134.
V  = (V * 262./ IPP_MAX_16U) - 140.
```

For `16s` data type:

```
L  = (L - IPP_MIN_16S)* 100./ IPP_MAX_16U
U  = ((U - IPP_MIN_16S)* 354./ IPP_MAX_16U) - 134.
V  = ((V - IPP_MIN_16S) * 262./ IPP_MAX_16U) - 140.
```

After that, conversion to `XYZ` format takes place as follows:

$$Y = Y_n * ((L + 16.) / 116.)^{**}3.$$

$$X = -9.* Y * u / ((u - 4.)* v - u * v)$$

$$Z = (9.* Y - 15*v*Y - v*X) / 3. * v$$

where

$$u = U / (13.* L) + u_n$$
$$v = V / (13.* L) + v_n$$

and

$$u_n = 4.*x_n / (-2.*x_n + 12.*y_n + 3.)$$
$$v_n = 9.*y_n / (-2.*x_n + 12.*y_n + 3.)$$

Here $x_n = 0.312713$, $y_n = 0.329016$ are the CIE chromaticity coordinates of the D65 white point, and $Y_n = 1.0$ is the luminance of the D65 white point.
After this intermediate conversion is done, the obtained `XYZ` image is then converted to the destination `RGB` format using equations defined for the `ippsXYZToRGB` function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGRToLab

Converts a BGR image to the Lab color model.

Syntax

```
IppStatus ippiBGRToLab_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToLab_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToLab` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the BGR image *pSrc* to the CIE Lab image *pDst* in two steps. First, the conversion is done into CIE XYZ format, using equations defined for the function `ippiRGBToXYZ`. After that, conversion to Lab image is performed in accordance with the following equations:

$$L = 116. * (Y/Y_n)^{1/3.} - 16 \quad \text{for } Y/Y_n > 0.008856$$
$$L = 903.3 * (Y/Y_n)^{1/3.} \quad \text{for } Y/Y_n \leq 0.008856$$

$$a = 500. * [f(X/X_n) - f(Y/Y_n)]$$

$$b = 200. * [f(Y/Y_n) - f(Z/Z_n)]$$

where

$$f(t) = t^{1/3.} - 16 \quad \text{for } t > 0.008856$$
$$f(t) = 7.787*t + 16/116 \quad \text{for } t \leq 0.008856$$

Here $Y_n = 1.0$, $X_n = 0.950455$, $Z_n = 1.088753$ for the D65 white point with the CIE chromaticity coordinates $x_n = 0.312713$, $y_n = 0.329016$.

The equations above are given on the assumption that initial B, G, R values are normalized to the range `[0..1]`. The computed values of the `L` component are in the range `[0..100]`, `a` and `b` component values are in the range `[-128..127]`.

These values are quantized and scaled to the 8-bit range of 0 to 255 for `ippiBGRToLab_8u_C3`:

```
L = L * 255./100.
a = (a + 128.)
b = (a + 128.)
```

or to the 16-bit range of 0 to 65535 for `ippiBGRToLab_8u16u_C3R`:

```
L = L * 65535./100.
a = (a + 128.)* 255
b = (a + 128.)* 255
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

LabToBGR

Converts a Lab image to the BGR color model.

Syntax

```
IppStatus ippiLabToBGR_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiLabToBGR_16u8u_C3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiLabToBGR` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the CIE Lab image *pSrc* to the `BGR` image *pDst* in two steps. First, the conversion is carried out into CIE XYZ format.

To accomplish it, `Lab` components are transformed back into their original range. This is done for different data types in the following way.

For `8u` data type:

```
L = L * 100./255.
a = a - 128.
b = b - 128.
```

For `16u` data type:

```
L = L * 100./65535.
a = (a/255. - 128.)
b = (b/255.) - 128.)
```

After that, conversion to `XYZ` format takes place as follows:

$Y = Y_n * P^{3.}$

$X = X_n * (P + a/500.)^{3.}$

$Z = Z_n * (P - b/200.)^{3.}$

where

$P = (L + 16)/116.$

After this intermediate conversion is done, the obtained `XYZ` image is then converted to the destination `BGR` format using equations defined for the `ippiXYZToRGB` function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToYCC

Converts an RGB image to the YCC color model.

Syntax

```
IppStatus ippiRGBToYCC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiRGBToYCC is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected R'B'G' image *pSrc* to the PhotoY'C'C' image *pDst* according to the following basic equations:

```
Y'  =  0.299*R' + 0.587*G' + 0.114*B'

C1' = -0.299*R' - 0.587*G' + 0.886*B' = B'- Y

C2' =  0.701*R' - 0.587*G' - 0.114*B' = R'- Y
```

The equations above are given on the assumption that `R',G'`, and `B'` values are normalized to the range `[0..1]`. In case of the floating-point data type, the input R'G'B' values must already be in the range `[0..1]`. For integer data types, normalization is done by the conversion function internally.

The computed `Y', C1', C2'` values are then quantized and converted to fit in the range `[0..1]` as follows:

```
Y'  = 1. / 1.402 * Y'
C1' = 111.4 / 255. * C1' + 156. / 255.
C2' = 135.64 /255. * C2' + 137. / 255.
```

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see Table 2-2 in Chapter 2).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

YCCToRGB

Converts a YCC image to the RGB color model.

Syntax

```
IppStatus ippiYCCToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYCCToRGB` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the PhotoY'C'C' image *pSrc* to the `R'B'G'` image *pDst*. The function `ippiYCCToRGB` first restores normal luminance and chrominance data as:

```
Y'  = 1.3584 * Y'
C1' = 2.2179 * (C1' - 156./255.)
C2' = 1.8215 * (C2' - 137./255.)
```

The equations above are given on the assumption that source `Y`, `C1`, and `C2` values are normalized to the range `[0..1]`. In case of the floating-point data type, the input YCC values must already be in the range `[0..1]`. For integer data types, normalization is done by the conversion function internally.

After that, YCC data are transformed into RGB format according to the following basic equations:

```
R' = Y' + C2'
G' = Y' - 0.194*C1' - 0.509*C2'
B' = Y' + C1'
```

In case of integer function flavors, the computed `R'B'G'` values are then scaled to the full range of the destination data type (see Table 2-2 in Chapter 2).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToHLS

Converts an RGB image to the HLS color model.

Syntax

```
IppStatus ippiRGBToHLS_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGBToHLS` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the `R'B'G'` image *pSrc* to the HLS image *pDst*. For function flavors operating on the floating point data, source RGB values must be in the range `[0..1]`.

The conversion algorithm from `RGB` to `HLS` can be represented in pseudocode as follows:

```
// Lightness:
M1 = max(R,G,B); M2 = min(R,G,B); L = (M1+M2)/2
// Saturation:
if M1 = M2 then // achromatics case
    S = 0
    H = 0
else // chromatics case
    if L <= 0.5 then
```

```
        S = (M1-M2) / (M1+M2)
    else
        S = (M1-M2) / (2-M1-M2)
// Hue:
Cr = (M1-R) / (M1-M2)
Cg = (M1-G) / (M1-M2)
Cb = (M1-B) / (M1-M2)
if R = M2 then H = Cb - Cg
if G = M2 then H = 2 + Cr - Cb
if B = M2 then H = 4 + Cg - Cr
H = 60*H
if H < 0 then H = H + 360
```

For floating point function flavors, the computed `H, L, S` values are scaled to the range `[0..1]`. In case of integer function flavors, these values are scaled to the full range of the destination data type (see Table 2-2 in Chapter 2).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

HLSToRGB

Converts an HLS image to the RGB color model.

Syntax

```
IppStatus ippiHLSToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiHLSToRGB` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the HLS image *pSrc* to the `R'B'G'` image *pDst*. For function flavors operating on the floating point data, source HLS values must be in the range `[0..1]`.

The conversion algorithm from `HLS` to `RGB` can be represented in pseudocode as follows:

```
if L <= 0.5 then
     M2 = L *(1 + S)
else
     M2 = L + S - L * S
M1 = 2 * L - M2
if S = 0 then
   R = G = B = L
else
     h = H + 120
     if h > 360 then
          h = h - 360
     if h  <  60 then
          R = ( M1 + ( M2 - M1 ) * h / 60)
     else if h < 180 then
          R = M2
     else if h < 240 then
          R = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
     else
          R = M1
     h = H
     if h  <  60 then
          G = ( M1 + ( M2 - M1 ) * h / 60
     else if h < 180 then
          G = M2
     else if h < 240 then
          G = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
     else
          G  = M1
     h = H - 120
     if h < 0 then
          h += 360
```

```
if h  <  60 then
     B = ( M1 + ( M2 - M1 ) * h / 60
else if h < 180 then
     B = M2
else if h < 240 then
     B = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
else
     B = M1
```

For floating point function flavors, the computed `R', G', B'` values are scaled to the range `[0..1]`. In case of integer function flavors, these values are scaled to the full range of the destination data type (see Table 2-2 in Chapter 2).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGRToHLS

Converts a BGR image to the HLS color model.

Syntax

```
IppStatus ippiBGRToHLS_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_AC4P4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_AP4C4R(const Ipp8u* const pSrc[4],
    int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiBGRToHLS_8u_AP4R(const Ipp8u* const pSrc[4], int srcStep,
    Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToHLS` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the `B'G'R'` image *pSrc* to the HLS image *pDst* according to the same formula as the function `ippiRGBToHLS` does.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

HLSToBGR

Converts an HLS image to the RGB color model.

Syntax

```
IppStatus ippiHLSToBGR_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AC4P4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4R(const Ipp8u* const pSrc[4], int srcStep,
    Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
```

```
IppStatus ippiHLSToBGR_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4C4R(const Ipp8u* const pSrc[4],
    int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiHLSToBGR` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the HLS image *pSrc* to the `B'G'R'` image *pDst* according to the same formula as the function `ippiHLSToRGB` does.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

RGBToHSV

Converts an RGB image to the HSV color model.

Syntax

```
IppStatus ippiRGBToHSV_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGBToHSV` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the `R'G'B'` image *pSrc* to the HSV image *pDst*.
The conversion algorithm from `RGB` to `HSV` can be represented in pseudocode as follows:

```
//  Value:
V = max(R,G,B);
//  Saturation:
temp = min(R,G,B);
if V = 0 then // achromatics case
    S = 0//          H = 0
else // chromatics case
    S = (V - temp)/V
//  Hue:
Cr = (V - R) / (V - temp)
Cg = (V - G) / (V - temp)
```

```
Cb = (V - B) / (V - temp)
if R = V then H = Cb - Cg
if G = V then H = 2 + Cr - Cb
if B = V then H = 4 + Cg - Cr
H = 60*H
if H < 0 then H = H + 360
```

The computed `H, S, V` values are scaled to the full range of the destination data type (see Table 2-2 in Chapter 2).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

HSVToRGB

Converts an HSV image to the RGB color model.

Syntax

```
IppStatus ippiHSVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

`8u_C3R`	`16u_C3R`
`8u_AC4R`	`16u_AC4R`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiHSVToRGB is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the R'G'B' image *pSrc* to the HSV image *pDst*.
The conversion algorithm from HSV to RGB can be represented in pseudocode as follows:

```
if S = 0 then
   R = G = B = V
else
    if H = 360 then
        H = 0
    else
        H = H/60
     I = floor(H)
     F = H - I;
     M = V * ( 1 - S);
     N = V * ( 1 - S * F);
     K = V * ( 1 - S * (1 - F));
     if(I == 0)then{ R = V;G = K;B = M;}
     if(I == 1)then{ R = N;G = V;B = M;}
     if(I == 2)then{ R = M;G = V;B = K;}
     if(I == 3)then{ R = M;G = N;B = V;}
     if(I == 4)then{ R = K;G = M;B = V;}
     if(I == 5)then{ R = V;G = M;B = N;}
```

The computed R', G', B' values are scaled to the full range of the destination data type (see Table 2-2 in Chapter 2).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

BGRToYCoCg

Converts a 24-bit BGR image to the YCoCg color model.

Syntax

```
IppStatus ippiBGRToYCoCg_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC [3], int yccStep, IppiSize roiSize);
IppStatus ippiBGRToYCoCg_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC [3], int yccStep, IppiSize roiSize);
```

Parameters

pBGR	Pointer to the source image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the source image.
pYCC	Array of pointers to the destination image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToYCoCg` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 24-bit `BGR` image *pSrc* to the YCoCg image *pDst* according to the following formulas:

```
Y  =  ((R + 2*G + B)+ 2)/4
Co =  ((R - B) + 1))/2
Cg =  (( - R + 2*G - B) + 2)/4
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

SBGRToYCoCg

Converts a 48-bit BGR image to the YCoCg color model.

Syntax

```
IppStatus ippiSBGRToYCoCg_<mod>(const Ipp16s* pBGR, int bgrStep,
    Ipp<dstDatatype>* pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for *mod*:

16s_C3P3R	16s32s_C3P3R
16s_C4P3R	16s32s_C4P3R

Parameters

pBGR	Pointer to the source image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the source image.
pYCC	Array of pointers to the destination image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToYCoCg` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 48-bit `BGR` image *pBGR* to the YCoCg image *pYCC* according to the following formulas:

```
Y  =  ((R + 2*G + B)+ 2)/4
Co =  ((R - B) + 1))/2
Cg =  (( - R + 2*G - B) + 2)/4
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is `NULL`.

YCoCgToBGR

Converts a YCoCg image to the 24-bit BGR image.

Syntax

```
IppStatus ippiYCoCgToBGR_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep,
    Ipp8u* pBGR,int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToBGR_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep,
    Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

Parameters

pYCC	Array of pointers to the source image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the source image.
pBGR	Pointer to the destination image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

The function `ippiYCoCrToBGR` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the `YCoCg` image *pYCC* to the 24-bit BGR image *pBGR* according to the following formulas:

```
R = Y + Co - Cg
G = Y + Cg
B = Y - Co - Cg
```

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

YCoCgToSBGR

Converts a YCoCg image to the 48-bit BGR image.

Syntax

Case 1: Conversion to 3-channel image.

```
ippiYCoCgToSBGR_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s*
    pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToSBGR_32s16s_P3C3R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

Case 2: Conversion to 4-channel image.

```
IppStatus ippiYCoCgToSBGR_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep,
    Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
IppStatus ippiYCoCgToSBGR_32s16s_P3C4R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

Parameters

pYCC	Array of pointers to the source image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the source image.
pBGR	Pointer to the destination image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

The function `ippiYCoCrToBGR` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg image *pYCC* to the 48-bit BGR image *pBGR* according to the following formulas:

```
R = Y + Co - Cg
G = Y + Cg
B = Y - Co - Cg
```

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

BGRToYCoCg_Rev

Converts a 24-bit BGR image to the YCoCg-R color model.

Syntax

```
IppStatus ippiBGRToYCoCg_Rev_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiBGRToYCoCg_Rev_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
```

Parameters

pBGR	Pointer to the source image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the source image.
pYCC	Array of pointers to the destination image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiBGRToYCoCg_rev is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 24-bit BGR image *pSrc* to the YCoCg-R image *pDst* according to the following formulas:

```
Co = R - B
t  = B + (Co >> 1)
Cg = G - t
Y  = t + (Cg >> 1)
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

SBGRToYCoCg_Rev

Converts a 48-bit BGR image to the YCoCg-R color model.

Syntax

```
IppStatus ippiSBGRToYCoCg_Rev_<mod>(const Ipp16s* pBGR, int bgrStep,
    Ipp<dstDatatype>* pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for *mod*:

16s_C3P3R	16s32s_C3P3R
16s_C4P3R	16s32s_C4P3R

Parameters

pBGR	Pointer to the source image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the source image.
pYCC	Array of pointers to the destination image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiBGRToYCoCg_Rev is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 48-bit BGR image *pBGR* to the YCoCg-R image *pYCC* according to the following formulas:

```
Co = R - B
t  = B + (Co >> 1)
Cg = G - t
Y  = t + (Cg >> 1)
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

YCoCgToBGR_Rev

Converts a YCoCg-R image to the 24-bit BGR image.

Syntax

```
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C3R(const Ipp16s* pYCC[3], int
    yccStep, Ipp8u* pBGR,int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C4R(const Ipp16s* pYCC[3], int
    yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

Parameters

pYCC	Array of pointers to the source image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the source image.
pBGR	Pointer to the destination image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

The function ippiYCoCrToBGR_Rev is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg-R image *pYCC* to the 24-bit BGR image *pBGR* according to the following formulas:

```
t = Y - (Cg >> 1)
G = Cg + t
B = t - (Co >> 1)
R = B + Co
```

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

YCoCgToSBGR_Rev

Converts a YCoCg-R image to the 48-bit BGR image.

Syntax

Case 1: Conversion to 3-channel image.

```
ippiYCoCgToSBGR_Rev_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep,
    Ipp16s*  pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C3R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

Case 2: Conversion to 4-channel image.

```
IppStatus ippiYCoCgToSBGR_Rev_16s_P3C4R(const Ipp16s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C4R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

Parameters

pYCC	Array of pointers to the source image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the source image.
pBGR	Pointer to the destination image ROI.

bgrStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

The function `ippiYCoCrToBGR_Rev` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg-R image *pYCC* to the 48-bit BGR image *pBGR* according to the following formulas:

```
t = Y - (Cg >> 1)
G = Cg + t
B = t - (Co >> 1)
R = B + Co
```

The fourth channel is created by setting channel values to the constant value *aval*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

Color - Gray Scale Conversions

RGBToGray

Converts an RGB image to gray scale using fixed transform coefficients.

Syntax

```
IppStatus ippiRGBToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32f_C3C1R
8u_AC4C1R	16u_AC4C1R	16s_AC4C1R	32f_AC4C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

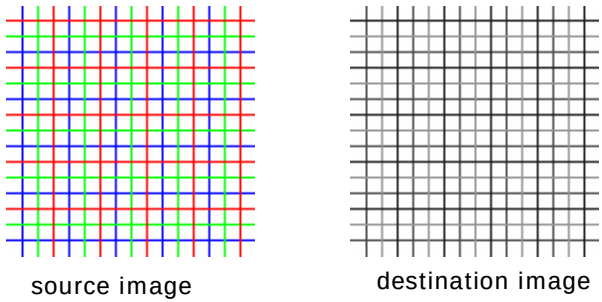
The function `ippiRGBToGray` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an RGB image to gray scale (see Figure 6-18) using the following basic equation to compute luma from nonlinear gamma-corrected red, green, and blue values:

$$Y' = 0.299 * R' + 0.587 * G' + 0.114 * B'$$

Note that the transform coefficients conform to the standard for the NTSC red, green, and blue CRT phosphors.

Figure 6-18 Converting an RGB Image to Gray Scale



Example 6-2 demonstrates how to use the function `ippiRGBToGray_8u_C3C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

Example 6-2 Using the Function ippiRGBToGray

```
Ipp8u  src[12*3] = { 255, 0,  0, 255, 0,  0, 255, 0,  0, 255, 0,  0,
                     0, 255, 0,  0, 255, 0,  0, 255, 0,  0, 255, 0,
                     0,  0, 255, 0,  0, 255, 0,  0, 255, 0,  0, 255};
Ipp8u dst[4*3];
IppiSize srcRoi = { 4, 3 };

ippiRGBToGray_8u_C3C1R ( src, 12, dst, 4, srcRoi );

Result:
255 0   0   255 0 0     255 0   0   255 0   0
0   255 0   0   255 0   0   255 0   0   255 0        src
0   0   255 0   0   255 0   0   255 0   0   255

 76  76  76  76
149 149 149 149    dst
 29  29  29  29
```

ColorToGray

Converts an RGB image to gray scale using custom transform coefficients.

Syntax

```
IppStatus ippiColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    coeffs[3]);
```

Supported values for *mod*:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32f_C3C1R
8u_AC4C1R	16u_AC4C1R	16s_AC4C1R	32f_AC4C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
coeffs	Transform coefficients.

Description

The function `ippiColorToGray` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function uses the following equation to convert an `RGB` image to gray scale:

`Y = coeffs[0] * R + coeffs[1] * G + coeffs[2] * B`

where the *coeffs* array contains user-defined transform coefficients which must be non-negative and satisfy the condition

$coeffs[0] + coeffs[1] + coeffs[2] \leq 1$

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

CFAToRGB

Restores the RGB image from the gray-scale CFA image.

Syntax

```
IppStatus ippiCFAToRGB_8u_C1C3R(const Ipp8u* pSrc, IppiRect srcRoi,
    IppiSize srcSize, int srcStep, Ipp8u* pDst, int dstStep,
    IppiBayerGrid grid, int interpolation);
IppStatus ippiCFAToRGB_16u_C1C3R(const Ipp16u* pSrc, IppiRect srcRoi,
    IppiSize srcSize, int srcStep, Ipp16u* pDst, int dstStep,
    IppiBayerGrid grid, int interpolation);
```

Parameters

pSrc	Pointer to the source image origin.

srcSize	Size of the source image.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

grid	Specifies the configuration of the Bayer grid in the source image. The following values are possible:
	`ippiBayerBGGR`
	`ippiBayerRGGB`
	`ippiBayerGBRG`
	`ippiBayerGRBG`

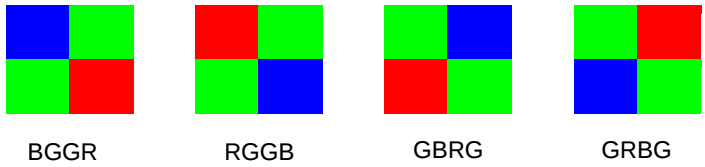
interpolation	Interpolation method, reserved, must be 0.

Description

The function `ippiCFAToRGB` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function transforms the one-channel gray-scale image *pSrc* that is produced by applying the color filter array (CFA) - an array of Bayer filters, to 24-bit three-channel RGB image. The order of the color component in the source image - Bayer grid - is specified by the parameter *grid*. Four possible values of this parameter correspond to the allowed variants of the Bayer grid (see Figure 6-19).

Figure 6-19 Possible Configurations of the Bayer Grids



Each element of the source image contains an intensity value for only one color component, two others are interpolated using neighbor elements. R and B values are interpolated linearly from the nearest neighbors of the same color. When interpolating R and B values on green pixel, the average values of the two nearest neighbors (above and below, or left and right) of the same colors are used. When interpolating R or B values on the blue or red pixel respectively, the average values of the four nearest blue (red) pixels cornering the red (blue) pixel are used. G values are interpolated using an adaptive interpolation [Sak98] from a pair of nearest neighbors (vertical or horizontal) and taking into account the correlation in the red (or blue) component. The pair is chosen depending on the values of the difference between the red (blue) pixels in the vertical and horizontal directions. If the difference is smaller in the vertical direction - a vertical pair of green pixels is used, if it is smaller in the horizontal direction - a horizontal pair is used. If the difference is the same, all four neighbors are used.

This interpolation requires border pixels for the input pixels near the horizontal or vertical edge of the image. The function uses the mirrored border of two edge rows or columns of the input image. In this case the G values is calculated as the average of four nearest green pixels.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *srcSize* has a field that is less than 2, or if the *roiSize* has a field with negative or zero value.
`ippStsBadArgErr`	Indicates an error condition if *grid* has an illegal value.

DemosaicAHD

Restores the RGB image from the gray-scale CFA image using AHD algorithm.

Syntax

```
IppStatus ippiDemosaicAHD_8u_C1C3R(const Ipp8u* pSrc, IppiRect srcRoi,
    IppiSize srcSize, int srcStep, Ipp8u* pDst, int dstStep,
    IppiBayerGrid grid, Ipp8u* pTmp, int tmpStep);
IppStatus ippiDemosaicAHD_16u_C1C3R(const Ipp16u* pSrc, IppiRect srcRoi,
    IppiSize srcSize, int srcStep, Ipp16u* pDst, int dstStep,
    IppiBayerGrid grid, Ipp16u* pTmp, int tmpStep);
```

Parameters

pSrc	Pointer to the source image.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
srcSize	Size of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

grid	Specifies the configuration of the Bayer grid in the source image. The following values are possible (see Figure 6-19):
	`ippiBayerBGGR`
	`ippiBayerRGGB`
	`ippiBayerGBRG`
	`ippiBayerGRBG`

pTmp	Pointer to the temporary image of (*srcRoi.width* + 6, 30) size.
tmpStep	Distance in bytes between starts of consecutive lines in the temporary image.

Description

The function `ippiDemosaicAHD` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function transforms the one-channel gray-scale image *pSrc* that is produced by applying the color filter array (CFA) to 24-bit three-channel RGB image using the adaptive homogeneity-directed demosaicing (AHD) algorithm [Hir05]. The algorithm requires the temporary image *pTmp* of size *srcRoi.width* +6.30.

The type of the Bayer grid (see Figure 6-19) is specified by the parameter *grid*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if the *srcSize* has a field that is less than 5, or if the *roiSize* has a field with negative or zero value.
ippStsBadArgErr	Indicates an error condition if *grid* has an illegal value.

Format Conversion

This section describes Intel IPP functions that perform image color conversion without changing the color space. These functions convert pixel-order images to planar format and vice versa, change the number of channels or planes, alter sampling formats and sequences of samples and planes. Several functions additionally perform filtering - deinterlacing and upsampling.

Intel IPP format conversion functions are specified mainly in the YCbCr color space, but as they do not transform color model they may be used to perform described types of conversion for any other color spaces with decoupled luminance and chrominance coordinates (YUV type).

RGBToRGB565, BGRToBGR565

Convert 24-bit per pixel image to the 16-bit image.

Syntax

```
IppStatus ippiRGBToRGB565_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToBGR565_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiRGBToRGB565` and `ippiBGRToBGR565` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

This function converts the 24-bit per pixel gamma-corrected `R'G'B'` (`B'G'R'`) image *pSrc* to the packed 16-bit image *pDst*: all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). The destination image has format `RGB565` (`BGR565`) - 5 bits for blue, 6 bits for green, 5 bits for red (see Figure 6-14 for more details).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of *srcStep* or *dstStep* is less than or equal to 0.

RGB565ToRGB, BGR565ToBGR

Convert 16-bit per pixel image to the 24-bit image.

Syntax

```
IppStatus ippiRGB565ToRGB_16u8u_C3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGR565ToBGR_16u8u_C3R(const Ipp16u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiRGB565ToRGB` and `ippiBGR565ToBGR` are declared in the `ippcc.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the 16-bit per pixel gamma-corrected `R'G'B'` (`B'G'R'`) image *pSrc* to the 24-bit image *pDst*. The source image has a packed format `RGB565` (`BGR565`), that is all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type) in the following order: 5 bits for blue, 6 bits for green, 5 bits for red (see Figure 6-14 for more details).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of *srcStep* or *dstStep* is less than or equal to 0.

YCbCr422

Converts 4:2:2 YCbCr image.

Syntax

```
IppStatus ippiYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.

dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 two-channel source image *pSrc* to the 4:2:2 three-plane image *pDst* and vice versa (see Table 6-2 and Table 6-3 for more details on 4:2:2 planar and pixel-order formats).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* of the first plane is less than 2, or *roiSize.height* is less than or equal to zero.

YCbCr422ToYCrCb422

Converts 4:2:2 YCbCr image to 4:2:2 YCrCb image.

Syntax

```
IppStatus ippiYCbCr422ToYCrCb422_8u_C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 2.

Description

The function `ippiYCbCr422ToYCrCb422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 `YCbCr` source image *pSrc* to the 4:2:2 `YCrCb` two-channel image *pDst* that has the following sequence of samples: `Y0`, `Cr0`, `Y1`, `Cb0`, `Y2`, `Cr1`, `Y3`, `Cb1`, ... (see Table 6-2). The source image can be either two-channel or three-plane (see Table 6-2 and Table 6-3).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2.

YCbCr422ToCbYCr422

Converts 4:2:2 YCbCr image to 4:2:2 CbYCr image.

Syntax

```
IppStatus ippiYCbCr422ToCbYCr422_8u_C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

The functions `ippiYCbCr422ToCbYCr422_8u_C2R` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the order of samples of the 4:2:2 two-channel image *pSrc* from `Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, ...` to `Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ...` (see Table 6-2) and stores the result in the *pDst*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

YCbCr422ToYCbCr420

Converts YCbCr image from 4:2:2 sampling format to 4:2:0 format.

Syntax

Case 1: Operation on planar data.

```
IppStatus ippiYCbCr422ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

Case 2: Conversion from pixel-order to planar data.

```
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr,int dstCbCrStep, IppiSize
    roiSize);
```

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.

srcStep	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
pDst	Array of pointers to the ROI in each plane for a three-plane destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
pDstY	Pointer to the ROI in the luminance plane for a two-plane destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
roiSize	Size of the ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiYCbCr422ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function convert the 4:2:2 image *pSrc* to the 4:2:0 image. The source image can be two-channel or three-plane , destination image always is planar with two or three planes (see Table 6-2 and Table 6-3). Two-plane image contains luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane *pDstY*, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... in the second plane *pDstCbCr*.

Example 6-3 shows how to use the function `ippiYCbCr422ToYCbCr420_8u_C2P3R`.

Example 6-3 Using the Function ippiYCbCr422ToYCbCr420_8u_C2P3R

```
{
  Ipp8u*   ImageI420[3];
  int      stepI420[3];
  Ipp8u*   ImageYUY2;
  int      stepYUY2;
  IppiSize roiSize = { 1024, 768};
  ImageI420[0] = ippiMalloc_8u_C1( roiSize.width, roiSize.height,
&(stepI420[0]));
  ImageI420[1] = ippiMalloc_8u_C1( roiSize.width, roiSize.height,
&(stepI420[1]));
  ImageI420[2] = ippiMalloc_8u_C1( roiSize.width, roiSize.height,
&(stepI420[2]));
  ImageYUY2    = ippiMalloc_8u_C2( roiSize.width, roiSize.height, &stepYUY2 );
  ippiYCbCr422ToYCbCr420_8u_C2P3R( ImageYUY2, stepYUY2, ImageI420, stepI420,
roiSize);

  ippiFree(ImageI420[0]);
  ippiFree(ImageI420[1]);
  ippiFree(ImageI420[2]);
  ippiFree(ImageYUY2);
}
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

YCbCr422To420_Interlace

Converts interlaced YCbCr image from 4:2:2 sampling format to 4:2:0 format.

Syntax

```
IppStatus ippiYCbCr422To420_Interlace_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
pDst	Array of pointers to the ROI in each plane for destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane fore destination image.
roiSize	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

Description

The function `ippiYCbCr422To420_Interlace` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts interlaced 4:2:2 image *pSrc* to the 4:2:0 image *pDst* (see Table 6-3).

The conversion is performed in accordance with the following formulas:

$$Y1_{dest} = Y1_{src};$$

$$Cb0(Cr0)_{dst} = (3*Cb0(Cr0)_{src} + Cb2(Cr2)_{src} + 2)/4;$$

$$Cb1(Cr1)_{dst} = (Cb1(Cr1)_{src} + 3*Cb3(Cr3)_{src} + 2)/4;$$

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2, or *roiSize.height* is less than 4.
`ippStsDoubleSize`	Indicates a warning if *roiSize.width* is not multiple of 2, or *roiSize.height* is not multiple of 4.

YCbCr422ToYCrCb420

Converts 4:2:2 YCbCr image to 4:2:0 YCrCb image.

Syntax

```
IppStatus ippiYCbCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiYCbCr422ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function convert the 4:2:2 two-channel image *pSrc* that has the following sequence of samples: `Y0`, `Cb0`, `Y1`, `Cr0`, `Y2`, `Cb1`, `Y3`, `Cr1`, ... to the 4:2:0 three-plane image *pDst* with the following order of pointers: `Y`-plane, `Cr`-plane, `Cb`-plane (see Table 6-2 and Table 6-3).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

YCbCr422ToYCbCr411

Converts YCbCr image from 4:2:2 sampling format to 4:1:1 format.

Syntax

Case 1: Operation on planar data.

```
IppStatus ippiYCbCr422ToYCbCr411_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

Case 2: Conversion from pixel-order to planar data.

```
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P2R(const Ipp8u* pSrc, int
    srcStep,  Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
pDst	Array of pointers to the ROI in each plane for a three-plane destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
pDstY	Pointer to the ROI in the luminance plane for a two-plane destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.

roiSize	Size of the ROI in pixels.

Description

The function `ippiYCbCr422ToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function convert the 4:2:2 image *pSrc* to the 4:1:1 image. The source image can be two-channel or three-plane (see Table 6-2 for more details), destination image always is planar with two or three planes (see Table 6-3 for more details). Two-plane image contains luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane *pDstY*, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... in the second plane *pDstCbCr*.

The value of the fields of the *roiSize* have certain limitations:

its width should be multiple of 4 and can not be less than 4 for operation on two-channel images;

its width should be multiple of 4 and can not be less than 4, and its height should be multiple of 2 and can not be less than 2 for three-plane to two-plane image conversion;

both height and width should be multiple of 2 and can not be less than 2 for operation on three-plane images.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if corresponding fileds of the *roiSize* is less than specified above values.

YCrCb422ToYCbCr422

Converts 4:2:2 YCrCb image to 4:2:2 YCbCr image.

Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the ROI in pixels, its width should be multiple of 2.

Description

The function `ippiYCrCb422ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 `YCrCb` two-channel image *pSrc* (see Table 6-2) to the 4:2:2 `YCbCr` three-plane image *pDst* (see Table 6-3).

Return Values

`ppStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2.

YCrCb422ToYCbCr420

Converts 4:2:2 YCrCb image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiYCrCb422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiYCrCb422ToYCbCr420_8u_C2P3R` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This functions converts 4:2:2 `YCrCb` two-channel image *pSrc* (see Table 6-2) to the 4:2:0 `YCbCr` three-plane image *pDst* (see Table 6-3).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

YCrCb422ToYCbCr411

Converts 4:2:2 YCrCb image to 4:1:1 YCbCr image.

Syntax

```
IppStatus ippiYCrCb422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the ROI in pixels, its width should be multiple of 4.

Description

The function `ippiYCrCb422ToYCbCr411_8u_C2P3R` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 `YCrCb` two-channel image *pSrc* (see Table 6-2) to the 4:1:1 `YCbCr` three-plane image *pDst* (see Table 6-3).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 4.

CbYCr422ToYCbCr422

Converts 4:2:2 CbYCr image to 4:2:2 YCbCr image.

Syntax

```
IppStatus ippiCbYCr422ToYCbCr422_8u_C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCbYCr422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels, its width should be multiple of 2.

Description

The function ippiCbYCr422ToYCbCr422 is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 CbYCr two-channel image *pSrc* to the 4:2:2 YCbCr two-channel or three-plane image *pDst* (see Table 6-2 and Table 6-3). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... Two-channel destination image has different sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, Y4, ...

Return Values

ppStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 2.

CbYCr422ToYCbCr420

Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiCbYCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiCbYCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDstY, int dstYStep,Ipp8u* pDstCbCr,int dstCbCrStep, IppiSize
    roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane for a three-plane destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
pDstY	Pointer to the ROI in the luminance plane for a two-plane destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiCbYCr422ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 `CbYCr` two-channel image *pSrc* to the 4:2:0 `YCbCr` two- or three-plane image *pDst* (see Table 6-2 and Table 6-3). The source image has the following sequence of samples: `Cb0`, `Y0`, `Cr0`, `Y1`, `Cb1`, `Y2`, `Cr1`, `Y3`, `Cb2`, ... Three-plane destination image has the following order of pointers: `Y`-plane, `Cb`-plane, `Cr`-plane. Two-plane destination image contains luminance samples `Y0`, `Y1`, `Y2`, ... in the first plane *pDstY*, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... in the second plane *pDstCbCr*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

CbYCr422ToYCbCr420_Interlace

Converts interlaced 4:2:2 CbYCr image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiCbYCr422ToYCbCr420_Interlace_8u_C2P3R(const Ipp8u* pSrc,
    int srcStep, Ipp8u* pDst[3],int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane for destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
roiSize	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

Description

The function `ippiCbYCr422ToYCbCr420_Interlace` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts interlaced 4:2:2 `CbYCr` two-channel image *pSrc* to the 4:2:0 `YCbCr` three-plane image *pDst* (see Table 6-2 and Table 6-3). The source image has the following sequence of samples: `Cb0`, `Y0`, `Cr0`, `Y1`, `Cb1`, `Y2`, `Cr1`, `Y3`, `Cb2`, ... Three-plane destination image has the following order of pointers: `Y`-plane, `Cb`-plane, `Cr`-plane.

The conversion is performed in accordance with the following formulas:

$$Y1_{dest} = Y1_{src};$$

$$Cb0(Cr0)_{dst} = (3*Cb0(Cr0)_{src} + Cb2(Cr2)_{src} + 2)/4;$$

$$Cb1(Cr1)_{dst} = (Cb1(Cr1)_{src} + 3*Cb3(Cr3)_{src} + 2)/4;$$

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2, or *roiSize.height* is less than 4.
`ippStsDoubleSize`	Indicates a warning if *roiSize.width* is not multiple of 2, or *roiSize.height* is not multiple of 4.

CbYCr422ToYCrCb420

Converts 4:2:2 CbYCr image to 4:2:0 YCrCb image.

Syntax

```
IppStatus ippiCbYCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3],int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.

dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiCbYCr422ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 `CbYCr` two-channel image *pSrc* to the 4:2:0 `YCrCb` three-plane image *pDst*. The source image has the following sequence of samples: `Cb0`, `Y0`, `Cr0`, `Y1`, `Cb1`, `Y2`, `Cr1`, `Y3`, `Cb2`, ... The destination image has the following order of pointers: `Y`-plane, `Cr`-plane, `Cb`-plane (see Table 6-2 and Table 6-3).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

CbYCr422ToYCbCr411

Converts 4:2:2 CbYCr image to 4:1:1 YCbCr image.

Syntax

```
IppStatus ippiCbYCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the ROI in pixels, its width should be multiple of 4.

Description

The function `ippiCbYCr422ToYCbCr411_8u_C2P3R` are declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 `CbYCr` two-channel image *pSrc* to the 4:1:1 `YCbCr` three-plane image *pDst*. The source image has the following sequence of samples: `Cb0`, `Y0`, `Cr0`, `Y1`, `Cb1`, `Y2`, `Cr1`, `Y3`, `Cb2`, ... The destination image has the following order of pointers: `Y`-plane, `Cb`-plane, `Cr`-plane (see Table 6-2 and Table 6-3).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 4.

YCbCr420

Converts 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep,
    IppiSize roiSize);
IppStatus ippiYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep,
    const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
pSrcY	Pointer to the ROI in the luminance plane for a two-plane source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.

pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane for a three-plane destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
pDstY	Pointer to the ROI in the luminance plane for a two-plane destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 three-plane (see Table 6-3) source image *pSrc* to the 4:2:0 two-plane image and vice versa. Two-plane image contains luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`,... in the second plane.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

YCbCr420ToYCbCr422

Converts YCbCr image from 4:2:0 sampling format to 4:2:2 format.

Syntax

```
IppStatusippiYCbCr420ToYCbCr422_8u_P3R(constIpp8u*pSrc[3],intsrcStep[3],
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr420ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr420ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize)
```

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
pSrcY	Pointer to the ROI in the luminance plane for a two-plane source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiYCbCr420ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 planar source image *pSrc* to the 4:2:2 image *pDst*. The source image can be two- or three-plane image (see Table 6-3). The first plane of the two-plane source image *pSrcY* contains luminance samples Y0, Y1, Y2, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... The destination image *pDst* can be three-plane or two-channel (see Table 6-2 and Table 6-3).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the *roiSize* is less than 2.

YCbCr420ToYCbCr422_Filter

Convert 4:2:0 image to 4:2:2 image with additional filtering.

Syntax

```
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize, int layout);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
pSrcY	Pointer to the ROI in the luminance plane for a two-plane source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.

pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels.
layout	Slice layout. Possible values:

`IPP_UPPER`	Upper (first) slice
`IPP_CENTER`	Middle slices
`IPP_LOWER`	Lowermost (last) slice
`IPP_LOWER && IPP_UPPER && IPP_CENTER`	Image is not sliced

Description

The function `ippiYCbCr420ToYCbCr422_Filter` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 planar source image *pSrc* to the 4:2:2 image *pDst* and performs additional filtering. The source image can be two- or three-plane image (see Table 6-3). The first plane of the two-plane source image *pSrcY* contains luminance samples `Y0`, `Y1`, `Y2`, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... The destination image *pDst* can be three-plane or two-channel (see Table 6-2 and Table 6-3).

The function flavors `ippiYCbCr420ToYCbCr422_Filter_8u_P3R` and `ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R` additionally perform the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation). In this case *roiSize.width* should be multiple of 2, and *roiSize.height* should be multiple of 8.

The function `ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R` additionally performs deinterlace filtering. Commonly it is used to process images that are divided into slices. In this case slice *layout* should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16.

CAUTION. The image slices should be processed exactly in the following order: first slice, intermediate slices, last slice.

The function may be applied to a not-sliced image as well. In this case *roiSize.width* and *roiSize.height* should be multiple of 2.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* has wrong value.

YCbCr420To422_Interlace

Converts interlaced YCbCr image from 4:2:0 sampling format to 4:2:2 format.

Syntax

```
IppStatus ippiYCbCr420To422_Interlace_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
pDst	Array of pointers to the ROI in each plane for destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for the destination image.
roiSize	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

Description

The function `ippiYCbCr420To422_Interlace` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts interlaced planar 4:2:0 source image *pSrc* to the 4:2:2 image *pDst*. Three-plane image has the following order of pointers: `Y`-plane, `Cb`-plane, `Cr`-plane.

The conversion is performed in accordance with the following formulas:

$$Yn_{dest} = Yn_{src};$$

$$Cb0(Cr0)_{dst} = (5*Cb0(Cr0)_{src} + 3*Cb2(Cr2)_{src} + 4)/8;$$

$$Cb1(Cr1)_{dst} = (7*Cb1(Cr1)_{src} + Cb3(Cr3)_{src} + 4)/8;$$

$$Cb2(Cr2)_{dst} = (Cb0(Cr0)_{src} + 7*Cb2(Cr2)_{src} + 4)/8;$$

$$Cb3(Cr3)_{dst} = (3*Cb1(Cr1)_{src} + 5*Cb3(Cr3)_{src} + 4)/8;$$

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2, or *roiSize.height* is less than 4.
`ippStsDoubleSize`	Indicates a warning if *roiSize.width* is not multiple of 2, or *roiSize.height* is not multiple of 4.

YCbCr420ToCbYCr422

Converts 4:2:0 YCbCr image to 4:2:2 CbYCr image.

Syntax

```
IppStatus ippiYCbCr420ToCbYCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep,
    const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize
    roiSize);
```

Parameters

pSrcY	Pointer to the ROI in the luminance plane of the source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.

pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane of the source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiYCbCr420ToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts planar 4:2:0 two-plane source image to the pixel-order 4:2:2 two-channel image. The first plane of the source image *pSrcY* contains luminance samples `Y0`, `Y1`, `Y2`, .., the second plane *pSrcCbCr* contains interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... The destination image *pDst* has the following sequence of samples: `Cb0`, `Y0`, `Cr0`, `Y1`, `Cb1`, `Y2`, `Cr1`, `Y3`, `Cb2`, ...

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

YCbCr420ToCbYCr422_Interlace

Converts interlaced 4:2:0 YCbCr image to 4:2:2 CbYCr image.

Syntax

```
IppStatus ippiYCbCr420ToCbYCr422_Interlace_8u_P3C2R(const Ipp8u*
    pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for source image.

srcStep	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

Description

The function `ippiYCbCr420ToCbYCr422_Interlace` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts interlaced planar 4:2:0 image to the pixel-order 4:2:2 two-channel image. Three-plane source image has the following order of pointers: `Y`-plane, `Cb`-plane, `Cr`-plane.The destination image *pDst* has the following sequence of samples: `Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ...`

The conversion is performed in accordance with the following formulas:

$$Yn_{dest} = Yn_{src};$$

$$Cb0(Cr0)_{dst} = (5*Cb0(Cr0)_{src} + 3*Cb2(Cr2)_{src} + 4)/8;$$

$$Cb1(Cr1)_{dst} = (7*Cb1(Cr1)_{src} + Cb3(Cr3)_{src} + 4)/8;$$

$$Cb2(Cr2)_{dst} = (Cb0(Cr0)_{src} + 7*Cb2(Cr2)_{src} + 4)/8;$$

$$Cb3(Cr3)_{dst} = (3*Cb1(Cr1)_{src} + 5*Cb3(Cr3)_{src} + 4)/8;$$

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 2, or *roiSize.height* is less than 4.
`ippStsDoubleSize`	Indicates a warning if *roiSize.width* is not multiple of 2, or *roiSize.height* is not multiple of 4.

YCbCr420ToYCrCb420

Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image.

Syntax

```
IppStatus ippiYCbCr420ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
```

Parameters

pSrcY	Pointer to the ROI in the luminance plane of the source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane of the source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiYCbCr420ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 two-plane source image *pSrc* to the 4:2:0 three-plane image *pDst*. The first plane of the source image *pSrcY* contains luminance samples `Y0`, `Y1`, `Y2`, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... The destination image *pDst* has the following order of pointers: `Y`-plane, `Cr`-plane, `Cb`-plane (see Table 6-3).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

YCbCr420ToYCrCb420_Filter

Convert 4:2:0 YCbCr image to 4:2:0 YCrCb image with deinterlace filtering.

Syntax

```
IppStatus ippiYCbCr420ToYCrCb420_Filter_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize, int layout);
```

Parameters

pSrcY	Pointer to the ROI in the luminance plane of the source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane of the source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.
layout	Slice layout. Possible values:

`IPP_UPPER`	Upper (first) slice
`IPP_CENTER`	Middle slices
`IPP_LOWER`	Lowermost (last) slice
`IPP_LOWER && IPP_UPPER && IPP_CENTER`	Image is not sliced

Description

The functions `ippiYCbCr420ToYCrCb420_Filter` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 two-plane source image to the 4:2:0 three-plane image. The first plane of the source image *pSrcY* contains luminance samples Y0, Y1, Y2, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ...
The destination image *pDst* has the following order of pointers: Y-plane, Cr-plane, Cb-plane.
The function additionally performs deinterlace filtering. Commonly it is used to process sliced images. In this case the slice *layout* should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16. The function may be applied to a not-sliced image as well.

CAUTION. The image slices should be processed exactly in the following order: first slice, intermediate slices, last slice.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the *roiSize* is less than 2.

YCbCr420ToYCbCr411

Converts YCbCr image from 4:2:0 sampling format to 4:1:1 format.

Syntax

```
IppStatus ippiYCbCr420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
pSrcY	Pointer to the ROI in the luminance plane for a two-plane source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane for a three-plane destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
pDstY	Pointer to the ROI in the luminance plane for a two-plane destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description

The function `ippiYCbCr420ToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 source image to the 4:1:1 destination image. The source two-plane image is converted to destination three-plane image and vice versa. First plane of the two-plane image contains luminance samples `Y0`, `Y1`, `Y2`, ..., the second plane contains interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... The three-plane image has the following order of pointers: `Y`-plane, `Cb`-plane, `Cr`-plane (see Table 6-3).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 4 or *roiSize.height* is less than 2.

YCrCb420ToYCbCr422

Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image.

Syntax

```
IppStatusippiYCrCb420ToYCbCr422_8u_P3R(constIpp8u*pSrc[3],intsrcStep[3],
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCrCb420ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels, height and width should be multiple of 2.

Description

The function ippiYCrCb420ToYCbCr422 is declared in the ippcc.h file. This function converts 4:2:0 YCrCb three-plane image *pSrc* to the 4:2:2 YCbCr three-plane or two-channel image *pDst* (see Table 6-2 and Table 6-3).

This function operates with ROI (see Regions of Interest in Intel IPP).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.

ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the *roiSize* is less than 2.

YCrCb420ToYCbCr422_Filter

Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image with additional filtering.

Syntax

```
IppStatus ippiYCrCb420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 2, its height should be multiple of 8.

Description

The function ippiYCrCb420ToYCbCr422_Filter is declared in the ippcc.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 YCrCb three-plane image *pSrc* to the 4:2:2 YCbCr three-plane image *pDst* (see Table 6-3).

Additionally this function performs the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if *roiSize.width* is less than 2 or *roiSize.height* is less than 8.

YCrCb420ToCbYCr422

Converts 4:2:0 YCrCb image to 4:2:2 CbYCr image.

Syntax

```
IppStatus ippiYCrCb420ToCbYCr422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiYCrCb420ToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 `YCrCb` three-plane image *pSrc* (see Table 6-3) to the 4:2:2 `CbYCr` two-channel image *pDst* with the following sequence of samples: `Cb0`, `Y0`, `Cr0`, `Y1`, `Cb1`, `Y2`, `Cr1`, `Y3`, `Cb2`, ... (see Table 6-2).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is `NULL`.
ippStsSizeErr	Indicates an error condition if any field of the *roiSize* is less than 2.

YCrCb420ToYCbCr420

Converts 4:2:0 YCrCb image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiYCrCb420ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep,
    IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDstY	Pointer to the ROI in the luminance plane of a destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane of the destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

The function `ippiYCrCb420ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 `YCrCb` three-plane image *pSrc* (see Table 6-3) to the 4:2:0 `YCbCr` two-plane image that contains luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane *pDstY*, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... in the second plane *pDstCbCr*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *roiSize* is less than 2.

YCrCb420ToYCbCr411

Converts 4:2:0 YCrCb image to 4:1:1 YCbCr image.

Syntax

```
IppStatus ippiYCrCb420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDstY	Pointer to the ROI in the luminance plane of a destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane of the destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description

The function `ippiYCrCb420ToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:0 `YCrCb` three-plane image *pSrc* (see Table 6-3) to the 4:1:1 `YCbCr` two-plane image that contains luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane *pDstY*, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... in the second plane *pDstCbCr*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 4 or *roiSize.height* is less than 2.

YCbCr411

Converts 4:1:1 YCbCr image.

Syntax

```
IppStatus ippiYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCrCb, int dstCbCrStep,
    IppiSize roiSize);
IppStatus ippiYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const
    Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst[3],
    int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
pSrcY	Pointer to the ROI in the luminance plane for a two-plane source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane for a three-plane destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
pDstY	Pointer to the ROI in the luminance plane for a two-plane destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.

roiSize	Size of the ROI in pixels, its width should be multiple of 4.

Description

The function `ippiYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:1:1 three-plane (see Table 6-3) source image *pSrc* to the 4:1:1 two-plane image and vice versa. Two-plane image contains luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... in the second plane.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 4.

YCbCr411ToYCbCr422

Converts 4:1:1 YCbCr image to 4:2:2 YCbCr image.

Syntax

```
IppStatus ippiYCbCr411ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr411ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr411ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY,
    int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3],
    int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr411ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY,
    int srcYStep, const Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.

pSrcY	Pointer to the ROI in the luminance plane for a two-plane source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels, its width should be multiple of 4.

Description

The function `ippiYCbCr411ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:1:1 planar source image *pSrc* to the 4:2:2 image *pDst*. The first plane of the two-plane source image *pSrcY* contains luminance samples `Y0`, `Y1`, `Y2`, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... The destination image *pDst* can be either three-plane (see Table 6-3) or two-channel image (see Table 6-2).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 4.

YCbCr411ToYCrCb422

Converts 4:1:1 YCbCr image to 4:2:2 YCrCb image.

Syntax

```
IppStatus ippiYCbCr411ToYCrCb422_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr411ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels, its width should be multiple of 4.

Description

The function `ippiYCbCr411ToYCrCb422` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:1:1 three-plane image *pSrc* to the 4:2:2 two-channel or three-plane image *pDst* with different order of components. The source image has the following order of pointers: `Y`-plane, `Cb`-plane, `Cr`-plane (see Table 6-3). The three-plane destination image has the following order of pointers: `Y`-plane, `Cr`-plane, `Cb`-plane (see Table 6-3), and two-channel destination image has the following sequence of samples: `Y0`, `Cr0`, `Y1`, `Cb0`, `Y2`, `Cr1`, `Y3`, `Cb1`, ... (see Table 6-2).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 4.

YCbCr411ToYCbCr420

Converts 4:1:1 YCbCr image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiYCbCr411ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr411ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
IppStatus ippiYCbCr411ToYCbCr420_8u_P2P3R(const Ipp8u* pSrcY,
    int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3],
    int dstStep[3], IppiSize roiSize);
```

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
pSrcY	Pointer to the ROI in the luminance plane for a two-plane source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane for a three-plane destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
pDstY	Pointer to the ROI in the luminance plane for a two-plane destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.

dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description

The function `ippiYCbCr411ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:1:1 planar source image *pSrc* (see Table 6-3) to the 4:2:0 planar image *pDst*. Both source and destination images can be three- or two-plane. Three-plane images has the following order of pointers: `Y`-plane, `Cb`-plane, `Cr`-plane (see Table 6-3). Two-plane images contain luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ... in the second plane.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 4 or *roiSize.height* is less than 2.

YCbCr411ToYCrCb420

Converts 4:1:1 YCbCr image to 4:2:0 YCrCb image.

Syntax

```
IppStatus ippiYCbCr411ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY,
    int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3],
    int dstStep[3], IppiSize roiSize);
```

Parameters

pSrcY	Pointer to the ROI in the luminance plane of the source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane of the source image.

srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description

The function `ippiYCbCr411ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:1:1 two-plane source image *pSrc* to the 4:2:0 three-plane image *pDst* with different order of components. The first plane of the source image *pSrcY* contains luminance samples `Y0`, `Y1`, `Y2`, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, ...The destination image has the following order of pointers: `Y`-plane, `Cr`-plane, `Cb`-plane (see Table 6-3),

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize.width* is less than 4 or *roiSize.height* is less than 2.

Color Twist

Color twist conversion functions use values of all color channels of a source pixel to compute the resultant destination channel value. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

For example, if (`r,g,b`) is a source pixel, then the destination pixel values (`R,G,B`) are computed as follows:

$$R = t_{11}*r + t_{12}*g + t_{13}*b + t_{14}$$
$$G = t_{21}*r + t_{22}*g + t_{23}*b + t_{24}$$

$$B = t_{31}*r + t_{32}*g + t_{33}*b + t_{34}$$

where

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \end{bmatrix}$$

is the color twist matrix. The color twist matrix used by Intel IPP functions is a matrix of size 3x4, or 4x4 with floating-point elements. The matrix elements are specific for each particular type of color conversion.

ColorTwist

Applies a color twist matrix to an image with floating-point pixel values.

Syntax

Case 1: Not-in-place operation on pixel-order data.

```
IppStatus ippiColorTwist_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f twist[3][4]);
```

Supported values for *mod*:

32f_C3R

32f_AC4R

```
IppStatus ippiColorTwist_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f twist[3][4]);
```

Case 2: Not-in-place operation on planar data.

```
IppStatus ippiColorTwist_32f_P3R(const Ipp32f* const pSrc[3], int srcStep,
    Ipp32f* const pDst[3], int dstStep, IppiSize roiSize,
    const Ipp32f twist[3][4]);
```

Case 3: In-place operation on pixel-order data.

```
IppStatus ippiColorTwist_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for *mod*:

32f_C3IR
32f_AC4IR

Case 4: In-place operation on planar data.

```
IppStatus ippiColorTwist_32f_IP3R(Ipp32f* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
twist	The array containing color-twist matrix elements.

Description

The function `ippiColorTwist` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function applies the color-twist matrix to all three color channels in the source image with floating-point pixel values to obtain the resulting data in the destination image. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.

ippStsNullPtrErr	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

ColorTwist32f

Applies a color twist matrix to an image with integer pixel values.

Syntax

Case 1: Not-in-place operation on pixel-order data.

```
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f twist[3][4]);
```

Supported values for *mod*:

8u_C3R	8s_C3R	16u_C3R	16s_C3R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R

Case 2: Not-in-place operation on planar data.

```
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* const pSrc[3],
    int srcStep, Ipp<datatype>* const pDst[3], int dstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for *mod*:

8u_P3R	8s_P3R	16u_P3R	16s_P3R

Case 3: In-place operation on pixel-order data.

```
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for *mod*:

8u_C3IR	8s_C3IR	16u_C3IR	16s_C3IR
8u_AC4IR	8s_AC4IR	16u_AC4IR	16s_AC4IR

Case 4: In-place operation on planar data.

```
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for *mod*:

8u_IP3R 8s_IP3R 16u_IP3R 16s_IP3R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
twist	The array containing color-twist matrix elements.

Description

The function `ippiColorTwist32f` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function applies the color-twist matrix to all three color channel values in the integer source image to obtain the resulting data in the destination image. For example, the conversion from `RGB` to `YCbCr` format can be done as

```
Y  =0.299*R   + 0.587*G    + 0.114*B

Cb =- 0.16874*R -  0.33126*G + 0.5*B     + 0.5

Cr =0.5*R     -  0.41869*G  - 0.08131*B + 0.5
```

which can be described in terms of the following color twist matrix:

```
 0.29900f   0.58700f   0.11400f  0.000f
-0.16874f  -0.33126f   0.50000f  128.0f
 0.50000f  -0.41869f  -0.08131f  128.0f
```

Color-twist matrices may also be used to perform many other color conversions.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

CompColorKey

Performs color keying of two images.

Syntax

Case 1: Operation on one-channel data.

```
IppStatus ippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int
    src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
    pDst, int dstStep, IppiSize roiSize, Ipp<datatype> colorKey);
```

Supported values for *mod*:

8u_C1R 16u_C1R 16s_C1R

Case 2: Operation on multi-channel data.

```
IppStatus ippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int
    src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
    pDst, int dstStep, IppiSize roiSize, Ipp<datatype> colorKey[3]);
```

Supported values for *mod*:

8u_C3R 16u_C3R 16s_C3R

```
IppStatus ippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int
    src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>*
    pDst, int dstStep, IppiSize roiSize, Ipp<datatype> colorKey[4]);
```

Supported values for *mod*:

8u_C4R 16u_C4R 16s_C4R

Parameters

pSrc1, *pSrc2*	Pointer to the source images ROI.

src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
colorKey	Value of the key color for 1-channel images, array of color values for multi-channel images.

Description

The function `ippiCompColorKey` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function replaces all areas of the source image *pSrc1* containing the specified key color *colorKey* with the corresponding pixels of the background image *pSrc2* and stores the result in the destination image *pDst*.

The Figure 6-20 shows example of how the function `ippiCompColorKey` works.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of the step values is less than or equal to 0.

Figure 6-20 Applying the Function `ippiCompColorKey` to the Sample Images

Src1

Src2

Dst

AlphaCompColorKey

Performs color keying and alpha composition of two images.

Syntax

```
IppStatus  ippiAlphaCompColorKey_8u_AC4R(const Ipp8u* pSrc1, int
    src1Step, Ipp8u alpha1, const Ipp8u* pSrc2, int src2Step, Ipp8u
    alpha2, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u
    colorKey[4], IppiAlphaType alphaType);
```

Parameters

pSrc1, pSrc2	Pointer to the source images ROI.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

roiSize	Size of the source and destination ROI in pixels.
colorKey	Array of color values.
alphaType	The type of composition to perform (without pre-multiplying). See Table 5-3 for more details.

Description

The function `ippiAlphaCompColorKey` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function replaces all areas of the source image *pSrc1* containing the specified key color *colorKey* with the corresponding pixels of the background image *pSrc2* and additionally performs alpha composition (see supported "Types of Image Compositing Operations") in accordance with the parameter *alphaType*. Note the alpha channel in the *pDst* is not changed after color keying.

The parameter *alphaType* should not be set to the values intended for operations with pre-multiplying.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of the step values is less than or equal to 0.
`ippStsAlphaTypeErr`	Indicates an error condition if *alphaType* specifies the unsupported type of composition.

Gamma Correction

GammaFwd

Performs gamma-correction of the source image with RGB data.

Syntax

Case 1: Not-in-place operation on integer pixel-order data.

```
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

Case 2: Not-in-place operation on integer planar data.

```
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* const pSrc[3],
    int srcStep, Ipp<datatype>* const pDst[3], int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_P3R	16u_P3R

Case 3: Not-in-place operation on floating-point pixel-order data.

```
IppStatus ippiGammaFwd_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

32f_C3R
32f_AC4R

Case 4: Not-in-place operation on floating-point planar data.

```
IppStatus ippiGammaFwd_32f_P3R (const Ipp32f* const pSrc[3], int srcStep,
    Ipp32f* const pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin,
    Ipp32f vMax);
```

Case 5: In-place operation on integer pixel-order data.

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3IR     16u_C3IR
8u_AC4IR    16u_AC4IR
```

Case 6: In-place operation on integer planar data.

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_IP3R     16u_IP3R
```

Case 7: In-place operation on floating-point pixel-order data.

```
IppStatus ippiGammaFwd_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

```
32f_C3IR
32f_AC4IR
```

Case 8: In-place operation on floating-point planar data.

```
IppStatus ippiGammaFwd_32f_IP3R (Ipp32f* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

roiSize	Size of the source and destination ROI in pixels.
vMin, *vMax*	Minimum and maximum values of the input floating-point data.

Description

The function `ippiGammaFwd` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs gamma-correction of the source image with RGB data. It uses the following basic equations to convert an `RGB` image to the gamma-corrected `R'G'B'` image:

for `R,G,B < 0.018`

$$R' = 4.5 * R$$
$$G' = 4.5 * G$$
$$B' = 4.5 * B$$

for `R,G,B ≥ 0.018`

$$R' = 1.099 * R^{0.45} - 0.099$$
$$G' = 1.099 * G^{0.45} - 0.099$$
$$B' = 1.099 * B^{0.45} - 0.099$$

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to 1/0.45 = 2.22 in conformity with [ITU709] specification.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsGammaRangeErr`	Indicates an error condition if the input data bounds are incorrect, that is *vMax* is less than or equal to *vMin*.

GammaInv

Converts a gamma-corrected R´G´B´ image back to the original RGB image.

Syntax

Case 1: Not-in-place operation on integer pixel-order data.

```
IppStatus ippiGammaInv_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3R      16u_C3R
8u_AC4R     16u_AC4R
```

Case 2: Not-in-place operation on integer planar data.

```
IppStatus ippiGammaInv_<mod>(const Ipp<datatype>* const pSrc[3],
    int srcStep, Ipp<datatype>* const pDst[3], int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R
```

Case 3: Not-in-place operation on floating-point pixel-order data.

```
IppStatus ippiGammaInv_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

```
32f_C3R
32f_AC4R
```

Case 4: Not-in-place operation on floating-point planar data.

```
IppStatus ippiGammaInv_32f_P3R (const Ipp32f* const pSrc[3], int srcStep,
    Ipp32f* const pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin,
    Ipp32f vMax);
```

Case 5: In-place operation on integer pixel-order data.

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3IR     16u_C3IR
8u_AC4IR    16u_AC4IR
```

Case 6: In-place operation on integer planar data.

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_IP3R     16u_IP3R
```

Case 7: In-place operation on floating-point pixel-order data.

```
IppStatus ippiGammaInv_<mod>(Ipp32f* pSrcDst, int srcDstStep,
     IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

```
32f_C3IR
32f_AC4IR
```

Case 8: In-place operation on floating-point planar data.

```
IppStatus ippiGammaInv_32f_IP3R (Ipp32f* const pSrcDst[3], int srcDstStep,
    IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
vMin, *vMax*	Minimum and maximum values of the input floating-point data.

Description

The function `ippiGammaInv`is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a gamma-corrected `R'G'B'` image back to the original `RGB` image. It uses the following equations:

for `R´,G´,B´ < 0.0812`

```
R = R´/4.5
G = G´/4.5
B = B´/4.5
```

for `R´,G´,B´ ≥ 0.0812`

$$R = \left(\frac{R' + 0.099}{1.099}\right)^{2.2}$$

$$G = \left(\frac{G' + 0.099}{1.099}\right)^{2.2}$$

$$B = \left(\frac{B' + 0.099}{1.099}\right)^{2.2}$$

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to 1/0.45 = 2.22 in conformity with [ITU709] specification.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsGammaRangeErr`	Indicates an error condition if the input data bounds are incorrect, that is *vMax* is less than or equal to *vMin*.

Intensity Transformation

The functions described in this section perfrom different types of intensity transformation including reduction of the intensity levels in each channel of the image, intensity transformation using lookup tables, and mapping high dynamic range image (HDRI) into low dynamic range image (LDRI).

ReduceBits

Reduces the bit resolution of an image.

Syntax

Case 1: Operation on data of the same source and destination bit depths.

```
IppStatus ippiReduceBits_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    int noise, IppiDitherType dtype, int levels);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Case 2: Operation on data of different source and destination bit depths.

```
IppStatus ippiReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    int noise, IppiDitherType dtype, int levels);
```

Supported values for *mod*:

16u8u_C1R	16s8u_C1R	32f8u_C1R	32f16u_C1R	32f16s_C1R
16u8u_C3R	16s8u_C3R	32f8u_C3R	32f16u_C3R	32f16s_C3R
16u8u_C4R	16s8u_C4R	32f8u_C4R	32f16u_C4R	32f16s_C4R
16u8u_AC4R	16s8u_AC4R	32f8u_AC4R	32f16u_AC4R	32f16s_AC4R

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
noise	The number specifying the amount of noise added. This parameter is set as a percentage of range `[0..100]`.
dtype	The type of dithering to be used. The following types are supported: `ippDitherNone` No dithering is done `ippDitherStucki` The Stucki's error diffusion dithering algorithm is used `ippDitherFS` The Floid-Steinberg error diffusion dithering algorithm is used `ippDitherJJN` The Jarvice-Judice-Ninke error diffusion dithering algorithm is used `ippDitherBayer` The Bayer's threshold dithering algorithm is used
levels	The number of output levels for halftoning (dithering); can be varied in the range `[2..(1<<depth)]`, where *depth* is the bit depth of the destination image.

Description

The function `ippiReduceBits` is declared in the `ippcc.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function reduces the number of intensity levels in each channel of the source image *pSrc* and places the results in respective channels of the destination image *pDst*. Note that for floating point source data type, RGB values must be in the range `[0..1]`.
The *levels* parameter sets the resultant number of intensity levels in each channel of the destination image.
If the *noise* value is greater than 0, some random noise is added to the threshold level used in computations. The amplitude of the noise signal is specified by the *noise* parameter set as a percentage of the destination image luminance range. For the 4x4 ordered dithering mode, the threshold value is determined by the dither matrix used, whereas for the error diffusion dithering mode the input threshold is set as half of the

range value, where
range = ((1<<*depth*) - 1)/(*levels* – 1)
and *depth* is the bit depth of the source image.

For floating-point data type, *range* = 1.0/(*levels* – 1).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsNoiseValErr	Indicates an error condition if *noise* has an illegal value.
ippStsDitherLevelsErr	Indicates an error condition if *levels* value is out of admissible range.

LUT

Maps an image by applying intensity transformation.

Syntax

Case 1: Not-in-place operation on one-channel integer data.

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for *mod*:

8u_C1R 16u_C1R 16s_C1R

Case 2: Not-in-place operation on multi-channel integer data.

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for *mod*:

```
8u_C3R      16u_C3R     16s_C3R
8u_AC4R     16u_AC4R    16s_AC4R
```

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for *mod*:

```
8u_C4R      16u_C4R     16s_C4R
```

Case 3: Not-in-place operation on one-channel floating-point data.

```
IppStatus ippiLUT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues,
    const Ipp32f* pLevels, int nLevels);
```

Case 4: Not-in-place operation on multi-channel floating-point data.

```
IppStatus ippiLUT_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues[3],
    const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for *mod*:

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiLUT_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues[4],
    const Ipp32f* pLevels[4], int nLevels[4]);
```

Case 5: In-place operation on one-channel integer data.

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);
```

Supported values for *mod*:

```
8u_C1IR     16u_C1IR    16s_C1IR
```

Case 6: In-place operation on multi-channel integer data.

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
```

Supported values for *mod*:

```
8u_C3IR     16u_C3IR    16s_C3IR
```

8u_AC4IR 16u_AC4IR 16s_AC4IR

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int
    nLevels[4]);
```

Supported values for *mod*:

8u_C4IR 16u_C4IR 16s_C4IR

Case 7: In-place operation on one-channel floating-point data.

```
IppStatus ippiLUT_32f_C1R(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
    const Ipp32f* pValues, const Ipp32f* pLevels, int nLevels);
```

Case 8: In-place operation on multi-channel floating-point data.

```
IppStatus ippiLUT_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
    const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for *mod*:

32f_C3IR

32f_AC4IR

```
IppStatus ippiLUT_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
    const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination mage ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source ROI in pixels.
pValues	Pointer to the array of intensity values. In case of multi-channel data, *pValues* is an array of pointers to the intensity values array for each channel.

pLevels	Pointer to the array of level values. In case of multi-channel data, *pLevels* is an array of pointers to the level values array for each channel.
nLevels	Number of levels, separate for each channel.

Description

The function ippiLUT is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs intensity transformation of the source image *pSrc* using the lookup table (LUT) specified by the arrays *pLevels* and *pValues*. Every source pixel *pSrc(x,y)* from the range [*pLevels*[*k*], *pLevels*[*k*+1]) is mapped to the destination pixel *pDst(x,y)* whose value is equal to the intensity *pValues*[*k*].

Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Number of level and intensity values is less than *nLevels* by one. Pixels in the *pSrc* image that are not in the range [*pLevels*[0], *pLevels*[*nLevels*-1]) are copied to the *pDst* image without any transformation.

Figure 6-21 shows particular curves that are used in all ippiLUT function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsMemAllocErr	Indicates an error when there is not enough memory for inner buffers.
ippStsLUTNofLevelsErr	Indicates an error when the *nLevels* is less than 2.

LUT_Linear

Maps an image by applying intensity transformation with linear interpolation.

Syntax

Case 1: Not-in-place operation on one-channel integer data.

```
IppStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R

Case 2: Not-in-place operation on multi-channel integer data.

```
IppStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for *mod* :

8u_C4R	16u_C4R	16s_C4R

Case 3: Not-in-place operation on one-channel floating-point data.

```
IppStatus ippiLUT_Linear_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues,
    const Ipp32f* pLevels, int nLevels);
```

Case 4: Not-in-place operation on multi-channel floating-point data.

```
IppStatus ippiLUT_Linear_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for *mod* :

32f_C3R

32f_AC4R

```
IppStatus ippiLUT_Linear_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

Case 5: In-place operation on one-channel integer data.

```
IppStatus ippiLUT_Linear_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);
```

Supported values for *mod* :

8u_C1IR 16u_C1IR 16s_C1IR

Case 6: In-place operation on multi-channel integer data.

```
IppStatus ippiLUT_Linear_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

8u_C3IR 16u_C1IR 16s_C1IR

8u_AC4IR 16u_C1IR 16s_C1IR

```
IppStatus ippiLUT_Linear_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4],
    int nLevels[4]);
```

Supported values for *mod* :

8u_C4IR 16u_C4IR 16s_C4IR

Case 7: In-place operation on one-channel floating-point data.

```
IppStatus ippiLUT_Linear_32f_C1R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int
    nLevels);
```

Case 8: In-place operation on multi-channel floating-point data.

```
IppStatus ippiLUT_Linear_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

32f_C3IR

32f_AC4IR

```
IppStatus ippiLUT_Linear_32f_C4R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4],
    int nLevels[4]);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination mage ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source ROI in pixels.
pValues	Pointer to the array of intensity values. In case of multi-channel data, *pValues* is an array of pointers to the intensity values array for each channel.
pLevels	Pointer to the array of level values. In case of multi-channel data, *pLevels* is an array of pointers to the level values array for each channel.
nLevels	Number of levels, separate for each channel.

Description

The function `ippiLUT_Linear` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs intensity transformation of the source image *pSrc* using the lookup table (LUT) with linear interpolation between two neighbor levels. LUT is specified by the arrays *pLevels* and *pValues*. Every source pixel *pSrc*(x,y) from the range [*pLevels*[*k*], *pLevels*[*k*+1]) is mapped to the destination pixel *pDst(x,y)* whose value is computed according to the following relation:

$$pDst(x, y) = pValues[k] + (pSrc(x, y) - pLevels[k]) \cdot \frac{pValues[k+1] - pValues[k]}{pLevels[k+1] - pLevels[k]}$$

Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Pixels in *pSrc* image that are not in the range [*pLevels*[0], *pLevels*[*nLevels*-1]) are copied to *pDst* image without any transformation.

Figure 6-21 shows particular curves that are used in all ippiLUT function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

Example 6-4 shows how to use the function ippiLUT_Linear_32f_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsMemAllocErr	Indicates an error when there is not enough memory for inner buffers.
ippStsLUTNofLevelsErr	Indicates an error when the *nLevels* is less than 2.

Example 6-4 Using the Function ippiLUT_Linear

```
void func_LUTLinear()
{
    Ipp32f pSrc[8*8];
    int srcStep = 8*sizeof(Ipp32f);
    IppiSize roiSize = {8, 8};

    Ipp32f pDst[8*8];
    int dstStep = 8*sizeof(Ipp32f);
    Ipp32f pLevels[5] = {0.0, 0.128, 0.256, 0.512, 1.0};
    Ipp32f pValues[5] = {0.2, 0.4, 0.6, 0.8, 1.0};
    int nLevels = 5;

    ippiImageJaehne_32f_C1R( pSrc, srcStep, roiSize);

    ippiLUT_Linear_32f_C1R( pSrc, srcStep, pDst, dstStep, roiSize,
pValues,pLevels, nLevels);
}
Result:
pSrc
0.00  0.26  0.65  0.82  0.82  0.65  0.26  0.00
0.26  0.82  1.00  0.98  0.98  1.00  0.82  0.26
0.65  1.00  0.89  0.74  0.74  0.89  1.00  0.65
```

Example 6-4 Using the Function `ippiLUT_Linear` (continued)

```
0.82  0.98  0.74  0.55  0.55  0.74  0.98  0.82
0.82  0.98  0.74  0.55  0.55  0.74  0.98  0.82
0.65  1.00  0.89  0.74  0.74  0.89  1.00  0.65
0.26  0.82  1.00  0.98  0.98  1.00  0.82  0.26
0.00  0.26  0.65  0.82  0.82  0.65  0.26  0.00


pDst
0.20  0.61  0.85  0.93  0.93  0.85  0.61  0.20
0.61  0.93  1.00  0.99  0.99  1.00  0.93  0.61
0.85  1.00  0.95  0.89  0.89  0.95  1.00  0.85
0.93  0.99  0.89  0.82  0.82  0.89  0.99  0.93
0.93  0.99  0.89  0.82  0.82  0.89  0.99  0.93
0.85  1.00  0.95  0.89  0.89  0.95  1.00  0.85
0.61  0.93  1.00  0.99  0.99  1.00  0.93  0.61
0.20  0.61  0.85  0.93  0.93  0.85  0.61  0.20
```

LUT_Cubic

Maps an image by applying intensity transformation with cubic interpolation.

Syntax

Case 1: Not-in-place operation on one-channel integer data.

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for *mod* :

8u_C1R	16u_C4IR	16s_C4IR

Case 2: Not-in-place operation on multi-channel integer data.

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for *mod* :

8u_C4R 16u_C4R 16s_C4R

Case 3: Not-in-place operation on one-channel floating-point data.

```
IppStatus ippiLUT_Cubic_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues,
    const Ipp32f* pLevels, int nLevels);
```

Case 4: Not-in-place operation on multi-channel floating-point data.

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues[3],
    const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for *mod* :

32f_C3R

32f_AC4R

```
IppStatus ippiLUT_Cubic_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues[4],
    const Ipp32f* pLevels[4], int nLevels[4]);
```

Case 5: In-place operation on one-channel integer data.

```
IppStatus ippiLUT_Cubic_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);
```

Supported values for *mod* :

8u_C1IR 16u_C1IR 16s_C1IR

Case 6: In-place operation on multi-channel integer data.

```
IppStatus ippiLUT_Cubic_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

8u_C3IR 16u_C3IR 16s_C3IR

8u_AC4IR 16u_AC4IR 16s_AC4IR

```
IppStatus ippiLUT_Cubic_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4],
    int nLevels[4]);
```

Supported values for *mod* :

8u_C4IR 16u_C4IR 16s_C4IR

Case 7: In-place operation on one-channel floating-point data.

```
IppStatus ippiLUT_Cubic_32f_C1R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int
    nLevels);
```

Case 8: In-place operation on multi-channel floating-point data.

```
IppStatus ippiLUT_Cubic_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

32f_C3IR

32f_AC4IR

```
IppStatus ippiLUT_Cubic_32f_C4R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4],
    int nLevels[4]);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination mage ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source ROI in pixels.
pValues	Pointer to the array of intensity values. In case of multi-channel data, *pValues* is an array of pointers to the intensity values array for each channel.

pLevels	Pointer to the array of level values. In case of multi-channel data, *pLevels* is an array of pointers to the level values array for each channel.
nLevels	Number of levels, separate for each channel.

Description

The function ippiLUT_Cubic is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs intensity transformation of the source image *pSrc* using the lookup table (LUT) with cubic interpolation between neighbor levels. LUT is specified by the arrays *pLevels* and *pValues*.

Every source pixel *pSrc*(x,y) from the range [*pLevels*[*k*], *pLevels*[*k*+1]) is mapped to the destination pixel *pDst*(x,y) whose value is computed as

$$pDst(x,y) = A*pSrc(x,y)^3 + B*pSrc(x,y)^2 + C*pSrc(x,y) + D$$

The function operates on the assumption that the cubic polynomial curve passes through the following four points:

(*pLevels*[*k*-1], *pValues*[*k*-1]),
(*pLevels*[*k*], *pValues*[*k*]),
(*pLevels*[*k*+1], *pValues*[*k*+1]),
(*pLevels*[*k*+2], *pValues*[*k*+2])

Based on that, coefficients A, B, C, D are computed by solving the following set of linear equations:

$$A*pLevels[k-1]^3 + B*pLevels[k-1]^2 + C*pLevels[k-1] + D = pValues[k-1]$$
$$A*pLevels[k]^3 + B*pLevels[k]^2 + C*pLevels[k] + D = pValues[k]$$
$$A*pLevels[k+1]^3 + B*pLevels[k+1]^2 + C*pLevels[k+1] + D = pValues[k+1]$$
$$A*pLevels[k+2]^3 + B*pLevels[k+2]^2 + C*pLevels[k+2] + D = pValues[k+2]$$

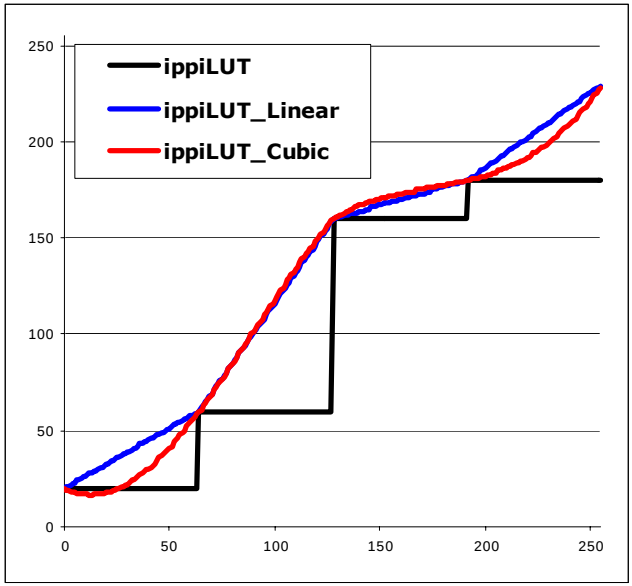
Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Pixels in *pSrc* image that are not in the range [*pLevels*[0], *pLevels*[*nLevels*-1]) are copied to *pDst* image without any transformation.

Figure 6-21 shows particular curves that are used in all ippiLUT function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsMemAllocErr	Indicates an error when there is not enough memory for inner buffers.
ippStsLUTNofLevelsErr	Indicates an error when the *nLevels* is less than 2.

Figure 6-21 Example Mapping Curves Used by the ippiLUT Function Flavors



LUTPalette, LUTPaletteSwap

Maps an image by applying intensity transformation in accordance with a palette table.

Syntax

Case 1: Operations on one-channel data

```
IppStatus ippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, const
    Ipp<dstDatatype>* pTable, int nBitSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R
8u32u_C1R	16u8u_C1R
	16u32u_C1R

```
IppStatus ippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, const Ipp8u*
    pTable, int nBitSize);
```

Supported values for *mod*:

8u24u_C1R	16u24u_C1R

Case 2: Operations on multi-channel data

```
IppStatus ippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const
    Ipp<datatype>* const pTable[3], int nBitSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

```
IppStatus ippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const
    Ipp<datatype>* const pTable[4], int nBitSize);
```

Supported values for *mod*:

8u_C4R	16u_C4R

```
IppStatus ippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    int alphaValue, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype>* const pTable[3], int nBitSize);
```

Supported values for *mod*:

8u_C3A0C4R 16u_C3A0C4R

```
IppStatus ippiLUTPaletteSwap_8u_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, int alphaValue, Ipp<datatype>* pDst, int dstStep, IppiSize
    roiSize, const Ipp<datatype>* const pTable[3], int nBitSize);
```

Supported values for *mod*:

8u_C3A0C4R 16u_C3A0C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source ROI in pixels.
pTable	Pointer to the palette table, or array of pointers to the palette tables for each source channel.
alphaValue	Constant value for the alpha channel.
nBitSize	Number of significant bits in the source image.

Description

The function `ippiLUTPalette` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs intensity transformation of the source image *pSrc* using the palette lookup table *pTable*. This table is a vector with $2^{nBitSize}$ elements that contain intensity values specified by the user. The function uses *nBitSize* lower bits of intensity value of each source pixel as an index in the *pTable* and assigns the correspondent intensity value from the table to the respective pixel in the destination image *pDst*. The number of significant bits *nBitSize* should be in the range [1, 8] for functions that operate on `8u` source images, and [1, 16] for functions that operate on `16u` source images.

Some function flavors that operate on the 3-channel source image additionally create a 4-th channel - alpha channel - in the destination image and place it at first position. The channel values of the alpha channel can be set to the arbitrary constant value *alhaValue*. If this value is less than 0 or greater than the upper boundary of the data range, the channel values are not set.

The function flavor **ippiLUTPaletteSwap** reverses the order of channels in the destination image.

Example 6-5 shows how to use the function ippiLUTPalette_8u32u_C1R.

Example 6-5 Using the Function ippiLUTPalette

```
void func_LUTPalette()
{
    Ipp8u pSrc[8*8];
    int srcStep = 8*sizeof(Ipp8u);
    IppiSize roiSize = {8, 8};

    Ipp32u pDst[8*8];
    int dstStep = 8*sizeof(Ipp32u);
    int nBitSize = 3;
    Ipp32u pTable[8] ={1, 2, 3, 4, 5, 6, 7, 8} ;

    ippiImageJaehne_8u_C1R( pSrc, srcStep, roiSize);

    ippiLUTPalette_8u32u_C1R( pSrc, srcStep, pDst, dstStep, roiSize,
pTable, nBitSize);
}
Result:
            pSrc

  0   67  165  209  209  165   67    0
 67  209  255  250  250  255  209   67
165  255  226  188  188  226  255  165
209  250  188  140  140  188  250  209
209  250  188  140  140  188  250  209
165  255  226  188  188  226  255  165
 67  209  255  250  250  255  209   67
  0   67  165  209  209  165   67    0

            pDst

  1    4    6    2    2    6    4    1
  4    2    8    3    3    8    2    4
  6    8    3    5    5    3    8    6
  2    3    5    5    5    5    3    2
  2    3    5    5    5    5    3    2
  6    8    3    5    5    3    8    6
  4    2    8    3    3    8    2    4
  1    4    6    2    2    6    4    1
```

Example 6-5 Using the Function `ippiLUTPalette` (continued)

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsOutOfRangeErr`	Indicates an error if *nBitSize* is out of the range.

Tone MapLinear, ToneMapMean

Maps an HDRI image to the LDRI image.

Syntax

```
IppStatus ippiToneMapLinear_32f8u_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiToneMapMean_32f8u_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the ROI in the HDRI source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the LDRI destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiToneMapLinear` and `ippiToneMapMean` are declared in the `ippcc.h` file. They both operates with ROI (see Regions of Interest in Intel IPP).

These functions converts the source high dynamic range image (HDRI) *pSrc* into low dynamic range image (LDRI) *pDst*. Pixel values of the source image should not be negative.
The function ippiToneMapLinear implements the Linear Scale-Factor method converted each source pixel *pSrc*[*i*] in accordance with the formula:

$$pSrc[i] = pSrc[i]/Lmax, \qquad Lmax = max\{pSrc[i]\}.$$

The function ippiToneMapMean implements the Mean Value method converted each source pixel *pSrc*[*i*] in accordance with the formula:

$$pSrc[i] = 0.5*pSrc[i]/Lave, \quad Lave = average\{pSrc[i]\}.$$

If the value of *Lmax* or *Lave* is less than 0, then the function does not perform the operation and returns the warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsNoOperation	Indicates a warning if the values of *Lmax* or *Lave* are less than 0.

Threshold and Compare Operations

7

This chapter describes the Intel® IPP image processing functions that operate on a pixel-by-pixel basis: threshold and compare functions.
Table 7-1 lists all functions of this group:

Table 7-1 Image Threshold and Compare Functions

Function Base Name	Operation
Thresholding	
Threshold	Performs thresholding of pixel values in an image, using the specified compare operation
Threshold_GT	Thresholds pixel values of an image, using the comparison for “greater than”
Threshold_LT	Thresholds pixel values of an image, using the comparison for “less than”
Threshold_Val	Thresholds pixel values of an image. Pixels that satisfy the compare condition are set to a specified value
Threshold_GTVal	Thresholds pixel values of an image. Pixels that are greater than threshold are set to a specified value
Threshold_LTVal	Thresholds pixel values of an image. Pixels that are less than threshold are set to a specified value
Threshold_LTValGTVal	Performs double thresholding of pixel values in an image
ComputeThreshold_Otsu	Computes the value of the Otsu threshold
Comparison	
Compare	Compares pixel values of two images using a specified compare operation
CompareC	Compares pixel values of a source image to a given value using a specified compare operation

Table 7-1 Image Threshold and Compare Functions (continued)

Function Base Name	Operation
CompareEqualEps	Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance *eps*
CompareEqualEpsC	Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance *eps*

Thresholding

The threshold functions change pixel values depending on whether they are less or greater than the specified *threshold*.

The type of comparison operation used to threshold pixel values is specified by the *ippCmpOp* parameter; this operation can be either “greater than” or “less than” (see Structures and Enumerators in chapter 2 for more information). For some thresholding functions the type of comparison operation is fixed.

If an input pixel value satisfies the compare condition, the corresponding output pixel is set to the fixed value that is specific for a given threshold function flavor. Otherwise, it is either not changed, or set to another fixed value, which is defined in a particular function description.

For images with multi-channel data, the compare conditions should be set separately for each channel.

Threshold

Performs thresholding of pixel values in an image buffer.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R     16s_C1R     32f_C1R
```

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R     32f_C3R
8u_AC4R     16u_AC4R    16s_AC4R    32f_AC4R
```

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

```
8u_C1IR     16u_C1IR    16s_C1IR    32f_C1IR
```

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp<datatype> threshold[3],
    IppCmpOp ippCmpOp);
```

Supported values for *mod* :

```
8u_C3IR     16u_C3IR    16s_C3IR    32f_C3IR
8u_AC4IR    16u_AC4IR   16s_AC4IR   32f_AC4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).

roiSize	Size of the source and destination ROI in pixels.
threshold	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.
ippCmpOp	The operation specified for comparing pixel values and the *threshold*. Comparison for either “less than” or “greater than” can be used.

Description

The function `ippiThreshold` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value using the *ippCmpOp* comparison operation.

If the result of the compare is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

Example 7-1 shows how yo use the function `ippiThreshold_8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep*, or *srcDstStep* has a zero or negative value
`ippStsNotSupportedModeErr`	Indicates an error if the comparison mode is not supported.

Example 7-1 Using the function `ippiThreshold`

```
void func_threshold()
{
    IppiSize ROI = {5,4};
    Ipp8u src[9*4] = {1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1};
    Ipp8u dst[9*4];
```

Example 7-1 Using the function `ippiThreshold` (continued)

```
    Ipp8u threshold = 6;
    ippiThreshold_8u_C1R(src, 9, dst, 9, ROI, threshold, ippCmpGreater);;

}
Result:
      dst
1  2  4  6  6  8  4  2  1
1  2  4  6  6  8  4  2  1
1  2  4  6  6  8  4  2  1
1  2  4  6  6  8  4  2  1
```

Threshold_GT

Performs thresholding of pixel values in an image , using the comparison for "greater than".

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold);
```

Supported values for *mod* :

8u_C1R 16u_C1R 16s_C1R 32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[3]);
```

Supported values for *mod* :

8u_C3R 16u_C3R 16s_C3R 32f_C3R
8u_AC4R 16u_AC4R 16s_AC4R 32f_AC4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for *mod* :

```
8u_C1IR     16u_C1IR    16s_C1IR    32f_C1IR
```

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for *mod* :

```
8u_C3IR     16u_C3IR    16s_C3IR    32f_C3IR
8u_AC4IR    16u_AC4IR   16s_AC4IR   32f_AC4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
threshold	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

Description

The function `ippiThreshold_GT` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs thresholding of pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “greater than”.

If the result of the compare is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep*, *dstStep*, or *srcDstStep* has a zero or negative value.

Threshold_LT

Performs thresholding of pixel values in an image buffer, using the comparison for "less than".

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R     16s_C1R     32f_C1R
```

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R     32f_C3R
8u_AC4R     16u_AC4R    16s_AC4R    32f_AC4R
```

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for *mod* :

```
8u_C1IR     16u_C1IR    16s_C1IR    32f_C1IR
```

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for *mod* :

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
threshold	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

Description

The function `ippiThreshold_LT` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs thresholding of pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “less than”. If the result of the compare is true, the corresponding output pixel is set to the *threshold* value.
Otherwise, it is set to the source pixel value.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep*, *dstStep*, or *srcDstStep* has a zero or negative value.

Threshold_Val

Performs thresholding of pixel values in an image buffer. Pixels that satisfy the compare condition are set to a specified value.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold, Ipp<datatype> value, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R     16s_C1R     32f_C1R
```

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[3], const Ipp<datatype> value[3], IppCmpOp ippCmpOp);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R     32f_C3R
8u_AC4R     16u_AC4R    16s_AC4R    32f_AC4R
```

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value,
    IppCmpOp ippCmpOp);
```

Supported values for *mod* :

```
8u_C1IR     16u_C1IR    16s_C1IR    32f_C1IR
```

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype>
    value[3], IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
threshold	The threshold value to use for each pixel.In case of multi-channel data, an array of 3 threshold values (one for each color channel) is used.
value	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of 3 output values (one for each color channel) is used.
ippCmpOp	The operation to use for comparing pixel values and the *threshold*. Comparison for either “less than” or “greater than” can be used.

Description

The function `ippiThreshold_Val` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value using the *ippCmpOp* comparison operation. If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep*, *dstStep*, or *srcDstStep* has a zero or negative value.

Threshold_GTVal

Performs thresholding of pixel values in an image. Pixels that are greater than threshold, are set to a specified value.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[3], const Ipp<datatype> value[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[4], const Ipp<datatype> value[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R     32f_C4R
```

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold,
    Ipp<datatype> value);
```

Supported values for *mod* :

```
8u_C1IR     16u_C1IR    16s_C1IR    32f_C1IR
```

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3],
    const Ipp<datatype> value[3]);
```

Supported values for *mod* :

```
8u_C3IR     16u_C3IR    16s_C3IR    32f_C3IR
8u_AC4IR    16u_AC4IR   16s_AC4IR   32f_AC4IR
```

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[4],
    const Ipp<datatype> value[4]);
```

Supported values for *mod* :

```
8u_C4IR     16u_C4IR    16s_C4IR    32f_C4IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.

threshold	The threshold value to use for each pixel.In case of multi-channel data, an array of threshold values (one for each channel) is used.
value	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

Description

The function `ippiThreshold_GTVal` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for "greater than".

If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep*, or *srcDstStep* has a zero or negative value.

Threshold_LTVal

Performs thresholding of pixel values in an image. Pixels that are less than threshold, are set to a specified value.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R     16s_C1R     32f_C1R
```

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R     32f_C3R
8u_AC4R     16u_AC4R    16s_AC4R    32f_AC4R
```

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R     32f_C4R
```

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold,
    Ipp<datatype> value);
```

Supported values for *mod* :

```
8u_C1IR     16u_C1IR    16s_C1IR    32f_C1IR
```

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[3], const Ipp<datatype> value[3]);
```

Supported values for *mod* :

```
8u_C3IR     16u_C3IR    16s_C3IR    32f_C3IR
8u_AC4IR    16u_AC4IR   16s_AC4IR   32f_AC4IR
```

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[4], const Ipp<datatype> value[4]);
```

Supported values for *mod* :

8u_C4IR 16u_C4IR 16s_C4IR 32f_C4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
threshold	The threshold value to use for each pixel.In case of multi-channel data, an array of threshold values (one for each channel) is used.
value	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

Description

The function ippiThreshold_LTVal is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “less than”. If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep*, *dstStep*, or *srcDstStep* has a zero or negative value.

Threshold_LTValGTVal

Performs double thresholding of pixel values in an image buffer.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> thresholdLT, Ipp<datatype> valueLT, Ipp<datatype>
    thresholdGT, Ipp<datatype> valueGT);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const
    Ipp<datatype> thresholdLT[3], const Ipp<datatype> valueLT[3], const
    Ipp<datatype> thresholdGT[3], const Ipp<datatype> valueGT[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp<datatype> thresholdLT,
    Ipp<datatype> valueLT, Ipp<datatype> thresholdGT,
    Ipp<datatype> valueGT);
```

Supported values for *mod* :

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype> thresholdLT[3],
    const Ipp<datatype> valueLT[3], const Ipp<datatype> thresholdGT[3],
    const Ipp<datatype> valueGT[3]);
```

Supported values for *mod* :

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
thresholdLT	The lower threshold value to use for each pixel.In case of multi-channel data, an array of 3 lower threshold values (one for each color channel) is used.
valueLT	The lower output value to be set for each pixel that is less than *thresholdLT*. In case of multi-channel data, an array of 3 lower output values (one for each color channel) is used.
thresholdGT	The upper threshold value to use for each pixel.In case of multi-channel data, an array of 3 upper threshold values (one for each color channel) is used.
valueGT	The upper output value to be set for each pixel that exceeds *thresholdGT*. In case of multi-channel data, an array of 3 upper output values (one for each color channel) is used.

Description

The function `ippiThreshold_LTValGTVal` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image *pSrc* using two specified levels *thresholdLT* and *thresholdGT*. Pixel values in the source image are compared to these levels. If the pixel value is less than *thresholdLT*, the corresponding output pixel is set to *valueLT*. If the pixel value is greater than *thresholdGT*, the output pixel is set to *valueGT*. Otherwise, it is set to the source pixel value.
The value of *thresholdLT* should be less than or equal to *thresholdGT*.

The code example below illustrates thresholding with two levels:

Example 7-2 Thresholding an Image

```
IppStatus threshold( void ) {
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   int i;
   for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
   return ippiThreshold_LTValGTVal_8u_C1IR( x, 5, roi, 2,1,6,7, );
}
```

The destination image *x* contains:

```
01 01 02 03 04
05 06 07 07 07
07 07 07 07 07
07 07 07 07 07
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsThresholdErr	Indicates an error when *thresholdLT* is greater than *thresholdGT*.
ippStsStepErr	Indicates an error condition if *srcStep*, *dstStep*, or *srcDstStep* has a zero or negative value.

ComputeThreshold_Otsu

Computes the value of the Otsu threshold.

Syntax

```
IppStatus ippiComputeThreshold_Otsu_8u_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize roiSize, Ipp8u* pThreshold);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the image ROI in pixels.
pThreshold	Pointer to the Otsu threshold value.

Description

The function `ippiComputeThreshold_Otsu` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function calculates the Otsu threshold [Otsu79] for the source image *pSrc* in according with the formula:

$$\min_{T}(\sigma(pSrc(x, y) \le T) + \sigma(pSrc(x, y) > T))$$

where $0 \le x < roiSize.width$, $0 \le y < roiSize.height$, T - Otsu threshold.

$$\sigma(pSrc(x, y)) = \sqrt{\frac{\sum_{\substack{x<roiSize.width \\ y<roiSize.height \\ x,y \ge 0}} (pSrc(x, y) - mean)^2}{roiSize.height \cdot roiSize.width}}$$

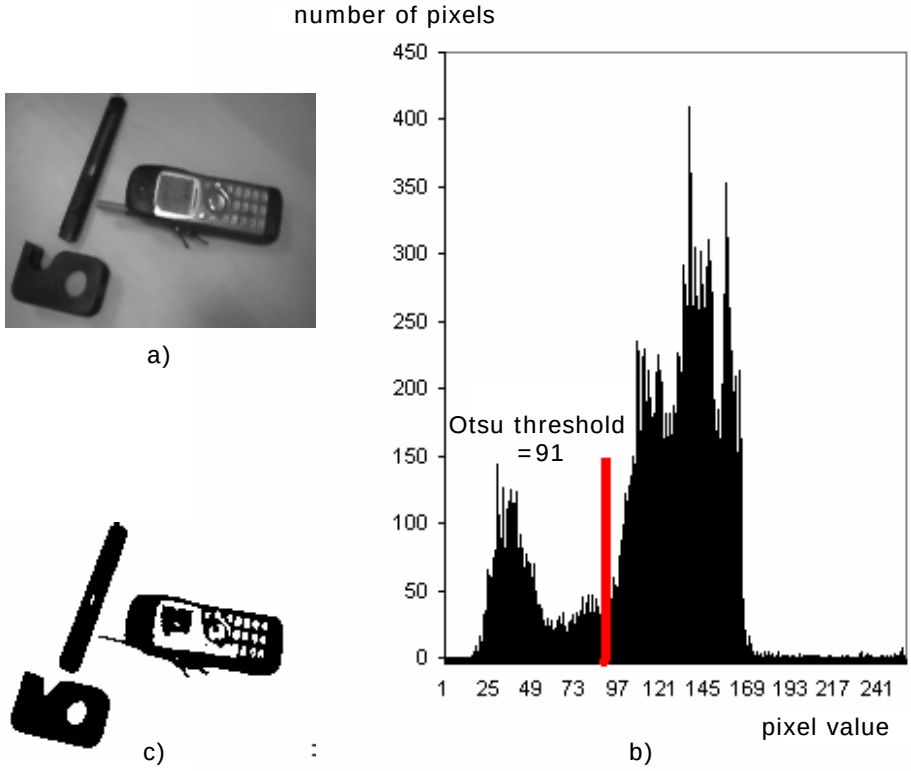
where

$$mean = \frac{\sum_{\substack{x<roiSize.width \\ y<roiSize.height \\ x,y \ge 0}} pSrc(x, y)}{roiSize.height \cdot roiSize.width}$$

The computed Otsu threshold can be used in the thresholding functions described above.

For example, the following figures shows how the Otsu threshold can be used for background/foreground selection. The Figure 7-1 a) shows the initial image, Figure 7-1 b) shows the histogram of the initial image with red line indicating the computed Otsu threshold. The Figure 7-1 c) demonstrates the resulting image obtained by applying to the source image the function `ippiThreshold_8u_C1R` with computed Otsu threshold value.

Figure 7-1 Using Otsu Threshold for Background/Foreground Selection



Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`roiSize`* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *`srcStep`* is less than or equal to 0.

Compare Operations

This section describes functions that compare images. Each compare function writes its results to a 1-channel `Ipp8u` output image. The output pixel is set to a non-zero value if the corresponding input pixel(s) satisfy the compare condition; otherwise, the output pixel is set to 0.

You can compare either two images, or an image and a constant value, using the following compare conditions: “greater”, “greater or equal”, “less”, “less or equal”, “equal”. Compare condition is specified as a function argument of `IppCmpOp` type (see Structures and Enumerators in chapter 2 for more information).

Images containing floating-point data can also be compared for being equal within a given tolerance *eps*.

For images with multi-channel data, the compare condition for a given pixel is true only when each color channel value of that pixel satisfies this condition.

Compare

Compares pixel values of two images using a specified compare operation.

Syntax

```
IppStatus ippiCompare_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp8u* pDst, int dstStep,
    IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc1, pSrc2	Pointers to the source image ROIs.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
ippCmpOp	Compare operation to be used for comparing the pixel values.

Description

The function `ippiCompare` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function compares the corresponding pixels of ROI in two source images *pSrc1*, *pSrc2* using the *ippCmpOp* compare operation, and writes the results to a 1-channel `Ipp8u` image *pDst*. If the result of the compare is true, the corresponding output pixel is set to an `IPP_MAX_8U` value; otherwise, it is set to 0.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *src1Step*, *src2Step*, or *dstStep* has a zero or negative value.

CompareC

Compares pixel values of a source image to a given value using a specified compare operation.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp8u* pDst, int dstStep, IppiSize roiSize,
    IppCmpOp ippCmpOp);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R      16s_C1R      32f_C1R
```

Case 2: Operation on multi-channel data

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp8u* pDst, int dstStep,
    IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp8u* pDst, int dstStep,
    IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
value	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
ippCmpOp	Compare operation to be used for comparing the pixel values.

Description

The function `ippiCompareC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function compares pixels of the each channel of the source image ROI *pSrc* to a given *value* specified for each channel using the *ippCmpOp* compare operation, and writes the results to a 1-channel `Ipp8u` image *pDst*. If the result of the compare is true, that is all pixels of all channels satisfy to the specified condition, then the corresponding output pixel is set to an `IPP_MAX_8U` value; otherwise, it is set to 0.

The following code example shows how to use comparison function and create a mask image:

Example 7-3 Comparing Image to a Constant Value

```
IppStatus compare ( void ) {
   Ipp8u x[5*4], y[5*4];
   IppiSize roi = {5,4};
   int i;
   for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
   return ippiCompareC_8u_C1R ( x, 5, 7, y, 5, roi,
ippCmpGreater );
}
```

The mask image *y* contains:

```
00 00 00 00 00
00 00 00 FF FF
FF FF FF FF FF
FF FF FF FF FF
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

CompareEqualEps

Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance `eps`.

Syntax

```
IppStatus ippiCompareEqualEps_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep,
    IppiSize roiSize, Ipp32f eps);
```

Supported values for *mod* :

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Parameters

pSrc1, pSrc2	Pointers to the source image ROIs.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
eps	The tolerance value.

Description

The function `ippiCompareEqualEps` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function tests if the corresponding pixels of ROI in two source images *pSrc1*, *pSrc2* are equal within the tolerance *eps*, and writes the results to a 1-channel `Ipp8u` image *pDst*. If the absolute value of difference of the pixel values in *pSrc1* and *pSrc2* is less than or equal to *eps*, then the corresponding pixel in *pDst* is set to an `IPP_MAX_8U` value;

otherwise the pixel in *pDst* is set to 0. For multi-channel images, the differences for all color channel values of a pixel must be within the tolerance *eps* for the compare condition to be true.

This function processes images with floating-point data only.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *src1Step*, *src2Step*, or *dstStep* has a zero or negative value.
ippStsEpsValErr	Indicates an error condition if *eps* has a negative value.

CompareEqualEpsC

Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance eps.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiCompareEqualEpsC_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f value, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

Case 2: Operation on multi-channel data

```
IppStatus ippiCompareEqualEpsC_<mod>(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    Ipp32f eps);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiCompareEqualEpsC_32f_C4R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    Ipp32f eps);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
value	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
eps	The tolerance value.

Description

The function `ippiCompareEqualEpsC` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function tests if pixel values of the source image ROI *pSrc* are equal to a given constant *value* within the tolerance *eps*, and writes the results to a 1-channel `Ipp8u` image *pDst*. If the absolute value of difference between the pixel value in *pSrc* and *value* is less than or equal to *eps*, then the corresponding pixel in *pDst* is set to an `IPP_MAX_8U` value; otherwise the pixel in *pDst* is set to 0. For multi-channel images, the differences between all color channel values of a pixel and the respective components of *value* must be within the tolerance *eps* for the compare condition to be true.

This function processes images with floating-point data only.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsEpsValErr`	Indicates an error condition if *eps* has a negative value.

Morphological Operation 8

This chapter describes the Intel® IPP image processing functions that perform morphological operations on images.

Table 8-1 lists functions described in more detail later in this chapter:

Table 8-1 Morphological Functions

Function Base Name	Operation
Dilate3x3	Performs dilation of an image using a 3x3 mask.
Erode3x3	Performs erosion of an image using a 3x3 mask.
Dilate	Performs dilation of an image using a general rectangular mask.
Erode	Performs erosion of an image using a general rectangular mask.
MorphologyInitAlloc	Allocates and initializes morphology state structures for the erosion or dilatation operation.
MorphologyFree	Frees memory allocated for the morphology state structure.
MorphologyInit	Initializes morphology state structures for the erosion or dilatation operation.
MorphologyGetSize	Computes the size of the morphology state structures for the erosion or dilatation operation.
DilateBorderReplicate	Performs dilation of an image.
ErodeBorderReplicate	Performs erosion of an image.
MorphAdvInitAlloc	Allocates and initializes morphology state structure for advanced morphology operations.
MorphAdvFree	Frees memory allocated for the advanced morphology state structure.

Table 8-1 Morphological Functions (continued)

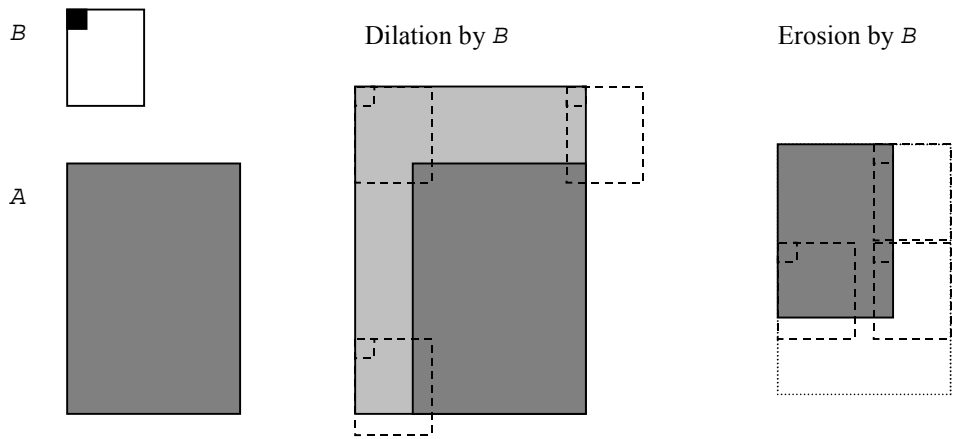
Function Base Name	Operation
MorphAdvInit	Initializes morphology state structure for advanced morphology operations.
MorphAdvGetSize	Computes the size of the advanced morphology state structure.
MorphOpenBorder	Perfoms opening operation of an image.
MorphCloseBorder	Perfoms closing operation of an image.
MorphTophatBorder	Performs top-hat operation of an image.
MorphBlackhatBorder	Performs black-hat operation of an image.
MorphGradientBorder	Calculates morphological gradient of an image.
MorphGrayInitAlloc	Allocates and initializes gray-kernel morphology state structure.
MorphGrayFree	Frees memory allocated for the gray-kernel morphology state structure.
MorphGrayInit	Initializes gray-kernel morphology state structure.
MorphGrayGetSize	Computes the size of the gray-kernel morphology state structure.
GrayDilateBorder	Performs gray-kernel dilation of an image.
GrayErodeBorder	Performs gray-kernel erosion of an image.
MorphReconstructGetBufferSize	Computes the size of the buffer for morphological reconstruction operation.
MorphReconstructDilate	Reconstructs an image by dilation.
MorphReconstructErode	Reconstructs an image by erosion.

Generally, the erosion and dilation smooth the boundaries of objects without significantly changing their area. Opening and closing smooth thin projections or gaps. Morphological operations use a structuring element (SE) that is a a user-defined rectangular mask, or for some functions - symmetric 3x3 mask.

In a more general sense, morphological operations involve an image A called the *object of interest* and a kernel element B called the *structuring element*. The image and structuring element could be in any number of dimensions, but the most common use is with a 2D binary image, or with a 3D gray scale image. The element B is most often a square or a

circle, but it could be of any shape. Just like in convolution, B is a kernel or template with an anchor point. Figure 8-1 shows dilation and erosion of object A by B. In the figure, B is rectangular with an anchor point at upper left shown as a dark square.

Figure 8-1 Dilation and Erosion of *A* by *B*



Let B_t is the SE with pixel t in the anchor position, $\bar{B}$ is transpose of the SE.

Dilation of binary image A $\{A(t) = 1, t \in A; 0$ - otherwise$\}$ by binary SE B is

$$A \oplus B = \{t : B_t \cap A \neq 0\}$$

It means that every pixel is in the set, if the intersection is not null. That is, a pixel under the anchor point of B is marked “on”, if at least one pixel of B is inside of A.

Erosion of binary image A by by binary SE B is

$$A \Theta B = \{t : B_t \subseteq A\}$$

That is, a pixel under the anchor of B is marked “on”, if B is entirely within A.

Generalization of dilation and erosion for the gray-scale image A and the binary SE B is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} A(u) \right\}, \qquad A \Theta B = \left\{ \min_{u \in B_t \cap A} A(u) \right\}$$

Generalization of dilation and erosion for the gray-scale image A and the gray-scale SE B is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} (A(u) + B(u-t)) \right\}, \qquad A \Theta B = \left\{ \min_{u \in B_t \cap A} (A(u) + B(u-t)) \right\}$$

Opening operation of A by B is $A \circ B = (A \oplus B) \Theta \bar{B}$.

Closing operation of A by B is $A \bullet B = (A \Theta B) \oplus \bar{B}$,

Top-hat operation of A by B is $A - A \circ B$.

Black-hat operation of A by B is $A \bullet B - A$.

Morphological gradient of A by B is $A \oplus B - A \Theta B$.

Morphological reconstruction [Vincent93] $\rho_A(C)$ of an image A from the image C, $A(t) \geq C(t) \forall t$ by dilation with the mask B is an image

$$C_k : k = \min_i C_i \equiv C_{i-1},\ C_0 = C,\ C_{i+1}(t) = \min\{(C_i \oplus B)(t), A(t)\}$$

Morphological reconstruction $\rho_A(C)$ of an image A from the image C, $A(t) \leq C(t) \forall t$ by erosion with the mask B is an image

$$C_k : k = \min_i C_i \equiv C_{i-1},\ C_0 = C,\ C_{i+1}(t) = \max\{(C_i \Theta B)(t), A(t)\}$$

Figure 8-2 presents the results of different morphological operations applied to the initial image . In these operations the SE is a matrix of 3x3 size with the following values:

$\begin{bmatrix} -8 & 0 & -8 \\ 0 & 8 & 0 \\ -8 & 0 & -8 \end{bmatrix}$ for common and advanced morphology, and $\begin{bmatrix} -5 & 0 & -5 \\ 0 & 5 & 0 \\ -5 & 0 & -5 \end{bmatrix}$ for gray morphology.

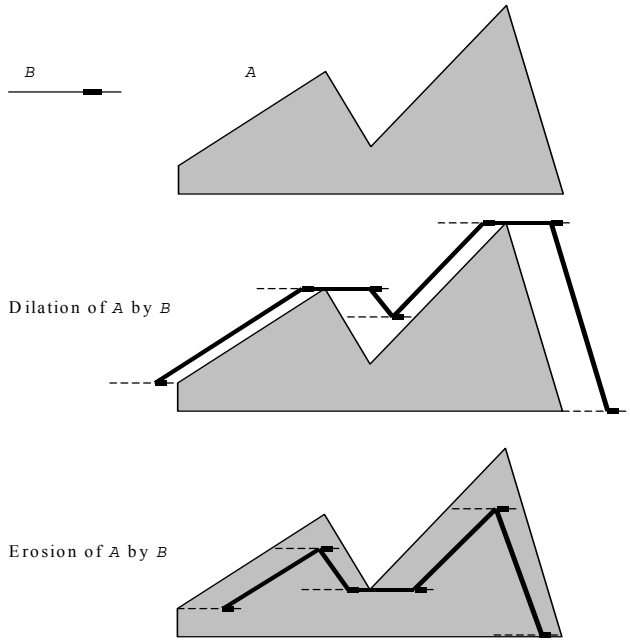
The anchor cell in the center cell (1, 1) of the matrix.

Figure 8-2 Morphological Operations Performed by the Intel IPP

Flat Structuring Elements for Grayscale Image

Erosion and dilation can be done in 3D space, that is, with gray levels. 3D structuring elements can be used, but the simplest and the best way is to use a flat structuring element B. Figure 8-3 is a 1D cross section of dilation and erosion of a grayscale image A by a flat structuring element B. In the figure, B has an anchor slightly to the right of the center as shown by the dark mark on B.

Figure 8-3 1D Cross Section of Dilation and Erosion of A by B



In Figure 8-3, dilation is mathematically $\sup_{y \in B_t} A$ and erosion is $\inf_{y \in B_t} A$.

Dilate3x3

Performs dilation of an image using a 3x3 mask.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiDilate3x3_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: In-place operation

```
IppStatus ippiDilate3x3_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod* :

8u_C1IR	16u_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROIs for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.

roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiDilate3x3` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a symmetric 3x3 mask.
Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its 8 neighboring pixels.
The code example Example 8-1 that shows how to the dilation function is given below:

Example 8-1 Dilation of a Dot

```
IppStatus dilate( void ) {
   Ipp8u x[7*5];
   IppiSize roi = {7,5};
   ippiSet_8u_C1R( 0, x, 7, roi );
   x[2*7+3] = 1;
   roi.width = roi.width - 2;
   roi.height = roi.height - 2;
   return ippiDilate3x3_8u_C1IR( x+7+1, 7, roi );
}
```

The destination image *x* contains:

```
00 00 00 00 00 00 00
00 00 01 01 01 00 00
00 00 01 01 01 00 00
00 00 01 01 01 00 00
00 00 00 00 00 00 00
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep*, *dstStep* or *srcDstStep* has a zero or negative value.

Erode3x3

Performs erosion of an image using a 3x3 mask.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiErode3x3_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: In-place operation

```
IppStatus ippiErode3x3_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod* :

8u_C1IR	16u_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROIs for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiErode3x3` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a symmetric 3x3 mask.
Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the minimum of the corresponding input pixel and its 8 neighboring pixels.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc, pDst,* or *pSrcDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep* or *srcDstStep* has a zero or negative value.

Dilate

Performs dilation of an image using a specified mask.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const char* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: In-place operation

```
IppStatus ippiDilate_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
```

Supported values for *mod* :

8u_C1IR	16u_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROIs for the in-place operation.

srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
pMask	Pointer to mask values. Only pixels that correspond to nonzero mask values are taken into account during operation.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

Description

The function `ippiDilate` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask *pMask* of size *maskSize* and alignment *anchor*.
Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values. In the four-channel image the alpha channel is not processed.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* or *maskSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep* or *srcDstStep* has a zero or negative value.
`ippStsZeroMaskValuesErr`	Indicates an error condition if all mask values are equal to zero.

Erode

Performs erosion of an image using a specified mask.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiErode_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const char* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: In-place operation

```
IppStatus ippiErode_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
```

Supported values for *mod* :

8u_C1IR	16u_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROIs for the in-place operation.

srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
pMask	Pointer to mask values. Only pixels that correspond to nonzero mask values are taken into account during operation.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

Description

The function `ippiErode` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask *pMask* of size *maskSize* and alignment *anchor*.
Source and destination images can have different size, but the ROI size is the same for both images (*roiSize*). The output pixel is set to the minimum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values. In the four-channel image the alpha channel is not processed.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* or *maskSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep* or *srcDstStep* has a zero or negative value.
`ippStsZeroMaskValuesErr`	Indicates an error condition if all mask values are equal to zero.

MorphologyInitAlloc

Allocates and initializes morphology state structure for erosion or dilation operation.

Syntax

```
IppStatus ippiMorphologyInitAlloc_<mod>(int roiWidth, const Ipp8u*
    pMask, IppiSize maskSize, IppiPoint anchor, IppiMorphState**
    ppState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

roiWidth	Maximal width of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
anchor	Coordinates of the anchor cell.
ppState	Pointer to the pointer to the morphology state structure.

Description

The function `ippiMorphologyInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory, initializes the morphology state structure and returns the pointer *ppState* to it. It is used by the functions `DilateBorderReplicate` and `ErodeBorderReplicate` that perform morphological operations on the source image pixels corresponding to non-zero values of the structuring element *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.

Example 8-2 shows how to use the function `ippiMorphologyInitAlloc_8u_C1R`.

WARNING. The structure can be used to process image with ROI that does not exceed the specified maximum width *roiWidth*

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *maskSize* has a field with zero or negative value, or if *roiWidth* is less than 1.
ippStsAnchorErr	Indicates an error if *anchor* is outside the mask.

MorphologyFree

Frees memory allocated for the morphology state structure.

Syntax

```
IppStatus ippiMorphologyFree(IppiMorphState* pState);
```

Parameters

pState	Pointer to the morphology state structure.

Description

The function ippiMorphologyFree is declared in the ippcv.h file. This function frees memory allocated by the function MorphologyInitAlloc for the morphology state structure *pState*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pState* is NULL.

MorphologyInit

Initializes morphology state structure for erosion or dilation operation.

Syntax

```
IppStatus ippiMorphologyInit_<mod>(int roiWidth, const Ipp8u* pMask,
    IppiSize maskSize, IppiPoint anchor, IppiMorphState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

roiWidth	Maximal width of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
anchor	Coordinates of the anchor cell.
pState	Pointer to the morphology state structure.

Description

The function `ippiMorphologyInit` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the morphology state structure *pState* in the external buffer. Its size should be computed by the function `MorphologyGetSize`. This structure is used by the functions `DilateBorderReplicate` and `ErodeBorderReplicate` that perform morphological operations on the source image pixels corresponding to non-zero values of

the structuring element (mask) *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.

WARNING. The structure can be used to processed image with ROI that does not exceed the specified maximum width *roiWidth*

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *maskSize* has a field with zero or negative value, or if *roiWidth* is less than 1.
ippStsAnchorErr	Indicates an error if *anchor* is outside the mask.

MorphologyGetSize

Computes the size of the morphology state structure.

Syntax

```
IppStatus ippiMorphologyGetSize_<mod>(int roiWidth, const Ipp8u* pMask,
    IppiSize maskSize, int* pSize);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

roiWidth	Maximal width of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
pSize	Pointer to the size of the morphology state structure.

Description

The function `ippiMorphologyGetSize` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the morphology state structure *pState*. This function should be run prior to the function `MorphologyInit`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *maskSize* has a field with zero or negative value, or if *roiWidth* is less than 1.

DilateBorderReplicate

Performs dilation of an image.

Syntax

```
IppStatus ippiDilateBorderReplicate_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphState* pState);
```

Supported values for *mod* :

`8u_C1R`	`32f_C1R`
`8u_C3R`	`32f_C3R`
`8u_C4R`	`32f_C4R`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border; following values are possible:

	`ippBorderRepl` Replicated border is used.
pState	Pointer to the morphology state structure.

Description

The function `ippiDilateBorderReplicate` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions `MorphologyInitAlloc` or `MorphologyInit` beforehand.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.

Example 8-2 shows how to use the function `ippiDilateBorderReplicate_8u_C1R`.

WARNING. The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width *roiWidth*

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value, or if *roiSize.width* is greater than maximum ROI *roiWidth* passed to the initialization functions.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
`ippStsBorderErr`	Indicates an error condition if *borderType* has an illegal value.

Example 8-2 Using the Morphlogy Functions

```
void func_morf()
{
    IppiMorphState* ppState;
    IppiSize roiSize = {5, 5};
    Ipp8u pMask[3*3] = {1, 1, 1,
```

Example 8-2 Using the Morphlogy Functions (continued)

```
                        1, 0, 1,
                        1, 1, 1};
    IppiSize maskSize = {3, 3};
    IppiPoint anchor = {1, 1};
    Ipp8u pSrc[5*5] = { 1, 2, 4, 1, 2,
                        5, 1, 2, 1, 2,
                        1, 2, 1, 2, 1,
                        2, 1, 5, 1, 2};
    int srcStep = 5;
    int dstStep = 5;
    Ipp8u pDst[5*5];
    int dstSize = 5;

    ippiMorphologyInitAlloc_8u_C1R( roiSize.width, pMask, maskSize, anchor,
                                        &ppState);
    ippiDilateBorderReplicate_8u_C1R( pSrc, srcStep,  pDst, dstStep, roiSize,
                                          ippBorderRepl, ppState);

    ippiMorphologyFree(ppState);
}
Result: pDst-> 5 5 4 4 2
               5 5 4 4 2
               5 5 5 5 2
               2 5 2 5 2
               2 5 5 5 2
```

ErodeBorderReplicate

Performs erosion of an image.

Syntax

```
IppStatus ippiErodeBorderReplicate_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border; following values are possible: `ippBorderRepl` Replicated border is used.
pState	Pointer to the morphology state structure.

Description

The function `ippiErodeBorderReplicate` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions `MorphologyInitAlloc` or `MorphologyInit` beforehand.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.

WARNING. The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width *roiWidth*

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value, or if *roiSize.width* is greater than maximum ROI *roiWidth* passed to the initialization functions.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.

ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
ippStsBorderErr	Indicates an error condition if *borderType* has an illegal value.

MorphAdvInitAlloc

Allocates and initializes morphology state structure for advanced morphology operations.

Syntax

```
IppStatus ippiMorphAdvInitAlloc_<mod>( IppiMorphAdvState** ppState,
    IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint
    anchor);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

ppState	Pointer to the pointer to the advanced morphology state structure.
roiSize	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
anchor	Coordinates of the anchor cell.

Description

The function ippiMorphAdvInitAlloc is declared in the ippcv.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory, initializes the advanced morphology state structure and returns the pointer *ppState* to it. It is used by the functions MorphOpenBorder, MorphCloseBorder, MorphTophatBorder, MorphBlackhatBorder, and MorphGradientBorder that perform advanced morphological operations (opening, closing, top-hat, black-hat, and gradient) on the source image pixels corresponding to

non-zero values of the structuring element *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.

WARNING. The structure can be used to processed image with ROI that does not exceed the specified maximum width and height *roiSize*

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *maskSize* or if *roiSize* has a field with zero or negative value.
ippStsAnchorErr	Indicates an error if *anchor* is outside the mask.

MorphAdvFree

Frees memory allocated for the advanced morphology state structure.

Syntax

```
IppStatus ippiMorphAdvFree(IppiMorphAdvState* pState);
```

Parameters

pState	Pointer to the advanced morphology state structure.

Description

The function ippiMorphAdvFree is declared in the ippcv.h file. This function frees memory allocated by the function MorphAdvInitAlloc for the advanced morphology state structure *pState*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pState* is NULL.

MorphAdvInit

Initializes morphology state structure for advanced morphology operations.

Syntax

```
IppStatus ippiMorphologyInit_<mod>( IppiMorphAdvState* pState, IppiSize
    roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

pState	Pointer to the advanced morphology state structure.
roiSize	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
anchor	Coordinates of the anchor cell.

Description

The function `ippiMorphAdvInit` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the advanced morphology state structure *pState* in the external buffer. Its size should be computed by the function `MorphAdvGetSize`. It is used by the functions `MorphOpenBorder`, `MorphCloseBorder`, `MorphTophatBorder`, `MorphBlackhatBorder`, and `MorphGradientBorder` that perform advanced morphological operations (opening, closing, top-hat, black-hat, and gradient) on the source

image pixels corresponding to non-zero values of the structuring element (mask) *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.

WARNING. The structure can be used to processed image with ROI that does not exceed the specified maximum width and height *roiSize*

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *maskSize* or if *roiSize* has a field with zero or negative value.
ippStsAnchorErr	Indicates an error if *anchor* is outside the mask.

MorphAdvGetSize

Computes the size of the advanced morphology state structure.

Syntax

```
IppStatus ippiMorphAdvGetSize_<mod>(IppiSize roiSize, const Ipp8u*
    pMask, IppiSize maskSize, int* pSize);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

roiSize	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
pSize	Pointer to the size of the advanced morphology state structure.

Description

The function `ippiMorphAdvGetSize` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the advanced morphology state structure *pState*. This function should be run prior to the function `MorphAdvInit`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *maskSize* or if *roiSize* has a field with zero or negative value.

MorphOpenBorder

Performs opening of an image.

```
IppStatus ippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

`8u_C1R`	`32f_C1R`
`8u_C3R`	`32f_C3R`
`8u_C4R`	`32f_C4R`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border; following values are possible: `ippBorderRepl` Replicated border is used.

pState	Pointer to the advanced morphology state structure.

Description

The function `ippiMorphOpenBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs opening of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions `MorphAdvInitAlloc` or `MorphAdvInit` beforehand.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

WARNING. The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBorderErr`	Indicates an error condition if *borderType* has an illegal value.

MorphCloseBorder

Performs closing of an image.

```
IppStatus ippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border; following values are possible: `ippBorderRepl` Replicated border is used.
pState	Pointer to the advanced morphology state structure.

Description

The function `ippiMorphCloseBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs closing of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions `MorphAdvInitAlloc` or `MorphAdvInit` beforehand.

The result is equivalent to successive erosion of the source image by the structured element (mask) and dilation by the reverted structured element.

WARNING. The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if *borderType* has an illegal value.

MorphTophatBorder

Performs top-hat operation on an image.

```
IppStatus ippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border; following values are possible: `ippBorderRepl` Replicated border is used.
pState	Pointer to the advanced morphology state structure.

Description

The function `ippiMorphTophatBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function perfoms a top-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions `MorphAdvInitAlloc` or `MorphAdvInit` beforehand.

The result is equivalent to the opening the source image and following subtraction from the initial source image.

WARNING. The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBorderErr`	Indicates an error condition if *borderType* has an illegal value.

MorphBlackhatBorder

Performs black-hat operation on an image.

```
IppStatus ippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border; following values are possible: `ippBorderRepl` Replicated border is used.
pState	Pointer to the advanced morphology state structure.

Description

The function `ippiMorphBlackhatBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs black-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions `MorphAdvInitAlloc` or `MorphAdvInit` beforehand.

The result is equivalent to the subtraction of the initial source image from the closed source image.

WARNING. The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if *borderType* has an illegal value.

MorphGradientBorder

Calculates morphological gradient of an image.

```
IppStatus ippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border; following values are possible: `ippBorderRepl` Replicated border is used.
pState	Pointer to the advanced morphology state structure.

Description

The function `ippiMorphGradientBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function calculates a morphological gradient of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions `MorphAdvInitAlloc` or `MorphAdvInit` beforehand.

The result is equivalent to the subtraction of the opened source image from the closed source image.

WARNING. The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBorderErr`	Indicates an error condition if *borderType* has an illegal value.

MorphGrayInitAlloc

Allocates and initializes morphology state structure for gray-kernel morphology operations.

Syntax

```
IppStatus ippiMorphGrayInitAlloc_8u_C1R(IppiMorphGrayState_8u** ppState,
    IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, IppiPoint
    anchor);
IppStatus ippiMorphGrayInitAlloc_32f_C1R(IppiMorphGrayState_32f**
    ppState, IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize,
    IppiPoint anchor);
```

Parameters

ppState	Pointer to the pointer to the advanced morphology state structure.
roiSize	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
anchor	Coordinates of the anchor cell.

Description

The function `ippiMorphGrayInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory, initializes the gray-kernel morphology state structure and returns the pointer *ppState* to it. It is used by the functions `GrayDilateBorder` and `GrayErodeBorder` that perform gray-kernel dilation and erosion of the source image pixels corresponding to non-zero values of the structuring element *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.

WARNING. The structure can be used to processed image with ROI that does not exceed the specified maximum width and height *roiSize*

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *maskSize* or if *roiSize* has a field with zero or negative value.
ippStsAnchorErr	Indicates an error if *anchor* is outside the mask.

MorphGrayFree

Frees memory allocated for the gray-kernel morphology state structure.

Syntax

```
IppStatus ippiMorphGrayFree_8u_C1R(IppiMorphAdvState_8u* pState);
IppStatus ippiMorphGrayFree_32f_C1R(IppiMorphAdvState_32f* pState);
```

Parameters

pState	Pointer to the gray-kernel morphology state structure.

Description

The function ippiMorphGrayFree is declared in the ippcv.h file. This function frees memory allocated by the function MorphGrayInitAlloc for the gray-kernel morphology state structure *pState*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pState* is NULL.

MorphGrayInit

Initializes morphology state structure for gray-kernel morphology operations.

Syntax

```
IppStatus ippiMorphGrayInit_8u_C1R(IppiMorphGrayState_8u* pState,
    IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, IppiPoint
    anchor);
IppStatus ippiMorphGrayInit_32f_C1R(IppiMorphGrayState_32f* pState,
    IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize, IppiPoint
    anchor);
```

Parameters

pState	Pointer to the pointer to the gray-kernel morphology state structure.
roiSize	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
anchor	Coordinates of the anchor cell.

Description

The function `ippiMorphGrayInit` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the gray-kernel morphology state structure *pState* in the external buffer. Its size should be computed by the function `MorphGrayGetSize`. It is used by the functions `GrayDilateBorder` and `GrayErodeBorder` that perform gray-kernel dilation and erosion of the source image pixels corresponding to non-zero values of the structuring element *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.

WARNING. The structure can be used to processed image with ROI that does not exceed the specified maximum width and height *roiSize*

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *maskSize* or if *roiSize* has a field with zero or negative value.
ippStsAnchorErr	Indicates an error if *anchor* is outside the mask.

MorphGrayGetSize

Computes the size of the gray-kernel morphology state structure.

Syntax

```
IppStatus ippiMorphGrayGetSize_8u_C1R(IppiSize roiSize, const Ipp32s*
    pMask, IppiSize maskSize, int* pSize);
IppStatus ippiMorphAdvGetSize_32f_C1R(IppiSize roiSize, const Ipp32f*
    pMask, IppiSize maskSize, int* pSize);
```

Parameters

roiSize	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
pSize	Pointer to the size of the advanced morphology state structure.

Description

The function ippiMorphGrayGetSize is declared in the ippcv.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the morphology state structure *pState* for gray-kernel dilation and erosion. This function should be run prior to the function MorphGrayInit.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if *maskSize* or if *roiSize* has a field with zero or negative value.

GrayDilateBorder

Performs gray-kernel dilation of an image.

Syntax

```
IppStatus ippiGrayDilateBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphGrayState_8u* pState);
IppStatus ippiGrayDilateBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphGrayState_32f* pState);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border; following values are possible: ippBorderRepl Replicated border is used.
pState	Pointer to the gray-kernel morphology state structure.

Description

The function ippiGrayDilateBorder is declared in the ippcv.h It operates with ROI (see Regions of Interest in Intel IPP).

This function performs gray-kernel dilation of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions MorphGrayInitAlloc or MorphGrayInit beforehand.

WARNING. The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if *borderType* has an illegal value.

GrayErodeBorder

Performs gray-kernel erosion of an image.

Syntax

```
IppStatus ippiGrayErodeBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphGrayState_8u* pState);
IppStatus ippiGrayErodeBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphGrayState_32f* pState);
```

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border; following values are possible: `ippBorderRepl` Replicated border is used.
pState	Pointer to the gray-kernel morphology state structure.

Description

The function `ippiGrayErodeBorder` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function performs gray-kernel erosion of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions `MorphGrayInitAlloc` or `MorphGrayInit` beforehand.

WARNING. The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBorderErr`	Indicates an error condition if *borderType* has an illegal value.

MorphReconstructGetBufferSize

Computes the size of the buffer for morphological reconstruction operation.

```
IppStatus ippiMorphReconstructGetBufferSize_8u_C1(IppiSize roiSize, int*
    pSize);
IppStatus ippiMorphReconstructGetBufferSize_32f_C1(IppiSize roiSize,
    int* pSize);
```

Parameters

roiSize	Maximal size of the image ROI in pixels, that can be processed using the buffer.
pSize	Pointer to the size of the buffer.

Description

The function `ippiMorphReconstructBufferGetSize` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size *pSize* of the buffer for the morphological reconstruction of thye source image. This buffer can be used by the functions and .

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSize* is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

MorphReconstructDilate

Reconstructs an image by dilation.

Syntax

```
IppStatus ippiMorphReconstructDilate_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u*
    pBuffer, IppiNorm norm);
IppStatus ippiMorphReconstructDilate_32f_C1IR(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f*
    pBuffer, IppiNorm norm);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pSrcDst	Pointer to the descreased and reconstructed image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.
roiSize	Size of the source and destination image ROI.
norm	Type of norm to form the mask for dilation; following values are possible: `ippiNormInf` Infinity norm (8-connectivity, 3x3 rectangular mask). `ippiNormL1` L1 norm (4-connectivity, 3x3 cross mask).
pBuffer	Pointer to the buffer.

Description

The function `ippiMorphReconstructDilate` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function performs morphological reconstruction of the decreased source image by dilation [Vincent93]. The operation is performed in the working buffer whose size should be computed using the function `MorphReconstructGetBufferSize` beforehand.

This operation allows to detect the regional maximums that con be used as markers for successive watershed segmentation.

The example Example 8-3 shows how the morphological reconstruction can be used to build markers of objects with different brightness. Some value (cap size) is subtracted from the initial image and then the subtracted image is reconstructed to the initial one. Thresholding and opening complete the building of markers. Figure 8-4 shows the results of these operations.

Example 8-3 Morphological Reconstruction

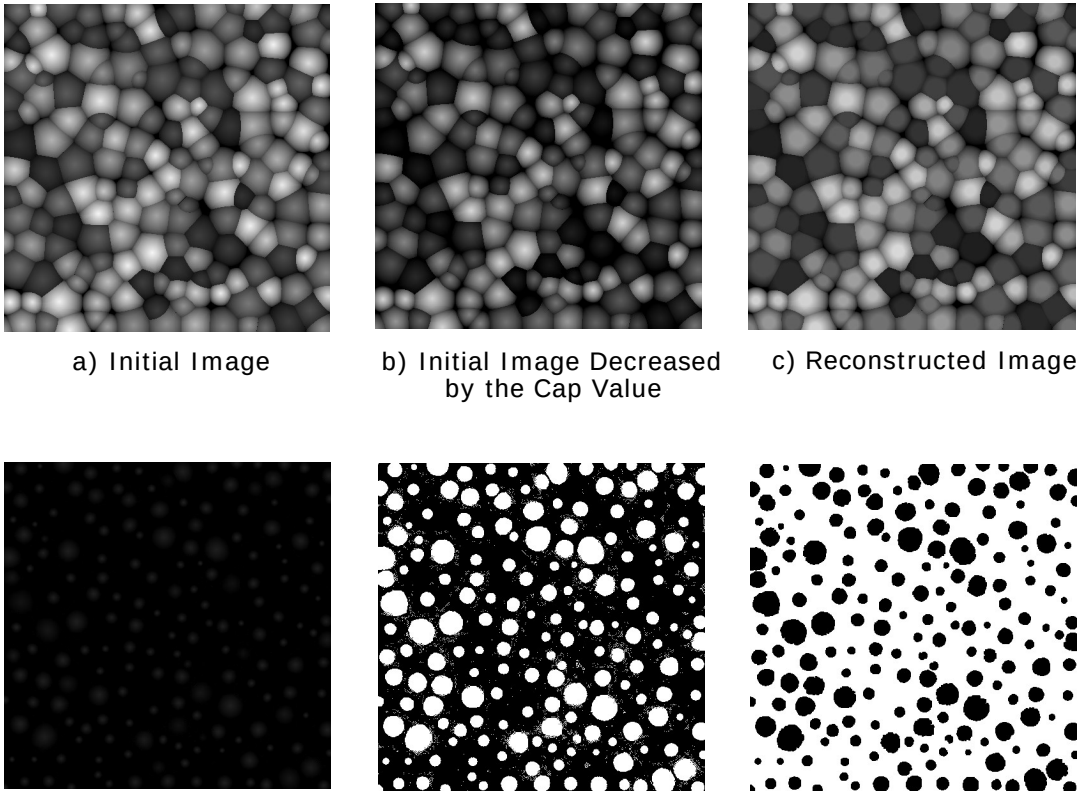
```
IppiMorphAdvState *state;
IppiSize roi, msize={3,3};
IppiPoint anchor={msize.width/2, msize.height/2};
Ipp8u *src, *dst, *img;
Ipp8u *buf, *mask={1,1,1,1,1,1,1,1,1};
Int step, size;

…

ippiMorphReconstructGetBufferSize_8u_C1(roi, &size);
buf = ippsMalloc(size);
ippiMorphAdvInitAlloc_8u_C1R(roi, mask, msize, anchor, &state);

ippiCopy_8u_C1R(src, step, dst, step, roi);
ippiCopy_8u_C1R(src, step, img, step, roi);
// subtract cap size
ippiSubC_8u_C1IRSfs(cap, dst, step, roi, 0);
// reconstruct image
ippiMorphReconstructDilate_8u_C1IR(src, step, dst, step, roi, buf, norm);
// get caps
ippiSub_8u_C1IRSfs(dst, step, img, step, roi, 0);
// caps to white
ippiThreshold_GTVal_8u_C1IR(img, step, roi, 0, 255);
// delete noise
ippiMorphOpenBorder_8u_C1R(img, step, dst, step, roi,
                              ippBorderRepl, state);
// revert to get markers
ippiXorC_8u_C1IR(0xff, dst, step, roi, 0);

ippiMorphAdvFree(state);
ippsFree(buf);
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *srcDstStep* is less than *roiSize.width* * <*pixelSize*>.

`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBadArgErr`	Indicates an error condition if *norm* has an illegal value.

Figure 8-4 Building Markers for Segmentation by the Morphological Reconstruction



a) Initial Image

b) Initial Image Decreased by the Cap Value

c) Reconstructed Image

d) Cap Image

e) Thresholded Caps

f) Markers

MorphReconstructErode

Reconstructs an image by erosion.

Syntax

```
IppStatus ippiMorphReconstructErode_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u*
    pBuf, IppiNorm norm);
IppStatus ippiMorphReconstructErode_32f_C1IR(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f*
    pBuf, IppiNorm norm);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pSrcDst	Pointer to the increased image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the increased image.
roiSize	Size of the source and destination image ROI.
norm	Type of norm to form the mask for erosion; following values are possible: `ippiNormInf` Infinity norm (8-connectivity, 3x3 rectangular mask). `ippiNormL1` L1 norm (4-connectivity, 3x3 cross mask).
pBuffer	Pointer to the buffer.

Description

The function `ippiMorphReconstructErode` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function performs morphological reconstruction of the increased source image by erosion [Vincent93]. The operation is performed in the working buffer whose size should be computed using the function `MorphReconstructGetBufferSize` beforehand.

This operation allows to detect the regional minimums that can be used as markers for successive watershed segmentation.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *srcDstStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBadArgErr`	Indicates an error condition if *norm* has an illegal value.

Filtering Functions

9

This chapter describes the Intel® IPP image processing functions that perform linear and non-linear filtering operations on an image.
Filtering can be used in a variety of image processing operations; for example, edge detection, blurring, noise removal, and feature detection.

Table 9-1 lists functions described in more detail later in this chapter:

Table 9-1 Filtering functions

Function Base Name	Operation
FilterBox	Blurs an image using a box filter
SumWindowRow	Sums pixel values in the row mask applied to the image.
SumWindowColumn	Sums pixel values in the column mask applied to the image.
FilterMin	Filters an image using a *min* filter
FilterMax	Filters an image using a *max* filter
FilterMinGetBufferSize	Computes the size of the working buffer for the minimum filter
FilterMaxGetBufferSize	Computes the size of the working buffer for the maximum filter
FilterMinBorderReplicate	Filters an image using a *min* filter with border replication
FilterMaxBorderReplicate	Filters an image using a *max* filter with border replication
FilterBilateralGetBufSize	Computes the size of the buffer for the bilateral filter
FilterBilateralInit	Initializes the bilateral filter structure
FilterBilateral	Performs bilateral filtering of the image

Table 9-1 Filtering functions (continued)

Function Base Name	Operation
DecimateFilterRow, DecimateFilterColumn	Decimates an image using the specified filter.
Median Filters	
FilterMedian	Applies a median filter to an image
FilterMedianHoriz	Filters an image using a median filter with a horizontal mask
FilterMedianVert	Filters an image using a median filter with a vertical mask
FilterMedianCross	Filters an image using a cross median filter
FilterMedianWeightedCenter3x3	Filters an image using a median filter with weighted center pixel
FilterMedianColor	Applies a color median filter to an image
General Linear Filters	
Filter	Filters an image using a general rectangular convolution kernel
FilterGetBufSize	Compute the size of working buffer for 64f general linear filter
Filter32f	Filters an image with integer data using a floating-point rectangular convolution kernel
Filter_Round16s, Filter_Round32s, Filter_Round32f	Perform filtering an image and rounding the result
FilterRoundGetBufSize16s, FilterRoundGetBufSize32s, FilterRoundGetBufSize32f	Computes the size of the working buffer for filters with rounding mode
Separable Filters	
FilterColumn	Filters an image using an integer column convolution kernel
FilterColumn32f	Filters an image using a floating-point column convolution kernel
FilterRow	Filters an image using an integer row convolution kernel
FilterRow32f	Filters an image using a floating-point row convolution kernel
FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize_Low	Compute the size of working buffer for rows filter with border
FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize_Low	Compute the size of working buffer for columns filter with border

Table 9-1 Filtering functions (continued)

Function Base Name	Operation
FilterRowBorderPipeline, FilterRowBorderPipeline_Low	Applies the filter with border to image rows
FilterColumnPipeline, FilterColumnPipeline_Low	Applies the filter with border to image columns
DotProdCol	Calculates the dot product of taps vector and columns of the specified set of rows
Wiener Filters	
FilterWienerGetBufferSize	Computes the size of the external buffer for ippiFilterWiener function.
FilterWiener	Filters an image using the Wiener algorithm
Convolution	
ConvFull	Performs full convolution of two images
ConvValid	Performs valid convolution of two images
Deconvolution	
DeconvFFTInitAlloc	Allocates and initializes state structure for FFT deconvolution
DeconvFFTFree	Frees memory allocated for the FFT deconvolution state structure
DeconvFFT	Performs FFT deconvolution of an image
DeconvLRInitAlloc	Allocates and initializes state structure for LR deconvolution
DeconvLRFree	Frees memory allocated for the LR deconvolution state structure
DeconvLR	Performs LR deconvolution of an image
Fixed Filters	
FilterPrewittHoriz	Filters an image using a horizontal Prewitt operator
FilterPrewittVert	Filters an image using a vertical Prewitt operator
FilterScharrHoriz	Filters an image using a horizontal Scharr operator
FilterScharrVert	Filters an image using a vertical Scharr operator
FilterSobelHoriz FilterSobelHorizMask	Filters an image using a horizontal Sobel operator

Table 9-1 Filtering functions (continued)

Function Base Name	Operation
FilterSobelVert FilterSobelVerMaskt	Filters an image using a vertical Sobel operator
FilterSobelHorizSecond	Filters an image using a second derivative horizontal Sobel operator.
FilterSobelVertSecond	Filters an image using a second derivative vertical Sobel operator.
FilterSobelCross	Filters an image using a second cross derivative Sobel operator.
FilterRobertsDown	Filters an image using a horizontal Roberts cross-gradient operator
FilterRobertsUp	Filters an image using a vertical Roberts cross-gradient operator
FilterLaplace	Filters an image using a Laplacian kernel
FilterGauss	Filters an image using a Gaussian kernel
FilterHipass	Applies a highpass filter to an image
FilterLowpass	Applies a lowpass filter to an image
FilterSharpen	Applies a sharpening filter to an image
Fixed Filters with Borders	
FilterScharrHorizGetBufferSize	Computes the size of the external buffer for the horizontal Scharr filter with border.
FilterScharrVertGetBufferSize	Computes the size of the external buffer for the vertical Scharr filter with border.
FilterSobelHorizGetBufferSize	Computes the size of the external buffer for the horizontal Sobel filter with border.
FilterSobelVertGetBufferSize, FilterSobelNegVertGetBufferSize	Computes the size of the external buffer for the vertical Sobel filter with border.
FilterSobelHorizSecondGetBufferSize	Computes the size of the external buffer for the second derivative horizontal Sobel filter with border.
FilterSobelVertSecondGetBufferSize	Computes the size of the external buffer for the second derivative vertical Sobel filter with border.
FilterSobelCrossGetBufferSize	Computes the size of the external buffer for the cross Sobel filter with border.
FilterLaplacianGetBufferSize	Computes the size of the external buffer for the Laplace filter with border.

Table 9-1 Filtering functions (continued)

Function Base Name	Operation
FilterGaussGetBufferSize	Computes the size of the external buffer for the Gaussian filter with border.
FilterLowpassGetBufferSize	Computes the size of the external buffer for the lowpass filter with border.
GenSobelKernel	Computes kernel for the Sobel filter to an image.
FilterScharrHorizBorder	Applies horizontal Scharr filter with border to an image.
FilterScharrVertBorder	Applies vertical Scharr filter with border to an image.
FilterSobelHorizBorder	Applies horizontal Sobel filter with border to an image.
FilterSobelVertBorder, FilterSobelNegVertBorder,	Applies vertical Sobel filter with border to an image.
FilterSobelHorizSecondBorder	Applies horizontal (second derivative) Sobel filter with border to an image.
FilterSobelVertSecondBorder	Applies vertical (second derivative) Sobel filter with border to an image.
FilterSobelCrossBorder	Applies second derivative cross Sobel filter with border to an image.
FilterLaplacianBorder	Applies Laplacian filter with border to an image.
FilterGaussBorder	Applies Gauss filter with border.
FilterLowpassBorder	Applies lowpass filter with border to an image.

Most of filtering function operate with ROI (see Regions of Interest in Intel IPP). The size of the source image ROI is equal to *dstRoiSize* the destination image ROI size. Most of the functions use different source and destination image buffers, that is, they are not in-place.

Borders

Filtering functions described later in this chapter perform neighborhood operations (see Neighborhood Operations in chapter 2). They operate on the assumption that for each pixel being processed, all referred neighborhood pixels necessary for the operation are also available.

The neighborhood for each given pixel is defined by the filter kernel (or mask) size and anchor cell position. Anchor cell (see Figure 9-1 a) is a fixed cell within the kernel, which is used for positioning the kernel with respect to the currently processed pixel of the source image. Specifically, the kernel is placed on the image in such a way that the anchor cell coincides with the input pixel.

As Fig. 9-1b illustrates, if the input pixel is near the horizontal or vertical edge of the image, the overlaid kernel may refer to neighborhood pixels that do not exist within the source image (that is, are located outside the image area).
The set of all boundary source image pixels that require such non-existent pixels to complete the neighborhood operation for the given kernel and anchor is shaded yellow in Figure 9-1 b), while the collection of all scanned external pixels (called *border* pixels) is shaded gray.

Hence, if you want to apply some filtering function to the whole source image, you need to first figure out what additional border pixels will be required for the operation, and then define in one way or another these non-existent pixels. To do this, you may either use Intel IPP functions `ippiCopyConstBorder`, `ippiCopyReplicateBorder`, or `ippiCopyWrapBorder` which fill the border of the extended image with the pixel values that you define, or apply your own extension method.

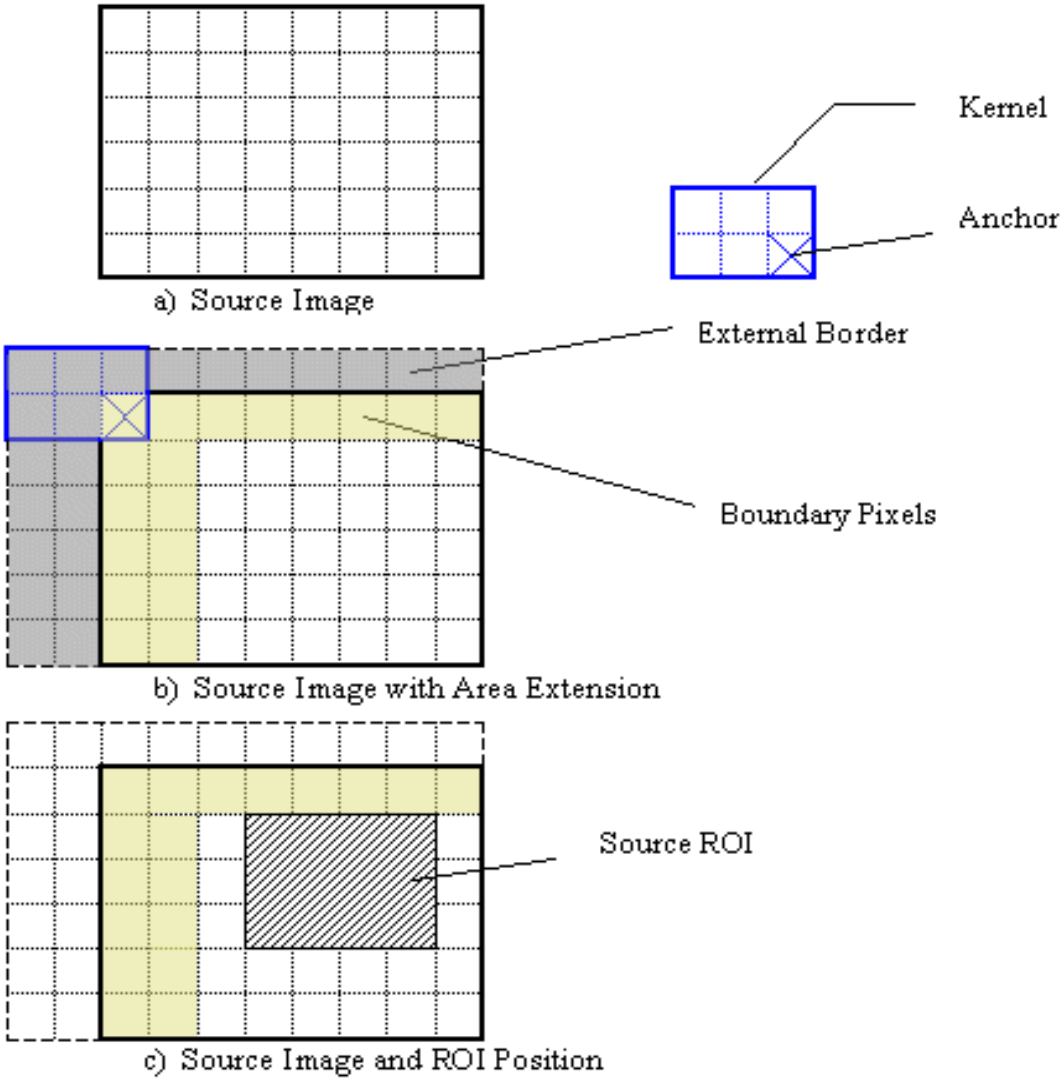
WARNING. If the required border pixels are not defined prior to the filtering function call, you may get memory violation errors.

On the other hand, if the filtering operation is to be carried out on a part of the source image, or ROI (see Regions of Interest in Intel IPP in chapter 2), then the necessity of extending the image area with border pixels depends upon the ROI size and position within the image.
It can be readily seen (Figure 9-1 c) that if ROI does not cover yellow (internal boundary) pixels, than no external pixels will be scanned, and border extension in not required.

If boundary pixels are part of ROI, then some area extension of the source image is still necessary.

The bottom-line is that to provide valid results of filtering operations, the application must check that ROI parameters passed to the filtering function have such values that all required neighborhood pixels actually exist in the image and define the missing pixels when necessary.

Figure 9-1 Borders for Neighborhood Operations



FilterBox

Blurs an image using a simple box filter.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiFilterBox_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Case 1: In-place operation

```
IppStatus ippiFilterBox_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
roiSize	Size of the source and destination ROI in pixels for the in-place operation.

pSrcDst	Pointer to the source and destination image ROIs for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

Description

The function `ippiFilterBox` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the output image as the average of all the input image pixels in the rectangular neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of smoothing or blurring the input image. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* or *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep*, *dstStep*, *or srcDstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *maskSize* has a field with zero or negative value.
`ippStsAnchorErr`	Indicates an error condition if *anchor* is outside the mask size.
`ippStsMemAllocErr`	Indicates a memory allocation error .

SumWindowRow

Sums pixel values in the row mask applied to the image.

Syntax

```
IppStatus ippiSumWindowRow_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int
    maskSize, int xAnchor);
```

Supported values for *mod* :

```
8u32f_C1R   16u32f_C1R  16s32f_C1R
8u32f_C3R   16u32f_C3R  16s32f_C3R
8u32f_C4R   16u32f_C4R  16s32f_C4R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
maskSize	Size of the horizontal row mask in pixels.
xAnchor	Anchor cell specifying the row mask alignment with respect to the position of the input pixel.

Description

The function `ippiSumWindowRow` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image ROI *pDst* as the sum of all the source image pixels in the horizontal row mask of size *maskSize* with the anchor cell *xAnchor* at the corresponding pixel in the source image ROI *pSrc*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if *maskSize* has a zero or negative value.
ippStsAnchorErr	Indicates an error condition if *xAnchor* is outside the mask size.

SumWindowColumn

Sums pixel values in the column mask applied to the image.

Syntax

```
IppStatus ippiSumWindowColumn_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int
    maskSize, int yAnchor);
```

Supported values for *mod* :

```
8u32f_C1R   16u32f_C1R  16s32f_C1R
8u32f_C3R   16u32f_C3R  16s32f_C3R
8u32f_C4R   16u32f_C4R  16s32f_C4R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
maskSize	Size of the vertical column mask in pixels.
anchor	Anchor cell specifying the column mask alignment with respect to the position of the input pixel.

Description

The function `ippiSumWindowColumn` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image ROI *pDst* as the sum of all the source image pixels in the vertical row mask of size *maskSize* with the anchor cell *yxAnchor* at the corresponding pixel in the source image ROI *pSrc*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *maskSize* has a zero or negative value.
`ippStsAnchorErr`	Indicates an error condition if *yAnchor* is outside the mask size.
`ippStsMemAllocErr`	Indicates a memory allocation error.

FilterMin

Applies the 'min' filter to an image.

Syntax

```
IppStatus ippiFilterMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod*:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`
`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`
`8u_C4R`	`16u_C4R`	`16s_C4R`	`32f_C4R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`	`32f_AC4R`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

Description

The function `ippiFilterMin` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image to the minimum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of decreasing the contrast in the image.
The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

The Example 9-1 illustrates the use of the `ippiFilterMin` function. Note that this function operates on the assumption that all neighborhood pixels needed from outside the ROI are available.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *maskSize* has a field with zero or negative value.
`ippStsAnchorErr`	Indicates an error condition if *anchor* is outside the mask size.

Example 9-1 Applying the Min Filter

```
IppStatus filterMin( void ) {
   Ipp8u x[5*4], y[5*4]={0};
   IppiSize img={5,4}, roi={3,2}, mask={3,3};
   IppiPoint anchor = {1,1};
   ippiSet_8u_C1R( 3, x, 5, img );
   /// set a hole (1) at row 1 and column 1.
   /// The value will be extended on filter
   /// mask area, depending on anchor
   x[5+1] = 1;
   /// ROI is inside the image.
   /// Offset pointers to jump at the ROI start
   return ippiFilterMin_8u_C1R( x+6, 5, y+6, 5, roi, mask, anchor );
}
The destination image y contains:
00 00 00 00 00
00 01 01 03 00
00 01 01 03 00
00 00 00 00 00
```

FilterMax

Applies the 'max' filter to an image.

Syntax

```
IppStatus ippiFilterMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

Description

The function `ippiFilterMax` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image to the maximum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of increasing the contrast in the image.
The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *maskSize* has a field with zero or negative value.
`ippStsAnchorErr`	Indicates an error condition if *anchor* is outside the mask size.

FilterMinGetBufferSize

Computes the size of the working buffer for the minimum filter.

Syntax

```
IppStatus ippiFilterMinGetBufferSize_<mod>(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

roiWidth	Image width (in pixels).
maskSize	Size of the mask in pixels.
pBufferSize	Pointer to the computed size of the buffer.

Description

The function `ippiFilterMinGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for `ippiFilterMinBorderReplicate` function. The buffer with the length *pBufferSize*[0] can be used to filter images with width equal to or less than *roiWidth*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer *pBufferSize* is NULL.
ippStsSizeErr	Indicates an error condition if *maskSize* has a field with zero or negative value, or if *roiWidth* is less than 1.

FilterMaxGetBufferSize

Computes the size of the working buffer for the maximum filter.

Syntax

```
IppStatus ippiFilterMaxGetBufferSize_<mod>(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

roiWidth	Image width (in pixels).
maskSize	Size of the mask in pixels.
pBufferSize	Pointer to the computed size of the buffer.

Description

The function `ippiFilterMaxGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for `ippiFilterMaxBorderReplicate` function. The buffer with the length *pBufferSize*[0] can be used to filter images with width equal to or less than *roiWidth*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer *pBufferSize* is NULL.
ippStsSizeErr	Indicates an error condition if *maskSize* has a field with zero or negative value, or if *roiWidth* is less than 1.

FilterMinBorderReplicate

Applies the 'min' filter with border replication to an image.

Syntax

```
IppStatus ippiFilterMinBorderReplicate_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiSize maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterMinBorderReplicate` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image to the minimum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of decreasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. Border pixels are chosen according to the Figure 4-2 (border replication). The function requires the external buffer *pBuffer*, its size should be previously computed by the function ippiFilterMinGetBufferSize.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* or *maskSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsAnchorErr	Indicates an error if *anchor* is outside the mask.

FilterMaxBorderReplicate

Applies the 'max' filter with border replication to an image.

Syntax

```
IppStatus ippiFilterMaxBorderReplicate_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiSize maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterMaxBorderReplicate` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image to the maximum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of increasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. Border pixels are chosen according to the Figure 4-2 (border replication). The function requires the external buffer *pBuffer*, its size should be previously computed by the function `ippiFilterMaxGetBufferSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* or *maskSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsAnchorErr`	Indicates an error if *anchor* is outside the mask.

FilterBilateralGetBufSize

Computes the size of the buffer for the bilateral filter.

Syntax

```
IppStatus ippiFilterBilateralGetBufSize_8u_C1R(IppiFilterBilateralType
    filter, IppiSize maxDstRoiSize, IppiSize maxKernelSize, int*
    pBufferSize);
```

Parameters

filter	Type of the bilateral filter; possible value `ippiFilterBilateralGauss` - Gaussian bilateral filter.
maxDstRoiSize	Maximum size of the destination image ROI.
maxKernelSize	Maximum size of the filter kernel.
pBufferSize	Pointer to the computed size of the buffer.

Description

The function `ippiFilterBilateralGetBufSize` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the size of the working buffer for the bilateral filter of the *filter* type that is used by the function `ippiFilterBilateralInit`.

The both dimensions of the filter kernel should be odd; if some of them size is even, the function changes its value to the nearest less odd number.

Example 9-2 shows how to use the function `ippiFilterBilateralGetBufSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if the pointer *pBufferSize* is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *maxDstRoiSize* has a field with zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *maxKernelSize* has a field with zero or negative value.
`ippStsNotSupportedModeErr`	Indicates an error condition if the type of the specified bilateral filter is not supported.

FilterBilateralInit

Initializes the bilateral filter structure.

Syntax

```
IppStatus ippiFilterBilateralInit_8u_C1R(IppiFilterBilateralType filter,
    IppiSize maxKernelSize, Ipp32f valSquareSigma, Ipp32f posSquareSigma,
    int stepInKernel, IppiFilterBilateralSpec* pSpec);
```

Parameters

filter	Type of the bilateral filter; possible value `ippiFilterBilateralGauss` - Gaussian bilateral filter.
maxKernelSize	Maximum size of the filter kernel.
valSquareSigma	Square of the sigma for differences of pixel values.
posSquareSigma	Square of the sigma for pixel positions.
stepInKernel	Processing step in the filter kernel.
pSpec	Pointer to the bilateral filter structure.

Description

The function `ippiFilterBilateralInit` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function initializes the state structure for the bilateral filter of the *filter* type in the external buffer. Size of the buffer must be computed by the function `ippiFilterBilateralGetBufSize` beforehand.

maxKernelSize specifies the maximum sizes of the filter kernel that should be the same as specified in the function `ippiFilterBilateralGetBufSize`.

The anchor cell is always the central cell of the kernel. Therefore, both dimensions of the filter kernel should be odd; if any of the sizes is even, the function changes its value to the nearest less odd number.

Coefficients of the bilateral filter kernel depend on their position in the kernel and on the value of some pixels of the source image laying in the kernel. Only pixels with both coordinates divisible by *stepInKernel* are used in calculations.

The value of the output pixel d is

$$d = \frac{\sum_{i,\ j} W1_{i,\ j} \cdot W2_{i,\ j} \cdot v_{i,\ j}}{\sum_{i,\ j} W1_{i,\ j} \cdot W2_{i,\ j}}$$

Here $v_{i,\ j}$ - value of a pixel in the kernel with coordinates i and j (coordinates of the central pixel are 0, 0), i, j = -(*kernelSize*-1)/2, -((*kernelSize*-1)/2)+1, ..., -1, 0, 1, ((*kernelSize*-1)/2)-1, (*kernelSize*-1)/2.

$$W1_{i,\ j} = Fun(valSquareSigma,\ (v_{i,\ j} - v_{0,\ 0})),$$

$$W2_{i,\ j} = Fun(posSquareSigma,\ MaxByAbs(i,\ j))$$

where

$v_{i,\ j} - v_{0,\ 0}$ - difference of pixel values,

$MaxByAbs(i,\ j) = \begin{cases} i, & \text{if } Abs(i) \geq Abs(j) \\ j, & \text{if } Abs(i) < Abs(j) \end{cases}$ - pixel position,

valSquareSigma and *posSquareSigma* - squares of values of sigma for pixel differences and their positions respectively.

Intel IPP functions implement only the Gaussian filtering, which reduces noise and smooths the image, but preserves edges. For the Gaussian filtering

$Fun(S,\ I) = \exp\left(-\frac{I^2}{2S^2}\right)$, S^2 - *valSquareSigma* or *posSquareSigma*, I - difference of pixel values or position.

Example 9-2 shows how to use the function `ippiFilterBilateralInit`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if the pointer *pBufferSize* is NULL.
`ippStsSizeErr`	Indicates an error condition if *maxDstRoiSize* has a field with zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *maxKernelSize* has a field with zero or negative value.
`ippStsNotSupportedModeErr`	Indicates an error condition if the type of the specified bilateral filter is not supported.

FilterBilateral

Performs bilateral filtering of the image.

Syntax

```
IppStatus ippiFilterBilateral_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize kernelSize,
    IppiFilterBilateralSpec* pSpec);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
kernelSize	Size of the filter kernel.
pSpec	Pointer to the bilateral filter structure.

Description

The function `ippiFilterBilateral` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies the bilateral filter with the rectangular kernel of the size *kernelSize* to the pixels of ROI of the source image *pSrc*. The anchor cell is always central cell of the kernel. Hence both dimensions of the filter kernel should be odd; if some of size is even, the function changes its value to the nearest less odd number.

The bilateral filter structure *pSpec* contains the parameters of filtering and must be initialized by the function `ippiFilterBilateralInit` beforehand. The size of the filter kernel should not be greater than the value *maxKernelSize* specified in the filter structure.

Example 9-2 shows how to use the function `ippiFilterBilateral`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if *kernelSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than or equal to 0.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid context structure is passed.

Example 9-2 Using Bilateral Filtering Functions

```
#include <stdlib.h>
#include <stdio.h>
#include "ippcore.h"
#include "ipps.h"
#include "ippi.h"

#define D_WIDTH     8
#define D_HEIGHT    8
#define D_MAX_W_KNL 5
#define D_MAX_H_KNL 5

int main( void )
{
    Ipp8u       *src, *src_start;
    Ipp8u       *dst1, *dst2, *dst3;
    Ipp8u       *ptmp;
    int         width_src, height_src;
    int         width_dst, height_dst;
    int         step_src;
    int         step_dst1, step_dst2, step_dst3;
    int         bufsize;
    int         stepInKernel;
    IppiSize    roi;
    IppiSize    kernel;
    int         i, j;
    Ipp32f      valsigma;
    Ipp32f      possigma;
    IppiFilterBilateralSpec *pSpec;


    width_src = D_WIDTH + D_MAX_W_KNL - 1;
    height_src = D_HEIGHT + D_MAX_H_KNL - 1;
    src = ippiMalloc_8u_C1( width_src, height_src, &step_src );
    for ( i = 0, ptmp = src; i < height_src; i++ ) {
        for ( j = 0; j < width_src; j++ ) {
            ptmp[j] = (Ipp8u)rand();
        }
```

Example 9-2 Using Bilateral Filtering Functions

```
        ptmp = (Ipp8u *)( (char *)ptmp + step_src );
    }
    src_start = (Ipp8u *)( (char *)src + (D_MAX_H_KNL>>1) * step_src );
    src_start += (D_MAX_W_KNL>>1);

    width_dst = D_WIDTH;
    height_dst = D_HEIGHT;
    dst1 = ippiMalloc_8u_C1( width_dst, height_dst, &step_dst1 );
    dst2 = ippiMalloc_8u_C1( width_dst, height_dst, &step_dst2 );
    dst3 = ippiMalloc_8u_C1( width_dst, height_dst, &step_dst3 );


    roi.width = D_WIDTH;
    roi.height = D_HEIGHT;
    kernel.width = D_MAX_W_KNL;
    kernel.height = D_MAX_H_KNL;
    ippiFilterBilateralGetBufSize_8u_C1R( ippiFilterBilateralGauss, roi,
                                          kernel, &bufsize);
    pSpec = (IppiFilterBilateralSpec *)ippsMalloc_8u( bufsize );

    valsigma = 16.f;
    possigma = 16.f;
    stepInKernel = 1;
    ippiFilterBilateralInit_8u_C1R( ippiFilterBilateralGauss, kernel, valsigma,
                                    possigma, stepInKernel, pSpec );
    ippiFilterBilateral_8u_C1R( src_start, step_src, dst1, step_dst1, roi,
                                kernel, pSpec );

    valsigma = 1.f;
    possigma = 1.f;
    stepInKernel = 1;
    ippiFilterBilateralInit_8u_C1R( ippiFilterBilateralGauss, kernel, valsigma,
                                    possigma, stepInKernel, pSpec );
    ippiFilterBilateral_8u_C1R( src_start, step_src, dst2, step_dst2, roi,
                                kernel, pSpec );


    valsigma = 16.f;
    possigma = 16.f;
    stepInKernel = 2;
    ippiFilterBilateralInit_8u_C1R( ippiFilterBilateralGauss, kernel, valsigma,
                                    possigma, stepInKernel, pSpec );
    ippiFilterBilateral_8u_C1R( src_start, step_src, dst3, step_dst3, roi,
                                kernel, pSpec );



    ippsFree( pSpec );

    printf("\n Src =");
    for ( i = 0, ptmp = src; i < height_src; i++ ) {
```

Example 9-2 Using Bilateral Filtering Functions

```
        printf("\n");
        for ( j = 0; j < width_src; j++ ) {
            printf("%3d ",ptmp[j]);
        }
        ptmp = (Ipp8u *)( (char *)ptmp + step_src );
    }

    printf("\n Dst1 =");
    for ( i = 0, ptmp = dst1; i < height_dst; i++ ) {
        printf("\n");
        for ( j = 0; j < width_dst; j++ ) {
            printf("%3d ",ptmp[j]);
        }
        ptmp = (Ipp8u *)( (char *)ptmp + step_dst1 );
    }
    printf("\n Dst2 =");
    for ( i = 0, ptmp = dst2; i < height_dst; i++ ) {
        printf("\n");
        for ( j = 0; j < width_dst; j++ ) {
            printf("%3d ",ptmp[j]);
        }
        ptmp = (Ipp8u *)( (char *)ptmp + step_dst2 );
    }
    printf("\n Dst3 =");
    for ( i = 0, ptmp = dst3; i < height_dst; i++ ) {
        printf("\n");
        for ( j = 0; j < width_dst; j++ ) {
            printf("%3d ",ptmp[j]);
        }
        ptmp = (Ipp8u *)( (char *)ptmp + step_dst3 );
    }

    ippiFree(dst3);
    ippiFree(dst2);
    ippiFree(dst1);
    ippsFree(src);
    return 0;
}
```

DecimateFilterRow, DecimateFilterColumn

Decimates an image by the rows or by the columns.

Syntax

```
IppStatus ippiDecimateFilterRow_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiFraction
    fraction);
IppStatus ippiDecimateFilterColumn_8u_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiFraction
    fraction);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source image ROI in pixels.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
fraction	Specifies how the decimating is performed. Possible values: `ippPolyphase_1_2`, `ippPolyphase_3_5`, `ippPolyphase_2_3`, `ippPolyphase_7_10`, `ippPolyphase_3_4`

Description

The functions `ippiDecimateFilterRow` and `ippiDecimateFilterColumn` are declared in the `ippi.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions perform decimating of the source image by the rows or by the columns respectively. The functions used the set of special internal polyphase filters. The parameter *fraction* specifies how the decimating is performed, for example, if the parameter is set to `ippPolyphase_3_5`, then each 5 pixels in the row (or column) of the source image give

3 pixels in the destination image, if the parameter is set to `ippPolyphase_1_2`, then each 2 pixels in the row (or column) of the source image give 1 pixel in the destination image, an so on.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`roiSize`* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *`srcStep`* or *`dstStep`* is less than or equal to 0.
`ippStsDecimateFractionErr`	Indicates an error condition if *`fraction`* has an illegal value.

Median Filters

The median filter functions perform non-linear filtering of a source image data.

These functions use either an arbitrary rectangular mask, or the following predefined masks of the `IppiMaskSize` type to filter an image:

`ippMskSize3x1`	Horizontal mask of length 3
`ippMskSize5x1`	Horizontal mask of length 5
`ippMskSize1x3`	Vertical mask of length 3
`ippMskSize3x3`	Square mask of size 3
`ippMskSize1x5`	Vertical mask of length 5
`ippMskSize5x5`	Square mask of size 5

The size of the neighborhood and coordinates of the anchor cell in the neighborhood depend on the *mask* mean. Table 9-2 lists the mask types with the corresponding neighborhood sizes and anchor cell coordinates. Note that in mask names the mask size is indicated in (*XY*) order.The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the mask

Table 9-2 Median Filter Mask, Neighborhood and Anchor Cell

Mask	Neighborhood Size		Anchor Cell
	Columns	Rows	
`ippMskSize3x1`	3	1	[1, 0]
`ippMskSize5x1`	5	1	[2, 0]
`ippMskSize1x3`	1	3	[0, 1]
`ippMskSize3x3`	3	3	[1, 1]
`ippMskSize1x5`	1	5	[0, 2]
`ippMskSize5x5`	5	5	[2, 2]

Median filters have the effect of removing the isolated intensity spikes and can be used to achieve noise reduction in an image.

For details on algorithms used in Intel IPP for median filtering, see [APMF].

FilterMedian

Filters an image using a median filter.

Syntax

```
IppStatus ippiFilterMedian_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	
8u_C4R	16u_C4R	16s_C4R	
8u_AC4R	16u_AC4R	16s_AC4R	

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

Description

The function `ippiFilterMedian` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders). The anchor cell is specified by its coordinates *anchor.x* and

anchor.y in the coordinate system associated with the top left corner of the kernel. The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

The Example 9-3 illustrates median filtering. Note that this filter removes noise and does not cut out signal brightness drops, as an averaging filter does.

Example 9-3 Applying the Median Filter to an Image

```
IppStatus filterMedian( void ) {
   IppiPoint anchor = { 1,1 };
   Ipp8u x[5*4], y[5*4]={0};
   IppiSize img={3,4}, roi={3,2}, mask={3,3};
   ippiSet_8u_C1R( 0x10, x, 5, img );
   /// raise the level of the signal,
   /// The edge will not be destroyed by the filter
   img.width = 5-3;
   ippiSet_8u_C1R( 0x40, x+3, 5, img );
   /// a spike, will be filtered
   x[5+1] = 0;
   /// roi is inside the image.
   /// Offset pointers to jump at the roi's beginning
   return ippiFilterMedian_8u_C1R( x+6, 5, y+6, 5, roi, mask, anchor );
}
The destination image y contains:
00 00 00 00 00
00 10 10 40 00
00 10 10 40 00
00 00 00 00 00
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if *maskSize* has a field with zero, negative, or even value.
ippStsAnchorErr	Indicates an error condition if *anchor* is outside the mask size.

FilterMedianHoriz

Filters an image using a horizontal median filter.

Syntax

```
IppStatusippiFilterMedianHoriz_<mod>(constIpp<datatype>*pSrc,intsrcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of `IppiMaskSize` type.

Description

The function `ippiFilterMedianHoriz` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The horizontal size of the neighborhood and the anchor cell coordinates depend on the *mask* mean, which can be either `ippMskSize3x1` or `ippMskSize5x1` (see Table 9-2). The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see Borders). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if *mask* has an illegal value.

FilterMedianVert

Filters an image using a vertical median filter.

Syntax

```
IppStatus ippiFilterMedianVert_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of IppiMaskSize type.

Description

The function `ippiFilterMedianVert` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The vertical size of the neighborhood and the anchor cell coordinates depend on the *mask* mean, which can be either `ippMskSize1x3` or `ippMskSize1x5` (see Table 9-2). The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see Borders). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

FilterMedianCross

Filters an image using a cross median filter.

Syntax

```
IppStatusippiFilterMedianCross_<mod>(constIpp<datatype>*pSrc,intsrcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod* :

`8u_C1R`	`16u_C1R`	`16s_C1R`
`8u_C3R`	`16u_C3R`	`16s_C3R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of `IppiMaskSize` type.

Description

The function `ippiFilterMedianCross` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the square mask of the predefined size, which can be either `ippMskSize3x3` or `ippMskSize5x5` (see Table 9-2). The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see Borders). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

FilterMedianWeightedCenter3x3

Filters an image using a median filter with weighted center pixel.

Syntax

```
IppStatus ippiFilterMedianWeightedCenter3x3_8u_C1R(const Ipp8u* pSrc,
    int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, int
    weight);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
weight	Weight of the pixel, must be an odd number.

Description

The function `ippiFilterMedianWeightedCenter3x3` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the fixed square mask of the size 3x3 with the anchor cell is the center cell of the mask. The parameter *weight* specifes the weight of the processed pixel, that is how many times its value will be included in the calculations. The value of this parameter should be odd. If it is even the function changes its value to the nearest less odd number and returns the warning message.

The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see Borders). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsWeightErr	Indicates an error condition if *weight* is less than or equal to 0.
ippStsEvenMedianWeight	Indicates a warning if *weight* has an even value.

FilterMedianColor

Filters an image using a color median filter.

Syntax

```
IppStatusippiFilterMedianColor_<mod>(constIpp<datatype>*pSrc,intsrcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod* :

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of IppiMaskSize type.

Description

The function `ippiFilterMedianColor` is declared in the `ippi.h` file.
When applied to a color image, the previously described median filtering functions process color planes of an image separately, and as a result any correlation between color components is lost. If you want to preserve this information, use the `ippiFilterMedianColor` function instead. For each input pixel, this function computes differences between red (R), green (G), and blue (B) color components of pixels in the *mask* neighborhood and the input pixel. The distance between the input pixel *i* and the neighborhood pixel *j* is formed as the sum of absolute values

```
abs (R(i)-R(j)) + abs (G(i)-G(j)) + abs (B(i)-B(j))
```

After scanning the entire neighborhood, the function sets the output value for pixel *i* as the value of the neighborhood pixel with the smallest distance to *i*.

The function `ippiFilterMedianColor` supports square masks of size either `ippMskSize3x3` or `ippMskSize5x5` and processes color images only. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

This function operates with ROI (see Regions of Interest in Intel IPP).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

General Linear Filters

These functions use a general rectangular kernel to filter an image. The kernel is a matrix of signed integers or single-precision real values. For each input pixel, the kernel is placed on the image in such a way that the fixed anchor cell within the kernel coincides with the input pixel. The anchor cell is usually a geometric center of the kernel, but can be skewed with respect to the geometric center.

A pointer to an array of kernel values is passed to filtering functions. These values are read in row-major order starting from the top left corner. There should be exactly *kernelSize.width* * *kernelSize.height* entries in this array. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the kernel.

The output value is computed as a sum of neighbor pixels' values, with kernel matrix elements used as weight factors. Note that summation formulas implement a convolution operation, which means that kernel coefficients are used in inverse order. Optionally, the output pixel values may be scaled. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Filter

Filters an image using a general rectangular kernel.

Syntax

Case 1: Operation on integer data

```
IppStatus ippiFilter_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, IppiSize kernelSize, IppiPoint anchor, int divisor);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Case 2: Operation on 32f data

```
IppStatus ippiFilter_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Case 3: Operation on 64f data

```
IppStatus ippiFilter_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp64f*
    pDst, int dstStep, IppiSize dstRoiSize, const Ipp64f* pKernel,
    IppiSize kernelSize, IppiPoint anchor, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the kernel values.
kernelSize	Size of the rectangular kernel in pixels.
anchor	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.
divisor	The integer value by which the computed result is divided (for operations on integer data only).
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilter` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function uses the general rectangular kernel of size *kernelSize* to filter an image ROI. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the kernel.

Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. In case of integer data, the result is divided by the fixed scaling factor *divisor*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

For function flavors that operate on integer data, the result value $Y_{i,j}$ for pixel $X_{i,j}$ inside image's ROI is computed according to the following formula:

$$Y_{i,\ j} = \frac{1}{divisor} \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} (X_{i+n-Q,\ j+m-P}) \times (K_{W-n-1,\ H-m-1})$$

where

$K_{n,m}$ are the kernel values;

H = *kernelSize.height* is the vertical size of the kernel;

W = *kernelSize.width* is the horizontal size of the kernel;

P = H - *anchor.y* - 1

Q = W - *anchor.x* - 1

Function flavors that accept floating-point data of `Ipp32f` and `Ipp64f` type use the same summation formula, but no scaling of the result is done.

Function flavor that operates on data of `Ipp64f` type requires the temporary working buffer. Its size must be computed using the function `ippiFilterGetBufSize` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pKernel* pointer is NULL.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* or *kernelSize* has a field with zero or negative value.
`ippStsDivisorErr`	Indicates an error condition if the *divisor* value is zero.
`ippStsStepErr`	Indicates an error condition if the *srcStep* is less than (*roiSize.width* + *kernelSize.width*)*sizeof(Ipp64f); or *dstStep* is less than *roiSize.width*sizeof(Ipp64f).

FilterGetBufSize

Computes the size of the working buffer

Syntax

```
IppStatus ippiFilterGetBufSize_64f_C1R(IppiSize kernelSize, int
    roiWidth, int* pSize);
```

Parameters

kernelSize	Size of the rectangular kernel in pixels.
roiWidth	Width of the image ROI in pixels.
pSize	Pointer to the size of the working buffer.

Description

The function `ippiFilterGetBufSize` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer *pSize* that is required for the function `ippiFilter` (flavor which operates on data of the `Ipp64f` type).

In some cases the function may return zero size of buffer.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *kernelSize* has a field with zero or negative value, or *roiWidth* is less than or equal to 0.

Filter32f

Filters an image with integer data using a floating-point rectangular kernel.

Syntax

```
IppStatus ippiFilter32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, IppiSize kernelSize, IppiPoint anchor);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the kernel values.
kernelSize	Size of the rectangular kernel in pixels.
anchor	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.

Description

The function `ippiFilter32f` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function uses the rectangular kernel of floating-point values to filter an image that consists of integer data. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom

right corner of the kernel. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. The summation formula for computing result values is similar to that used by `ippiFilter` function, but no scaling of the result is done. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pKernel* pointer is NULL.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* or *kernelSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one the step value is less than *roiSize.width*sizeof(pSrc)*numberOfChannels*.

Filter_Round16s, Filter_Round32s, Filter_Round32f

Perform filtering an image and rounding the result.

Syntax

Case 1: Filters with integer 16s kernel

```
IppStatus ippiFilter_Round16s_<mod>(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s* pKernel,
    IppiSize kernelSize, IppiPoint anchor, int divisor, IppRoundMode
    roundMode, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u_C1R
8u_C3R
8u_C4R
8u_AC4R
```

Case 2: Filters with integer 32s kernel

```
IppStatus ippiFilter_Round32s_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, const
    Ipp32s* pKernel, IppiSize kernelSize, IppiPoint anchor, int divisor,
    IppRoundMode roundMode, Ipp8u* pBuffer);
```

Supported values for *mod*:

16u_C1R	16s_C1R
16u_C3R	16s_C3R
16u_C4R	16s_C4R
16u_AC4R	16s_AC4R

Case 3: Filters with floating-point 32f kernel

```
IppStatus ippiFilter_Round32f_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, const
    Ipp32f* pKernel, IppiSize kernelSize, IppiPoint anchor, IppRoundMode
    roundMode, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the kernel values.
kernelSize	Size of the rectangular kernel in pixels.
anchor	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.
divisor	The integer value by which the computed result is divided (for operations on integer data only).
roundMode	Rounding mode, the following values are possible:

ippRndZero	Specifies that floating-point values must be truncated toward zero.
ippRndNear	Specifies that floating-point values must be rounded to the nearest even integer.
ippRndFinancial	Specifies that floating-point values must be rounded down to the nearest integer if decimal value is less than 0.5, or rounded up to the nearest integer if decimal value is equal or greater than 0.5.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilter_Round` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function uses the general rectangular kernel of size *kernelSize* to filter an image ROI. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the kernel. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel.

In case of integer kernel, the result is divided by the fixed scaling factor *divisor* (see formula above) and then is rounded using the rounding method specified by the parameter *roundMode*.

In case of floating-point kernel the result is rounded using the rounding method specified by the parameter *roundMode*.
To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

This function requires the temporary working buffer. Its size must be computed using the one of the functions `ippiFilterRoundGetBufSize16s`, `ippiFilterRoundGetBufSize32s`, `ippiFilterRoundGetBufSize32f` beforehand.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* or *kernelSize* has a field with zero or negative value.

ippStsDivisorErr	Indicates an error condition if the *divisor* value is zero, the execution is interrupted.
ippStsStepErr	Indicates an error condition if the *srcStep* is less than (*roiSize.width* + *kernelSize.width*-1)*sizeof(*pSrc*); or *dstStep* is less than *roiSize.width*sizeof(*pSrc*).
ippStsRoundModeNotSupportedErr	Indicates an error condition if the *roundMode* has an illegal value.

FilterRoundGetBufSize16s, FilterRoundGetBufSize32s, FilterRoundGetBufSize32f

Computes the size of the working buffer for filters with rounding mode.

Syntax

Case 1: Filters with integer 16s kernel

```
IppStatus, ippiFilterRoundGetBufSize16s_<mod>(IppiSize kernelSize, int
    roiWidth, int* pSize);
```

Supported values for *mod*:

```
8u_C1R
8u_C3R
8u_C4R
8u_AC4R
```

Case 2: Filters with integer 32s kernel

```
IppStatus, ippiFilterRoundGetBufSize32s_<mod>(IppiSize kernelSize, int
    roiWidth, int* pSize);
```

Supported values for *mod*:

```
16u_C1R      16s_C1R
16u_C3R      16s_C3R
16u_C4R      16s_C4R
16u_AC4R     16s_AC4R
```

Case 3: Filters with floating-point 32f kernel

```
IppStatus, ippiFilterRoundGetBufSize32f_<mod>(IppiSize kernelSize, int
    roiWidth, int* pSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the kernel values.
kernelSize	Size of the rectangular kernel in pixels.
anchor	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.
divisor	The integer value by which the computed result is divided (for operations on integer data only).
roundMode	Rounding mode, the following values are possible:

Description

The functions `ippiFilterRoundGetBufSize16s`, `ippiFilterRoundGetBufSize32s`, `ippiFilterRoundGetBufSize32f` are declared in the `ippi.h` file. They operates with ROI (see Regions of Interest in Intel IPP).

These functions compute the size of the working buffer *pSize* that is required for the function `ippiFilter_Round16s`, `ippiFilter_Round16s`, `ippiFilter_Round16s` respectively.

In some cases the functions may return zero size of buffer.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *`pSize`* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`kernelSize`* has a field with zero or negative value, or *`roiWidth`* is less than or equal to 0.

Separable Filters

Separable filters use a spatial kernel consisting of a single column (as in `FilterColumn` function) or a single row (as in `FilterRow` function) to filter the source image.

FilterColumn

Filters an image using a spatial kernel that consists of a single column.

Syntax

Case 1: Operation on integer data

```
IppStatus ippiFilterColumn_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32s* pKernel, int kernelSize, int yAnchor, int divisor);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Case 2: Operation on floating-point data

```
IppStatus ippiFilterColumn_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    int kernelSize, int yAnchor);
```

Supported values for *mod*:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the kernel values.
kernelSize	Size of the kernel in pixels.
yAnchor	Anchor cell specifying the kernel vertical alignment with respect to the position of the input pixel.
divisor	The integer value by which the computed result is divided (for operations on integer data only).

Description

The function `ippiFilterColumn` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function uses the vertical column kernel of size *kernelSize* to filter an image ROI. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *yAnchor*. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. In case of integer data, the result is divided by the fixed scaling factor *divisor*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

For function flavors that operate on integer data, the result value $Y_{i,j}$ for pixel $X_{i,j}$ inside image's ROI is computed according to the following formula:

$$Y_{i,\ j} = \frac{1}{divisor} \sum_{m=0}^{H-1} X_{i,\ j+m-P} \times K_{H-m-1}$$

where:
K_m are the kernel values;
H = *kernelSize* is the size of the vertical column kernel;
$P = H$ - *yAnchor* - 1

Function flavors that accept input data of `Ipp32f` type use the same summation formula, but no scaling of the result is done.

Example 9-4 show how to use the function `ippiFilterColumn_8u_C1R`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsDivisorErr	Indicates an error condition if the *divisor* value is zero.

Example 9-4 Using the Function ippiFilterColum

```
Ipp8u src[4*3] = {
       1, 2, 3, 4,
       1, 2, 3, 4,
       1, 2, 3, 4
    };
Ipp8u dst[4*3];
IppiSize srcRoi = { 4, 3 };
Ipp32s kern[] = { 1, -2, 1 };    // using "Sobel" kernel
int kernelSize = 3;
int yAnchor = 2;
int divisor = 2;

ippiFilterColumn_8u_C1R ( src, 4, dst, 4, srcRoi, kern, kernelSize,
yAnchor, divisor);

Result:
1 2 3 4
1 2 3 4       src
1 2 3 4

1 -2 1    kern

0   0   0   0
102 101 101 100      dst
0   0   0   0
```

FilterColumn32f

Filters an image with integer data using a floating-point column kernel.

Syntax

```
IppStatus ippiFilterColumn32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, int kernelSize, int yAnchor);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the column kernel values.
kernelSize	Size of the kernel in pixels.
yAnchor	Anchor cell specifying the kernel vertical alignment with respect to the position of the input pixel.

Description

The function `ippiFilterColumn32f` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function uses the vertical column kernel of floating-point values to filter an image that consists of integer data. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *yAnchor*. Note that kernel coefficients are used in inverse order. The sum

is written to the destination pixel. The summation formula for computing result values is similar to that used by `ippiFilterColumn` function, but no scaling of the result is done. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.

FilterRow

Filters an image using a spatial kernel that consists of a single row.

Syntax

Case 1: Operation on integer data

```
IppStatus ippiFilterRow_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32s* pKernel, int kernelSize, int xAnchor, int divisor);
```

Supported values for *mod*:

`8u_C1R`	`16u_C1R`	`16s_C1R`
`8u_C3R`	`16u_C3R`	`16s_C3R`
`8u_C4R`	`16u_C4R`	`16s_C4R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`

Case 2: Operation on floating-point data

```
IppStatus ippiFilterRow_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int kernelSize,
    int xAnchor);
```

Supported values for *mod*:

`32f_C1R`

`32f_C3R`

`32f_C4R`

32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the kernel values.
kernelSize	Size of the kernel in pixels.
xAnchor	Anchor cell specifying the kernel horizontal alignment with respect to the position of the input pixel.
divisor	The integer value by which the computed result is divided (for operations on integer data only).

Description

The function `ippiFilterRow` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function uses the horizontal row kernel of size *kernelSize* to filter an image. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *xAnchor*. Note that kernel coefficients are used in inverse order.

The sum is written to the destination pixel. In case of integer data, the result is divided by the fixed scaling factor *divisor*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

For function flavors that operate on integer data, the result value $Y_{i,j}$ for pixel $X_{i,j}$ inside image's ROI is computed according to the following formula:

$$Y_{i,\ j} = \frac{1}{divisor} \sum_{n=0}^{W-1} X_{i+n-Q,\ j} \times K_{W-n-1}$$

where

K_n are the kernel values;

W = *kernelSize* is the size of the horizontal row kernel;

Q = *W* - *xAnchor* - 1

Function flavors that accept input data of Ipp32f type use the same summation formula, but no scaling of the result is done.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if *pSrc*, *pDst*, or *pKernel* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsDivisorErr	Indicates an error condition if the *divisor* value is zero.

FilterRow32f

Filters an image with integer data using a floating-point row kernel.

Syntax

```
IppStatus ippiFilterRow32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, int kernelSize, int xAnchor);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R
8u_AC4R	16u_AC4R	16s_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the row kernel values.
kernelSize	Size of the kernel in pixels.
xAnchor	Anchor cell specifying the kernel horizontal alignment with respect to the position of the input pixel.

Description

The function `ippiFilterRow32f` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function uses the horizontal row kernel of floating-point values to filter an image that consists of integer data. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *xAnchor*. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. The summation formula for computing result values is similar to that used by `ippiFilterRow` function, but no scaling of the result is done. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.

FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize_Low

Compute the size of working buffer for the row filter.

Syntax

```
IppStatus ippiFilterRowBorderPipelineGetBufferSize_<mod>(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

Supported values for *mod*:

```
8u16s_C1R    16s_C1R    32f_C1R
8u16s_C3R    16s_C3R    32f_C3R
```

```
IppStatus ippiFilterRowBorderPipelineGetBufferSize_Low_<mod>(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

Supported values for *mod*:

```
16s_C1R
16s_C3R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
kernelSize	Size of the kernel in pixels.
pBufferSize	Pointer to the computed size of the buffer.

Description

The functions `ippiFilterRowBorderPipelineGetBufferSize` and `ippiFilterRowBorderPipelineGetBufferSize_Low` are declared in the `ippcv.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions compute the size of the working buffer required for the functions `ippiFilterRowBorderPipeline` and `ippiFilterRowBorderPipeline_Low` respectively. The buffer with the length *pBufferSize*[0] can be used to filter images with width equal to or less than *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the pointer *pBufferSize* is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *maskSize* has a field with zero or negative value, or if *roiWidth* is less than 1.

FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize_Low

Compute the size of working buffer for the column filter.

Syntax

```
IppStatus ippiFilterColumnPipelineGetBufferSize_<mod>(IppiSize roiSize,
    int kernelSize, int* pBufferSize);
```

Supported values for *mod*:

```
16s_C1R      16s8u_C1R    16s8s_C1R    32f_C1R
16s_C3R      16s8u_C3R    16s8s_C3R    32f_C3R
```

```
IppStatus ippiFilterColumnPipelineGetBufferSize_Low_<mod>(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

Supported values for *mod*:

```
16s_C1R
16s_C3R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
kernelSize	Size of the kernel in pixels.
pBufferSize	Pointer to the computed size of the buffer.

Description

The functions `ippiFilterColumnPipelineGetBufferSize` and `ippiFilterColumnPipelineGetBufferSize_Low` are declared in the `ippcv.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions compute the size of the working buffer required for the functions `ippiFilterColumnPipeline` and `ippiFilterColumnPipeline_Low` respectively. The buffer with the length *pBufferSize*[0] can be used to filter images with width equal to or less than *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if the pointer *pBufferSize* is NULL.
ippStsSizeErr	Indicates an error condition if *maskSize* has a field with zero or negative value, or if *roiWidth* is less than 1.

FilterRowBorderPipeline, FilterRowBorderPipeline_Low

Apply the filter with border to image rows.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>*
    pSrc, int srcStep, Ipp<dstDatatype>** ppDst, IppiSize roiSize, const
    Ipp<dstDatatype>* pKernel, int kernelSize, int xAnchor,
    IppiBorderType borderType, Ipp<srcDatatype> borderValue, int divisor,
    Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u16s_C1R    16s_C1R
```

```
IppStatus ippiFilterRowBorderPipeline_Low_16s_C1R(const Ipp16s* pSrc,
    int srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel,
    int kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s
    borderValue, int divisor, Ipp8u* pBuffer);
```

Case 2: Operation on one-channel floating point data

```
IppStatus ippiFilterRowBorderPipeline_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f
    borderValue, Ipp8u* pBuffer);
```

Case 3: Operation on three-channel integer data

```
IppStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>*
    pSrc, int srcStep, Ipp<dstDatatype>** ppDst, IppiSize roiSize, const
    Ipp<dstDatatype>* pKernel, int kernelSize, int xAnchor,
    IppiBorderType borderType, Ipp<srcDatatype> borderValue[3], int
    divisor, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u16s_C3R    16s_C3R
```

```
IppStatus ippiFilterRowBorderPipeline_Low_16s_C3R(const Ipp16s* pSrc,
    int srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel,
    int kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s
    borderValue[3], int divisor, Ipp8u* pBuffer);
```

Case 4: Operation on three-channel floating point data

```
IppStatus ippiFilterRowBorderPipeline_32f_C3R(const Ipp32f* pSrc, int
    srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f
    borderValue[3], Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
ppDst	Double pointer to the destination inage ROI.
roiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the row kernel values.
kernelSize	Size of the kernel in pixels.
xAnchor	Anchor value specifying the kernel row alignment with respect to the position of the input pixel.
borderType	Type of border (see Borders); following values are possible: `ippBorderZero` Values of all border pixel are set to zero. `ippBorderConst` Values of all border pixels are set to constant. `ippBorderRepl` Replicated border is used. `ippBorderWrap` Wrapped border is used `ippBorderMirror` Mirrored border is used `ippBorderMirrorR` Mirrored border with replication is used
borderValue	The constant value (constant vector in case of three-channel data) to assign to the pixels in the constant border (not applicable for other border's type).
divisor	Value by which the computed result is divided (for operations on integer data only).
pBuffer	Pointer to the working buffer.

Description

The functions `ippiFilterRowBorderPipeline` and `ippiFilterRowBorderPipeline_Low` are declared in the `ippcv.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

The function `ippiFilterRowBorderPipeline_Low` performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the horizontal row filter of the separable convolution kernel to the source image *pSrc*. The filter coefficients are placed in the reversed order. For integer data:

$$ppDst[i][j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

and for floating point data:

$$ppDst[i][j] = \sum_{k=0}^{kernelSize-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

Here `j = 0, ...` *roiSize.width*`-1`, `i=0,...`*roiSize.height*`-1`. The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel *pSrc*[*i*, *l*] $l \notin [0, roiSize.width-1]$ are set in accordance with the specified parameters *borderType* and *borderValue*.
This function can be used to organize the separable convolution as a step of the image processing pipeline (see Example 9-5).

The functions requires the external buffer *pBuffer*, its size should be previously computed by the functions `ippiFilterRowBorderPipelineGetBufferSize` and `ippiFilterRowBorderPipelineGetBufferSize_Low` respectively

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsAnchorErr	Indicates an error condition if *xAnchor* has a wrong value.
ippStsBorderErr	Indicates an error condition if *borderType* has a wrong value.
ippStsBadArgErr	Indicates an error condition if *divisor* is equal to 0.

FilterColumnPipeline, FilterColumnPipeline_Low

Apply the filter to image columns.

Syntax

Case 1: Operation on integer data

```
IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<srcDatatype>** ppSrc,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, const
    Ipp<srcDatatype>* pKernel, int kernelSize, int divisor, Ipp8u*
    pBuffer);
```

Supported values for *mod*:

16s_C1R	16s8u_C1R	16s8s_C1R
16s_C3R	16s8u_C3R	16s8s_C3R

```
IppStatus ippiFilterColumnPipeline_Low_16s_C1R(const Ipp16s** ppSrc,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s*
    pKernel, int kernelSize, int divisor, Ipp8u* pBuffer);
IppStatus ippiFilterColumnPipeline_Low_16s_C3R(const Ipp16s** ppSrc,
    Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s*
    pKernel, int kernelSize, int divisor, Ipp8u* pBuffer);
```

Case 2: Operation on floating-point data

```
IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<datatype>** ppSrc,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, const
    Ipp<datatype>* pKernel, int kernelSize, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
```

Parameters

ppSrc	Double pointer to the source image ROI.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
pKernel	Pointer to the row kernel values.
kernelSize	Size of the kernel in pixels.
divisor	Value by which the computed result is divided (for operations on integer data only).
pBuffer	Pointer to the working buffer.

Description

The functions `ippiFilterColumnPipeline` and `ippiFilterColumnPipeline_Low` are declared in the `ippcv.h` file. The operates with ROI (see Regions of Interest in Intel IPP).

The function `ippiFilterColumnPipeline_Low` performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the column filter of the separable convolution kernel to the source image *pSrc*. The filter coefficients are placed in the reversed order. For integer data:

$$pDst[i, j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i+k, j] \cdot pKernel[k]$$

and for floating point data:

$$pDst[i, j] = \sum_{k=0}^{kernelSize-1} pSrc[i+k, j] \cdot pKernel[k]$$

Here `j = 0, ...` *dstRoiSize.width*`-1, i=0,...`*dstRoiSize.height*`-1`.

The size of the source image is

(*dstRoiSize.height* + *kernelSize* - 1) * *dstRoiSize.width*.

The functions requires the external buffer *pBuffer*, its size should be previously computed by the functions ippiFilterColumnPipelineGetBufferSize and ippiFilterColumnPipelineGetBufferSize_Low respectively.

These functions can be used to organize the separable convolution as a step of image processing pipeline (see Example 9-5).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error condition if *divisor* is equal to 0.

Example 9-5 Separable Convolution 3x3 by One Row

```
void Separable_3x3(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep,
                   IppiSize roiSize, Ipp16s* pKerX, Ipp16s* pKerY) {
   Ipp16s **get, *dst=pDst;
   const Ipp16s *src=pSrc;
   IppiSize roi;
   int todo=roiSize.height,sizeRow,sizeCol,bufLen;
   int mStep=(roiSize.width+7)&(~7),sStep=srcStep>>1,dStep=dstStep>>1;
   Ipp8u *pBufRow, *pBufCol;
   ippiFilterRowBorderPipelineGetBufferSize_Low_16s_C1R(roiSize,3,&sizeRow);
   ippiFilterColumnPipelineGetBufferSize_Low_16s_C1R(roiSize,3,&sizeCol);
   bufLen = mStep*3*sizeof(Ipp16s)+4*sizeof(Ipp16s*);
   pBufRow = ippsMalloc_8u(sizeRow);
   pBufCol = ippsMalloc_8u(sizeCol);
   get = (Ipp16s**)ippsMalloc_8u(bufLen);
   get[0]=get[1]=(Ipp16s*)(get+4);
   get[2]=get[1]+mStep;
   get[3]=get[2]+mStep;
   roi.width  = roiSize.width;
   roi.height = 1;
   ippiFilterRowBorderPipeline_Low_16s_C1R(src, srcStep, get, roi, pKerX,
      3, 1, ippBorderRepl, 0, 1, pBufRow);
```

Example 9-5 Separable Convolution 3x3 by One Row (continued)

```
   if (--todo) {
      get[2] = get[0];
   } else {
      get[2] = get[0] + mStep; get[3] = get[2] + mStep;
      for (; todo>0; src+=sStep, dst+=dStep, todo--) {
        ippiFilterRowBorderPipeline_Low_16s_C1R(src, srcStep, get+2, roi, pKerX,
            3, 1, ippBorderRepl, 0, 1, pBufRow);
         ippiFilterColumnPipeline_Low_16s_C1R(get, dst, dstStep, roi, pKerY,
            3, 1, pBufCol);
         get[0] = get[1]; get[1] = get[2]; get[2] = get[3]; get[3] = get[0];
      }
   }
   ippiFilterColumnPipeline_Low_16s_C1R(get, dst, dstStep, roi, pKerY,
      3, 1, pBufCol);

   ippsFree(pBufRow);
   ippsFree(pBufCol);
   ippsFree(get);
}
```

DotProdCol

Calculates the dot product of taps vector and columns of the specified set of rows.

Syntax

```
IppStatus ippiDotProdCol_32f_L2(const Ipp32f* const ppSrcRow[], const
    Ipp32f* pTaps, int tapsLen, Ipp32f* pDst, int width);
```

Parameters

ppSrcRow	Pointer to the set of rows.
pTaps	Pointer to the taps vector.
tapsLen	Length of taps vector, is equal to the number of rows.
pDst	Pointer to the destination row.
width	Width of the source and destination rows.

Description

The function `ippiDotProdCol` is declared in the `ippi.h` file.

This function calculates the dot product of filter taps vector *pTaps* and columns of the specified set of the rows *ppSrcRow*. The computations are performed in accordance with the following formula:

$$pDst[i] = \sum_{j=0}^{tapsLen-1} ppSrcRow[j][i] \cdot pTaps[j]$$

It is useful for external vertical filtering pipeline implementation.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *width* is less than or equal to 0.

Wiener Filters

Intel IPP functions described in this section perform adaptive noise-removal filtering of an image using Wiener filter [Lim90]. The adaptive filter is more selective than a comparable linear filter in preserving edges and other high frequency parts of an image. Wiener filters are commonly used in image processing applications to remove additive noise from degraded images, to restore a blurry image, and in similar operations.

These functions use a pixel-wise adaptive Wiener method based on statistics estimated from a local neighborhood (mask) of arbitrary size for each pixel.

FilterWienerGetBufferSize

Computes the size of the external buffer for ippiFilterWiener *function.*

Syntax

```
IppStatus ippiFilterWienerGetBufferSize(IppiSize dstRoiSize,
    IppiSize maskSize, int channels, int* pBufferSize);
```

Parameters

dstRoiSize	Size of the destination ROI in pixels.
maskSize	Size of the mask in pixels.
channels	Number of channels in the image.
pBufferSize	Pointer to the computed value of the external buffer size.

Description

The function `ippiFilterWienerGetBufferSize` is declared in the `ippi.h` file. This function computes the size in bytes of an external memory buffer that is required for the function `ippiFilterWiener`, and stores the result in the *pBufferSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if one of the fields of *dstRoiSize* has zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if one of the fields of *maskSize* has value less than or equal to 1.
`ippStsNumChannelsErr`	Indicates an error condition if *channels* is not 1, 3 or 4.

FilterWiener

Filters an image using the Wiener algorithm.

Syntax

Case 1: Operation on one-channel images

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor, Ipp32f noise[1], Ipp8u*
    pBuffer);
```

Supported values for *mod* :

```
8u_C1R        16s_C1R       32f_C1R
```

Case 2: Operation on multi-channel images

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp32f noise[3], Ipp8u* pBuffer);
```

Supported values for *mod* :

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp32f noise[4], Ipp8u* pBuffer);
```

Supported values for *mod* :

8u_C4R	16s_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
noise	Noise level value or array of the noise level values in case of multi-channel image. This value must be in the range [0,1].
pBuffer	Pointer to the external work buffer.

Description

The function `ippiFilterWiener` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function performs adaptive filtering of the image degraded by constant power additive noise. For each pixel of the input image *pSrc,* the function estimates the local image mean μ and variance σ in the rectangular neighborhood (mask) of size *maskSize* with anchor cell *anchor* centered on the pixel. The anchor cell is

specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the mask.
The following formulas are used in computations:

$$\mu_{i,\ j} = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,\ n}$$

$$\sigma_{i,\ j}{}^{2} = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,\ n}^{2} - \mu_{i,\ j}^{2}$$

Here $\mu_{i,j}$ and $\sigma_{i,j}$ stand for local mean and variance for pixel $X_{i,j}$, respectively, and H,W are the vertical and horizontal sizes of the mask, respectively.

The corresponding value for the output pixel $Y_{i,j}$ is computed as:

$$Y_{i,\ j} = \mu_{i,\ j} + \frac{\sigma_{i,\ j}^{2} - \nu^{2}}{\sigma^{2}} \cdot [X_{i,\ j} - \mu_{i,\ j}]$$

and stored in the *pDst*. Here ν^2 is the noise variance, specified for each channel by the noise level parameter *noise*. If this parameter is not defined (*noise* = 0), then the function estimates the noise level by averaging through the image of all local variances $\sigma_{i,j}$, and stores the corresponding values in the *noise* for further use.

The function `ippiFilterWiener` uses the external work buffer *pBuffer*, which must be allocated before the function call. To determine the required buffer size, the function `ippiFilterWienerGetBufferSize` can be used.

Example 9-6 shows how to use the function `ippiFilterWiener_32f_C1R`, result is presentes on Figure 9-2.

Return Values

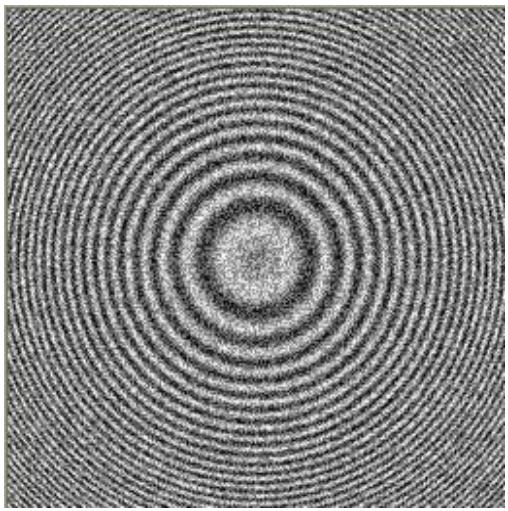
`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if one of the fields of *dstRoiSize* has zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if one of the fields of *maskSize* has value less than or equal to 1.
ippStsNoiseRangeErr	Indicates an error condition if one of the *noise* values is less than 0 or greater than 1.

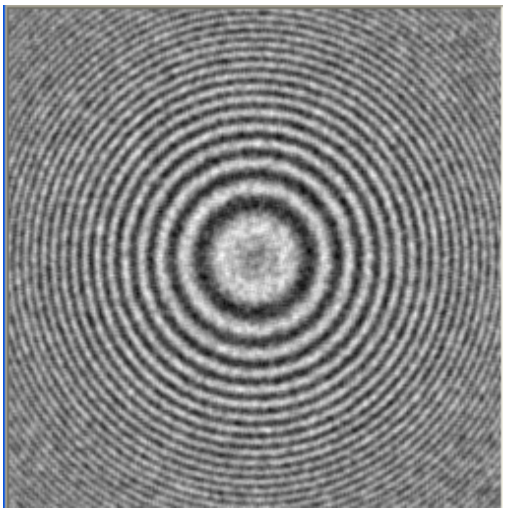
Example 9-6 Using the Wiener Filter function

```
void func_wiener()
{
    Ipp32f pSrc[256*256];
    Ipp32f pDst[256*256];
    int srcStep = 256*sizeof(Ipp32f);
    int dstStep = 256*sizeof(Ipp32f);
    IppiSize roiSize = {256, 256};
    IppiSize dstRoiSize = {256, 256};
    IppiSize maskSize = {3, 3};
    int pBufferSize;
    int channels = 1;
    unsigned int pSeed = 3;
    IppiPoint anchor = {1, 1};
    Ipp32f noise[1] = {0.5};

    ippiImageJaehne_32f_C1R(pSrc, srcStep, roiSize); // source image
    //source image + RandUniform noise
ippiAddRandUniform_Direct_32f_C1IR(pSrc, srcStep, roiSize, 0, 1, &pSeed);

    ippiFilterWienerGetBufferSize( dstRoiSize, maskSize, channels,
&pBufferSize);

    Ipp8u* pBuffer = ippsMalloc_8u(pBufferSize);
    ippiFilterWiener_32f_C1R( pSrc, srcStep, pDst, dstStep, dstRoiSize,
maskSize, anchor, noise, pBuffer);
    ippsFree(pBuffer);
}
```

Figure 9-2 Applying the function `ippiFilterWiener`



source image + noise (generated by the function ippiAddRandom)



destination image

Convolution

Intel IPP functions described in this section perform two-dimensional finite linear convolution operation between two source images and write the result into the destination image. Convolution is used to perform many common image processing operations including sharpening, blurring, noise reduction, embossing, and edge enhancement.
For convenience, in this section we shall represent any digital image f as a matrix with M_f columns and N_f rows that contains pixel values $f[i,j]$, $0 \le i < M_f$, $0 \le j < N_f$.

ConvFull

Performs a full convolution of two images.

Syntax

Case 1: Operation on integer data

```
IppStatus ippiConvFull_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize src2Size, Ipp<datatype>* pDst, int dstStep, int divisor);
```

Supported values for *mod*:

```
8u_C1R      16s_C1R
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

Case 2: Operation on floating-point data

```
IppStatus ippiConvFull_<mod>(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step,
    IppiSize src2Size, Ipp32f* pDst, int dstStep);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_AC4R
```

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.

src1Size, src2Size	Sizes in pixels of the source images.
pDst	Pointer to the destination buffer ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
divisor	The integer value by which the computed result is divided (for operations on integer data only).

Description

The function `ippiConvFull` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function performs full two-dimensional finite linear convolution operation between two source images pointed to by *pSrc1* and *pSrc2*. If we denote the first source image as a matrix *f* of size M_f by N_f and the second source image as a matrix *g* of size M_g by N_g, then the destination image *h* obtained as a result of function operation will have size M_h by N_h, where $M_h = M_f + M_g - 1$ and $N_h = N_f + N_g - 1$.

The function `ippiConvFull` implements the following equation to compute values *h* [*i,j*] of the destination image:

$$h[i, j] = \frac{1}{divisor} \sum_{l=0}^{N_h-1} \sum_{k=0}^{M_h-1} f[k, l] \times g[i-k, j-l] \quad ,$$

where $0 \le i < M_h$, $0 \le j < N_h$ and

$$f[k, l] = \begin{cases} f[k, l], & 0 \le k < M_f; \quad 0 \le l < N_f \\ 0 & , otherwise \end{cases}$$

$$g[i-k, j-l] = \begin{cases} g[i-k, j-l], & 0 \le i-k < M_g; \quad 0 \le j-l < N_g \\ 0 & , otherwise \end{cases}$$

Function flavors that accept input data of `Ipp32f` type use the same summation formula, but no scaling of the result is done (*divisor* = 1 is assumed).

To illustrate the function operation, for the source images *f, g* of size 3 x 5 represented as

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad g = f$$

the resulting convolution image *h* is of size 5 x 9 and contains the following data:

$$h = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 2 & 2 & 0 & 0 \\ 3 & 4 & 6 & 4 & 2 \\ 2 & 2 & 4 & 2 & 2 \\ 3 & 6 & 11 & 6 & 3 \\ 2 & 2 & 4 & 2 & 2 \\ 2 & 4 & 6 & 4 & 3 \\ 0 & 0 & 2 & 2 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

Example 9-7 shows how to use the function `ippiConvFull_16s_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc1*, *pSrc2*, or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *src1Size* or *src2Size* has zero or negative value.
`ippStsStepErr`	Indicates an error condition if *src1Step*, *src2Step*, or *dstStep* has zero or negative value.
`ippStsDivisorErr`	Indicates an error condition if *divisor* has zero value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation failed.

ConvValid

Performs a valid convolution of two images.

Syntax

Case 1: Operation on integer data

```
IppStatus ippiConvValid_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize src2Size, Ipp<datatype>* pDst, int dstStep, int divisor);
```

Supported values for *mod* :

```
8u_C1R      16s_C1R
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

Case 2: Operation on floating-point data

```
IppStatus ippiConvValid_<mod>(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step,
    IppiSize src2Size, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* :

```
32f_C1R
32f_C3R
32f_AC4R
```

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
src1Size, src2Size	Sizes in pixels of the source images.
pDst	Pointer to the destination buffer ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
divisor	The integer value by which the computed result is divided (for operations on integer data only).

Description

The function `ippiConvValid` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function performs valid two-dimensional finite linear convolution operation between two source images pointed to by *pSrc1* and *pSrc2*.
If we denote the first source image as a matrix *f* of size M_f by N_f and the second source image as a matrix *g* of size M_g by N_g, then the destination image *h* obtained as a result of function operation will have size M_h by N_h, where $M_h = |M_f - M_g| + 1$ and $N_h = |N_f - N_g| + 1$.

The function `ippiConvFull` implements the following equation to compute values *h* [*i,j*] of the destination image:

$$h[i, j] = \frac{1}{divisor} \sum_{l=0}^{N_g-1} \sum_{k=0}^{M_g-1} f[i+k, j+l] \times g[M_g-k-1, N_g-l-1],$$

where $0 \le i < M_h$, $0 \le j < N_h$.

We assume here that $M_f \ge M_g$ and $N_f \ge N_g$. In case when $M_f < M_g$ and $N_f < N_g$, the subscript index *g* in this equation must be replaced with index *f*. For any other combination of source image sizes, the function `ippiConvValid` performs no operation.

Note that the above formula gives the same result as in the case of `ippiConvFull` function, but produces only that part of the convolution image which is computed without using zero-padded values.

Function flavors that accept input data of `Ipp32f` type use the same summation formula, but no scaling of the result is done (*divisor* = 1 is assumed).

Example 9-7 shows how to use the function `ippiConvValid_16s_C1R`.

To illustrate the function operation, for the source images *f, g* of size 3 x 5 represented as

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad g = f,$$

the resulting convolution image *h* is of size 1 x 1 and contains the following data:

$$h = \begin{bmatrix} 11 \end{bmatrix}.$$

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if *pSrc1*, *pSrc2*, or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *src1Size* or *src2Size* has zero or negative value.
ippStsStepErr	Indicates an error condition if *src1Step*, *src2Step*, or *dstStep* has zero or negative value.
ippStsDivisorErr	Indicates an error condition if *divisor* has zero value.
ippStsMemAllocErr	Indicates an error condition if memory allocation failed.

Example 9-7 Using the Convolution Functions

```
Ipp8u src1[8*4]={ 0, 1, 2, 3, 4, 5, 6, 7,
                  0, 1, 2, 3, 4, 5, 6, 7,
                  7, 6, 5, 4, 3, 2, 1, 0,
                  7, 6, 5, 4, 3, 2, 1, 0};
Ipp8u src2[3*3];
Ipp8u dst1[6*6];
Ipp8u dst2[2*2];
IppiSize src1Size = { 4, 4 };
IppiSize src2Size = { 3, 3 };
int divisor = 2;
int sign = 1;

ippiGenSobelKernel_16s ( src2, 9, 8, sign);    // using "Sobel" kernel

ippiConvFull_16s_C1R ( src1, 8*sizeof(Ipp16s), src1Size, src2,
                 3*sizeof(Ipp16s), src2Size, dst1, 6*sizeof(Ipp16s), 2 );
ippiConvValid_16s_C1R ( src1, 8*sizeof(Ipp16s),  src1Size,  src2,
                 3*sizeof(Ipp16s), src2Size, dst2, 2*sizeof(Ipp16s), 2 );

Result:
0 1 2 3
0 1 2 3
7 6 5 4     src1
7 6 5 4

  1  -8   28
-56  70  -56     src2
 28  -8    1


   0    0   -3    8   16   42
   0  -28  -24  -34   65  -42
```

Example 9-7 Using the Convolution Functions

```
   4  -39   80   58   92  -26
-192   66  -26   -4   43  -54        Output of ippiConvFull  (dst1)
 -98  133  -76  -66  -14 -110
  98   56   50   39  -14    2

        80  59    Output of ippiConvValid (dst2)
       -26  -5
```

Deconvolution

Functions described in this section perform image deconvolution. They can be used for restoring the degraded image, in particular image that was obtained by applying the convolution operation with known kernel. The Intel IPP functions implements two methods: Fourier deconvolution (noniterative method) [see for example, Puetter2005], and Richardson-Lucy method (iterative method) [Richardson72]. Border pixels of a source image are restored before deconvolution.

DeconvFFTInitAlloc

Allocates and initializes state structure for FFT deconvolution.

Syntax

```
IppStatus ippiDeconvFFTInitAlloc_32f_C1R(IppiDeconvFFTState_32f_C1R**
    ppDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int
    FFTorder, Ipp32f threshold);
IppStatus ippiDeconvFFTInitAlloc_32f_C3R(IppiDeconvFFTState_32f_C3R**
    ppDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int
    FFTorder, Ipp32f threshold);
```

Parameters

ppDeconvFFTState	Double pointer to the FFT deconvolution state structure.
pKernel	Pointer to the kernel array.
kernelSize	Size of the kernel.
FFTorder	Order of the created FFT state structure.
threshold	Value of the threshold level (to except dividing by zero).

Description

The function `ippiDeconvFFTInitAlloc` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory, initializes the deconvolution state structure and returns the pointer *ppDeconvFFTState* to it. This structure is used by the function `ippiDeconvFFT` that performs deconvolution of the source image using FFT.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *kernelSize* is less than or equal to 0, or if *kernelSize* is greater than $2^{FFTorder}$.
ippStsBadArgErr	Indicates an error condition if *threshold* is less than or equal to 0.

DeconvFFTFree

Frees memory allocated for the FFT deconvolution state structure.

Syntax

```
IppStatus ippiDeconvFFTFree_32f_C1R(IppiDeconvFFTState_32f_C1R*
    pDeconvFFTState)
IppStatus ippiDeconvFFTFree_32f_C3R(IppiDeconvFFTState_32f_C3R*
    pDeconvFFTState);
```

Parameters

pDeconvFFTState	Pointer to the FFT deconvolution state structure.

Description

The function ippiDeconvFFTFree is declared in the ippi.h file. This function frees memory allocated by the function ippiDeconvFFTInitAlloc for the FFT deconvolution state structure *pDeconvFFTState*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pDeconvFFTState* is NULL.

DeconvFFT

Performs FFT deconvolution of an image.

Syntax

```
IppStatus ippiDeconvFFT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C1R*
    pDeconvFFTState);
IppStatus ippiDeconvFFT_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C3R*
    pDeconvFFTState);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
pDeconvFFTState	Pointer to the FFT deconvolution state structure.

Description

The function `ippiDeconvFFT` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs deconvolution of the source image *pSrc* using FFT with parameters specified in the FFT deconvolution state structure *pDeconvFFTState* and stores results to the destination image *pDst*. The FFT deconvolution state structure must be initialized by calling the function `ippiDeconvFFTInitAlloc` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

DeconvLRInitAlloc

Allocates and initializes state structure for LR deconvolution.

Syntax

```
IppStatus ippiDeconvLRInitAlloc_32f_C1R(IppiDeconvLR_32f_C1R**
    ppDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi,
    Ipp32f threshold);
IppStatus ippiDeconvLRInitAlloc_32f_C3R(IppiDeconvLR_32f_C3R**
    ppDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi,
    Ipp32f threshold);
```

Parameters

ppDeconvLR	Double pointer to the LR deconvolution state structure.
pKernel	Pointer to the kernel array.
kernelSize	Size of the kernel.
maxRoi	Maximum size of the image ROI in pixels.
threshold	Value of the threshold level (to except dividing by zero).

Description

The function `ippiDeconvLRInitAlloc` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory, initializes the deconvolution state structure and returns the pointer *ppDeconvLR* to it. This structure is used by the function `ippiDeconvLR` that performs deconvolution of the source image using Lucy-Richardson algorithm.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *kernelSize* is less than or equal to 0, or if *kernelSize* is greater than *maxRoi.width* or *maxRoi.height*, or if any field of the *maxRoi* is less than or equal to 0.
`ippStsBadArgErr`	Indicates an error condition if *threshold* is less than or equal to 0.

DeconvLRFree

Frees memory allocated for the LR deconvolution state structure.

Syntax

```
IppStatus ippiDeconvLRFree_32f_C1R(IppiDeconvLR_32f_C1R* pDeconvLR)
IppStatus ippiDeconvLRFree_32f_C3R(IppiDeconvLR_32f_C3R* pDeconvLR);
```

Parameters

pDeconvLR	Pointer to the LR deconvolution state structure.

Description

The function `ippiDeconvLRFree` is declared in the `ippi.h` file. This function frees memory allocated by the function `ippiDeconvLRInitAlloc` for the LR deconvolution state structure *pDeconvLR*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDeconvLR* is `NULL`.

DeconvLR

Performs LR deconvolution of an image.

Syntax

```
IppStatus ippiDeconvLR_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, int numIter,
    IppiDeconvLR_32f_C1R* pDeconvLR);
IppStatus ippiDeconvLR_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, int numIter,
    IppiDeconvLR_32f_C3R* pDeconvLR);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
numIter	Number of algorithm iterations.
pDeconvLR	Pointer to the LR deconvolution state structure.

Description

The function `ippiDeconvLR` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs deconvolution of the source image *pSrc* using Lucy-Richardson algorithm with parameters specified in the state structure *pDeconvLR* and stores results to the destination image *pDst*. The Lucy-Richardson deconvolution state structure must be initialized by calling the function `ippiDeconvLRInitAlloc` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roi.width* or *roi.height* is less than or equal to 0, or if *roi.width* is greater than (*maxRoi.width - kernelSize*), or *roi.height* is greater than (*maxRoi.height - kernelSize*).
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * *<pixelSize>*
`ippStsNotEvenStepErr`	Indicates an error condition if steps for floating-point images are not divisible by 4.
`ippStsBadArgErr`	Indicates an error condition if *numIter* is less than or equal to 0.

Fixed Filters

The fixed filter functions perform linear filtering of a source image using one of the predefined convolution kernels. The supported fixed filters and their respective kernel sizes are listed in the following table:

Table 9-3 Types of the Fixed Filter Functions

Fixed Filter Type	Kernel Size
Horizontal Prewitt operator	3x3
Vertical Prewitt operator	3x3
Horizontal Scharr operator	3x3
Vertical Scharr operator	3x3
Horizontal Sobel operator	3x3 or 5x5
Vertical Sobel operator	3x3 or 5x5
Second derivative horizontal Sobel operator	3x3 or 5x5
Second derivative vertical Sobel operator	3x3 or 5x5
Second cross derivative Sobel operator	3x3 or 5x5
Horizontal Roberts operator	3x3
Vertical Roberts operator	3x3
Laplacian highpass filter	3x3 or 5x5
Gaussian lowpass filter	3x3 or 5x5
Highpass filter	3x3 or 5x5
Lowpass filter	3x3 or 5x5
Sharpening filter	3x3

Using fixed filter functions with predefined kernels is more efficient as it eliminates the need to create the convolution kernel in your application program.

NOTE. For all fixed filter functions, to ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

NOTE. The anchor cell is the center cell of the kernel for all fixed filters.

FilterPrewittHoriz

Filters an image using a horizontal Prewitt kernel.

Syntax

```
IppStatus ippiFilterPrewittHoriz_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiFilterPrewittHoriz` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a horizontal Prewitt operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
 1   1   1
 0   0   0
-1  -1  -1
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter has the effect of enhancing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

FilterPrewittVert

Filters an image using a vertical Prewitt kernel.

Syntax

```
IppStatus ippiFilterPrewittVert_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiFilterPrewittVert` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a vertical Prewitt operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
-1  0  1
-1  0  1
-1  0  1
```

The anchor cell is the center cell (red) of the kernel. The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of enhancing vertical edges of an image.

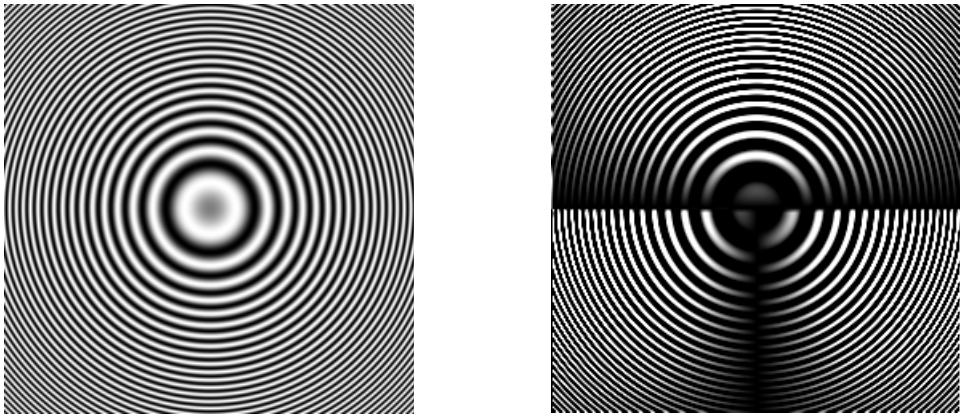
To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Figure 9-3 shows the result of filtering the sample image using the horizontal (upper part) and vertical (lower part) Prewitt operators. All pixels with negative values are zeroed because the function for `unsigned char` values was used.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

Figure 9-3 Using Prewitt Operator for Image Filtering



Sample Image Filtered Image

FilterScharrHoriz

Filters an image using a horizontal Scharr kernel.

Syntax

```
IppStatus ippiFilterScharrHoriz_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

```
8u16s_C1R     8s16s_C1R     32f_C1R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiFilterScharrHoriz` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a horizontal Scharr operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
 3   10   3
 0    0   0
-3  -10  -3
```

The anchor cell is the center cell (red) of the kernel. The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter has the effect of both enhancing and smoothing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

FilterScharrVert

Filters an image using a vertical Scharr kernel.

Syntax

```
IppStatus ippiFilterScharrVert_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u16s_C1R 8s16s_C1R 32f_C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiFilterScharrVert` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a vertical Scharr operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
  3   0   -3
 10   0  -10
  3   0   -3
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

FilterSobelHoriz, FilterSobelHorizMask

Filters an image using a horizontal Sobel kernel.

Syntax

```
IppStatus ippiFilterSobelHoriz_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiFilterSobelHoriz_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

8u16s_C1R	8s16s_C1R

```
IppStatus ippiFilterSobelHorizMask_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of `IppiMaskSize` type.

Description

The functions `ippiFilterSobelHoriz` and `ippiFilterSobelHorizMask` are declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). These functions apply a horizontal Sobel operator to an image ROI. The appropriate kernel is the matrix of 3x3 size, or either 3x3 or 5x5 size in accordance with *mask* parameter of the corresponding function flavors. The kernels have the following values:

```
                                 1   4   6   4   1
  1   2   1                      2   8  12   8   2
  0   0   0        or            0   0   0   0   0
 -1  -2  -1                     -2  -8 -12  -8  -2
                                -1  -4  -6  -4  -1
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter has the effect of both enhancing and smoothing horizontal edges of an image.
To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

FilterSobelVert, FilterSobelVertMask

Filter an image using a vertical Sobel kernel.

Syntax

```
IppStatus ippiFilterSobelVert_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiFilterSobelVert_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

8u16s_C1R	8s16s_C1R

```
IppStatus ippiFilterSobelVertMask_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of `IppiMaskSize` type.

Description

The functions `ippiFilterSobelVert` and `ippiFilterSobelVertMask` are declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). These functions apply a vertical Sobel operator to an image ROI. The appropriate kernel is the matrix of 3x3 size, or either 3x3 or 5x5 size in accordance with *mask* parameter of the corresponding function flavors. The kernels have the following values:

				-1	-2	0	2	1
-1	0	1		-4	-8	0	8	4
-2	0	2	or	-6	-12	0	12	6
-1	0	1		-4	-8	0	8	4
				-1	-2	0	2	1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if *mask* has an illegal value.

FilterSobelHorizSecond

Filters an image using a second derivative horizontal Sobel operator.

Syntax

```
IppStatus ippiFilterSobelHorizSecond_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

```
8u16s_C1R     8s16s_C1R     32f_C1R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.

mask	Predefined mask of `IppiMaskSize` type.

Description

The function `ippiFilterSobelHorizSecond` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a second derivative horizontal Sobel operator to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

$$\begin{matrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{matrix} \quad \text{or} \quad \begin{matrix} 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 \end{matrix}$$

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter has the effect of both enhancing and smoothing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

FilterSobelVertSecond

Filters an image using a second derivative vertical Sobel operator.

Syntax

```
IppStatus ippiFilterSobelVertSecond_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

```
8u16s_C1R     8s16s_C1R     32f_C1R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of `IppiMaskSize` type.

Description

The function `ippiFilterSobelVertSecond` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a second derivative vertical Sobel operator to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

```
                                 1   0  -2   0   1
1  -2   1                        4   0  -8   0   4
2  -4   2          or            6   0 -12   0   6
1  -2   1                        4   0  -8   0   4
                                 1   0  -2   0   1
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Example 9-8 shows how to use the function `ippiFilterSobelVertSecond_8u16s_C1R`.

Return Values

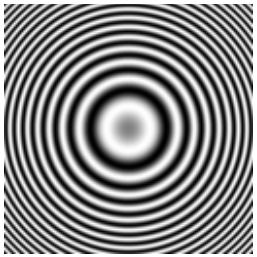
`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

Example 9-8 Using the Function `ippiFilterSobelVertSecond`

```
Ipp8u pSrc[8*8]={
                  4, 4, 4, 4, 4, 4, 4, 4,
                  4, 3, 3, 3, 3, 3, 3, 4,
                  4, 3, 2, 2, 2, 2, 3, 4,
                  4, 3, 2, 1, 1, 2, 3, 4,
                  4, 3, 2, 1, 1, 2, 3, 4,
                  4, 3, 2, 2, 2, 2, 3, 4,
                  4, 3, 3, 3, 3, 3, 3, 4,
                  4, 4, 4, 4, 4, 4, 4, 4
                 };
Ipp16s pDst[8*8];
IppiSize Roi = { 8, 8 };

ippiFilterSobelVertSecond_8u16s_C1R ( pSrc, 8, pDst, 8*sizeof( Ipp16s ),
Roi,  ippMskSize3x3 );

result:
4 4 4 4 4 4 4 4
4 3 3 3 3 3 3 4
4 3 2 2 2 2 3 4
4 3 2 1 1 2 3 4
4 3 2 1 1 2 3 4          pSrc
4 3 2 2 2 2 3 4
4 3 3 3 3 3 3 4
4 4 4 4 4 4 4 4
```

Example 9-8 Using the Function `ippiFilterSobelVertSecond` (continued)

```
 399 1 0    0   0     0    1 -201
 197 2 1    0   0     1    2   -3
  -4 1 2    1   1     2    1   -4
  -4 0 1    3   3     1    0   -4        pDst
  -4 0 1    3   3     1    0   -4
  -4 1 2    1   1     2    1   -4
  -3 2 1    0   0     1    2  197
-201 1 0 -12  75 -300  220  593

for example, on real picture it can looks like this

    size 128x128        size  126x126
    roi  126x126         roi   126x126
                        ippMskSize3x3
```



FilterSobelCross

Filters an image using a second cross derivative Sobel operator.

Syntax

```
IppStatus ippiFilterSobelCross_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

```
8u16s_C1R     8s16s_C1R     32f_C1R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of `IppiMaskSize` type.

Description

The function `ippiFilterSobelCross` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a second cross derivative Sobel operator to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

```
                                -1  -2   0   2   1
 -1   0   1                     -2  -4   0   4   2
  0   0   0        or            0   0   0   0   0
  1   0  -1                      2   4   0  -4  -2
                                 1   2   0  -2  -1
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

FilterRobertsDown

Filters an image using a horizontal Roberts kernel.

Syntax

```
IppStatus ippiFilterRobertsDown_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiFilterRobertsDown` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a horizontal Roberts operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
0  0  0
0  1  0
0  0 -1
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter gives the gross approximation of the pixel values' gradient in the horizontal direction.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

FilterRobertsUp

Filters an image using a vertical Roberts kernel.

Syntax

```
IppStatus ippiFilterRobertsUp_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiFilterRobertsUp` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a vertical Roberts operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
 0  0  0
 0  1  0
-1  0  0
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.This filter gives the gross approximation of the pixel values' gradient in the vertical direction.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

FilterLaplace

Filters an image using a Laplacian kernel.

Syntax

```
IppStatus ippiFilterLaplace_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R	8u16s_C1R	8s16s_C1R
8u_C3R	16s_C3R	32f_C3R		
8u_C4R	16s_C4R	32f_C4R		
8u_AC4R	16s_AC4R	32f_AC4R		

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of `IppiMaskSize` type.

Description

The function `ippiFilterLaplace` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a highpass Laplacian filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

```
                                -1 -3 -4 -3 -1
 -1  -1  -1                     -3  0  6  0 -3
 -1   8  -1        or           -4  6 20  6 -4
 -1  -1  -1                     -3  0  6  0 -3
                                -1 -3 -4 -3 -1
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter helps locate zero crossings in an image.
To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

FilterGauss

Filters an image using a Gaussian kernel.

Syntax

```
IppStatus ippiFilterGauss_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of `IppiMaskSize` type.

Description

The function `ippiFilterGauss` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a lowpass Gaussian filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size.

The 3x3 filter uses the kernel:

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

These filter coefficients correspond to a 2-dimensional Gaussian distribution with standard deviation 0.85.

The 5x5 filter uses the kernel:

```
 2/571  7/571  12/571  7/571  2/571
 7/571 31/571  52/571 31/571  7/571
12/571 52/571 127/571 52/571 12/571
 7/571 31/571  52/571 31/571  7/571
 2/571  7/571  12/571  7/571  2/571
```

These filter coefficients correspond to a 2-dimensional Gaussian distribution with standard deviation 1.0.
The anchor cell is the center cell of the kernels (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Example 9-9 shows how to use the function `ippiFilterGauss_8u_C1R`.

Return Values

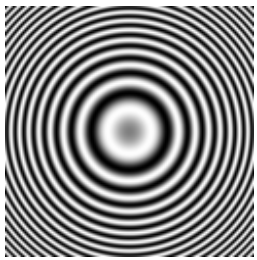
`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

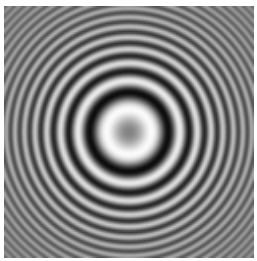
Example 9-9 Using the Gauss Filter

```
Ipp8u src[9*9] = {
      0, 1, 2, 3, 4, 5, 6, 7, 0,
      1, 2, 3, 4, 5, 6, 7, 0, 1,
      2, 3, 4, 5, 6, 7, 0, 1, 2,
            3, 4, 5, 6, 7, 0, 1, 2, 3,
      4, 5, 6, 7, 0, 1, 2, 3, 4,
      5, 6, 7, 0, 1, 2, 3, 4, 5,
                  6, 7, 0, 1, 2, 3, 4, 5, 6,
      7, 0, 1, 2, 3, 4, 5, 6, 7,
      0, 1, 2, 3, 4, 5, 6, 7, 0
```

Example 9-9 Using the Gauss Filter (continued)

```
   };
Ipp8u dst[9*9];
IppiSize roiSize = { 9, 9 };

ippiFilterGauss_8u_C1R ( src, 9, dst, 9, roiSize, ippMskSize5x5 );

Result:
0 1 2 3 4 5 6 7 0
1 2 3 4 5 6 7 0 1
2 3 4 5 6 7 0 1 2
3 4 5 6 7 0 1 2 3        src
4 5 6 7 0 1 2 3 4
5 6 7 0 1 2 3 4 5
6 7 0 1 2 3 4 5 6
7 0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 0

80 62 58 59 60 60 60 57 44
26 15 14 15 15 16 15 12 09
05 04 04 05 05 04 03 02 02
03 04 05 05 04 03 02 02 03
04 05 05 04 03 02 02 03 04        dst
05 05 04 03 02 02 03 04 05
05 04 03 02 02 03 04 05 08
12 13 13 13 13 14 15 18 26
39 48 53 54 52 50 50 53 63

for example, on real picture it can looks like this

  size  128x128          size  124x124
  roi   124x124          roi   124x124
                        ippMskSize5x5
```

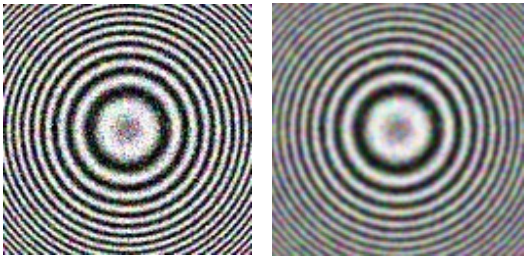




```
+ ippiAddRandGauss_Direct_8u_C3IR (.., 2, 50, 57328065)

   size 128x128     size  124x124
   roi   124x124    roi   124x124
```

Example 9-9 Using the Gauss Filter (continued)

```
ippMskSize5x5
```



FilterHipass

Filters an image using a highpass filter.

Syntax

```
IppStatus ippiFilterHipass_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.

mask	Predefined mask of `IppiMaskSize` type.

Description

The function `ippiFilterHipass` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a highpass filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

```
                          -1 -1 -1 -1 -1
 -1 -1 -1                 -1 -1 -1 -1 -1
 -1  8 -1     or           -1 -1 24 -1 -1
 -1 -1 -1                 -1 -1 -1 -1 -1
                          -1 -1 -1 -1 -1
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter attenuates low-frequency components and thus sharpens an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has an illegal value.

FilterLowpass

Filters an image using a lowpass filter.

Syntax

```
IppStatus ippiFilterLowpass_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of `IppiMaskSize` type.

Description

The function `ippiFilterLowpass` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a lowpass filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size.

The 3x3 filter uses the following kernel:

```
1/9  1/9  1/9
1/9  1/9  1/9
1/9  1/9  1/9
```

The 5x5 filter uses the following kernel:

```
 1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.
This filter blurs an image by averaging the pixel values over some neighborhood.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if *mask* has an illegal value.

FilterSharpen

Filters an image using a sharpening filter.

Syntax

```
IppStatus ippiFilterSharpen_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiFilterSharpen` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies a sharpening filter to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
-1/8  -1/8  -1/8
-1/8  16/8  -1/8
-1/8  -1/8  -1/8
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI. This filter enhances high-frequency components and thus sharpens an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

Fixed Filters with Border

This section describes the fixed filter functions that perform linear filtering of a source image using one of the predefined convolution kernels like the filter functions from the previous section do. Difference is that these functions automatically create a required border and define appropriate pixel values.

FilterScharrHorizGetBufferSize

Computes the size of the external buffer for the horizontal Scharr filter with border.

Syntax

```
IppStatus ippiFilterScharrHorizGetBufferSize_<mod>(IppiSize roiSize, int*
    pBufferSize);
```

Supported values for *mod* :

```
8u8s_C1R      8u16s_C1R     32f_C1R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
pBufferSize	Pointer to the buffer size.

Description

The function `ippiFilterScharrHorizGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the function `ippiFilterScharrHorizBorder`. This buffer *pBufferSize*[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pBufferSize* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

FilterScharrVertGetBufferSize

Computes the size of the external buffer for the vertical Scharr filter with border.

Syntax

```
IppStatus ippiFilterScharrVertGetBufferSize_<mod>(IppiSize roiSize,
    int* pBufferSize);
```

Supported values for *mod* :

```
8u8s_C1R      8u16s_C1R     32f_C1R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
pBufferSize	Pointer to the buffer size.

Description

The function `ippiFilterScharrVertGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the function `ippiFilterScharrVertBorder`. This buffer *pBufferSize*[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

FilterSobelHorizGetBufferSize

Computes the size of the external buffer for the horizontal Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelHorizGetBufferSize_<mod>(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

Supported values for *mod* :

```
8u8s_C1R      8u16s_C1R     32f_C1R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
mask	Predefined mask of `IppiMaskSize` type.
pBufferSize	Pointer to the buffer size.

Description

The function `ippiFilterSobelHorizGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterSobelHorizBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see Table 9-3). This buffer *pBufferSize*`[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has a wrong value.

FilterSobelVertGetBufferSize, FilterSobelNegVertGetBufferSize

Computes the size of the external buffer for the vertical Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelVertGetBufferSize_<mod>(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);

IppStatus ippiFilterSobelNegVertGetBufferSize_<mod>(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

Supported values for *mod* :

```
8u8s_C1R      8u16s_C1R     32f_C1R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
mask	Predefined mask of `IppiMaskSize` type.
pBufferSize	Pointer to the buffer size.

Description

The functions `ippiFilterSobelVertGetBufferSize` and `ippiFilterSobelNegVertGetBufferSize` are declared in the `ippcv.h` file. These functions compute the size of the external buffer that is required for the filter functions `ippiFilterSobelVertBorder, FilterSobelNegVertBorder,` and `ippiFilterSobelNegVertBorder` respectively.The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see Table 9-3). This buffer *pBufferSize*[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has a wrong value.

FilterSobelHorizSecondGetBufferSize

Computes the size of the external buffer for the second derivative horizontal Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelHorizSecondGetBufferSize_<mod>(IppiSize
    roiSize, IppiMaskSize mask, int* pBufferSize);
```

Supported values for *mod* :

```
8u8s_C1R      8u16s_C1R     32f_C1R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
mask	Predefined mask of `IppiMaskSize` type.
pBufferSize	Pointer to the buffer size.

Description

The function `ippiFilterSobelHorizSecondGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterSobelHorizSecondBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see Table 9-3). This buffer *pBufferSize*[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has a wrong value.

FilterSobelVertSecondGetBufferSize

Computes the size of the external buffer for the second derivative vertical Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelVertSecondGetBufferSize_<mod>(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

Supported values for *mod* :

```
8u8s_C1R      8u16s_C1R     32f_C1R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
mask	Predefined mask of `IppiMaskSize` type.
pBufferSize	Pointer to the buffer size.

Description

The function `ippiFilterSobelVertSecondGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterSobelVertSecondBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see Table 9-3). This buffer *pBufferSize*`[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has a wrong value.

FilterSobelCrossGetBufferSize

Computes the size of the external buffer for the cross Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelCrossGetBufferSize_<mod>(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

Supported values for *mod* :

```
8u8s_C1R      8u16s_C1R     32f_C1R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
mask	Predefined mask of `IppiMaskSize` type.
pBufferSize	Pointer to the buffer size.

Description

The function `ippiFilterSobelCrossGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterSobelCrossBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see Table 9-3). This buffer *pBufferSize*[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has a wrong value.

FilterLaplacianGetBufferSize

Computes the size of the external buffer for the Laplace filter with border.

Syntax

```
IppStatus ippiFilterLaplacianGetBufferSize_<mod>(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

Supported values for *mod* :

```
8u8s_C1R      8u16s_C1R     32f_C1R
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
mask	Predefined mask of `IppiMaskSize` type.
pBufferSize	Pointer to the buffer size.

Description

The function `ippiFilterLaplacianGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterLaplacianBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see Table 9-3). This buffer *pBufferSize*`[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has a wrong value.

FilterGaussGetBufferSize

Computes the size of the external buffer for the Gaussian filter with border.

Syntax

```
IppStatus ippiFilterGaussGetBufferSize_32f_C1R(IppiSize roiSize, int
    kernelSize, int* pBufferSize);
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
kernelSize	Size of the Gaussian kernel, odd, greater than or equal to 3.
pBufferSize	Pointer to the buffer size.

Description

The function `ippiFilterGaussGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterGaussBorder`. This buffer with the length *pBufferSize*[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*, and/or the kernel size is equal to or less than *kernelSize*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pBufferSize* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsBadArgErr	Indicates an error condition if *kernelSize* is even or less than 3.

FilterLowpassGetBufferSize

Computes the size of the external buffer for the lowpass filter with border.

Syntax

```
IppStatus ippiFilterLowpassGetBufferSize_8u_C1R(IppiSize roiSize, IppiMaskSize
    mask, int* pBufferSize);
```

```
IppStatus ippiFilterLowpassGetBufferSize_32f_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

Parameters

roiSize	Maximum size of the source and destination image ROI.
mask	Predefined mask of `IppiMaskSize` type.
pBufferSize	Pointer to the buffer size.

Description

The function `ippiFilterLowpassGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function `ippiFilterLowpassBorder`. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see Table 9-3). This buffer *pBufferSize*[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if *mask* has a wrong value.

GenSobelKernel

Computes kernel for the Sobel filter.

Syntax

```
IppStatus ippiGenSobelKernel_16s(Ipp16s* pDst, int kernelSize, int dx, int
    sign);
IppStatus ippiGenSobelKernel_32f(Ipp32f* pDst, int kernelSize, int dx, int
    sign);
```

Parameters

pDst	Pointer to the destination vector.
kernelSize	Size of the Sobel kernel.

dx	Order of derivative.
sign	Specifies signs of kernel elements.

Description

The function `ippiGenSobelKernel` is declared in the `ippcv.h` file. This function computes the one dimensional Sobel kernel. Kernel coefficients are equal to coefficients of the polynomial

$$(1+x)^{kernelSize-dx-1} \cdot (x-1)^{dx}$$

If the *sign* parameter is negative, then signs of kernel coefficients are changed. Kernel calculated by this function can be used to filter images by high order Sobel filter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *kernelSize* is less than 3 or is even.
`ippStsBadArgErr`	Indicates an error condition if *dx* is equal to or less than *kernelSize*, or *dx* is negative.

FilterScharrHorizBorder

Applies horizontal Scharr filter with border.

Syntax

```
IppStatus ippiFilterScharrHorizBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp8u
    borderValue, int divisor, Ipp8u* pBuffer);
IppStatus ippiFilterScharrHorizBorder_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
    Ipp8u borderValue, Ipp8u* pBuffer);
IppStatus ippiFilterScharrHorizBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
    Ipp32f borderValue, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border (see Borders); following values are possible: `ippBorderConst` Values of all border pixels are set to constant. `ippBorderRepl` Replicated border is used. `ippBorderWrap` Wrapped border is used `ippBorderMirror` Mirrored border is used `ippBorderMirrorR` Mirrored border with replication is used
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
divisor	The constant value that output pixel value is divided by.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterScharrHorizBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies the horizontal Scharr filter (y-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. For the function flavor `ippiFilterScharrHorizBorder_32f_C1R` the source image can be used as the destination image.
The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is a matrix of 3x3 size with the anchor in the center cell (red) and the following values:

```
 3     10     3

 0      0     0

-3    -10    -3
```

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterScharrHorizGetBufferSize` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error if *borderType* or *divisor* has wrong value.

FilterScharrVertBorder

Applies vertical Scharr filter with border.

Syntax

```
IppStatus ippiFilterScharrVertBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp8u
    borderValue, int divisor, Ipp8u* pBuffer);
IppStatus ippiFilterScharrVertBorder_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp8u
    borderValue, Ipp8u* pBuffer);
IppStatus ippiFilterScharrVertBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp32f
    borderValue, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
borderType	Type of border (see Borders); following values are possible: ippBorderConst — Values of all border pixels are set to constant. ippBorderRepl — Replicated border is used. ippBorderWrap — Wrapped border is used ippBorderMirror — Mirrored border is used ippBorderMirrorR — Mirrored border with replication is used
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).

divisor	The constant value that output pixel value is divided by.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterScharrVertBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies the vertical Scharr filter (x-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. For the function flavor `ippiFilterSobelHorizGetBufferSize` the source image can be used as the destination image. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is a matrix of 3x3 size with the anchor in the center cell (red) and the following values:

```
  3     0     -3

 10     0    -10

  3     0     -3
```

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterScharrVertGetBufferSize` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if *borderType* or *divisor* has wrong value.

FilterSobelHorizBorder

Applies horizontal Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelHorizBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType
    borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterSobelHorizBorder_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
IppStatus ippiFilterSobelHorizBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
mask	Type of the filter kernel.
borderType	Type of border (see Borders); following values are possible:

`ippBorderConst`	Values of all border pixels are set to constant.
`ippBorderRepl`	Replicated border is used.
`ippBorderWrap`	Wrapped border is used
`ippBorderMirror`	Mirrored border is used
`ippBorderMirrorR`	Mirrored border with replication is used

borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
divisor	The constant value that output pixel value is divided by.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterSobelHorizBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies the horizontal Sobel filter (y-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. For the function flavor `ippiFilterSobelHorizBorder_32f_C1R` the source image can be used as the destination image. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

				1	4	6	4	1
1	2	1		2	8	12	8	2
0	0	0	or	0	0	0	0	0

```
-1  -2  -1                  -2  -8 -12  -8  -4

                            -1  -4  -6  -4  -1
```

The function requires the working buffer *pBuffer* whose size should be computed by the function ippiFilterSobelHorizGetBufferSize beforehand.

Example 14-1 shows how the function ippiFilterSobelHorizBorder_8u16s_C1R can be used for edge detection.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error if *borderType* or *divisor* has wrong value.
ippStsMaskErr	Indicates an error condition if *mask* has wrong value.

FilterSobelVertBorder, FilterSobelNegVertBorder,

Applies vertical Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelVertBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
IppStatus ippiFilterSobelVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype>
    borderValue, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u16s_C1R 32f_C1R

```
IppStatus ippiFilterSobelNegVertBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterSobelNegVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u*
    pBuffer);
```

Supported values for *mod* :

8u16s_C1R 32f_C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
mask	Type of the filter kernel.
borderType	Type of border (see Borders); following values are possible: `ippBorderConst` Values of all border pixels are set to constant `ippBorderRepl` Replicated border is used. `ippBorderWrap` Wrapped border is used `ippBorderMirror` Mirrored border is used `ippBorderMirrorR` Mirrored border with replication is used
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
divisor	The constant value that output pixel value is divided by.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterSobelVertBorder` and `ippiFilterSobelNegVertBorder` are declared in the `ippcv.h` file. They operate with ROI (see Regions of Interest in Intel IPP). These functions apply the vertical Sobel filter (x-derivative) to the source image ROI *pSrc* and stores results to the destination image ROI of the same size *pDst*. Source image can

be used as the destination image if they have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The anchor cell is the center cell of the kernel (red).

The function `ippiFilterSobelVertBorder` uses the kernels with the following coefficients:

```
                                -1   -2   0   2   1
 -1   0   1                     -4   -8   0   8   4
 -2   0   2        or           -6  -12   0  12   6
 -1   0   1                     -4   -8   0   8   4
                                -1   -2   0   2   1
```

The function `ippiFilterSobelNegVertBorder` uses the kernels which coefficients are the same in magnitude but opposite in sign:

```
                                 1   2   0  -2  -1
  1   0  -1                      4   8   0  -8  -4
  2   0  -2        or            6  12   0 -12  -6
  1   0  -1                      4   8   0  -8  -4
                                 1   2   0  -2  -1
```

Both functions require the working buffer *pBuffer* whose size must be computed beforehand by the functions `ippiFilterSobelVertGetBufferSize`, `FilterSobelNegVertGetBufferSize` and `ippiFilterSobelNegVertGetBufferSize` respectively.

Example 14-1 shows how the function `ippiFilterSobelNegVertBorder_8u16s_C1R` can be used for edge detection.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>

ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error if *borderType* or *divisor* has wrong value.
ippStsMaskErr	Indicates an error condition if *mask* has wrong value.

FilterSobelHorizSecondBorder

Applies horizontal (second derivative) Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelHorizSecondBorder_8u8s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
IppStatus ippiFilterSobelHorizSecondBorder_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue,
    Ipp8u* pBuffer);
```

Supported values for *mod* :

8u16s_C1R 32f_C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
mask	Type of the filter kernel.
borderType	Type of border (see Borders); following values are possible:

ippBorderConst	Values of all border pixels are set to constant.
ippBorderRepl	Replicated border is used.

	`ippBorderWrap` Wrapped border is used `ippBorderMirror` Mirrored border is used `ippBorderMirrorR` Mirrored border with replication is used
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
divisor	The constant value that output pixel value is divided by.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterSobelHorizSecondBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies the second derivative horizontal Sobel filter (y-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*.The kernels have the following values with the anchor in the center cell (red):

$$\begin{matrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{matrix} \quad \text{or} \quad \begin{matrix} 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 \end{matrix}$$

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterSobelHorizSecondGetBufferSize` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if *borderType* or *divisor* has wrong value.
`ippStsMaskErr`	Indicates an error condition if *mask* has wrong value.

FilterSobelVertSecondBorder

Applies vertical (second derivative) Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelVertSecondBorder_8u8s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
IppStatus ippiFilterSobelVertSecondBorder_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue,
    Ipp8u* pBuffer);
```

Supported values for *mod* :

8u16s_C1R 32f_C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
mask	Type of the filter kernel.
borderType	Type of border (see Borders); following values are possible: ippBorderConst Values of all border pixels are set to constant. ippBorderRepl Replicated border is used. ippBorderWrap Wrapped border is used ippBorderMirror Mirrored border is used ippBorderMirrorR Mirrored border with replication is used
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
divisor	The constant value that output pixel value is divided by.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterSobelVertSecondBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies the second derivative vertical Sobel filter (x-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

```
                                  1   0  -2   0   1
1  -2   1                         4   0  -8   0   4
2  -4   2          or             6   0 -12   0   6
1  -2   1                         4   0  -8   0   4
                                  1   0  -2   0   1
```

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterSobelVertSecondGetBufferSize` beforehand.

Example 9-10 shows how to use the `ippiFilterSobelVertSecondBorder_8u16s_C1R` function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if *borderType* or *divisor* has wrong value.
`ippStsMaskErr`	Indicates an error condition if *mask* has wrong value.

Example 9-10 Using the Second Derivative Vertical Sobel Filter

```
Ipp8u pSrc[8*8]={
                  4, 4, 4, 4, 4, 4, 4, 4,
                  4, 3, 3, 3, 3, 3, 3, 4,
                  4, 3, 2, 2, 2, 2, 3, 4,
```

Example 9-10 Using the Second Derivative Vertical Sobel Filter (continued)

```
                    4, 3, 2, 1, 1, 2, 3, 4,
                    4, 3, 2, 1, 1, 2, 3, 4,
                    4, 3, 2, 2, 2, 2, 3, 4,
                    4, 3, 3, 3, 3, 3, 3, 4,
                    4, 4, 4, 4, 4, 4, 4, 4
                  };
Ipp16s pDst[8*8];
IppiSize Roi = { 8, 8 };
Ipp8u borderValue = 3;
int pBufferSize;

ippiFilterSobelVertSecondGetBufferSize_8u16s_C1R(Roi, ippMskSize3x3,
&pBufferSize);

Ipp8u* pBuffer = ippiMalloc_8u_C1 ( 8, 8, &pBufferSize );

ippiFilterSobelVertSecondBorder_8u16s_C1R ( pSrc, 8, pDst,
8*sizeof(Ipp16s), Roi, ippMskSize3x3, ippBorderConst, borderValue,
pBuffer );

result:
4 4 4 4 4 4 4 4
4 3 3 3 3 3 3 4
4 3 2 2 2 2 3 4
4 3 2 1 1 2 3 4
4 3 2 1 1 2 3 4     pSrc
4 3 2 2 2 2 3 4
4 3 3 3 3 3 3 4
4 4 4 4 4 4 4 4


-4 1 0 0 0 0 1 -4
-7 2 1 0 0 1 2 -7
-8 1 2 1 1 2 1 -8
-8 0 1 3 3 1 0 -8
-8 0 1 3 3 1 0 -8     pDst
-8 1 2 1 1 2 1 -8
-7 2 1 0 0 1 2 -7
-4 1 0 0 0 0 1 -4
```

FilterSobelCrossBorder

Applies second derivative cross Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelCrossBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
IppStatus ippiFilterSobelCrossBorder_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u*
    pBuffer);
```

Supported values for *mod* :

8u16s_C1R 32f_C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
mask	Type of the filter kernel.
borderType	Type of border (see Borders); following values are possible: ippBorderConst Values of all border pixels are set to constant. ippBorderRepl Replicated border is used. ippBorderWrap Wrapped border is used ippBorderMirror Mirrored border is used ippBorderMirrorR Mirrored border with replication is used
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
divisor	The constant value that output pixel value is divided by.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterSobelCrossBorder` is declared in the `ippcv.h` file.
It operates with ROI (see Regions of Interest in Intel IPP). This function applies the second derivative cross Sobel filter (xy-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*.The kernels have the following values with the anchor in the center cell (red):

				-1	-2	0	2	1
-1	0	1		-2	-4	0	4	2
0	0	0	or	0	0	0	0	0
1	0	-1		2	4	0	-4	-2
				1	2	0	-2	-1

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterSobelCrossGetBufferSize` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if *borderType* or *divisor* has wrong value.
`ippStsMaskErr`	Indicates an error condition if *mask* has wrong value.

FilterLaplacianBorder

Applies Laplacian filter with border.

Syntax

```
IppStatus ippiFilterLaplacianBorder_8u8s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, int divisor, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLaplacianBorder_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u*
    pBuffer);
```

Supported values for *mod* :

`8u16s_C1R` `32f_C1R`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
mask	Type of the filter kernel.
borderType	Type of border (see Borders); following values are possible: `ippBorderConst` Values of all border pixels are set to constant. `ippBorderRepl` Replicated border is used. `ippBorderWrap` Wrapped border is used `ippBorderMirror` Mirrored border is used `ippBorderMirrorR` Mirrored border with replication is used
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
divisor	The constant value that output pixel value is divided by.
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterLaplacianBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies the laplacian filter to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

$$\begin{matrix} 2 & 0 & 2 \\ 0 & -8 & 0 \\ 2 & 0 & 2 \end{matrix} \quad \text{or} \quad \begin{matrix} 2 & 4 & 4 & 4 & 2 \\ 4 & 0 & -8 & 0 & 4 \\ 4 & -8 & -24 & -8 & 4 \\ 4 & 0 & -8 & 0 & 4 \\ 2 & 4 & 4 & 4 & 2 \end{matrix}$$

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterLaplacianGetBufferSize` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if *borderType* or *divisor* has wrong value.
`ippStsMaskErr`	Indicates an error condition if *mask* has wrong value.

FilterGaussBorder

Applies Gauss filter with border.

Syntax

```
IppStatus ippiFilterGaussBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, int kernelSize, Ipp32f sigma,
    IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
kernelSize	Specifies the size of the Gaussian kernel, must be odd and greater than or equal to 3.
sigma	Standard deviation of the Gaussian kernel.
borderType	Type of border (see Borders); following values are possible: `ippBorderConst` Values of all border pixels are set to constant. `ippBorderRepl` Replicated border is used. `ippBorderWrap` Wrapped border is used `ippBorderMirror` Mirrored border is used `ippBorderMirrorR` Mirrored border with replication is used
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterGaussBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies the Gaussian filter to the source image ROI *pSrc*. The kernel of this filter is the matrix of size *kernelSize***kernelSize* with the standard deviation *sigma*. The values of the elements of the Gaussian kernel are calculated according to the formula:

$$G(i, j) = \exp\left\{-\frac{(K/2-i)^2+(K/2-j)^2}{2\sigma^2}\right\}, \qquad i, j = 0, \ldots K$$

and then are normalized. The anchor cell is the center of the kernel.

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFilterGaussGetBufferSize` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4.
`ippStsBorderErr`	Indicates an error condition if *borderType* has wrong value.
`ippStsBadArgErr`	Indicates an error condition if *kernelSize* is less than 3, or *sigma* is less than or equal to0.

FilterLowpassBorder

Applies lowpass filter with border.

Syntax

```
IppStatus ippiFilterLowpassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLowpassBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
mask	Type of the filter kernel.
borderType	Type of border (see Borders); following values are possible: `ippBorderConst` Values of all border pixels are set to constant. `ippBorderRepl` Replicated border is used. `ippBorderWrap` Wrapped border is used `ippBorderMirror` Mirrored border is used `ippBorderMirrorR` Mirrored border with replication is used
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
pBuffer	Pointer to the working buffer.

Description

The function `ippiFilterLowpassBorder` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function applies the lowpass filter (blur operation) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The anchor cell is the center cell (red) of the kernel.

The 3x3 filter uses the kernel with the following values:

```
1/9  1/9  1/9
1/9  1/9  1/9
1/9  1/9  1/9
```

The 5x5 filter uses the kernel with the following values:

```
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
```

The function requires the working buffer *pBuffer* whose size should be computed by the function ippiFilterLowpassGetBufferSize beforehand.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if *borderType* has wrong value.
ippStsMaskErr	Indicates an error condition if *mask* has wrong value.

Image Linear Transforms

10

This chapter describes the Intel® IPP image processing functions that perform linear transform operations on an image buffer.

These operations include Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), and Discrete Cosine Transform (DCT).

Table 10-1 lists the Intel IPP linear transform functions.:

Table 10-1 Image Linear Transform Functions

Function Base Name	Operation
Fourier Transforms	
FFTInitAlloc	Allocates memory and fills in context data needed for the image FFT functions to operate.
FFTFree	Deallocates memory used by the FFT context structure.
FFTGetBufSize	Determines the size of an external work buffer that can be used by the FFT functions.
FFTFwd	Applies forward Fast Fourier Transform to an image.
FFTInv	Applies inverse Fast Fourier Transform to complex source data and stores results in a destination image.
DFTInitAlloc	Allocates memory and fills in context data needed for the image DFT functions to operate.
DFTFree	Deallocates memory used by the DFT context structure.
DFTGetBufSize	Determines the size of an external work buffer that can be used by the DFT functions.
DFTFwd	Applies forward Discrete Fourier Transform to an image.

Table 10-1 Image Linear Transform Functions (continued)

Function Base Name	Operation
DFTInv	Applies inverse Discrete Fourier Transform to complex source data and stores results in a destination image
MulPack	Multiplies two source images with data in packed format and stores the result a destination image in packed format.
MulPackConj	Multiplies two source images with data in packed format and stores the result in a destination image that is complex-conjugate to that obtained with the `ippiMulPack` function.
Magnitude	Computes the magnitude of elements of complex data images.
MagnitudePack	Computes the magnitude of elements of an image in packed format.
Phase	Computes the phase of elements of complex data images.
PhasePack	Computes the phase of elements of an image in packed format.
PolarToCart	Converts an image in the polar coordinate form to Cartesian coordinate form.
PackToCplxExtend	Converts an image in packed format to a complex data image.
CplxExtendToPack	Converts a complex data image to an image in packed format.
WinBartlett, WinBartlettSep	Applies Bartlett window function to the image.
WinHamming, WinHammingSep	Applies Hamming window function to the image.
Discrete Cosine Transforms	
DCTFwdInitAlloc	Allocates memory and fills in context data needed for the forward DCT function to operate
DCTInvInitAlloc	Allocates memory and fills in context data needed for the inverse DCT function to operate
DCTFwdFree	Deallocates memory used by the forward DCT context structure.
DCTInvFree	Deallocates memory used by the inverse DCT context structure.

Table 10-1 Image Linear Transform Functions (continued)

Function Base Name	Operation
DCTFwdGetBufSize	Determines the size of an external work buffer that can be used by the forward DCT function
DCTInvGetBufSize	Determines the size of an external work buffer that can be used by the inverse DCT function
DCTFwd	Performs a forward DCT of an image
DCTInv	Performs an inverse DCT of an image
DCT8x8Fwd	Performs a forward DCT on a buffer of 8x8 size
DCT8x8Inv, DCT8x8Inv_A10	Performs an inverse DCT on a buffer of 8x8 size
DCT8x8FwdLS	Performs a forward DCT on a 2D buffer of 8x8 size with level shift
DCT8x8InvLSClip	Performs an inverse DCT on a 2D buffer of 8x8 size with level shift
DCT8x8To2x2Inv, DCT8x8To4x4Inv	Perform an inverse DCT on a 2D buffer of 8x8 size with further downsampling to 2x2 or 4x4 size
DCT8x8Inv_2x2 DCT8x8Inv_4x4	Perform an inverse DCT on a top left quadrant 2x2 or 4x4 of the buffer of 8x8 size

To speed up performance, linear transform functions use precomputed auxiliary data that is needed for computation of the transforms (that is, tables of twiddle factors for FFT functions). This data is calculated by the respective initialization functions and passed to the transform functions in context structures specific for each type of transform.

Most linear transform functions in Intel IPP have code branches that implement different algorithms to compute the results. You can choose the desired code variety to be used by the transform function by setting the *hint* argument to one of the following values that are listed in Table 10-2 :

Table 10-2 Hint Arguments for Linear Transform Functions

Value	Description
ippAlgHintNone	The computation algorithm will be chosen by the internal function logic.
ippAlgHintFast	Fast algorithm must be used. The output results will be less accurate.
ippAlgHintAccurate	High accuracy algorithm must be used. The function will need more time to execute.

Intel IPP linear transform functions can use external work buffers for storing data and intermediate results, which eliminates the need to allocate and free internal memory buffers and thus helps to further increase function performance. To determine the required work buffer size, use one of the respective support functions specific for each transform type. In case when no external buffer is specified, the transform functions handle memory allocation internally.

All Intel IPP linear transform functions except DCT of 8x8 size work on images with floating point data only.

Fourier Transforms

Intel IPP functions that compute FFT and DFT can process both real and complex images. Function flavors operating on real data are distinguished by R suffix present in function-specific modifier of their full name, whereas complex flavors' names include C suffix (see Function Naming in Chapter 2).

The results of computing the Fourier transform can be normalized by specifying the appropriate value of *flag* argument for context initialization. This parameter sets up a pair of matched normalization factors to be used in forward and inverse transforms as listed in the following table:

Table 10-3 Normalization Factors for Fourier Transform Results

Value of *flag* Argument	Normalization Factors	
	Forward Transform	Inverse Transform
IPP_FFT_DIV_FWD_BY_N	1/MN	1
IPP_FFT_DIV_INV_BY_N	1	1/MN
IPP_FFT_DIV_BY_SQRTN	1/sqrt(MN)	1/sqrt(MN)
IPP_FFT_NODIV_BY_ANY	1	1

In this table, N and M denote the length of Fourier transform in the x- and y-directions, respectively (or, equivalently, the number of columns and rows in the 2D array being transformed).

For the FFT, these lengths must be integer powers of 2, that is $N=2^{orderX}$, $M=2^{orderY}$, where power exponents are known as order of FFT.

For the DFT, N and M can take on arbitrary integer non-negative values.

Real - Complex Packed (RCPack2D) Format

The forward Fourier transform of a real two-dimensional image data yields a matrix of complex results which has conjugate-symmetric properties. Intel IPP functions use packed format RCPack2D for storing and retrieving data of this type. Accordingly, real flavors of the inverse Fourier transform functions convert packed complex conjugate-symmetric data back to its real origin.

The RCPack2D format exploits the complex conjugate symmetry of the transformed data to store only a half of the resulting Fourier coefficients. For the `N` by `M` transform, the respective FFT and DFT functions actually store real and imaginary parts of the complex Fourier coefficients `A(i,j)` for `i = 0,...,M-1; j = 0,... N/2` in a single real array of dimensions (`N`, `M`). The RCPack2D storage format is slightly different for odd and even `M` and is arranged in accordance with the following tables:

Table 10-4 RCPack2D Storage for Odd Number of Rows

Re A(0,0)	Re A(0,1)	Im A(0,1)	...	Re A(0,(N-1)/2)	Im A(0,(N-1)/2)	Re A(0,N/2)
Re A(1,0)	Re A(1,1)	Im A(1,1)	...	Re A(1,(N-1)/2)	Im A(1,(N-1)/2)	Re A(1,N/2)
Im A(1,0)	Re A(2,1)	Im A(2,1)	...	Re A(2,(N-1)/2)	Im A(2,(N-1)/2)	Im A(1,N/2)
...	...	...	...	...	...	...
Re A(M/2,0)	Re A(M-2,1)	Im A(M-2,1)	...	Re A(M-2,(N-1)/2)	Im A(M-2,(N-1)/2)	Re A(M/2,N/2)
Im A(M/2,0)	Re A(M-1,1)	Im A(M-1,1)	...	Re A(M-1,(N-1)/2)	Im A(M-1,(N-1)/2)	Im A(M/2,N/2)

Table 10-5 RCPack2D Storage for Even Number of Rows

Re A(0,0)	Re A(0,1)	Im A(0,1)	...	Re A(0,(N-1)/2)	Im A(0,(N-1)/2)	Re A(0,N/2)
Re A(1,0)	Re A(1,1)	Im A(1,1)	...	Re A(1,(N-1)/2)	Im A(1,(N-1)/2)	Re A(1,N/2)
Im A(1,0)	Re A(2,1)	Im A(2,1)	...	Re A(2,(N-1)/2)	Im A(2,(N-1)/2)	Im A(1,N/2)
...	...	...	...	...	...	...
Re A(M/2-1,0)	Re A(M-3,1)	Im A(M-3,1)	...	Re A(M-3,(N-1)/2)	Im A(M-3,(N-1)/2)	Re A(M/2-1,N/2)
Im A(M/2-1,0)	Re A(M-2,1)	Im A(M-2,1)	...	Re A(M-2,(N-1)/2)	Im A(M-2,(N-1)/2)	Im A(M/2-1,N/2)
Re A(M/2,0)	Re A(M-1,1)	Im A(M-1,1)	...	Re A(M-1,(N-1)/2)	Im A(M-1,(N-1)/2)	Re A(M/2,N/2)

The shaded columns to the right side of the tables indicate values for even `N` only.

Note the above tables show the arrangement of coefficients for one channel. For multichannel images the channel coefficients are clustered and stored consecutively, for example, for 3-channel image they are stored in the following way: `C1-Re A(0,0); C2-Re A(0,0); C3-Re A(0,0); C1-Re A(0,1); C2-Re A(0,1); C3-Re A(0,1); C1-Im A(0,1); C2-Im A(0,1); ...`

The remaining Fourier coefficients are obtained using the following relationships based on conjugate-symmetric properties:

```
A(i,j) = conj(A(M-i,N-j))    i = 1,..., M-1;   j = 1,..., N-1
A(0,j) = conj(A(0,N-j))      j = 1,..., N-1
A(i,0) = conj(A(M-i,0))      i = 1,..., M-1
```

FFTInitAlloc

Allocates memory and fills in context data needed for the image FFT functions to operate.

Syntax

```
IppStatus ippiFFTInitAlloc_R_32s (IppiFFTSpec_R_32s** ppFFTSpec,
    int orderX, int orderY, int flag, IppHintAlgorithm hint);
IppStatus ippiFFTInitAlloc_R_32f (IppiFFTSpec_R_32f** ppFFTSpec,
    int orderX, int orderY, int flag, IppHintAlgorithm hint);
IppStatus ippiFFTInitAlloc_C_32fc (IppiFFTSpec_C_32fc** ppFFTSpec,
    int orderX, int orderY, int flag, IppHintAlgorithm hint);
```

Parameters

ppFFTSpec	Pointer to the pointer to the FFT context structure being initialized.
orderX, orderY	Order of the FFT in x- and y- directions, respectively.
flag	Flag to choose the results normalization option.
hint	Option to select the algorithmic implementation of the transform function (see Table 10-2).

Description

The function `ippiFFTInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure *pFFTSpec* needed to compute the forward and inverse FFT of a two-dimensional image data.

The ippiFFTFwd and ippiFFTInv functions called with the pointer to the initialized *pFFTSpec* structure as an argument will compute the fast Fourier transform with the following characteristics:

- length $N=2^{orderX}$ in x-direction by $M=2^{orderY}$ in y-direction
- results normalization mode as set by *flag* (see Table 10-3)
- computation algorithm indicated by *hint*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pFFTSpec* pointer is NULL.
ippStsFftOrderErr	Indicates an error condition if the FFT order value is illegal.
ippStsFFTFlagErr	Indicates an error condition if *flag* has an illegal value.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

FFTFree

Deallocates memory used by the FFT context structure.

Syntax

```
IppStatus ippiFFTFree_R_32s (IppiFFTSpec_R_32s* pFFTSpec);
IppStatus ippiFFTFree_R_32f (IppiFFTSpec_R_32f* pFFTSpec);
IppStatus ippiFFTFree_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec);
```

Parameters

pFFTSpec	Pointer to the FFT context structure that has to be released.

Description

The function ippiFFTFree is declared in the ippi.h file. This function releases memory used by the previously initialized FFT context structure *pFFTSpec*.
Call this function after your application program has finished computing the fast Fourier transforms.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pFFTSpec* pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid *pFFTSpec* structure is passed.

FFTGetBufSize

Determines the size of external work buffer that can be used by the FFT functions.

Syntax

```
IppStatus ippiFFTGetBufSize_R_32s (IppiFFTSpec_R_32s* pFFTSpec,
    int* pSize);
IppStatus ippiFFTGetBufSize_R_32f (IppiFFTSpec_R_32f* pFFTSpec,
    int* pSize);
IppStatus ippiFFTGetBufSize_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec,
    int* pSize);
```

Parameters

pFFTSpec	Pointer to the previously initialized FFT context structure.
pSize	Pointer to the size of the external work buffer to be used by the FFT functions.

Description

The function ippiFFTGetBufSize is declared in the ippi.h file. This function determines the size in bytes of an external memory buffer that can be used by FFT functions with context *pFFTSpec* as a workspace during calculations.
If you want to use external buffering, call this function after *pFFTSpec* initialization.
Use the computed *pSize* value to set the size in bytes of an external buffer to be allocated by means of the functions ippMalloc or ippsMalloc_8u. Note that the allocated memory can be freed only by the functions ippFree or ippsFree respectively. Refer to the chapter 3 “Support Functions” of the *Intel IPP Reference Manual*, vol.1 for more details about the memory allocation functions.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pFFTSpec* or *pSize* pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid *pFFTSpec* structure is passed.

FFTFwd

Applies forward Fast Fourier Transform to an image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiFFTFwd_RToPack_<mod> (const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u32s_C1RSfs
8u32s_C3RSfs
8u32s_C4RSfs
8u32s_AC4RSfs
```

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiFFTFwd_RToPack_<mod> (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,
    Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Case 3: Not-in-place operation on complex data

```
IppStatus ippiFFTFwd_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,
    Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec,
    Ipp8u* pBuffer);
```

Case 4: In-place operation on floating-point data

```
IppStatus ippiFFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

Case 5: In-place operation on complex data

```
IppStatus ippiFFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,
    const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
scaleFactor	Scale factor (see Integer Result Scaling).
pBuffer	Pointer to the external work buffer, can be `NULL`.

Description

The function `ippiFFTFwd` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a forward FFT on each channel of the source image ROI *pSrc* (*pSrcDst* for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer *pDst* (*pSrcDst* for in-place flavors). The size of ROI is N x M, it is specified by the parameters *orderX*, *orderY* calling the function ippiFFTInitAlloc.
The function flavor ippiFFTFwd_RToPack that operates on images with real data takes advantage of the symmetry property and stores the output data in RCPack2D format . It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with AC4 descriptor do not process alpha channel.

The function flavor ippiFFTFwd_CToC that operates on images with complex data does not perform any packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

The forward FFT functions use the previously initialized *pFFTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* that allows to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing FFT. On modern Intel® architectures the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function ippiFFTGetBufSize beforehand. If the external buffer is not specified (*pBuffer* is set to NULL), then the function itself allocates the memory needed for operation.

The Example 10-1 illustrates the use of the forward FFT function for processing real data. The Example 10-2 explains FFT processing of complex data. Note that input values are the same as in the previous example, but specified in complex domain.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc*, *pDst*, or *pFFTSpec* pointer is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.

ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid *pFFTSpec* structure is passed.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

Example 10-1 Fast Fourier Transform of a Real Image

```
IppStatus fft( void ) {
   Ipp32f src[64] = {0}, dst[64];
   IppiFFTSpec_R_32f *spec;
   IppStatus status;
   src[0] = -3; src[9] = 1;
   ippiFFTInitAlloc_R_32f(&spec, 3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate);
   status = ippiFFTFwd_RToPack_32f_C1R( src, 8*sizeof(Ipp32f), dst,
                                        8*sizeof(Ipp32f), spec, 0 );
   ippiFFTFree_R_32f( spec );
   return status;
}
On exit from the ippiFFTFwd_RToPack function, the destination buffer contains the
following data in RCPack2D format:
-2.00  -2.29  -0.71  -3.00  -1.00  -3.71  -0.71  -4.00
-2.29  -3.00  -1.00  -3.71  -0.71  -4.00  +0.00  -3.71
-0.71  -3.71  -0.71  -4.00  +0.00  -3.71  +0.71  +0.71
-3.00  -4.00  +0.00  -3.71  +0.71  -3.00  +1.00  -3.00
-1.00  -3.71  +0.71  -3.00  +1.00  -2.29  +0.71  +1.00
-3.71  -3.00  +1.00  -2.29  +0.71  -2.00  +0.00  -2.29
-0.71  -2.29  +0.71  -2.00  +0.00  -2.29  -0.71  +0.71
-4.00  -2.00  +0.00  -2.29  -0.71  -3.00  -1.00  -2.00
```

Example 10-2 Fast Fourier Transform of a Complex Image

```
IppStatus fft_cplx( void ) {
   Ipp32fc src[64] = {0}, dst[64], m3 = {-3,0}, one = {1,0};
   IppiFFTSpec_C_32fc *spec;
   IppStatus status;
   src[0] = m3; src[9] = one;
   ippiFFTInitAlloc_C_32fc( &spec, 3, 3, IPP_FFT_DIV_INV_BY_N,
ippAlgHintAccurate);
  status = ippiFFTFwd_CToC_32fc_C1R( src, 8*sizeof(Ipp32fc), dst,
                                     8*sizeof(Ipp32fc), spec, 0 );
   ippiFFTFree_C_32fc( spec );
   return status;
}
On exit from the ippiFFTFwd_CToC function, the destination buffer contains the
following data (real and imaginary part of complex numbers are given in
comma-delimited format):

-2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7
-2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0
-3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0,-0.0 -2.3,-0.7
-3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0,-0.0 -2.3,-0.7 -3.0,-1.0
-4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7
-3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0
-3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0 -3.7, 0.7
```

Example 10-2 Fast Fourier Transform of a Complex Image

```
-2.3, 0.7 -2.0,-0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0
```

FFTInv

Applies an inverse FFT to complex source data and stores results in a destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiFFTInv_PackToR_<mod>(const Ipp32s* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32s8u_C1RSfs
32s8u_C3RSfs
32s8u_C4RSfs
32s8u_AC4RSfs
```

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiFFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,
    Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Case 3: Not-in-place operation on complex data

```
IppStatus ippiFFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep,
    Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec,
    Ipp8u* pBuffer);
```

Case 4: In-place operation on floating-point data

```
IppStatus ippiFFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

Case 5: In-place operation on complex data

```
IppStatus ippiFFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,
    const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
pFFTSpec	Pointer to the previously initialized FFT context structure.
scaleFactor	Scale factor (see Integer Result Scaling).
pBuffer	Pointer to the external work buffer, can be NULL.

Description

The function ippiFFTInv is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs an inverse FFT on each channel of the source image *pSrc* (*pSrcDst* for in-place flavors) and writes the restored image data into the corresponding channel of the destination image buffer *pDst* (*pSrcDst* for in-place flavors). The size of ROI is N x M,

it is specified by the parameters *orderX*, *orderY* calling the function ippiFFTInitAlloc. For function flavor ippiFFTInv_PackToR, the input buffer must contain data in RCPack2D format.

The inverse FFT functions use the previously initialized *pFFTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* that allows to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing FFT. On modern Intel® architectures the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function ippiFFTGetBufSize beforehand. If the external buffer is not specified (*pBuffer* is set to NULL), then the function itself allocates the memory needed for operation.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc*, *pDst*, or *pFFTSpec* pointer is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid *pFFTSpec* structure is passed.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

DFTInitAlloc

Allocates memory and fills in context data needed for the image DFT functions to operate.

Syntax

```
IppStatus ippiDFTInitAlloc_R_32s (IppiDFTSpec_R_32s** ppDFTSpec,
    IppiSize roiSize, int flag, IppHintAlgorithm hint);
IppStatus ippiDFTInitAlloc_R_32f (IppiDFTSpec_R_32f** ppDFTSpec,
    IppiSize roiSize, int flag, IppHintAlgorithm hint);
```

```
IppStatus ippiDFTInitAlloc_C_32fc (IppiDFTSpec_C_32fc** ppDFTSpec,
    IppiSize roiSize, int flag, IppHintAlgorithm hint);
```

Parameters

ppDFTSpec	Pointer to pointer to the DFT context structure being initialized.
roiSize	Size of the source and destination ROI in pixels.
flag	Flag to choose the results normalization option.
hint	Option to select the algorithmic implementation of the transform function (see Table 10-2).

Description

The function `ippiDFTInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure *pDFTSpec* needed to compute the forward and inverse DFT of a two-dimensional image data. The `ippiDFTFwd` and `ippiDFTInv` functions called with the pointer to the initialized *pDFTSpec* structure as an argument will compute the discrete Fourier transform for points in the ROI of size *roiSize*, with results normalization mode set according to *flag* (see Table 10-3), and computation algorithm indicated by *hint*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDFTSpec* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsFFTFlagErr`	Indicates an error condition if *flag* has an illegal value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

DFTFree

Deallocates memory used by the DFT context structure.

Syntax

```
IppStatus ippiDFTFree_R_32s (IppiDFTSpec_R_32s* pDFTSpec);
IppStatus ippiDFTFree_R_32f (IppiDFTSpec_R_32f* pDFTSpec);
```

```
IppStatus ippiDFTFree_C_32fc (IppiDFTSpec_C_32fc* pDFTSpec);
```

Parameters

pDFTSpec	Pointer to the DFT context structure that has to be released

Description

The function `ippiDFTFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized DFT context structure *pDFTSpec*. Call this function after your application program has finished computing the discrete Fourier transforms.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDFTSpec* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid *pDFTSpec* structure is passed.

DFTGetBufSize

Determines the size of work buffer that can be used by the DFT functions to operate.

Syntax

```
IppStatus ippiDFTGetBufSize_R_32s(IppiDFTSpec_R_32s* pDFTSpec,
    int* pSize);
IppStatus ippiDFTGetBufSize_R_32f(IppiDFTSpec_R_32f* pDFTSpec,
    int* pSize);
IppStatus ippiDFTGetBufSize_C_32fc(IppiDFTSpec_C_32fc* pDFTSpec,
    int* pSize);
```

Parameters

pDFTSpec	Pointer to the previously initialized DFT context structure.
pSize	Pointer to the size of the external work buffer to be used by the DFT functions.

Description

The function `ippiDFTGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by DFT functions with context *pDFTSpec* as a workspace during calculations.

If you want to use external buffering, call this function after *pDFTSpec* initialization. Use the computed *pSize* value as the size in bytes of an external buffer to be allocated by means of the functions `ippMalloc` or `ippsMalloc_8u`. Note that the allocated memory can be freed only by the functions `ippFree` or `ippsFree` respectively. Refer to the chapter 3 "Support Functions" of the *Intel IPP Reference Manual*, vol.1 for more details about the memory allocation functions.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDFTSpec* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid *pDFTSpec* structure is passed.

DFTFwd

Applies forward discrete Fourier transform to an image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiDFTFwd_RToPack_<mod> (const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u32s_C1RSfs
8u32s_C3RSfs
8u32s_C4RSfs
8u32s_AC4RSfs
```

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiDFTFwd_RToPack_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Case 3: Not-in-place operation on complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep,
    Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec,
    Ipp8u* pBuffer);
```

Case 4: In-place operation on floating-point data

```
IppStatus ippiDFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

Case 5: In-place operation on complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,
    const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.

srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
pDFTSpec	Pointer to the previously initialized DFT context structure.
scaleFactor	Scale factor (see Integer Result Scaling).
pBuffer	Pointer to the external work buffer, can be NULL.

Description

The function ippiDFTFwd is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP) of the size specified by the function ippiDFTInitAlloc.

This function performs a forward DFT on each channel of the source image ROI *pSrc* (*pSrcDst* for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer *pDst* (*pSrcDst* for in-place flavors).
The function flavor ippiDFTFwd_RToPack that operates on images with real data takes advantage of the symmetry property and stores the output data in RCPack2D format . It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with AC4 descriptor do not process alpha channel.
The function flavor ippiDFTFwd_CToC that operates on images with complex data performs no packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.
The forward DFT functions use the previously initialized *pDFTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* that allows to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DFT. On modern Intel® architectures the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function ippiDFTGetBufSize beforehand. If the external buffer is not specified (*pBuffer* is set to NULL), then the function itself allocates the memory needed for operation.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc*, *pDst*, or *pDFTSpec* pointer is NULL.

ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid *pDFTSpec* structure is passed.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

DFTInv

Applies an inverse DFT to complex source data and stores results in a destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiDFTInv_PackToR_<mod>(const Ipp32s* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32s8u_C1RSfs
32s8u_C3RSfs
32s8u_C4RSfs
32s8u_AC4RSfs
```

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiDFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Case 3: Not-in-place operation on complex data

```
IppStatus ippiDFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep,
    Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec,
    Ipp8u* pBuffer);
```

Case 4: In-place operation on floating-point data

```
IppStatus ippiDFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

Case 5: In-place operation on complex data

```
IppStatus ippiDFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,
    const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
pDFTSpec	Pointer to the previously initialized DFT context structure.
scaleFactor	Scale factor (see Integer Result Scaling).
pBuffer	Pointer to the external work buffer, can be NULL.

Description

The function ippiDFTInv is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP) of the size specified by the function ippiDFTInitAlloc.

This function performs an inverse DFT on each channel of the input buffer *pSrc* (*pSrcDst* for in-place flavors) and writes the restored image data into the corresponding channel of the output image buffer *pDst* (*pSrcDst* for in-place flavors).
For function flavor ippiDFTInv_PackToR, the input buffer must contain data in RCPack2D format.

The inverse DFT functions use the previously initialized *pDFTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* that allows to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DFT. On modern Intel® architectures the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function ippiDFTGetBufSize beforehand. If the external buffer is not specified (*pBuffer* is set to NULL), then the function itself allocates the memory needed for operation.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc*, *pDst*, or *pDFTSpec* pointer is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid *pDFTSpec* structure is passed.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

MulPack

Multiplies two source images in packed format.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiMulPack_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int
    dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

16s_C1RSfs	32s_C1RSfs
16s_C3RSfs	32s_C3RSfs

```
16s_C4RSfs      32s_C4RSfs
16s_AC4RSfs     32s_AC4RSfs
```

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Case 3: In-place operation on integer data

```
IppStatus ippiMulPack_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int dstSrcStep, IppiSize roiSize,
    int scaleFactor);
```

Supported values for *mod*:

```
16s_C1IRSfs   32s_C1IRSfs
16s_C3IRSfs   32s_C3IRSfs
16s_C4IRSfs   32s_C4IRSfs
16s_AC4IRSfs  32s_AC4IRSfs
```

Case 4: In-place operation on floating-point data

```
IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst,
    int dstSrcStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

Parameters

pSrc1, *pSrc2*	Pointer to the ROI in the source images.
src1Step, *src2Step*	Distance in bytes between starts of consecutive lines in the source images.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrc	Pointer to the first source image ROI for the in-place operation.
srcStep	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
pSrcDst	Pointer to the second source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiMulPack` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies corresponding pixel values of two source images, *A* and *B* represented in RCPack2D format and stores the result into the destination image *C* in packed format also. The multiplying is performed according to the following formulas:

```
ReC = ReA*ReB - ImA*ImB;
ImC = ImA*ReB + ImB*ReA
```

Not-in-place flavors multiply pixel values of ROI in the source images *pSrc1* and *pSrc2*, and store result in the *pDst*.

In-place flavors multiply pixel values of ROI in the source images *pSrc* and *pSrcDst*, and store result in the *pSrcDst*.

In case of operations on integer data, the resulting values are scaled by *scaleFactor* (see Integer Result Scaling).
Note that the functions with `AC4` descriptor do not process the alpha channel.

This function can be used in image filtering operations that include FFT transforms.
The Example 10-3 illustrates how the horizontal edges of an image can be detected. First, both the source image and the filter are transformed into the frequency domain by applying the forward FFT function which yields results in packed format. Then, actual

filtering is performed through multiplying these packed data on a point by point basis using the function `ippiMulPack`. Finally, filtered data is transformed to the time domain by means of the inverse FFT function.

Example 10-3 Using `ippiMulPack` Function in Image Filtering

```
IppStatus mulpack( void ) {
   Ipp32f tsrc[64], tflt[64], tdst[64];
   Ipp32f fsrc[64], fflt[64], fdst[64];
   IppiFFTSpec_R_32f *spec;
   const IppiSize roiSize8x8 = { 8, 8 }, roiSize3x3 = { 3, 3 };
   const Ipp32f filter[3*3] = {-1,-1,-1, 0,0,0, 1,1,1};
   ippiSet_32f_C1R( 0, tsrc, 8*sizeof(Ipp32f), roiSize8x8 );
   ippiAddC_32f_C1IR( 1, tsrc+8+1, 8*sizeof(Ipp32f), roiSize3x3 );
   ippiSet_32f_C1R( 0, tflt, 8*sizeof(Ipp32f), roiSize8x8 );
   ippiCopy_32f_C1R( filter, 3*sizeof(Ipp32f), tflt, 8*sizeof(Ipp32f),
                     roiSize3x3 );
   ippiFFTInitAlloc_R_32f( &spec, 3, 3, IPP_FFT_DIV_INV_BY_N,
                           ippAlgHintAccurate );
   ippiFFTFwd_RToPack_32f_C1R( tsrc, 8*sizeof(Ipp32f), fsrc,
                               8*sizeof(Ipp32f), spec, 0 );
   ippiFFTFwd_RToPack_32f_C1R( tflt, 8*sizeof(Ipp32f), fflt,
                               8*sizeof(Ipp32f), spec, 0 );
   ippiMulPack_32f_C1R( fsrc, 8*sizeof(Ipp32f), fflt, 8*sizeof(Ipp32f),
                        fdst, 8*4, roiSize8x8);
   ippiFFTInv_PackToR_32f_C1R( fdst, 8*sizeof(Ipp32f), tdst,
                               8*sizeof(Ipp32f), spec, 0 );
   ippiFFTFree_R_32f( spec );
   return 0;
}
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`roiSize`* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

MulPackConj

Multiplies a source image by the complex conjugate image with data in packed format and stores the result in the destination buffer in the packed format.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiMulPackConj_<mod>(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step,Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Case 2: In-place operation

```
IppStatus ippiMulPackConj_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

Parameters

pSrc1, *pSrc2*	Pointer to the ROI in the source images.
src1Step, *src2Step*	Distance in bytes between starts of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

pSrc	Pointer to the first source image ROI for the in-place operation.
srcStep	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
pSrcDst	Pointer to the second source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiMulPackConj` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies pixel values of the source image *A* by the corresponding pixel values of the complex conjugate image *A**, represented in RCPack2D format The result of the operation is written into the destination buffer in packed format also.

Not-in-place flavors multiply pixel values of ROI in the source images *pSrc1* and *pSrc2*, and store result in the *pDst*.

In-place flavors multiply pixel values of ROI in the source images *pSrc* and *pSrcDst*, and store result in the *pSrcDst*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

Magnitude

Computes magnitude of elements of a complex data image.

Syntax

```
IppStatus ippiMagnitude_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32fc32f_C1R    32fc32f_C3R
```

```
IppStatus ippiMagnitude_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
16uc16u_C1RSfs   16uc16u_C3RSfs
16sc16s_C1RSfs   16sc16s_C3RSfs
32sc32s_C1RSfs   32sc32s_C3RSfs
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiMagnitude` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes magnitude of elements of the source image *pSrc* given in complex data format, and stores results in the destination image *pDst*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
ippStsSizeErr	Indicates an error condition if width or height of images is less than or equal to zero.

MagnitudePack

Computes magnitude of elements of an image in packed format.

Syntax

```
IppStatus ippiMagnitudePack_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod*:

```
32f_C1R        32f_C3R
```

```
IppStatus ippiMagnitudePack_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    int scaleFactor);
```

Supported values for *mod*:

```
16s_C1RSfs      16s_C3RSfs
32s_C1RSfs      32s_C3RSfs
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiMagnitudePack` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes magnitude of elements of the source image *pSrc* given in RCPack2D format and stores results in the destination image *pDst*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
`ippStsSizeErr`	Indicates an error condition if width or height of images is less than or equal to zero.

Phase

Computes the phase of elements of a complex data image.

Syntax

```
IppStatus ippiPhase_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32fc32f_C1R    32fc32f_C3R
```

```
IppStatus ippiPhase_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
16uc16u_C1RSfs   16uc16u_C3RSfs
16sc16s_C1RSfs   16sc16s_C3RSfs
32sc32s_C1RSfs   32sc32s_C3RSfs
```

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiPhase` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the phase in radiansof elements of a source image *pSrc* given in complex data format, and stores results in the destination image *pDst*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
`ippStsSizeErr`	Indicates an error condition if width or height of images is less than or equal to zero.

PhasePack

Computes the phase of elements of an image in packed format.

Syntax

```
IppStatus ippiPhasePack_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod*:

```
32f_C1R        32f_C3R
```

```
IppStatus ippiPhasePack_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);
```

Supported values for *mod*:

16s_C1RSfs	32s_C1RSfs
16s_C3RSfs	32s_C3RSfs

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
scaleFactor	The factor for integer result scaling (see Integer Result Scaling).

Description

The function `ippiPhasePack` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the phase of elements of a source image *pSrc* given in RCPack2D format and stores results in the destination image *pDst*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
ippStsSizeErr	Indicates an error condition if width or height of images is less than or equal to zero.

PolarToCart

Converts an image in the polar coordinate form to Cartesian coordinate form.

Syntax

```
IppStatus ippiPolarToCart_<mod>(const Ipp32f* pSrcMagn,
    const Ipp32f* pSrcPhase, int srcStep, IppiSize roiSize, Ipp32fc*
    pDst, int dstStep);
```

Supported values for *mod*:

```
32fc_C1R        32fc_C3R
```

```
IppStatus ippiPolarToCart_<mod>(const Ipp<sourceDatatype>* pSrcMagn,
    const Ipp<sourceDatatype>* pSrcPhase, int srcStep, int
    phaseFixedPoint, IppiSize roiSize, Ipp<dstDatatype>* pDst, int
    dstStep);
```

Supported values for *mod*:

```
16sc_C1R        16sc_C3R
32sc_C1R        32sc_C3R
```

Parameters

pSrcMagn	Pointer to the buffer containing magnitudes of the source image.
pSrcPhase	Pointer to the buffer containing phase values of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source buffers.
phaseFixedPoint	Specified the position of the decimal fixed point for the phase.
roiSize	Size of the source and destination image ROI.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.

Description

The function `ippiPolarToCart` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the polar coordinate of the source image stored in the arrays of magnitudes *pSrcMagn* and phase values *pSrcPhase* to the destination image *pDst* in complex-data format (in Cartesian coordinate form).

Example 10-4 shows how to use the function `ippiPolarToCart_32fc_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
`ippStsSizeErr`	Indicates an error condition if *srcSize* has a field with value less than 1.

Example 10-4 Using the Function `ippiPolarToCart`

```
void func_polartocart()
{
    Ipp32f pSrcMagn[2*2] = {1.0, 0.0, 2.1, 3.2};

    Ipp32f pSrcPhase[2*2] = {0.0, 2.0, 1.6,-1.0};
    Ipp32fc pDst[2*2] = {0};
    int srcStep = 2*sizeof(Ipp32f);
    int dstStep = 2*sizeof(Ipp32fc);
    IppiSize roiSize = {2, 2};

    ippiPolarToCart_32fc_C1R(pSrcMagn,  pSrcPhase, srcStep, roiSize,
                                 pDst, dstStep);
}

Result: pDst - > (1.0, 0.0)  (0.0, 0.0)  (-0.1, 2.1) (1.7, -2.7)
```

PackToCplxExtend

Converts an image in packed format to a complex data image.

Syntax

```
IppStatus ippiPackToCplxExtend_32s32sc_C1R(const Ipp32s* pSrc,
    IppiSize srcSize, int srcStep, Ipp32sc* pDst, int dstStep);
```

```
IppStatus ippiPackToCplxExtend_32f32fc_C1R(const Ipp32f* pSrc,
    IppiSize srcSize, int srcStep, Ipp32fc* pDst, int dstStep);
```

Parameters

pSrc	Pointer to the source image ROI.
srcSize	Size in pixels of the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source buffer.
pDst	Pointer to the destination image buffer.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.

Description

The function `ippiPackToCplxExtend` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the source image *pSrc* in RCPack2D format to complex data format and stores the results in *pDst*, which is a matrix with complete set of the Fourier coefficients. Note that if the *pSrc* in RCPack2D format is a real array of dimensions (`NxM`), then the *pDst* is a real array of dimensions (`2xNxM`). This should be taken into account when allocating memory for the function operation.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
`ippStsSizeErr`	Indicates an error condition if *srcSize* has field with zero or negative value.

CplxExtendToPack

Converts a complex data image to an image in packed format.

Syntax

```
IppStatus ippiCplxExtendToPack_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, IppiSize srcSize, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for *mod*:

16sc16s_C1R	32sc32s_C1R	32fc32f_C1R
16sc16s_C3R	32sc32s_C3R	32fc32f_C3R

Parameters

pSrc	Pointer to the source image ROI.
srcSize	Size in pixels of the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source buffer.
pDst	Pointer to the destination image buffer.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.

Description

The function `ippiCplxExtendToPack` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the source image *pSrc* in complex data format to RCPack2D format and stores the results in *pDst*, which is a real array of dimensions (NxM). The *pSrc* is a matrix with complete set of the Fourier coefficients.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.

`ippStsSizeErr`	Indicates an error condition if *srcSize* has field with zero or negative value.

Windowing Functions

This section describes Intel IPP windowing functions used in image processing. A window is a mathematical function by which pixel values are multiplied to prepare an image for the subsequent analysis. This procedure is often called 'windowing'. In fact, a window function approaches zero towards the edges of the image avoiding strong distortions of spectral densities in the Fourier domain.

The Intel IPP provides two following types of window functions:

- Bartlett window function
- Hamming window function

These functions generate the window samples and applied them to the specified image. To obtain the window samples themselves, you should apply the desired function to the image with all pixel values set to 1.0. As the windowing operation is very time consuming, it may be useful if you want to apply the same window to the multiple images. In this case use one of the image multiplication functions (`ippiMul`) to multiply the pixel values of the image by the window samples.

WinBartlett, WinBartlettSep

Applies Bartlett window function to the image.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiWinBartlett_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R          16u_C1R         32f_C1R
```

```
IppStatus ippiWinBartlettSep_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R          16u_C1R         32f_C1R
```

Case 2: In-place operation

```
IppStatus ippiWinBartlett_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1IR         16u_C1IR        32f_C1IR
```

```
IppStatus ippiWinBartlettSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1IR         16u_C1IR        32f_C1IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiWinBartlett` and `ippiWinBartlettSep` are declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

These functions compute the Bartlett (triangle) window samples, multiply pixel values of the source image *pSrc* (*pSrcDst* for in-place flavors) with these samples, and store results in the destination image *pDst* (*pSrcDst* for in-place flavors).

The Bartlett window function for one-dimensional case with *M* elements is defined as follows:

$$w_{bartlett}(i) = \begin{cases} \frac{2i}{M-1}, & 0 \le i \le \frac{M-1}{2} \\ 2 - \frac{2i}{M-1}, & \frac{M-1}{2} < i \le M-1 \end{cases}$$

ippiWinBartlettSep. This flavor applies the window function successively to the rows and then to the columns of the image.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
ippStsSizeErr	Indicates an error condition if width or height of images is less than or equal to zero.

WinHamming, WinHammingSep

Applies Hamming window function to the image.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiWinHamming_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R          16u_C1R         32f_C1R
```

```
IppStatus ippiWinHammingSep_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R          16u_C1R         32f_C1R
```

Case 2: In-place operation

```
IppStatus ippiWinHamming_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1IR        16u_C1IR       32f_C1IR
```

```
IppStatus ippiWinHammingSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1IR        16u_C1IR       32f_C1IR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiWinHamming` and `ippiWinHammingSep` are declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

These functions compute the Bartlett (triangle) window samples, multiply pixel values of the source image *pSrc* (*pSrcDst* for in-place flavors) with these samples, and store results in the destination image *pDst* (*pSrcDst* for in-place flavors).

The Hamming window function for one-dimensional case with *M* elements is defined as follows:

$$w_{hamming}(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{len-1}\right)$$

ippiWinBartlettSep. This flavor applies the window function successively to the rows and then to the columns of the image.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *`pSrc`* or *`pDst`* pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if *`srcStep`* or *`dstStep`* value is zero or negative.
`ippStsSizeErr`	Indicates an error condition if width or height of images is less than or equal to zero.

Discrete Cosine Transforms

Discrete Cosine Transform (DCT) of a real 2D image yields output results that are also real, which eliminates the need to use packed format for storing the transformed data. However, forward and inverse DCT functions `ippDCTFwd` and `ippDCTInv` need different context data structures to be initialized and filled in prior to their use. Consequently, the required workspace buffer size is different for these functions. In case of using an external buffer, its size must be determined by previously calling the respective support function. DCT functions that use context structures implement the modified computation algorithm proposed in [Rao90].

The DCT functions `ippiDCT8x8Fwd` and `ippiDCT8x8Inv` working on a fixed 8x8 image buffer need no context data or external workspace buffers. Functions `ippiDCT8x8Inv` meet IEEE-1180 standard requirements (see [IEEE]).

Intel IPP Discrete Cosine Transform functions working on a fixed 8x8 image buffer use Feig and Winograd algorithm ([Feig92]) modified for taking advantage of SIMD instructions. For details on algorithms used in DCT transforms and for more references, see [AP922].

DCTFwdInitAlloc

Allocates memory and fills in context data needed for the forward DCT function to operate.

Syntax

```
IppStatus ippiDCTFwdInitAlloc_32f (IppiDCTFwdSpec_32f** ppDCTSpec,
    IppiSize roiSize, IppHintAlgorithm hint);
```

Parameters

ppDFTSpec	Pointer to the forward DCT context structure being initialized.
roiSize	Size of the source and destination ROI in pixels.
hint	Option to select the algorithmic implementation of the transform function.

Description

The function `ippiDCTFwdInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure *pDCTSpec* needed to compute the forward DCT of a two-dimensional image data. The `ippiDCTFwd` function called with the pointer to the initialized *pDCTSpec* structure as an argument will compute the forward discrete cosine transform for points in the ROI of size *roiSize*, using computation algorithm indicated by *hint* parameter (see Table 10-2).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDCTSpec* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

DCTInvInitAlloc

Allocates memory and fills in context data needed for the inverse DCT function to operate.

Syntax

```
IppStatus ippiDCTInvInitAlloc_32f (IppiDCTInvSpec_32f** ppDCTSpec,
    IppiSize roiSize, IppHintAlgorithm hint);
```

Parameters

ppDFTSpec	Pointer to the inverse DCT context structure being initialized.
roiSize	Size of the source and destination ROI in pixels.
hint	Option to select the algorithmic implementation of the transform function.

Description

The function `ippiDCTInvInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure *pDCTSpec* needed to compute the inverse DCT of a two-dimensional image data. The `ippiDCTInv` function called with the pointer to

the initialized *pDCTSpec* structure as an argument will compute the inverse discrete cosine transform for points in the ROI of size *roiSize*, using computation algorithm indicated by *hint* (see Table 10-2).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pDCTSpec* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

DCTFwdFree

Deallocates memory used by the forward DCT context structure.

Syntax

```
IppStatus ippiDCTFwdFree_32f (IppiDCTFwdSpec_32f* pDCTSpec);
```

Parameters

pDCTSpec	Pointer to the forward DCT context structure that has to be released.

Description

The function ippiDCTFwdFree is declared in the ippi.h file. This function releases memory used by the previously initialized forward DCT context structure *pDCTSpec*. Call this function after your application program has finished computing the forward discrete cosine transforms.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pDCTSpec* pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid *pDCTSpec* structure is passed.

DCTInvFree

Deallocates memory used by the inverse DCT context structure.

Syntax

```
IppStatus ippiDCTInvFree_32f (IppiDCTInvSpec_32f* pDCTSpec);
```

Parameters

pDCTSpec	Pointer to the inverse DCT context structure that has to be released.

Description

The function `ippiDCTInvFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized inverse DCT context structure *pDCTSpec*. Call this function after your application program has finished computing the inverse discrete cosine transforms.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDCTSpec* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid *pDCTSpec* structure is passed.

DCTFwdGetBufSize

Determines the size of work buffer can be used by the forward DCT function to operate.

Syntax

```
IppStatus ippiDCTFwdGetBufSize_32f (IppiDCTFwdSpec_32f* pDCTSpec,
    int* pSize);
```

Parameters

pDCTSpec	Pointer to the previously initialized forward DCT context structure.

pSize	Pointer to the size of the external work buffer to be used by the forward DCT function.

Description

The function `ippiDCTFwdGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by the forward DCT function with context *pDCTSpec* as a workspace during calculations.
If you want to use external buffering, call this function after *pDCTSpec* initialization.
Use the computed *pSize* value as the size in bytes of an external buffer to be allocated by means of the functions `ippMalloc` or `ippsMalloc_8u`. Note that the allocated memory can be freed only by the functions `ippFree` or `ippsFree` respectively. Refer to the chapter 3 "Support Functions" of the *Intel IPP Reference Manual*, vol.1 for more details about the memory allocation functions.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDCTSpec* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid *pDCTSpec* structure is passed.

DCTInvGetBufSize

Determines the size of work buffer that can be used by the inverse DCT function to operate.

Syntax

```
IppStatus ippiDCTInvGetBufSize_32f (IppiDCTInvSpec_32f* pDCTSpec,
    int* pSize);
```

Parameters

pDCTSpec	Pointer to the previously initialized inverse DCT context structure.
pSize	Pointer to the size of the external work buffer to be used by the inverse DCT function.

Description

The function `ippiDCTInvGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by the inverse DCT function with context *pDCTSpec* as a workspace during calculations.

If you want to use external buffering, call this function after *pDCTSpec* initialization. Use the computed *pSize* value as the size in bytes of an external buffer to be allocated by means of the functions `ippMalloc` or `ippsMalloc_8u`. Note that the allocated memory can be freed only by the functions `ippFree` or `ippsFree` respectively. Refer to the chapter 3 "Support Functions" of the *Intel IPP Reference Manual*, vol.1 for more details about the memory allocation functions.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pDCTSpec* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid *pDCTSpec* structure is passed.

DCTFwd

Applies a forward discrete cosine transform to an image.

Syntax

```
IppStatus ippiDCTFwd_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, const IppiDCTFwdSpec_32f* pDCTSpec, Ipp8u*
    pBuffer);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pDCTSpec	Pointer to the previously initialized forward DCT context structure.
pBuffer	Pointer to the external work buffer, can be NULL.

Description

The function ippiDCTFwd is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP) of the size that is specified by the function ippiDCTFwdInitAlloc.

This function performs a forward DCT on each channel of the source image *pSrc* and writes the result into the corresponding channel of the destination image buffer *pDst*. Note that the function flavor with AC4 descriptor does not process alpha channel.
This function uses the previously initialized *pDCTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* that allows to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing forward DCT. On modern Intel® architectures the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function ippiDCTFwdGetBufSize beforehand. If the external buffer is not specified (*pBuffer* is set to NULL), then the function itself allocates the memory needed for operation.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc*, *pDst*, or *pDCTSpec* pointer is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid *pDCTSpec* structure is passed.

ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

DCTInv

Applies an inverse discrete cosine transform to an image.

Syntax

```
IppStatus ippiDCTInv_<mod> (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDCTInvSpec_32f* pDCTSpec, Ipp8u*
    pBuffer);
```

Supported values for *mod* :

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pDCTSpec	Pointer to the previously initialized inverse DCT context structure.
pBuffer	Pointer to the external work buffer, can be NULL.

Description

The function ippiDCTInv is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP) of the size that is specified by the function ippiDCTInvInitAlloc.

This function performs an inverse DCT on each channel of the input image *pSrc* and writes the result into the corresponding channel of the output image buffer *pDst*. Note that the function flavor with AC4 descriptor does not process alpha channel.
This function uses the previously initialized *pDCTSpec* context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer *pBuffer* that allows to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing inverse DCT. On modern Intel® architectures the use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function ippiDCTInvGetBufSize beforehand. If the external buffer is not specified (*pBuffer* is set to NULL), then the function itself allocates the memory needed for operation.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc*, *pDst*, or *pDCTSpec* pointer is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* value is zero or negative.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid *pDCTSpec* structure is passed.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

DCT8x8Fwd

Performs a forward DCT on a 2D buffer of 8x8 size.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<datatype>* pSrc,
    Ipp<datatype>* pDst);
```

Supported values for *mod*:

```
16s_C1      32f_C1
```

Case 2: Not-in-place operation with ROI

```
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst);
```

Supported values for *mod*:

```
16s_C1R      8u16s_C1R
```

Case 3: In-place operation

```
IppStatus ippiDCT8x8Fwd_<mod>(Ipp<datatype>* pSrcDst );
```

Supported values for *mod*:

```
16s_C1I      32f_C1I
```

Parameters

pSrc	Pointer to the source image buffer.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer for operations with ROI.
pDst	Pointer to the destination image buffer.
pSrcDst	Pointer to the source and destination image for in-place operations.

Description

The function `ippiDCT8x8Fwd` is declared in the `ippi.h` file. Some flavors operate with ROI (see Regions of Interest in Intel IPP).

This function computes the forward discrete cosine transform of short integer or floating-point data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

The code Example 10-5 illustrates the use of `ippiDCT8x8Fwd` function.

Example 10-5 Forward DCT of 8x8 Size

```
IppStatus dct16s( void ) {
   Ipp16s x[64] = {0};
   IppiSize roi = {8,8};
   int i;
   for( i=0; i<8; ++i ) {
      ippiSet_16s_C1R( (Ipp16s)i, x+8*i+i, 8*sizeof(Ipp16s), roi );
       --roi.width;
       --roi.height;
   }
   return ippiDCT8x8Fwd_16s_C1I( x );
}
```

Example 10-5 Forward DCT of 8x8 Size (continued)

The destination image *x* contains:

```
18 -9 -2 -1 -1  0  0  0
-9  7  0  0  0  0  0  0
-2  0  2  0  0  0  0  0
-1  0  0  1  0  0  0  0
 0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is NULL.
`ippStsStepErr`	Indicates an error condition if *srcStep* value is zero or negative.

DCT8x8Inv, DCT8x8Inv_A10

Performs an inverse DCT on a 2D buffer of 8x8 size.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiDCT8x8Inv_<mod>(const Ipp<datatype>* pSrc,
    Ipp<datatype>* pDst);
```

Supported values for *mod*:

```
16s_C1      32f_C1
```

```
IppStatus ippiDCT8x8Inv_A10_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

Case 2: Not-in-place operation with ROI

```
IppStatus ippiDCT8x8Inv_<mod>(const Ipp<srcDatatype>* pSrc,
    Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for *mod*:

```
16s_C1R     16s8u_C1R
```

Case 3: In-place operation

```
IppStatus ippiDCT8x8Inv_<mod>(Ipp<datatype>* pSrcDst);
```

Supported values for *mod*:

```
16s_C1I     32f_C1I
```

```
IppStatus ippiDCT8x8Inv_A10_16s_C1I(Ipp16s* pSrcDst);
```

Parameters

pSrc	Pointer to the source image.
pDst	Pointer to the destination buffer.
dstStep	Distance in bytes between starts of consecutive lines in the destination buffer for operations with ROI.
pSrcDst	Pointer to the source and destination image for in-place operations.

Description

The function `ippiDCT8x8Inv` is declared in the `ippi.h` file. Some flavors operate with ROI (see Regions of Interest in Intel IPP).

This function computes the inverse discrete cosine transform of data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

CAUTION. Source data for 16s functions must be the result of the forward discrete cosine transform of data from the range [-512, 511] for flavors with `A10` modifier (`ippiDCT8x8Inv_A10`), and from the range [-256, 255] for flavors without `A10` modifier (`ippiDCT8x8Inv`); they can not be arbitrary data from the range [-32768, 32767].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pDst*, or *pSrcDst* pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* value is zero or negative.

DCT8x8FwdLS

Performs a forward DCT on a 2D buffer of 8x8 size with prior data conversion and level shift.

Syntax

```
IppStatus ippiDCT8x8FwdLS_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, Ipp16s addVal);
```

Parameters

pSrc	Pointer to the source image buffer.
pDst	Pointer to the destination buffer.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
addVal	The level shift value.

Description

The function `ippiDCT8x8FwdLS` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function first converts data in the buffer *pSrc* from unsigned `Ipp8u` type to the signed `Ipp16s` type and then performs level shift operation by adding the constant value *addVal* to each sample. After that, the function performs the forward discrete cosine transform of the modified data. The result is stored in *pDst*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* value is zero or negative.

DCT8x8InvLSClip

Performs an inverse DCT on a 2D buffer of 8x8 size with further data conversion and level shift.

Syntax

```
IppStatus ippiDCT8x8InvLSClip_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst,
    int dstStep, Ipp16s addVal, Ipp8u clipDown, Ipp8u clipUp);
```

Parameters

pSrc	Pointer to the source image buffer.
pDst	Pointer to the destination image buffer.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
addVal	The level shift value.
clipDown	The lower bound for the range of output values.
clipUp	The upper bound for the range of output values.

Description

The function `ippiDCT8x8InvLSClip` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function performs the inverse discrete cosine transform of the buffer *pSrc*. After completing the DCT, this function performs level shift operation by adding the constant value *addVal* to each sample. Finally, the function converts data from the signed `Ipp16s` type to the unsigned `Ipp8u` type. The output data are clipped to the range [*clipDown* .. *clipUp*]. The result is stored in the destination buffer *pDst*.

CAUTION. Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they can not be arbitrary data from the range [-32768, 32767].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsStepErr	Indicates an error condition if *dstStep* value is zero or negative.

DCT8x8Inv_2x2, DCT8x8Inv_4x4

Perform an inverse DCT on a top left quadrant of size 2x2 or 4x4 in the 2D buffer of size 8x8.

Syntax

```
IppStatus ippiDCT8x8Inv_2x2_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8Inv_4x4_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8Inv_2x2_16s_C1I(Ipp16s* pSrcDst);
IppStatus ippiDCT8x8Inv_4x4_16s_C1I(Ipp16s* pSrcDst);
```

Parameters

pSrc	Pointer to the source image.
pDst	Pointer to the destination buffer.
pSrcDst	Pointer to the source and destination buffer for in-place operations.

Description

The functions ippiDCT8x8Inv_2x2 and ippiDCT8x8Inv_4x4 are declared in the ippi.h file. These functions compute the inverse discrete cosine transform of non-zero elements in the top left quadrant of size 2x2 or 4x4 in the 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

CAUTION. Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they can not be arbitrary data from the range [-32768, 32767].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

DCT8x8To2x2Inv, DCT8x8To4x4Inv

Perform an inverse DCT on a 2D buffer of 8x8 size with further downsampling to 2x2 or 4x4 size.

Syntax

```
IppStatus ippiDCT8x8To2x2Inv_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8To4x4Inv_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8To2x2Inv_16s_C1I(Ipp16s* pSrcDst);
IppStatus ippiDCT8x8To4x4Inv_16s_C1I(Ipp16s* pSrcDst);
```

Parameters

pSrc	Pointer to the source image.
pDst	Pointer to the destination buffer.
pSrcDst	Pointer to the source and destination buffer for in-place operations.

Description

The functions `ippiDCT8x8To2x2Inv` and `ippiDCT8x8To4x4Inv` are declared in the `ippi.h` file. These functions compute the inverse discrete cosine transform of the 2D buffer *pSrc* of 8x8 size. Then the functions perform downsampling of the result by averaging to the destination buffer *pDst* of size 2x2 or 4x4.

In-place flavors of the functions perform operations on the source and destination buffer *pSrcDst*.

CAUTION. Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they can not be arbitrary data from the range [-32768, 32767].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

Image Statistics Functions 11

This chapter describes the Intel® IPP image processing functions that can be used to compute the following statistical parameters of an image:

- sum, integrals, mean and standard deviation of pixel values
- intensity histogram of pixel values
- minimum and maximum pixel values
- spatial and central moments of order 0 to 3
- the infinity, *L1*, and *L2* norms of the image pixel values and of the differences between pixel values of two images
- relative error values for the infinity, *L1*, and *L2* norms of differences between pixel values of two images
- universal image quality index
- proximity measures of an image and a template (another image).

Table 11-1 lists the image statistics functions.:

Table 11-1 Image Statistics Functions

Function Base Name	Operation
`Sum`	Computes the sum of image pixel values
`Integral`	Transforms an image to the integral representation
`SqrIntegral`	Transforms an image to integral and integral of pixel squares representations
`TiltedIntegral`	Transforms an image to the tilted integral representation
`TiltedSqrIntegral`	Transforms an image to tilted integral and tilted integral of pixel squares representations
`Mean`	Computes the mean of image pixel values

Table 11-1 Image Statistics Functions (continued)

Function Base Name	Operation
Mean_StdDev	Computes the mean and standard deviation of image pixel values
RectStdDev	Computes the standard deviation of the integral images.
TiltedRectStdDev	Computes the standard deviation of the tilted integral images.
HistogramRange	Computes the intensity histogram of an image
HistogramEven	Computes the intensity histogram of an image with equal bins
CountInRange	Counts the number of pixels within the given intensity range
Minimum and Maximum Operations	
Min	Computes the minimum of image pixel values
MinIndx	Computes the minimum of image pixel values, and retrieves the coordinates of pixels with minimal intensity values
Max	Computes the maximum of image pixel values
MaxIndx	Computes the maximum of image pixel values, and retrieves the coordinates of pixels with maximal intensity values
MinMax	Computes the minimum and maximum of image pixel values
MinMaxIndx	Calculates minimum and maximum pixel values and their indexes
MaxEvery	Computes maximum value for each pair of pixels of two images
MinEvery	Computes minimum value for each pair of pixels of two images
FindPeaks3x3GetBufferSize	Computes the size of the working buffer for the peak search
FindPeaks3x3	Finds coordinates of peaks (maximums or minimums) with absolute value exceeding threshold value
Moments	
MomentInitAlloc	Allocates memory and initializes the moment context structure

Table 11-1 Image Statistics Functions (continued)

Function Base Name	Operation
MomentFree	Deallocates the previously initialized structure that stores image moments
MomentGetStateSize	Computes the size of the external buffer for the moment context structure
MomentInit	Initializes the moment context structure
Moments	Computes all image moments of order 0 to 3 and Hu moment invariants
GetSpatialMoment	Retrieves image spatial moment computed by ippiMoments
GetCentralMoment	Retrieves image central moment computed by ippiMoments
GetNormalizedSpatialMoment	Retrieves the normalized value of the image spatial moment computed by ippiMoments
GetNormalizedCentralMoment	Retrieves the normalized value of the image central moment computed by ippiMoments
GetHuMoments	Retrieves image Hu moment invariants computed by ippiMoments
Norms	
Norm_Inf	Computes the infinity norm of image pixel values
Norm_L1	Computes the L1- norm of image pixel values
Norm_L2	Computes the L2- norm of image pixel values
NormDiff_Inf	Computes the infinity norm of differences between pixel values of two images
NormDiff_L1	Computes the L1- norm of differences between pixel values of two images
NormDiff_L2	Computes the L2- norm of differences between pixel values of two images
NormRel_Inf	Computes the relative error for the infinity norm of differences between pixel values of two images
NormRel_L1	Computes the relative error for the L1- norm of differences between pixel values of two images.
NormRel_L2	Computes the relative error for the L2- norm of differences between pixel values of two images.
Image Quality Index	
QualityIndex	Computes the universal image quality index

Table 11-1 Image Statistics Functions (continued)

Function Base Name	Operation
Proximity Measures	
SqrDistanceFull_Norm	Computes normalized full Euclidean distance between an image and a template.
SqrDistanceSame_Norm	Computes normalized Euclidean distance between an image and a template.
SqrDistanceValid_Norm	Computes normalized valid Euclidean distance between an image and a template.
CrossCorrFull_Norm	Computes normalized full cross-correlation between an image and a template.
CrossCorrSame_Norm	Computes normalized cross-correlation between an image and a template.
CrossCorrValid_Norm	Computes normalized valid cross-correlation between an image and a template.
CrossCorrValid	Computes valid cross-correlation between an image and a template.
CrossCorrFull_NormLevel	Computes normalized full correlation coefficients between an image and a template.
CrossCorrSame_NormLevel	Computes normalized correlation coefficients between an image and a template.
CrossCorrValid_NormLevel	Computes normalized valid correlation coefficients between an image and a template.

Sum

Computes the sum of image pixel values.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pSum);
```

Supported values for *mod*:

```
8u_C1R      16u_C1R     16s_C1R
```

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiSum_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pSum, IppHintAlgorithm hint);
```

Case 3: Operation on multi-channel integer data

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f sum[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R
8u_AC4R     16u_AC4R    16s_AC4R
```

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f sum[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R
```

Case 4: Operation on multi-channel floating-point data

```
IppStatus ippiSum_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f sum[3], IppHintAlgorithm hint);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiSum_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f sum[4], IppHintAlgorithm hint);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
pSum	Pointer to the computed sum of pixel values.
sum	Array containing computed sums of channel values of pixels in the source buffer.
hint	Option to select the algorithmic implementation of the function.

Description

The function `ippiSum` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the sum of pixel values *pSum* for the source image *pSrc* using algorithm indicated by the *hint* argument (see Table 11-3). In case of a multi-channel image, the sum is computed over each channel and stored in the array *sum*.

The code Example 11-1 demonstrates the use of `ippiSum` function:

Example 11-1 Summing Up Pixel Values of an Image

```
IppStatus sum( void ) {
   Ipp64f sum;
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   ippiSet_8u_C1R( 1, x, 5, roi );
   return ippiSum_8u_C1R( x, 5, roi, &sum);
}
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pSum* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

Integral

Transforms an image to the integral representation.

Syntax

```
IppStatus ippiIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
    int dstStep, IppiSize roiSize, Ipp32s val);
IppStatus ippiIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, Ipp32f val);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination integral image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of source and destination image ROI in pixels.
val	The value to add to *pDs*t image pixels

Description

The function `ippiIntegral` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function transform a source image *pSrc* to the integral image *pDst*. Pixel values of the destination image *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{k<i} \sum_{l<j} pSrc[k, l]$$

where *i*,*j* are coordinates of the destination image pixels (see Figure 11-1) varying in the range *i* = 1 ,..., *roiSize.height*, *j* = 0,..., *roiSize.width*. Pixel values of zero row and column of *pDst* (*i*=0) is set to *val*.

The size of the destination images is (*roiSize.width* + 1) x (*roiSize.height* + 1).

Figure 11-1 shows what pixels (red circles) of the source image are used in computation new pixel values in the *i*,*j* coordinates.

For large images the result of summation can exceed the upper bound of the output data type. Table 11-2 lists the maximum image size for different function flavors and values. The code Example 11-2 demonstrates how to use the function `ippiIntegral`.

Figure 11-1 Operation of the Integral and TiltedIntegral functions

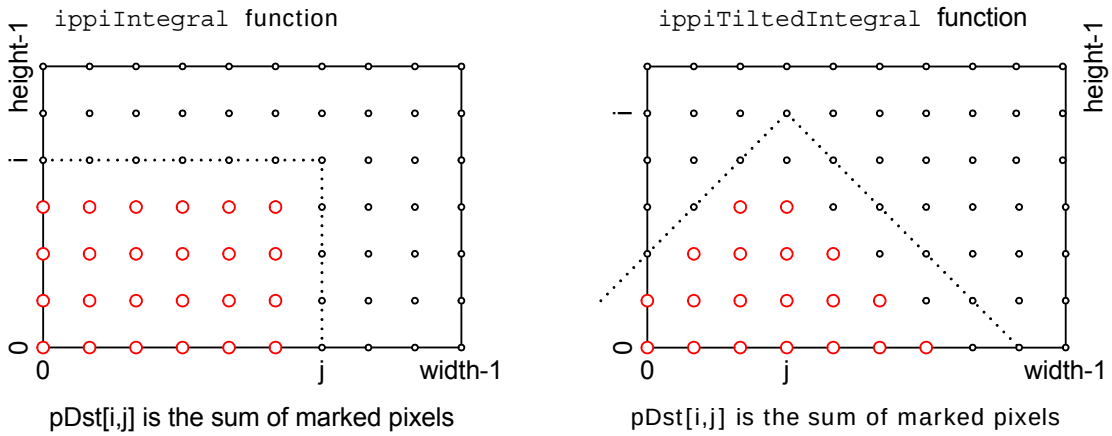


Table 11-2 Maximum Image Size for Integral Functions

Function Flavor	Value val	Maximum Image Size
ippiIntegral_8u32s_C1R	0	$(2^{31}-1)/255$
	-2^{31}	$2^{32}/255$
ippiIntegral_8u32f_C1R	0	$2^{24}/255$
	-2^{24}	$(2^{25}+1)/255$

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pSrc* or *pDst* is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* is less than *roiSize.width* * *<pixelSize>*, or *dstStep* is less than *(roiSize.width+1)* * *<pixelSize>* .
ippStsNotEvenStepErr	Indicates an error condition if one *dstStep* is not divisible by 4.

Example 11-2 Using the function ippiIntegral

```
void func_inegral()
{
    Ipp8u pSrc[5*4];
    Ipp32s pDst[6*5];
    IppiSize ROI = {5,4};
    ippiSet_8u_C1R(1,pSrc,5,ROI);
    Ipp32s val = 1;
    ippiIntegral_8u32s_C1R(pSrc, 5, pDst, 6*4, ROI, val);
}
result:
pSrc -> 1 1 1 1 1            pDst -> 1 1 1 1  1  1
        1 1 1 1 1                    1 2 3 4  5  6
        1 1 1 1 1                    1 3 5 7  9  11
        1 1 1 1 1                    1 4 7 10 13 16
                                     1 5 9 13 17 21
```

SqrIntegral

Transforms an image to integral and integral of pixel squares representations.

Syntax

```
IppStatus ippiSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val,
    Ipp32s valSqr);
IppStatus ippiSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val,
    Ipp64f valSqr);
IppStatus ippiSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val,
    Ipp64f valSqr);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination integral image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSqr	Pointer to the ROI of the destination integral image of pixel squares.
sqrStep	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
roiSize	Size of source image ROI in pixels.
val	The value to add to *pDst* image pixels.
valSqr	The value to add to *pSqr* image pixels

Description

The function `ippiSqrIntegral` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function builds two destination image: integral image *pDst* and integral image of pixel squares *pSqr*.
Pixel values of *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{k<i} \sum_{l<j} pSrc[k, l]$$

Pixel values of *pSqr* are computed using pixel values of the source image *pSrc* and the specified value *valSqr* in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{k<i} \sum_{l<j} pSrc[k, l]^2$$

where *i*,*j* are coordinates of the destination image pixels (see Figure 11-1)varying in the range *i* = 1, ..., *roiSize.height*, *j* = 0,..., *roiSize.width*. Pixel values of zero row and column are set to *val* for *pDst*, and to *valSqr* for *pSqr*. The size of both destination images is (*roiSize.width* + 1) x (*roiSize.height* + 1).

Figure 11-1 shows what pixels (red circles) of the source image are used in computation new pixel values in the *i*,*j* coordinates.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* is less than *roiSize.width* * *<pixelSize>*, or *dstStep* or *sqrStep* is less than *(roiSize.width+1)* * *<pixelSize>* .
ippStsNotEvenStepErr	Indicates an error condition if *dstStep* is not divisible by 4, or *sqrStep* is not divisible by 8.

TiltedIntegral

Transforms an image to the tilted integral representation.

Syntax

```
IppStatus ippiTiltedIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize, Ipp32s val);
IppStatus ippiTiltedIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f val);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination integral image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of source image ROI in pixels.
val	The value to add to pDst image pixels

Description

The function `ippiTiltedIntegral` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function transform a source image *pSrc* to the tilted integral image *pDst*. Pixel values of the destination image *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+l \le i+j-2 \\ k-l \le i-j+1}} pSrc[k, l]$$

where *i*,*j* are coordinates of the destination image pixels (see Figure 11-1)varying in the range *i* = 2 ,..., *roiSize.height* + 1, *j* = 0,..., *roiSize.width* + 1. Pixel values of rows 0 and 1 of the destination image *pDst* (*i*=0) is set to *val*.

The size of the destination images is (*roiSize.width* + 2) x (*roiSize.height* + 2).

Figure 11-1 shows what pixels (red circles) of the source image are used in computation new pixel values in the *i*,*j* coordinates.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pSrc* or *pDst* is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* is less than *roiSize.width* * *<pixelSize>*, or *dstStep* is less than *(roiSize.width+2)* * *<pixelSize>* .
`ippStsNotEvenStepErr`	Indicates an error condition if one *dstStep* is not divisible by 4.

TiltedSqrIntegral

Transforms an image to tilted integral and tilted integral of pixel squares representations.

Syntax

```
IppStatus ippiTiltedSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize,
    Ipp32s val, Ipp32s valSqr);
IppStatus ippiTiltedSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize,
    Ipp32s val, Ipp64f valSqr);
IppStatus ippiTiltedSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize,
    Ipp32f val, Ipp64f valSqr);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination integral image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSqr	Pointer to the ROI of the destination integral image of pixel squares.
sqrStep	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
roiSize	Size of source image ROI in pixels.
val	The value to add to *pDst* image pixels.
valSqr	The value to add to *pSqr* image pixels

Description

The function `ippiTitledSqrIntegral` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function builds two destination image: tilted integral image *pDst* and tilted integral image of pixel squares *pSqr*.

Pixel values of *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+l \le i+j-2 \\ k-l \le i-j+1}} pSrc[k, l]$$

Pixel values of *pSqr* are computed using pixel values of the source image *pSrc* and the specified value *valSqr* in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{\substack{k+l \le i+j-2 \\ k-l \le i-j+1}} pSrc[k, l]^2$$

where *i*,*j* are coordinates of the destination image pixels (see Figure 11-1)varying in the range *i* = 2 ,..., *roiSize.height*, *j* = 0,..., *roiSize.width*. Pixel values of zero and first rows (*i*=0,1) are set to *val* for *pDst*, and to *valSqr* for *pSqr*. The size of both destination images is (*roiSize.width* + 2) x (*roiSize.height* + 2).

Figure 11-1 shows what pixels (red circles) of the source image are used in computation new pixel values in the *i*,*j* coordinates.

Return Values

ippStsNoEr	Indicates no error.
ippStsNullPtrErr	ndicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* is less than *roiSize.width* * *<pixelSize>*, or *dstStep* or *sqrStep* is less than *(roiSize.width+2)* * *<pixelSize>* .
ippStsNotEvenStepErr	Indicates an error condition if *dstStep* is not divisible by 4, or *sqrStep* is not divisible by 8.

Mean

Computes the mean of image pixel values.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pMean);
```

Supported values for *mod*:

```
8u_C1R      16u_C1R     16s_C1R
```

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiMean_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pMean, IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean);
```

Supported values for *mod*:

```
8u_C1MR     8s_C1MR     16u_C1MR    32f_C1MR
```

Case 4: Operation on multi-channel integer data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f mean[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R
8u_AC4R     16u_AC4R    16s_AC4R
```

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f mean[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R
```

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiMean_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f mean[3], IppHintAlgorithm hint);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiMean_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f mean[4], IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean);
```

Supported values for *mod* :

```
8u_C3CMR    8s_C3CMR    16u_C3CMR    32f_C3CMR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pMean	Pointer to the computed mean of pixel values.
mean	Array containing computed mean values for each channel of a multi-channel image.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiMean` that perform masked operations are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. This function operates with ROI (see Regions of Interest in Intel IPP). It computes the mean (average) of pixel values *pMean* for the source image *pSrc*. Computation algorithm is specified by the *hint* argument (see Table 11-3). For non-masked operations on a multi-channel image (**Case 4, 5**), the mean is computed over each channel and stored in the array *mean*.

In the mask multi-channel mode (**Case 6**), the mean is computed for a single channel of interest specified by *coi*.
The following code example shows how to use the `ippiMean` function:

Example 11-3 Computing the Mean of Pixel Values

```
IppStatus mean( void ) {
   Ipp64f mean;
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   ippiSet_8u_C1R( 3, x, 5, roi );
   return ippiMean_8u_C1R( x, 5, roi, &mean );
}
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition in mask mode, if *srcStep* or *maskStep* is less than *roiSize.width* * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
`ippStsCOIErr`	Indicates an error when *coi* is not 1, 2, or 3.

Mean_StdDev

Computes the mean and standard deviation of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for *mod*:

8u_C1R 8s_C1R 16u_C1R 32f_C1R

Case 2: Masked operation on one-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean,
    Ipp64f* pStdDev);
```

Supported values for *mod*:

8u_C1MR 8s_C1MR 16u_C1MR 32f_C1MR

Case 3: Operation on multi-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for *mod*:

8u_C3CR 8s_C3CR 16u_C3CR 32f_C3CR

Case 4: Masked operation on multi-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for *mod*:

8u_C3CMR 8s_C3CMR 16u_C3CMR 32f_C3CMR

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pMean	Pointer to the computed mean of pixel values.
pStdDev	Pointer to the computed standard deviation of pixel values in the image.

Description

The function `ippiMean_StdDev` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the mean and standard deviation of pixel values in the ROI of the source image *pSrc*. In the mask mode, the computation is done over pixels selected by nonzero mask values.

In the multi-channel mode, the mean is computed for a single channel of interest specified by *coi*.

If any of the parameters *pMean* or *pStdDev* is not required, the zero pointer is to be passed to the corresponding parameter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when *pSrc* or *pMask* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *maskStep* is less than *roiSize.width* * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error condition if steps for floating-point images cannot be divided by 4.
`ippStsCOIErr`	Indicates an error when *coi* is not 1, 2, or 3.

RectStdDev

Computes the standard deviation of the integral images.

Syntax

```
IppStatus ippiRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const
    Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect);
IppStatus ippiRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep,
    const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect, int scaleFactor);
IppStatus ippiRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep,
    const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect);
```

Parameters

pSrc	Pointer to the ROI in the source integral image.
srcStep	Distance in bytes between starts of consecutive lines in the source integral image.
pSqr	Pointer to the ROI in the source integral image of pixel squares.
sqrStep	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of destination image ROI in pixels.
rect	Rectangular window.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiRectStdDev` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the standard deviation for each pixel in the rectangular window *rect* using the integral image *pSrc* and integral image of pixel squares *pSqr*. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{max\left(0, \frac{sumSqr \cdot numPix - sum^2}{numPix^2}\right)}$$

where *i*,*j* are coordinates of the destination image pixels varying in the range *i* = 0 ,..., *roiSize.height* - 1, *j* = 0,..., *roiSize.width* - 1;

sum = *pSrc*[*i* + *rect.y* + *rect.height*, *j* + *rect.x* + *rect.width*] -
- *pSrc*[*i* + *rect.y*, *j* + *rect.x* + *rect.width*] -
- *pSrc*[*i* + *rect.y* + *rect.height*, *j* + *rect.x*] + *pSrc*[*i* + *rect.y*, *j* + *rect.x*];

sumSqr = *pSqr*[*i* + *rect.y* + *rect.height*, *j* + *rect.x* + *rect.width*] -
- *pSqr*[*i* + *rect.y*, *j* + *rect.x* + *rect.width*] -
- *pSqr*[*i* + *rect.y* + *rect.height*, *j* + *rect.x*] + *pSqr*[*i* + *rect.y*, *j* + *rect.x*];

numPix = *rect.height* * *rect.width*.

The minimum size of each source images *pSrc* and *pSqr* should be (*roiSize.width* + *rect.x* + *rect.width*) x (*roiSize.height* + *rect.y* + *rect.height*).

The source images *pSrc* and *pSqr* can be obtained by using the functions `ippiIntegral` or `ippiSqrIntegral`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsSizeErr`	Indicates an error condition if *rect.width* or *rect.height* is less than or equal to zero, or if *rect.x* or *rect.y* is less than zero.

ippStsStepErr	Indicates an error condition if *srcStep* or *sqrStep* is less than *(roiSize.width+rect.x+rect.width+1) * <pixelSize>*, or *dstStep* is less than *roiSize.width * <pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if *sqrStep* is not divisible by 8, or one of *pSrc* and *dstStep* is not divisible by 4.

TiltedRectStdDev

Computes the standard deviation of the tilted integral images.

Syntax

```
IppStatus ippiTiltedRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep,
    const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect);
IppStatus ippiTiltedRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int
    srcStep, const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep,
    IppiSize roiSize, IppiRect rect, int scaleFactor);
IppStatus ippiTiltedRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int
    srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep,
    IppiSize roiSize, IppiRect rect);
```

Parameters

pSrc	Pointer to the ROI in the source integral image.
srcStep	Distance in bytes between starts of consecutive lines in the source integral image.
pSqr	Pointer to the ROI in the source integral image of pixel squares.
sqrStep	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of destination image ROI in pixels.
rect	Rectangular window.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiTiltedRectStdDev` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the standard deviation for each pixel in the rectangular window *rect* using the tilted integral image *pSrc* and tilted integral image of pixel squares *pSqr*. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{max\left(0, \frac{sumSqr \cdot numPix - sum^2}{numPix^2}\right)}$$

where *i*,*j* are coordinates of the destination image pixels varying in the range *i* = 0 ,..., *roiSize.height* - 1, *j* = 0,..., *roiSize.width* - 1;

```
sum = pSrc[i + rect.x - rect.y + rect.height + rect.width,
          j +  rect.x + rect.y - rect.height + rect.width] -
      - pSrc[i + rect.x - rect.y + rect.width, j + rect.x + rect.y + rect.width]
      -
      - pSrc[i + rect.x - rect.y + rect.height, j + rect.x - rect.y -
      rect.height] + pSrc[i + rect.x - rect.y, j + rect.x + rect.y];
```

```
sumSqr = pSqr[i + rect.x - rect.y + rect.height + rect.width,
             j + rect.x + rect.y - rect.height + rect.width] -
      - pSqr[i + rect.x - rect.y + rect.width, j + rect.x + rect.y +
      rect.width] -
      - pSqr[i + rect.x - rect.y + rect.height, j + rect.x - rect.y -
      rect.height] + pSqr[i + rect.x - rect.y, j + rect.x + rect.y];
```

numPix = 2 * *rect.height* * *rect.width*.

The minimum size of each source images *pSrc* and *pSqr* should be (*roiSize.width* + *rect.height* + *rect.width* - 2) x (*roiSize.height* + *rect.x* + *rect.y* + *rect.height* + *rect.width* - 2).

The source images *pSrc* and *pSqr* can be obtained by using the functions `ippiTiltedIntegral` or `ippiTiltedSqrIntegral`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsSizeErr	Indicates an error condition if *rect.width* or *rect.height* is less than or equal to zero, or if *rect.x* or *rect.y* is less than zero.
ippStsStepErr	Indicates an error condition if *srcStep* or *sqrStep* is less than *(roiSize.width+rect.x+rect.width+1) * <pixelSize>*, or *dstStep* is less than *roiSize.width * <pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if *sqrStep* is not divisible by 8, or one of *pSrc* and *dstStep* is not divisible by 4.

HistogramRange

Computes the intensity histogram of an image.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiHistogramRange_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist, const Ipp32s* pLevels,
    int nLevels);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R      16s_C1R
```

Case 2: Operation on multi-channel integer data

```
IppStatus ippiHistogramRange_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R      16s_C3R
8u_AC4R     16u_AC4R     16s_AC4R
```

```
IppStatus ippiHistogramRange_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[4], const Ipp32s* pLevels[4],
    int nLevels[4]);
```

Supported values for *mod* :

8u_C4R 16u_C4R 16s_C4R

Case 3: Operation on one-channel floating-point data

```
IppStatus ippiHistogramRange_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist, const Ipp32f* pLevels, int nLevels);
```

Case 4: Operation on multi-channel floating-point data

```
IppStatus ippiHistogramRange_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3],  const Ipp32f* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

32f_C3R

32f_AC4R

```
IppStatus ippiHistogramRange_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[4], const Ipp32f* pLevels[4],
    int nLevels[4]);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
pHist	Pointer to the computed histogram. In case of multi-channel data, *pHist* is an array of pointers to the histogram for each channel.
pLevels	Pointer to the array of level values. In case of multi-channel data, *pLevels* is an array of pointers to the level values array for each channel.
nLevels	Number of levels, separate for each channel.

Description

The function `ippiHistogramRange` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the intensity histogram of an image in the ranges specified by the array (or arrays, in case of multi-channel data) *pLevels*. The histograms computed separately for each channel are stored in the arrays *pHist*. Before calling this function, the array *pLevels* should be initialized to determine bounds of the bins for histogram computing.
Length of the arrays *pLevels* and *pHist* is defined by the *nLevels* parameter. Since

nLevels is the number of levels, the number of histogram bins is *nLevels* - 1. Similarly, the number of values in the array *pHist* is also *nLevels* - 1.
The meaning of *pHist* and *pLevels* values can be illustrated by the following example:
pHist[*k*] is a number of source image pixels *pSrc(x,y)* that satisfy the condition

$$pLevels[k] \le pSrc(x,y) < pLevels[k+1].$$

The following code example shows how to use the `ippiHistogramRange` function:

Example 11-4 Computing the Histogram of an Image

```
// Compute histogram of an image for 4 bins in the range [28, 127]
{
    Ipp8u img[WIDTH*HEIGHT];
    IppiSize imgSize = {WIDTH, HEIGHT};
    const Ipp32s levels[5] = {28, 50, 70, 90, 128};
    Ipp32s histo[5];

    ippiHistogramRange_8u_C1R(img, WIDTH, imgSize, levels, histo, 5);

    // after executing of the function the array histo
    // will contain a histogram in specified range.
    // Actually, four result values will be stored

    // compute cumulative histogram
    for (i = 1; i < 4; i ++)
        histo[i] += histo[i-1];
    }
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is NULL.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

ippStsMemAllocErr	Indicates an error if there is not enough memory for the inner histogram.
ippStsHistoNofLevelsErr	Indicates an error if *nLevels* is less than 2.

HistogramEven

Computes the intensity histogram of an image using equal bins.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiHistogramEven_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist, Ipp32s* pLevels,
    int nLevels, Ipp32s lowerLevel, Ipp32s upperLevel);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R     16s_C1R
```

Case 2: Operation on multi-channel integer data

```
IppStatus ippiHistogramEven_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[3], Ipp32s* pLevels[3],
    int nLevels[3], Ipp32s lowerLevel[3], Ipp32s upperLevel[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R
8u_AC4R     16u_AC4R    16s_AC4R
```

```
IppStatus ippiHistogramEven_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[4],  Ipp32s* pLevels[4],
    int nLevels[4], Ipp32s lowerLevel[4], Ipp32s upperLevel[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

roiSize	Size of the source ROI in pixels
pHist	Pointer to the computed histogram. In case of multi-channel data, *pHist* is an array of pointers to the histogram for each channel.
pLevels	Pointer to the array of level values. In case of multi-channel data, *pLevels* is an array of pointers to the level values array for each channel.
nLevels	Number of levels, separate for each channel.
lowerLevel	Lower level boundary, separate for each channel.
upperLevel	Upper level boundary, separate for each channel.

Description

The function `ippiHistogramEven` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the intensity histogram of an image in the ranges specified by the values *lowerLevel* (inclusive), *upperLevel* (exclusive), and *nLevels*. The function operates on the assumption that all histogram bins have the same width and equal boundary values of the bins (levels).
The function stores computed histograms in the array *pHist* separately for each channel. The calculated levels are stored in the array *pLevels*.

Length of the arrays *pLevels* and *pHist* is defined by the *nLevels* parameter. Since *nLevels* is the number of levels, the number of histogram bins is *nLevels* - 1. Similarly, the number of values in the array *pHist* is also *nLevels* - 1.
The meaning of *pHist* and *pLevels* values can be illustrated by the following example:
pHist[*k*] is a number of source image pixels *pSrc(x,y)* that satisfy the condition

$$pLevels[k] \le pSrc(x,y) < pLevels[k+1].$$

Example 11-5 shows how to use the `ippiHistogramEven` function.

Example 11-5 Computing the Even Histogram of an Image

```
// Compute histogram of an image for 4 bins in the range [28, 127];
// compute level values for the bins.
{
Ipp8u img[WIDTH*HEIGHT];
IppiSize imgSize = {WIDTH, HEIGHT};
Ipp32s levels[5], histo[5];

ippiHistogramEven_8u_C1R(img, WIDTH, imgSize, levels,
                                              histo, 5, 28, 128);

// When the function completes operation the array histo will      // contain a
histogram in specified range, and the array
// levels will contain the level values {28, 53, 78, 103, 128}
}
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsMemAllocErr	Indicates an error if there is not enough memory for the inner histogram.
ippStsHistoNofLevelsErr	Indicates an error if *nLevels* is less than 2.

CountInRange

Computes the number of pixels within the given intensity range.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, int* counts, Ipp<datatype> lowerBound,
    Ipp<datatype> upperBound);
```

Supported values for *mod* :

8u_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, int counts[3], Ipp<datatype> lowerBound[3],
    Ipp<datatype> upperBound[3]);
```

Supported values for *mod* :

8u_C3R	32f_C3R
8u_AC4R	32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
counts	The computed number of pixels within the given intensity range. An array of 3 values in case of multi-channel data.
lowerBound	Lower limit of the intensity range.
upperBound	Upper limit of the intensity range.

Description

The function `ippiCountInRange` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the number of pixels in the image which have intensity values in the range between *lowerBound* and *upperBound* (inclusive).

In case of a multi-channel image, pixels are counted within intensity range for each color channel separately, and the array *counts* of three resulting values is returned. The alpha channel values, if present, are not processed.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.
`ippStsRangeErr`	Indicates an error condition if *lowerBound* exceeds *upperBound*.

Min

Computes the minimum of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype>* pMin);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> min[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> min[4]);
```

Supported values for *mod* :

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
pMin	Pointer to the minimum pixel value (for one-channel data).
min	Array containing minimum channel values of pixels in the source buffer (for multi-channel data).

Description

The function `ippiMin` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the minimum pixel value *pMin* for the source image *pSrc*. In case of a multi-channel image, the minimum is computed over each channel and stored in the array *min*.

The code Example 11-2 demonstrates how to use the function `ippiMin`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pMin* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

Example 11-6 Using the function ippiMin

```
Ipp8u src[4*1] = { 40, 20, 60, 80 };
IppiSize roiSize = { 4, 1 };
Ipp8u min;

ippiMin_8u_C1R ( src, 4, roiSize, &min );

result: min = 20
```

MinIndx

Computes the minimum of image pixel values and retrieves the x and y coordinates of pixels with minimal intensity values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype>* pMin, int* pIndexX, int* pIndexY);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R     16s_C1R     32f_C1R
```

Case 2: Operation on multi-channel data

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> min[3], int indexX[3], int indexY[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R     32f_C3R
8u_AC4R     16u_AC4R    16s_AC4R    32f_AC4R
```

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> min[4], int indexX[4], int indexY[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R     32f_C4R
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
pMin	Pointer to the minimum pixel value (for one-channel data).
min	Array containing minimum color channel values of pixels in the source buffer (for multi-channel data).
pIndexX, pIndexY	Pointers to the x and y coordinates of the pixel with minimum value.
indexX, indexY	Arrays containing the x and y coordinates of pixels with minimum channel values.

Description

The function `ippiMinIndx` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the minimum pixel value *pMin* for the source image *pSrc*. In case of a multi-channel image, the minimum is computed over each channel and stored in the array *min*. The function also retrieves the *x* and *y* coordinates of pixels on which the minimum is reached. If several pixels have equal minimum value, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, *indexX*[k] and *indexY*[k] are the *x* and *y* coordinates of the pixel that has the minimal intensity value of the k-th channel, k = 1,2,3,4.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of *pSrc, pMin, pIndexX,* or *pIndexY* pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

Max

Computes the maximum of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype>* pMax);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> max[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> max[4]);
```

Supported values for *mod* :

8u_C4R	16u_C4R	16s_C4R	32f_C4

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels
pMax	Pointer to the maximum pixel value (for one-channel data).
max	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).

Description

The function `ippiMax` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the maximum pixel value *pMax* for the source image *pSrc*. In case of a multi-channel image, the maximum is computed over each channel and stored in the array *max*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pMax* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

MaxIndx

Computes the maximum of image pixel values and retrieves the x and y coordinates of pixels with maximal intensity values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype>* pMax, int* pIndexX, int* pIndexY);
```

Supported values for *mod* :

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`

Case 2: Operation on multi-channel data

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> max[3], int indexX[3], int indexY[3]);
```

Supported values for *mod* :

`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`	`32f_AC4R`

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> max[4], int indexX[4], int indexY[4]);
```

Supported values for *mod* :

8u_C4R 16u_C3R 16s_C3R 32f_C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
pMax	Pointer to the maximum pixel value (for one-channel data).
max	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).
pIndexX, pIndexY	Pointers to the x and y coordinates of the pixel with maximum value.
indexX, indexY	Arrays containing the x and y coordinates of pixels with maximum channel values.

Description

The function ippiMaxIndx is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the maximum pixel value *pMax* for the source image *pSrc*. In case of a multi-channel image, the maximum is computed over each channel and stored in the array *max*. The function also retrieves the *x* and *y* coordinates of pixels on which the maximum is reached. If several pixels have equal maximum value, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, *indexX*[k] and *indexY*[k] are the *x* and *y* coordinates of the pixel that has the maximal intensity value of the k-th channel, k = 1,2,3,4.

The code Example 11-2 demonstrates how to use the function ippiMaxIndx.

Example 11-7 Using the function ippiMaxIndx

```
Ipp8u src[4*1] = { 40, 20, 60, 80 };
IppiSize roiSize = { 4, 1 };
Ipp8u max;
int IndexX;
int IndexY;

ippiMaxIndx_8u_C1R ( src, 4, roiSize, &max, &IndexX, &IndexY );

result: max = 80  IndexX = 3 IndexY = 0
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of *pSrc*, *pMax*, *pIndexX*, or *pIndexY* pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

MinMax

Computes the minimum and maximum of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype>* pMin, Ipp<datatype>* pMax);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> min[3], Ipp<datatype> max[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> min[4], Ipp<datatype> max[4]);
```

Supported values for *mod* :

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

roiSize	Size of the source ROI in pixels.
pMin, pMax	Pointers to the minimum and maximum pixel values (for one-channel data).
min, max	Arrays containing minimum and maximum channel values of pixels in the source buffer (for multi-channel data).

Description

The function `ippiMinMax` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the minimum and maximum pixel values *pMin* and *pMax* for the source image *pSrc*. In case of a multi-channel image, the minimum and maximum is computed over each channel and stored in the arrays *min* and *max*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc*, *pMin*, or *pMax* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

MinMaxIndx

Calculates minimum and maximum pixel values and their indexes in selected image rectangle.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal,
    IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for *mod* :

`8u_C1R` `8s_C1R` `16u_C1R` `32f_C1R`

Case 2: Masked operation on one-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal,
    Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for *mod* :

8u_C1MR 8s_C1MR 16uC1MR 32f_C1MR

Case 3: Operation on multi-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal,
    IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for *mod* :

8u_C3CR 8s_C3CR 16u_C3CR 32f_C3CR

Case 4: Masked operation on multi-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
    IppiPoint* pMaxIndex);
```

Supported values for *mod* :

8u_C3CMR 8s_C3CMR 16u_C3CMR 32f_C3CMR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the image ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pMinVal	Pointer to the variable that returns the value of the minimum pixel.
pMaxVal	Pointer to the variable that returns the value of the maximum pixel.
pMinIndex	Pointer to the variable that returns the index of the minimum value found.
pMaxIndex	Pointer to the variable that returns the index of the maximum value found.

Description

The function `ippiMinMaxIndx` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function finds minimum and maximum pixel values and their indexes in an image ROI or in an arbitrary image region defined by nonzero mask values.

If there are several minima and maxima in the selected area, the function returns the top leftmost positions.

If the specified region in the mask mode is empty, that is, the mask image is filled with zeros, then the function returns {*minIndex*, *maxIndex*} = {0, 0}, *minVal* = *maxVal* = 0.

If any of the parameters *pMinVal*, *pMaxVal*, *pMinIndex*, or *pMaxIndex* is not required, the zero pointer is to be passed to the corresponding parameter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pMask* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error for masked operations when *srcStep* or *maskStep* is less than *roiSize.width* * *<pixelSize>* .
`ippStsNotEvenStepErr`	Indicates an error when steps for floating-point images cannot be divided by 4.
`ippStsCOIErr`	Indicates an error when *coi* is not 1, 2, or 3.

MaxEvery

Computes maximum value for each pair of pixels of two images.

Syntax

```
IppStatus ippiMaxEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep,
          Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the first source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the first source image.
pSrcDst	Pointer to the second source and destination image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the second source and destination image.
roiSize	Size of the image ROI in pixels.

Description

The function `ippiMaxEvery` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the maximum value for each pair of the corresponding pixels of two source images *pSrc* and *pSrcDst*, and stores the result in the *pSrcDst*:

pSrcDst(*i*,*j*) = max(*pSrcDst*(*i*,*j*), *pSrcDst*(*i*,*j*)).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *srcDstStep* is less than *roiSize.width*<pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images are not divisible by 4.

MinEvery

Computes minimum value for each pair of pixels of two images.

Syntax

```
IppStatus ippiMinEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep,
          Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the first source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the first source image.
pSrcDst	Pointer to the second source and destination image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the second source and destination image.
roiSize	Size of the image ROI in pixels.

Description

The function `ippiMinEvery` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the minimum value for each pair of the corresponding pixels of two source images *pSrc* and *pSrcDst*, and stores the result in the *pSrcDst*:

pSrcDst(*i*,*j*) = min(*pSrcDst*(*i*,*j*), *pSrcDst*(*i*,*j*)).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *srcDstStep* is less than *roiSize.width*<pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images are not divisible by 4.

FindPeaks3x3GetBufferSize

Computes the size of the working buffer for the peak search.

Syntax

```
IppStatus ippiFindPeaks3x3GetBufferSize_32s_C1R(int roiWidth, int* pBufferSize);
IppStatus ippiFindPeaks3x3GetBufferSize_32f_C1R(int roiWidth, int* pBufferSize);
```

Parameters

roiWidth	Maximum width of the image, in pixels.
pBufferSize	Pointer to the size of the working buffer.

Description

The function `ippiFindPeaks3x3GetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the function `ippiFindPeaks3x3`. The buffer with the length *pBufferSize*[0] can be used to filter images with width that is less than or equal to *roiWidth*.

Example 11-8 shows how to use the function `ippiFindPeaks3x3GetBufferSize_32f_C1R`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer *pBufferSize* is NULL.
ippStsSizeErr	Indicates an error condition if *roiWidth* is less than 1.

FindPeaks3x3

Finds coordinates of peaks (maximums or minimums) with absolute value exceeding threshold value.

Syntax

```
IppStatus ippiFindPeaks3x3_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize
    roiSize, Ipp32s threshold, IppiPoint* pPeak, int maxPeakCount, int*
    pPeakCount, IppiNorm norm, int border, Ipp8u* pBuffer);
IppStatus ippiFindPeaks3x3_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize
    roiSize, Ipp32f threshold, IppiPoint* pPeak, int maxPeakCount, int*
    pPeakCount, IppiNorm norm, int border, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the first source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the first source image.
roiSize	Size of the image ROI in pixels.
threshold	Threshold value.
pPeak	Pointer to the coorditanes peaks [*maxPeakCount*].
maxPeakCount	Maximum number of peaks.
pPeakCount	Pointer to the number of the detected peaks.
border	Border value, only pixel with distance from the edge of the image greater than *border* are processed.
norm	Specifies type of the norm to form the mask for extremum search: `ippiNormInf` Infinity norm (8-connectivity, 3x3 rectangular mask); `ippiNormL1` L1 norm (4-connectivity, 3x3 cross mask).
pBuffer	Pointer to the working buffer.

Description

The function `ippiFindPeaks3x3` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function detects local maximum and minimum pixels in the source image:

$$pSrc(i_m, j_m) = \max_{(k,\ l) \in O(i_m,\ j_m)} pSrc(k, l), \quad pSrc(i_m, j_m) \geq threshold$$

$$pSrc(i_m, j_m) = \min_{(k,\ l) \in O(i_m,\ j_m)} pSrc(k, l), \quad |pSrc(i_m, j_m)| \geq threshold$$

and stores their coordinates in the *pPeak* array
pPeak[m].x = jm, *pPeak*[m].y = im, m = 0, ... *pPeakCount*[0], *pPeakCount*[0] ≤ *maxPeakCount*

The neighborhood $O(i, j)$ for the extremum search is defined by the parameter *norm*. The number of detected extremums is returned in *pPeakCount*[0]. The operation is stopped when the *maxPeakCount* extremums are found.

The function requires the working buffer *pBuffer* whose size should be computed by the function ippiFindPeaks3x3GetBufferSize beforehand.

Example 11-8 shows how to use the function ippiFindPeaks3x3_32f_C1R for calculations of the SIFT (Scale Invariant Feature Transform) features [Lowe04].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value; or if *maxPeakCount* is less than or equal to 0; or if *border* is less than 1 or greater than one of 0.5*roiSize.width or of 0.5*roiSize.height.
ippStsStepErr	Indicates an error condition if *srcStep* is less than *roiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if *srcStep* is not divisible by 4.

Example 11-8 Simplified Peak Search for Calculation of SIFT Features

```
Ipp32s sigma[NIMAGE];        //  sigmas of Gauss blur (increasing)
Ipp32s kerSize[NIMAGE];      //  kernel sizef of Gauss blur (increasing)
Ipp32f *pDst[NIMAGE];        //  blurred images
Ipp32f *pSrc;                //  initial image
Ipp8u *pBuffer;              //  working buffer
```

Example 11-8 Simplified Peak Search for Calculation of SIFT Features

```
          IppiPoint *pPeak;           //  array of peaks
          IppiSize roi;               //  image size
          int step;                   //  row step in bytes
          int bufSize;                //  working buffer size
          int maxPeaks=500;           //  max peaks number for image
          int peakCount;              //  peaks number for image
          Ipp32f threshold=0.02f;     //  peak thresjold

          ippiFilterGaussGetBufferSize_32f_C1R(roi,kerSize[NIMAGE-1],&bufSize);
          pBuffer = ippsMalloc_8u(bufSize);
         for (i=0; i<NIMAGE; i++) {  //blur inital image with different Gauss kernels
             ippiFilterGaussBorder_32f_C1R(pSrc,step,pDst[i],step,roi,kerSize[i],
                sigma[i],ippBorderRepl,0,pBuffer);
          }
          ippsFree(pBuffer);
        for (i=0; i<NIMAGE-1; i++) { //  calulate difference of Gaussian (DOG) images
             ippiSub_32f_C1IR(pDst[i+1],step,pDst[i],step,roi);
          }
          ippiFindPeaks3x3GetBufferSize_32f_C1R(roi.width,&bufSize);
          pBuffer = ippsMalloc_8u(bufSize);
          pPeak = (IppiPoint*)ippsMalloc_8u(maxPeaks*sizeof(IppiPoint));
          for (i=1; i<NIMAGE-2; i++) {  // find and process peaks in DOG images
             ippiFindPeaks3x3_32f_C1R(pDst[i],step,roi,threshold,pPeak,maxPeaks,
                &peakCount,ippiNormL1,kerSize[0]/2,pBuffer);
             for (j=0; j<peakCount; j++) {
                Ipp32f pix = pDst[i][pPeak[j].y*step/sizeof(Ipp32f)+pPeak[j].x];
                Ipp32f pup = pDst[i+1][pPeak[j].y*step/sizeof(Ipp32f)+pPeak[j].x];
                Ipp32f pdn = pDst[i-1][pPeak[j].y*step/sizeof(Ipp32f)+pPeak[j].x];
                if (pix>0) if ((pix<pup)||(pix<pdn)) continue;
                if (pix<0) if ((pix>pup)||(pix>pdn)) continue;
//              calculate SIFT feature description for a peak pPeak[j] in DOG image
pDst[i]
             }
          }
          ippsFree(pBuffer);
          ippsFree(pPeak);
```

Image Moments

Spatial and central moments are important statistical properties of an image. The spatial moment $M_U(m,n)$ of order (m,n) is defined as follows:

$$M_U(m, n) = \sum_j \sum_k x_k^m y_j^n P_{j,k}$$

where the summation is performed for all rows and columns in the image; $P_{j,k}$ are pixel values; x_k and y_j are pixel coordinates; m and n are integer power exponents that define the moment order.

The central moment $U_U(m,n)$ is the spatial moment computed relative to the "center of gravity" (x_0, y_0):

$$U_U(m, n) = \sum_j \sum_k (x_k - x_0)^m (y_j - y_0)^n P_{j,k}$$

where $x_0 = M_U(1,0)/M_U(0,0)$ and $y_0 = M_U(0,1)/M_U(0,0)$.

The normalized spatial moment $M(m,n)$ and central moment $U(m,n)$ are defined as follows:

$$M(m, n) = \frac{M_U(m, n)}{M_U(0, 0)^{\frac{m+n+2}{2}}}$$

$$U(m, n) = \frac{U_U(m, n)}{U_U(0, 0)^{\frac{m+n+2}{2}}}$$

The Intel IPP functions support moments of order (m, n) with $0 \leq m + n \leq 3$. The computation of seven invariant Hu moments derived from the second and third order moments is also supported. All computed moments are stored in context structures of type `IppiMomentState_64s` (for integer versions) or `IppiMomentState_64f` (for floating point versions).

Most Intel IPP functions for computing image moments have code branches that implement different algorithms to compute the results. You can choose the desired code variety to be used by the given function by setting the *hint* argument to one of the following values that are listed in Table 11-3:

Table 11-3 Hint Arguments for Image Moment Functions

Value	Description
ippAlgHintNone	The computation algorithm will be chosen by the internal function logic.
ippAlgHintFast	Fast algorithm must be used. The output results will be less accurate.
ippAlgHintAccurate	High accuracy algorithm must be used. The function will need more time to execute.

MomentInitAlloc

Allocates memory and initializes the moment context structure.

Syntax

```
IppStatus ippiMomentInitAlloc_64f(IppiMomentState_64f** pState,
    IppHintAlgorithm hint);
IppStatus ippiMomentInitAlloc_64s(IppiMomentState_64s** pState,
    IppHintAlgorithm hint);
```

Parameters

pState	Pointer to the structure for storing moment values.
hint	Option to select the algorithmic implementation of the function.

Description

The function ippiMomentInitAlloc is declared in the ippi.h file. This function initializes the structure that is needed for the function ippiMoments to store the computed image moments. Different functions are used to allocate structures for integer and floating-point data. Computation algorithm is specified by *hint* argument (see Table 11-3).

If the initialization is successful, the `ippiMomentInitAlloc` function sets the *pState* pointer to the allocated structure of `IppiMomentState` type.

If memory allocation failed, the function returns `ippStsMemAllocErr` error status code.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsMemAllocErr`	Indicates an error condition in case of memory allocation failure.

MomentFree

Frees memory allocated by the function `ippiMomentInitAlloc`.

Syntax

```
IppStatus ippiMomentFree_64f(IppiMomentState_64f* pState);
IppStatus ippiMomentFree_64s(IppiMomentState_64s* pState);
```

Parameters

pState	Pointer to the structure for image moments storage.

Description

The function `ippiMomentFree` is declared in the `ippi.h` file. Use this function to deallocate the previously initialized *pState* structure that stored the image moments data. Different functions are used to deallocate structures for integer and floating-point data.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pState* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid structure is passed.

MomentGetStateSize

Computes the size of the external buffer for the moment context structure.

Syntax

```
IppStatus ippiMomentGetStateSize_64s(IppHintAlgorithm hint, int* pSize);
```

Parameters

pSize	Pointer to the computed value of the buffer size.
hint	Option to select the algorithmic implementation of the function.

Description

The function `ippiMomentGetStateSize` is declared in the `ippi.h` file. Use this function to determine the size of the external work buffer for the moment context structure to be initialized by the function `ippiMomentInit`. Computation algorithm is specified by *hint* argument (see Table 11-3).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSize* pointer is `NULL`.

MomentInit

Initializes the moment context structure.

Syntax

```
IppStatus ippiMomentInit_64s(IppiMomentState_64s* pState,
    IppHintAlgorithm hint);
```

Parameters

pState	Pointer to the structure for storing moment values.
hint	Option to select the algorithmic implementation of the function.

Description

The function `ippiMomentInit` is declared in the `ippi.h` file. This function initializes the structure that is needed for the function `ippiMoments` to store the computed image moments. Computation algorithm is specified by *hint* argument (see Table 11-3).

The structure is allocated in the external buffer. The size of this buffer can be computed by the function `ippiMomentGetStateSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pState* pointer is `NULL`.

Moments

Computes all image moments of order 0 to 3 and Hu moment invariants.

Syntax

Case 1: Computation of floating-point results

```
IppStatus ippiMoments64f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, IppiMomentState_64f* pState);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Computation of integer results

```
IppStatus ippiMoments64s_<mod>(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, IppiMomentState_64s* pState);
```

Supported values for *mod* :

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
pState	Pointer to the structure that stores image moments.

Description

The function `ippiMoments` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes all spatial and central moments of order 0 to 3 for the source image *pSrc*. The seven Hu moment invariants are also computed. Different functions, `ippiMoments64s` and `ippiMoments64f`, are used to compute image moments in integer and floating-point formats, respectively.

The `ippiMoments` function computes spatial moment values relative to the image point referred to by *pSrc*. Note that this point is the ROI origin and may not coincide with the entire image origin. If you need to obtain spatial moment values relative to the actual image origin, use `ippiGetSpatialMoment` functions to recalculate them.

The moments' values are stored in the *pState* structure. To retrieve a particular moment value, use one of the functions described in the sections that follow.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pState* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid structure is passed.

GetSpatialMoment

Retrieves image spatial moment of the specified order, computed by `ippiMoments`.

Syntax

```
IppStatus ippiGetSpatialMoment_64f(const IppiMomentState_64f* pState,
    int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,
    Ipp64f* pValue);
IppStatus ippiGetSpatialMoment_64s(const IppiMomentState_64s* pState,
    int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,
    Ipp64s* pValue, int scaleFactor);
```

Parameters

pState	Pointer to the structure that stores image moments.
mOrd, nOrd	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \le mOrd + nOrd \le 3$.
nChannel	The channel for which the moment is returned.
roiOffset	Offset in pixels of the ROI origin (top left corner) from the image origin.
pValue	Pointer to the retrieved moment value.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiGetSpatialMoment` is declared in the `ippi.h` file. This function returns the pointer *pValue* to the spatial moment that was previously computed by the `ippiMoments` function. All spatial moment values are computed by `ippiMoments` relative to the image ROI origin.
You may also obtain spatial moment values relative to different point in the image, using the appropriate *roiOffset* settings.

The moment order is specified by the integer exponents *mOrd, nOrd*.

Different functions are used to retrieve spatial moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pState* or *pValue* pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.
ippStsSizeErr	Indicates an error condition if *mOrd + nOrd* is greater than 3, or *nChannel* has an illegal value.

GetNormalizedSpatialMoment

Retrieves the normalized value of the image spatial moment computed by ippiMoments.

Syntax

```
IppStatus ippiGetNormalizedSpatialMoment_64f(IppiMomentState_64f*
    pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,
    Ipp64f* pValue);
IppStatus ippiGetNormalizedSpatialMoment_64s(IppiMomentState_64s*
    pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,
    Ipp64s* pValue, int scaleFactor);
```

Parameters

pState	The structure that stores image moments.
mOrd, nOrd	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \le mOrd + nOrd \le 3$.
nChannel	The channel for which the moment is returned.
roiOffset	Offset in pixels of the ROI origin (top left corner) from the image origin.
pValue	Pointer to the returned normalized moment value.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiGetNormalizedSpatialMoment` is declared in the `ippi.h` file. This function normalizes the spatial moment value that was previously computed by the `ippiMoments` function, and returns the pointer *pValue* to the normalized moment. See Image Moments for details of moments normalization. The moment order (*mOrd, nOrd*) is specified by integer power exponents.
All spatial moment values are computed by `ippiMoments` relative to the image ROI origin.You may also obtain normalized spatial moment values relative to different point in the image, using the appropriate *roiOffset* settings.

Different functions are used to retrieve normalized spatial moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pState* or *pValue* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid structure is passed.
`ippStsMoment00ZeroErr`	Indicates an error condition if *M*(*0,0*) value is close to zero.
`ippStsSizeErr`	Indicates an error condition if *mOrd + nOrd* is greater than 3, or *nChannel* has an illegal value.

GetCentralMoment

Retrieves image central moment computed by `ippiMoments`.

Syntax

```
IppStatus ippiGetCentralMoment_64f(const IppiMomentState_64f* pState,
    int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
IppStatus ippiGetCentralMoment_64s(const IppiMomentState_64s* pState,
    int mOrd, int nOrd, int nChannel, Ipp64s* pValue,
    int scaleFactor);
```

Parameters

pState	The structure that stores image moments.
mOrd, nOrd	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \le mOrd + nOrd \le 3$.
nChannel	The channel for which the moment is returned.
pValue	Pointer to the returned moment value.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiGetCentralMoment` is declared in the `ippi.h` file. This function returns the pointer *pValue* to the central moment previously computed by the `ippiMoments` function. The moment order is specified by the integer exponents *mOrd, nOrd.*

Different functions are used to retrieve central moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor.*

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pState* or *pValue* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid structure is passed.
`ippStsSizeErr`	Indicates an error condition if *mOrd + nOrd* is greater than 3, or *nChannel* has an illegal value.

GetNormalizedCentralMoment

Retrieves the normalized value of the image central moment computed by `ippiMoments`.

Syntax

```
IppStatus ippiGetNormalizedCentralMoment_64f(IppiMomentState_64f*
    pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
```

```
IppStatus ippiGetNormalizedCentralMoment_64s(IppiMomentState_64s*
    pState, int mOrd, int nOrd, int nChannel, Ipp64s* pValue,
    int scaleFactor);
```

Parameters

pState	The structure that stores image moments.
mOrd, nOrd	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \le mOrd + nOrd \le 3$.
nChannel	The channel for which the moment is returned.
pValue	Pointer to the returned moment value.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiGetNormalizedCentralMoment` is declared in the `ippi.h` file. This function normalizes the central moment value that was previously computed by the `ippiMoments` function, and returns the pointer *pValue* to the normalized moment. The moment order (*mOrd, nOrd*) is specified by the integer power exponents. See Image Moments for details of moments normalization.

Different functions are used to retrieve normalized central moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pState* or *pValue* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid structure is passed.
`ippStsMoment00ZeroErr`	Indicates an error condition if $M(0,0)$ value is close to zero.

GetHuMoments

Retrieves image Hu moment invariants computed by `ippiMoments` *function.*

Syntax

```
IppStatus ippiGetHuMoments_64f(IppiMomentState_64f* pState,
    int nChannel, IppiHuMoment_64f* pHm);
IppStatus ippiGetHuMoments_64s(IppiMomentState_64s* pState,
    int nChannel, IppiHuMoment_64s* pHm, int scaleFactor);
```

Parameters

pState	Pointer to the structure that stores image moments.
nChannel	The channel for which the moment is returned.
pHm	Pointer to the array containing the Hu moment invariants.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiGetHuMoments` is declared in the `ippi.h` file. This function returns the pointer *pHm* to the array of seven Hu moment invariants previously computed by the `ippiMoments` function.
Different functions are used to retrieve Hu moments in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

The code Example 11-9 shows how to compute Hu moment values for images that have horizontal and vertical lines.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pState* or *pHm* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid structure is passed.
`ippStsMoment00ZeroErr`	Indicates an error condition `if` $M(0,0)$ value is close to zero.

Example 11-9 Computing Hu moments

```
IppStatus mom8u( void ) {
   IppStatus stH, stV;
   Ipp8u x[64];
   IppiSize roiA={8,8}, roiH={8,1}, roiV={1,8};
   IppiHuMoment_64f hmH, hmV;
   IppiMomentState_64f* ctx;

   stH = ippiMomentInitAlloc_64f( &ctx, ippAlgHintNone );

   ippiSet_8u_C1R( 0, x, 8, roiA );
   ippiSet_8u_C1R( 3, x, 8, roiH );
   stH = ippiMoments64f_8u_C1R( x, 8, roiA, ctx );
   ippiGetHuMoments_64f( ctx, 0, hmH );

   ippiSet_8u_C1R( 0, x, 8, roiA );
   ippiSet_8u_C1R( 3, x, 8, roiV );
   stV = ippiMoments64f_8u_C1R( x, 8, roiA, ctx );
   ippiGetHuMoments_64f( ctx, 0, hmV );

   ippiMomentFree_64f( ctx );
   return ippStsNoErr==stH ? stV : stH;
}
```

Image Norms

The functions described in this section compute the following norms of the image pixel values:

- Infinity norm (the largest absolute pixel value).
- L1 norm (the sum of absolute pixel values)
- L2 norm (the square root of the sum of squared pixel values)

Functions of this group also help you compute the norm of differences in pixel values of two input images as well as the relative error for two input images.

Norm_Inf

Computes the infinity norm of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R     16s_C1R     32s_C1R     32f_C1R
```

Case 2: Masked operation on one-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

```
8u_C1MR     8s_C1MR     16u_C1MR    32f_C1MR
```

Case 3: Operation on multi-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R     32f_C3R
8u_AC4R     16u_AC4R    16s_AC4R    32f_AC4R
```

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u_C4R 16u_C4R 16s_C4R 32f_C4R

Case 4: Masked operation on multi-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

8u_C3CMR 8s_C3CMR 16u_C3CMR 32f_C3CMR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed infinity norm of pixel values.
value	An array containing the computed infinity norms of channel values in case of multi-channel data.
pNorm	Pointer to the computed norm value in the mask mode.

Description

The flavors of the function `ippiNorm_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP) and computes the infinity norm *pValue* (*pNorm* for the mask mode) for the source image *pSrc*. This norm is defined as the largest absolute pixel value in an image. In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (**Case 3**), the norm is computed separately for each channel and stored in the array *value* .

In the mask multi-channel mode (**Case 4**), the norm is computed for a single channel of interest specified by *coi*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if *srcStep* or *maskStep* is less than *roiSize.width* * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when *coi* is not 1, 2, or 3.

Norm_L1

Computes the L1- norm of image pixel values.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u_C1R 16u_C1R 16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNorm_L1_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u_C1MR 8s_C1MR 16u_C1MR 32f_C1MR

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R
8u_AC4R     16u_AC4R    16s_AC4R
```

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R
```

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNorm_L1_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNorm_L1_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

```
8u_C3CMR    8s_C3CMR    16u_C3CMR   32f_C3CMR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.

coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed L1- norm of pixel values.
value	An array containing the computed L1- norms of channel values in case of multi-channel data.
pNorm	Pointer to the computed norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiNorm_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L1- norm *pValue* (*pNorm* in mask mode) for the source image *pSrc*. This norm is defined as the sum of absolute pixel values in an image. Computation algorithm is specified by the *hint* argument (see Table 11-3). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (**Case 4, 5**), the norm is computed separately for each channel and stored in the array *value* .

In the mask multi-channel mode (**Case 6**), the norm is computed for a single channel of interest specified by *coi*.

The code Example 11-10 demonstrates how an image norm can be computed.

Example 11-10 Computing the Image Norm

```
IppStatus norm( void ) {
   Ipp64f sum, normL1;
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   ippiSet_8u_C1R( 1, x, 5, roi );
   ippiSum_8u_C1R( x, 5, roi, &sum);
   return ippiNorm_L1_8u_C1R( x, 5, roi, &normL1 );
}
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

ippStsStepErr	Indicates an error condition in mask mode, if *srcStep* or *maskStep* is less than *roiSize.width* * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when *coi* is not 1, 2, or 3.

Norm_L2

Computes the L2- norm of image pixel values.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R     16s_C1R
```

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNorm_L2_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

```
8u_C1MR     8s_C1MR     16u_C1MR    32f_C1MR
```

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R
8u_AC4R     16u_AC4R    16s_AC4R
```

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R
```

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNorm_L2_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNorm_L2_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

```
8u_C3CMR    8s_C3CMR    16u_C3CMR   32f_C3CMR
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed L2- norm of pixel values.
value	An array containing the computed L2- norms of channel values in case of multi-channel data.
pNorm	Pointer to the computed norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiNorm_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L2- norm *pValue* (*pNorm* in mask mode) for the source image *pSrc*. This norm is defined as the square root of the sum of squared pixel values in an image. Computation algorithm is specified by the *hint* argument (see Table 11-3). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (**Case 4,5**), the norm is computed separately for each channel and stored in the array *value* .

In the mask multi-channel mode (**Case 6**), the norm is computed for a single channel of interest specified by *coi*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition in mask mode, if *srcStep* or *maskStep* is less than *roiSize.width* * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
`ippStsCOIErr`	Indicates an error when *coi* is not 1, 2, or 3.

NormDiff_Inf

Computes the infinity norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Masked operation on one-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR

Case 3: Operation on multi-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Case 4: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

8u_C3CMR	8s_C3CMR	16u_C3CMR	32f_C3CMR

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distance in bytes between starts of consecutive lines in the source images.
pMask	Pointer to the mask image.

maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
pValue	Pointer to the computed infinity norm of difference between pixel values.
value	An array containing the computed infinity norms of difference between corresponding channel values in case of multi-channel data.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pNorm	Pointer to the computed norm value in the mask mode.

Description

The flavors of the function `ippiNormDiff_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the infinity norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source images *pSrc1* and *pSrc2*. This norm is defined as the largest absolute value of differences:

norm = max |*pSrc1-pSrc2*|

In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Case 3**), the norm is computed separately for each pair of corresponding channels and stored in the array *value* .

In the mask multi-channel mode (**Case 4**), the norm is computed for a single channel of interest specified by *coi*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition in mask mode, if *src1Step*, *src2Step*, or *maskStep* is less than *roiSize.width* * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
`ippStsCOIErr`	Indicates an error when *coi* is not 1, 2, or 3.

NormDiff_L1

Computes the L1- norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormDiff_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask,
    int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R
```

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNormDiff_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask,
    int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for *mod* :

```
8u_C3CMR    8s_C3CMR    16u_C3CMR   32f_C3CMR
```

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distance in bytes between starts of consecutive lines in the source images.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed L1- norm of difference between pixel values.
value	An array containing the computed L1- norms of difference between channel values in case of multi-channel data.
pNorm	Pointer to the computed norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiNormDiff_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L1-norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source image buffers *pSrc1* and *pSrc2*. This norm is defined as the sum of absolute values of differences:

norm = $\sum |pSrc1 - pSrc2|$

Computation algorithm is specified by the *hint* argument (see Table 11-3). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Case 4,5**), the norm is computed separately for each pair of the corresponding channels and stored in the array *value* .

In the mask multi-channel mode (**Case 6**), the norm is computed for a single channel of interest specified by *coi*.

Example 11-11 shows how to use the function `ippiNormDiff_L1_8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition in mask mode, if *src1Step*, *src2Step*, or *maskStep* is less than *roiSize.width* * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
`ippStsCOIErr`	Indicates an error when *coi* is not 1, 2, or 3.

Example 11-11 Using the function `ippiNormDiff_L1`

```
void func_normdiff_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};
```

Example 11-11 Using the function ippiNormDiff_L1

```
    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
    ippiSet_8u_C1R(2, pSrc2, src2Step, roi);

    ippiNormDiff_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);
}

Result -> 20.0
```

NormDiff_L2

Computes the L2- norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R     16s_C1R
```

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormDiff_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask,
    int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

```
8u_C1MR     8s_C1MR     16u_C1MR    32f_C1MR
```

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R
8u_AC4R     16u_AC4R    16s_AC4R
```

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R
```

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNormDiff_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask,
    int maskStep, IppiSize roiSize, int coi, Ipp64f* norm);
```

Supported values for *mod* :

```
8u_C3CMR    8s_C3CMR    16u_C3CMR   32f_C3CMR
```

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distance in bytes between starts of consecutive lines in the source images.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.

roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed L2- norm of difference between pixel values.
value	An array containing the computed L2- norms of difference between channel values in case of multi-channel data.
pNorm	Pointer to the computed norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiNormDiff_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L2-norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source image buffers *pSrc1* and *pSrc2*.

This norm is defined as the square root of the sum of squared differences:

$$\text{norm} = \sqrt{\sum |pSrc1 - pSrc2|^2} \quad .$$

Computation algorithm is specified by the *hint* argument (see Table 11-3). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Case 4,5**), the norm is computed separately for each pair of the corresponding channels and stored in the array *value* .

In the mask multi-channel mode (**Case 6**), the norm is computed for a single channel of interest specified by *coi*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition in mask mode, if *src1Step*, *src2Step*, or *maskStep* is less than *roiSize.width* * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
`ippStsCOIErr`	Indicates an error when *coi* is not 1, 2, or 3.

NormRel_Inf

Computes the relative error for the infinity norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Masked operation on one-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u_C1MR	8s_C1MR	16u_C1MR	32f_C1MR

Case 3: Operation on multi-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Case 4: Masked operation on multi-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

8u_C3CMR 8s_C3CMR 16u_C3CMR 32f_C3CMR

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distance in bytes between starts of consecutive lines in the source images.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
pValue	Pointer to the computed relative error value.
value	An array containing the computed relative error values for separate channels in case of multi-channel data.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pNorm	Pointer to the computed relative norm value in the mask mode.

Description

The flavors of the function `ippiNormRel_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the infinity norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the largest absolute pixel value in an image.
The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the infinity norm of the second source image buffer *pSrc2*.
In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Case 3**), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (**Case 4**), the relative norm is computed for a single channel of interest specified by *coi*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if *src1Step*, *src2Step*, or *maskStep* is less than *roiSize.width* * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when *coi* is not 1, 2, or 3.
ippStsDivByZero	Indicates a warning when the infinity norm of *pSrc2* has a zero value.

NormRel_L1

Computes the relative error for the L1 norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f* pValue);
```

Supported values for *mod* :

```
8u_C1R      16u_C1R      16s_C1R
```

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormRel_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

```
8u_C1MR     8s_C1MR     16u_C1MR    32f_C1MR
```

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f value[3]);
```

Supported values for *mod* :

```
8u_C3R      16u_C3R     16s_C3R
8u_AC4R     16u_AC4R    16s_AC4R
```

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f value[4]);
```

Supported values for *mod* :

```
8u_C4R      16u_C4R     16s_C4R
```

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiNormRel_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);
```

Supported values for *mod* :

```
8u_C3CMR    8s_C3CMR    16u_C3CMR   32f_C3CMR
```

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distance in bytes between starts of consecutive lines in the source images.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed relative error value.
value	An array containing the computed relative error values for separate channels in case of multi-channel data.
pNorm	Pointer to the computed relative norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiNormRel_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L1- norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the sum of absolute pixel values in an image. The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the L1- norm of the second source image buffer *pSrc2*. Computation algorithm is specified by the *hint* argument (see Table 11-3). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Cases 4, 5**), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (**Case 6**), the relative norm is computed for a single channel of interest specified by *coi*.

Example 11-12 shows how to use the function `ippiNormRel_L1_8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if *src1Step*, *src2Step*, or *maskStep* is less than *roiSize.width* * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when *coi* is not 1, 2, or 3.
ippStsDivByZero	Indicates a warning when the L1 norm of *pSrc2* has a zero value.

Example 11-12 Using the function ippiNormRel_L1

```
void func_normrel_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};

    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
    ippiSet_8u_C1R(2, pSrc2, src2Step, roi);
    ippiNormRel_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);

}
Result -> 0.5
```

NormRel_L2

Computes the relative error for the L2 norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u_C1R 16u_C1R 16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormRel_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u_C1MR 8s_C1MR 16u_C1MR 32f_C1MR

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f value[3]);
```

Supported values for *mod* :

8u_C3R 16u_C3R 16s_C3R

8u_AC4R 16u_AC4R 16s_AC4R

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f value[4]);
```

Supported values for *mod* :

8u_C4R 16u_C4R 16s_C4R

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
```

Supported values for *mod* :

32f_C3R

32f_AC4R

```
IppStatus ippiNormRel_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for *mod* :

8u_C3CMR 8s_C3CMR 16u_C3CMR 32f_C3CMR

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distance in bytes between starts of consecutive lines in the source images.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed relative error value.
value	An array containing the computed relative error values for separate channels in case of multi-channel data.
pNorm	Pointer to the computed relative norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiNormRel_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L2- norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the square root of the sum of squared pixel values in an image. The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the L2- norm of the second source image buffer *pSrc2*. Computation algorithm is specified by the *hint* argument (see Table 11-3). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Cases 4, 5**), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (**Case 6)**, the relative norm is computed for a single channel of interest specified by *coi*.

Example 11-13 shows how to use the function ippiNormRel_L2_8u_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if *src1Step*, *src2Step*, or *maskStep* is less than *roiSize.width* * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when *coi* is not 1, 2, or 3.
ippStsDivByZero	Indicates a warning when the L2 norm of *pSrc2* has a zero value.

Example 11-13 Using the function ippiNormRel_L2

```
void func_normrel_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};

    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
    ippiSet_8u_C1R(10, pSrc2, src2Step, roi);
    ippiNormRel_L2_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);

}
Result -> 0.9
```

Image Quality Index

Intel IPP functions described in this section compute the universal image quality index [Wang02] that may be used as image and video quality distortion measure. It is mathematically defined by modeling the image distortion relative to the reference image as a combination of three factors: loss of correlation, luminance distortion, and contrast distortion.

If two images *f* and *g* are considered as a matrices with *M* column and *N* rows containing pixel values *f*[*i,j*], *g*[*i,j*], respectively ($0 \geq i > M$, $0 \geq j > N$), the universal image quality index *Q* may be calculated as a product of three components:

$$Q = \frac{\sigma_{fg}}{\sigma_f \sigma_g} \cdot \frac{2\bar{f}\bar{g}}{(\bar{f})^2 + (\bar{g})^2} \cdot \frac{2\sigma_f \sigma_g}{\sigma_f^2 + \sigma_g^2}$$

where

$$\bar{f} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f[i,j] \qquad \bar{g} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} g[i,j]$$

$$\sigma_{fg} = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})(g[i,j] - \bar{g})$$

$$\sigma_f^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})^2 \qquad \sigma_g^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (g[i,j] - \bar{g})^2$$

The first component is the correlation coefficient, which measures the degree of linear correlation between images *f* and *g*. It varies in the range [-1, 1]. The best value 1 is obtained when *f* and *g* are linearly related, which means that *g*[*i,j*] = *af*[*i,j*]+*b* for all possible values of *i* and *j*.

The second component, with a value range of [0, 1], measures how close the mean luminance is between images.

Since σ_f and σ_g can be considered as estimates of the contrast of *f* and *g*, the third component measures how similar the contrasts of the images are. The value range for this component is also [0, 1].

The range of values for the index *Q* is [-1, 1]. The best value 1 is achieved if and only if the images are identical.

QualityIndex

Computes the universal image quality index.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiQualityIndex_<mod>(const Ipp<srcDatatype>* pSrc1,
    int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp<dstDatatype>* pQualityIndex);
```

Supported values for *mod*:

```
8u32f_C1R   16u32f_C1R  32f_C1R
```

Case 2: Operation on multi-channel data

```
IppStatus ippiQualityIndex_<mod>(const Ipp<srcDatatype>* pSrc1,
    int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp<dstDatatype> qualityIndex[3]);
```

Supported values for *mod*:

```
8u32f_C3R   16u32f_C3R  32f_C3R
8u32f_AC4R  16u32f_AC4R 32f_AC4R
```

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distance in bytes between starts of consecutive lines in the source images.
roiSize	Size of the source ROI in pixels.
pQualityIndex	Pointer to the computed quality index value.
qualityIndex	An array containing the computed quality index values for separate channels in case of multi-channel data.

Description

The function `ippiQualityIndex` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the universal image quality index for two images *pSrc1* and *pSrc2* according to the formula in the introduction section above.The computed value of the index for one-channel image is stored in *pQualityIndex*. For multi-channel images, the quality index is computed separately for each channel and stored in the array *qualityIndex*.

Example 11-14 shows how to use the function `ippiQualityIndex_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *src1Step* or *src2Step* has a zero or negative value.
`ippStsQualityIndexErr`	Indicates an error condition if pixel values of one of the images are constatnt.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

Example 11-14 Using of the function `ippiQualityIndex`

```
void func_qualityindex()
{
    Ipp32f pSrc1[256*256];
    Ipp32f pSrc2[256*256];
    int src1Step = 256*sizeof(Ipp32f);
    int src2Step = 256*sizeof(Ipp32f);;
    Ipp32f pQIndex;
    IppiSize roiSize = {256,256};
    unsigned int pSeed = 3;

// creating reference image (image1) and distorted image (image2)
    ippiImageJaehne_32f_C1R(pSrc1, src1Step, roiSize); // image1
    ippiImageJaehne_32f_C1R(pSrc2, src2Step, roiSize);
    ippiAddRandUniform_Direct_32f_C1IR(pSrc2, src2Step, roiSize, 0, 1, &pSeed);
//image2 = image1 + RandUniform

    ippiQualityIndex_32f_C1R(pSrc1, src1Step, pSrc2, src2Step, roiSize,
                                &pQIndex);
}
```

Example 11-14 Using of the function `ippiQualityIndex`

```
Result:
```

image1 (reference)

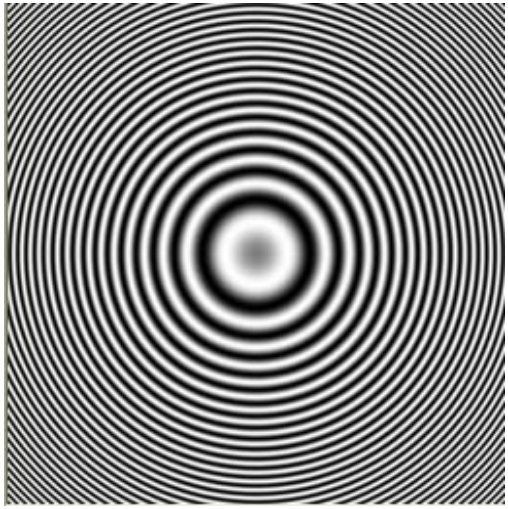
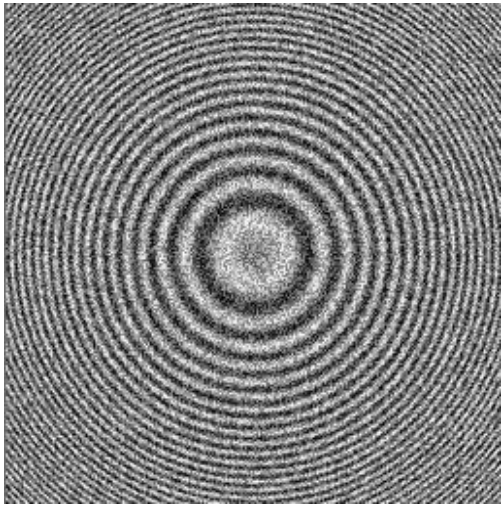


image2



```
Universal image quality index of the distorted image2 -> pQIndex = 0.6
```

Image Proximity Measures

The functions described in this section compute the proximity (similarity) measure between an image and a template (another image). These functions may be used as feature detection functions, as well as the components of more sophisticated techniques.

There are several ways to compute the measure of similarity between two images. One way is to compute the Euclidean distance, or sum of the squared distances (SSD), of an image and a template. The smaller is the value of SSD at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The squared Euclidean distance $S_{tx}(r,c)$ between a template and an image for the pixel in row r and column c is given by the equation:

$$S_{tx}(r,\ c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} \left[t(j,\ i) - x\left(r + j - \frac{tplRows}{2},\ c + i - \frac{tplCols}{2}\right)\right]^2$$

where $x(r,c)$ is the image pixel value in row r and column c, and $t(j,i)$ is the template pixel value in row j and column i; template size is *tplCols* by *tplRows* and its center is positioned at (r,c).

The other similarity measure is the cross-correlation function: the higher is the cross-correlation at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The cross-correlation $R_{tx}(r,c)$ between a template and an image at the pixel in row r and column c is computed by the equation :

$$R_{tx}(r,\ c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t(j,\ i) \cdot x\left(r + j - \frac{tplRows}{2},\ c + i - \frac{tplCols}{2}\right)$$

The cross-correlation function is dependent on the brightness variation across the image. To avoid this dependence, the correlation coefficient function is used instead. It is defined as:

$$G_{tx}(r,\ c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} [t(j,\ i) - \bar{t}] \cdot \left[x\left(r + j - \frac{tplRows}{2},\ c + i - \frac{tplCols}{2}\right) - \bar{x}(r,\ c)\right]$$

where $\bar{t}$ is the mean of the template, and $\bar{x}$ is the mean of the image in the region just under the template.

All Intel IPP proximity functions compute **normalized** values of SSD, cross-correlation and correlation coefficient that are defined as follows:

normalized SSD: $\sigma_{tx}(r, c)$

$$\sigma_{tx}(r, c) = \frac{S_{tx}(r, c)}{\sqrt{R_{xx}(r, c) R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

normalized cross-correlation $\rho_{tx}(r, c)$:

$$\rho_{tx}(r, c) = \frac{R_{tx}(r, c)}{\sqrt{R_{xx}(r, c) R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

Here R_{xx} and R_{tt} denote the auto-correlation of the image and the template, respectively:

$$R_{xx}(r, c) = \sum_{j=r-\frac{tplRows-1}{2}}^{r+\frac{tplRows-1}{2}} \sum_{i=c-\frac{tplCols-1}{2}}^{c+\frac{tplCols-1}{2}} x_{j,i} x_{j,i}$$

$$R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t_{j,i} t_{j,i}$$

Normalized correlation coefficient $\gamma_{tx}(r,c)$:

$$\gamma_{tx}(r, c) = \frac{G_{tx}(r, c)}{\sqrt{G_{xx}(r, c) G_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

Here G_{xx} and G_{tt} denote the auto-correlations of the image and the template without constant brightness component, respectively:

$$G_{xx}(r, c) = \sum_{j=r-\frac{tplRows-1}{2}}^{r+\frac{tplRows-1}{2}} \sum_{i=c-\frac{tplCols-1}{2}}^{c+\frac{tplCols-1}{2}} [x_{j,i} - \bar{x}(r, c)]^2$$

$$G_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} (t_{j,i} - \bar{t})^2$$

SqrDistanceFull_Norm

Computes normalized Euclidean distance between an image and a template.

Syntax

Case 1: Operation with integer output

```
IppStatus ippiSqrDistanceFull_Norm_<mod>(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize,
    Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

Case 2: Operation on data with floating-point output

```
IppStatus ippiSqrDistanceFull_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* :

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiSqrDistanceFull_Norm` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes and returns the sum of squared distances (SSD) between the source and template images, that is, this function computes the normalized SSD value $\sigma_{tx}(r,c)$ for each pixel in source and template buffers and stores the computed value in the corresponding pixel of the output image. The size of the resulting matrix with normalized SSD coefficients is

$$(W_s + W_t - 1) * (H_s + H_t - 1),$$

where W_s, H_s denote the width and height of the source image, and W_t, H_t denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $\sigma_{tx}(r,c)$ in the introduction section above).
There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded..

NOTE. For multi-channel images each channel is processed separately

Example 11-15 shows how to use the function `ippiSqrDistanceFull_Norm_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.
`ippStsStepErr`	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

Example 11-15 Using the function `ippiSqrDistanceFull_Norm`

```
void finc_sqr_distance()
{
    Ipp32f pSrc[5*4];
    Ipp32f pTpl[5*4];
    Ipp32f pDst[9*7];

    IppiSize srcRoiSize = {5,4};
    IppiSize tplRoiSize = {5,4};
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 5*sizeof(Ipp32f);
    int dstStep = 9*sizeof(Ipp32f);

    ippiSet_32f_C1R(2.0, pSrc, srcStep, srcRoiSize);
    ippiSet_32f_C1R(1.0, pTpl, tplStep, tplRoiSize);

    ippiSqrDistanceFull_Norm_32f_C1R( pSrc, srcStep, srcRoiSize, pTpl, tplStep,
                                      tplRoiSize, pDst, dstStep);
}
Result

1

1.58 1.12  0.91  0.79  0.71  0.79  0.91  1.12  1.58
```

Example 11-15 Using the function `ippiSqrDistanceFull_Norm`

```
1.29 0.91  0.75  0.65  0.58  0.65  0.75  0.91  1.92
1.12 0.79  0.65  0.56  0.50  0.56  0.65  0.79  1.12
1.29 0.91  0.75  0.65  0.58  0.65  0.75  0.91  1.29
1.58 1.12  0.91  0.79  0.71  0.79  0.91  1.12  1.58
2.24 1.58  1.29  1.12  1.00  1.12  1.29  1.58  2.24
```

1.

SqrDistanceSame_Norm

Computes normalized Euclidean distance between an image and a template.

Syntax

Case 1: Operation with integer output

```
IppStatus ippiSqrDistanceSame_Norm_<mod>(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

8u_C1RSfs

8u_C3RSfs

8u_C4RSfs

8u_AC4RSfs

Case 2: Operation on data with floating-point output

```
IppStatus ippiSqrDistanceSame_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* ::

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiSqrDistanceSame_Norm` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the normalized SSD values $\sigma_{tx}(r,c)$ for the pixels that belong only to the source image. The size of the resulting matrix with normalized SSD coefficients is equal to the size of the source image:

$W_s * H_s$,

where W_s, H_s denote the width and height of the source image, respectively.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $\sigma_{tx}(r, c)$ in the introduction section above).
There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded..

NOTE. For multi-channel images each channel is processed separately

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.
ippStsStepErr	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

SqrDistanceValid_Norm

Computes normalized Euclidean distance between an image and a template.

Syntax

Case 1: Operation with integer output

```
IppStatus ippiSqrDistanceValid_Norm_<mod>(const Ipp8u* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs

Case 2: Operation on data with floating-point output

```
IppStatus ippiSqrDistanceValid_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int
    tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* ::

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiSqrDistanceValid_Norm` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the normalized SSD values $\sigma_{tx}(r,c)$ for the pixels of the source image without zero-padded edges. The size of the resulting matrix with normalized SSD values is

$$(W_s - W_t + 1) * (H_s - H_t + 1),$$

where W_s, H_s denote the width and height of the source image, and W_t, H_t denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $\sigma_{tx}(r,c)$ in the introduction section above)..

NOTE. For multi-channel images each channel is processed separately

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.

ippStsStepErr	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

CrossCorrFull_Norm

Computes normalized cross-correlation between an image and a template.

Syntax

Case 1: Operation with integer output

```
IppStatus ippiCrossCorrFull_Norm_<mod>(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrFull_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod*:

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.

pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiCrossCorrFull_Norm` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes and returns the cross correlation between the source and template images, that is, this function computes the normalized cross-correlation value $\rho_{tx}(r,c)$ for each pixel in source and template buffers and stores the computed value in the corresponding pixel of the output image. The size of the resulting matrix with normalized cross-correlation coefficients is

$$(W_s + W_t - 1) * (H_s + H_t - 1),$$

where W_s, H_s denote the width and height of the source image, and W_t, H_t denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $\rho_{tx}(r,c)$ in the introduction section above.)
There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded.

NOTE. For multi-channel images, each channel is processed separately

Application Notes

Intel IPP does not provide *absolute* accuracy for rather sophisticated functions. As these functions are intended for `32f` data type, all calculations are performed in floats. To avoid divisions by zeroes as well as overflows and underflows, substitutions are implemented for very small denominator values in the course of the normalization phase. Output for these functions is always correct when the source image pixels are normalized - that is they must be in the range 0.0f - 1.0f. The following code example demonstrates how the normalization can be done to have a correct result.

```
ippiMinMax_32f_C1R( pSrc, srcStep, tmpRoi, &min, &max );

normC = 1.0f/max;

ippiMulC_32f_C1R( normC, pSrc ..);
```

The only requirement is that both the nominator and the denominator (see the formulas above) must exceed `#define IPP_EPS23 ( 1.19209289e-07f )`, otherwise substitution is used.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.
`ippStsStepErr`	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

CrossCorrSame_Norm

Computes normalized cross-correlation between an image and a template.

Syntax

Case 1: Operation with integer output

```
IppStatus ippiCrossCorrSame_Norm_<mod>(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
```

```
8u_AC4RSfs
```

Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrSame_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* ::

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiCrossCorrSame_Norm` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the normalized cross-correlation values $\rho_{tx}(r,c)$ for the pixels that belong only to the source image. The size of the resulting matrix with normalized cross-correlation values is equal to the size of the source image:

$$W_s * H_s,$$

where W_s, H_s denote the width and height of the source image, respectively.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $\rho_{tx}(r,c)$ in the introduction section above.)
There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded..

NOTE. For multi-channel images each channel is processed separately

Example 11-16 shows how to use the function `ippiCrossCorrSame_Norm_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.
`ippStsStepErr`	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

Example 11-16 Using the function `ippiCrossCorrSame_Norm`

```
void func_CrossCorrSame_Norm()
{
    Ipp32f pSrc[5*4] = { 1.0, 2.0, 1.5, 4.1, 3.6,
                         0.2, 3.2, 2.5, 1.5, 10.0,
                         5.0, 6.8, 0.5, 4.1, 1.1,
                         7.1, 4.2, 2.2, 8.7, 10.0};
    Ipp32f pTpl[3*3] = {2.1, 3.5, 7.7,
                        0.4, 2.3, 5.5,
                        1.4, 2.8, 3.1};
    Ipp32f pDst[5*4];

    IppiSize srcRoiSize = {5,4};
    IppiSize tplRoiSize = {3,3};
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 3*sizeof(Ipp32f);
    int dstStep = 5*sizeof(Ipp32f);

    ippiCrossCorrSame_Norm_32f_C1R(pSrc, srcStep, srcRoiSize, pTpl, tplStep,
tplRoiSize, pDst, dstStep);
}

Result: pDst ->
```

Example 11-16 Using the function `ippiCrossCorrSame_Norm`

```
0.53  0.54  0.58  0.50  0.30
0.68  0.62  0.68  0.83  0.38
0.77  0.55  0.60  0.81  0.42
0.81  0.46  0.70  0.62  0.24
```

CrossCorrValid_Norm

Computes normalized cross-correlation between an image and a template.

Syntax

Case 1: Operation with integer output

```
IppStatus ippiCrossCorrValid_Norm_<mod>(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrValid_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int
    tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* ::

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiCrossCorrValid_Norm` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the normalized cross-correlation values $\rho_{tx}(r,c)$ for the pixels of the source image without zero-padded edges. The size of the resulting matrix with normalized cross-correlation values is

$$(W_s - W_t + 1) * (H_s - H_t + 1),$$

where W_s, H_s denote the width and height of the source image, and W_t, H_t denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $\rho_{tx}(r,c)$ in the introduction section above.).

NOTE. For multi-channel images each channel is processed separately

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.
`ippStsStepErr`	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

CrossCorrValid

Computes cross-correlation between an image and a template.

Syntax

```
IppStatus ippiCrossCorrValid_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int
    tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* ::

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiCrossCorrValid` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the cross-correlation values $R_{tx}(r,c)$ for the pixels of the source image without zero-padded edges. The size of the resulting matrix with cross-correlation values is

$$(W_s - W_t + 1) * (H_s - H_t + 1),$$

where W_s, H_s denote the width and height of the source image, and W_t, H_t denote the

width and height of the template.
The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $R_{tx}(r,c)$ in the introduction section above.).

NOTE. For multi-channel images each channel is processed separately

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.
ippStsStepErr	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

CrossCorrFull_NormLevel

Computes normalized correlation coefficient between an image and a template.

Syntax

Case 1: Operation with integer output

```
IppStatus ippiCrossCorrFull_NormLevel_<mod>(const Ipp8u* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs

Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrFull_NormLevel_<mod>(const Ipp<srcDatatype>*
    pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* ::

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiCrossCorrFull_NormLevel` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes and returns the correlation coefficient between the source and template images, that is, this function computes the normalized correlation coefficient $\gamma_{tx}(r,c)$ for each pixel in source and template buffers and stores the computed value in the corresponding pixel of the output image. The size of the resulting matrix with normalized cross-correlation coefficients is

$$(W_s + W_t - 1) * (H_s + H_t - 1),$$

where W_s, H_s denote the width and height of the source image, and W_t, H_t denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $\gamma_{tx}(r,c)$ in the introduction section above.)
There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.
ippStsStepErr	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

CrossCorrSame_NormLevel

Computes normalized correlation coefficient between an image and a template.

Syntax

Case 1: Operation with integer output

```
IppStatus ippiCrossCorrSame_NormLevel_<mod>(const Ipp8u* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs

Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrSame_NormLevel_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int
    tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* ::

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiCrossCorrSame_NormLevel` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the normalized correlation coefficients $\gamma_{tx}(r,c)$ for the pixels that belong only to the source image. The size of the resulting matrix with normalized correlation coefficients is equal to the size of the source image:

$W_s * H_s$,

where W_s, H_s denote the width and height of the source image, respectively.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $\gamma_{tx}(r,c)$ in the introduction section above.)
There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded..

NOTE. For multi-channel images each channel is processed separately

Example 11-17 shows how to use the function `ippiCrossCorrSame_NormLevel_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.
`ippStsStepErr`	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

Example 11-17 Using the function `ippiCrossCorrSame_NormLevel`

```
void func_CrossCorrSame_NormLevel()
{
    Ipp32f pSrc[5*4] = { 1.0, 2.0, 1.5, 4.1, 3.6,
                         0.2, 3.2, 2.5, 1.5, 10.0,
                         5.0, 6.8, 0.5, 4.1, 1.1,
                         7.1, 4.2, 2.2, 8.7, 10.0};

    Ipp32f pTpl[3*3] = {2.1, 3.5, 7.7,
                        0.4, 2.3, 5.5,
                        1.4, 2.8, 3.1};
    Ipp32f pDst[5*4];

    IppiSize srcRoiSize = {5, 4};
    IppiSize tplRoiSize = {3, 3};
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 3*sizeof(Ipp32f);
    int dstStep = 5*sizeof(Ipp32f);

    ippiCrossCorrSame_NormLevel_32f_C1R(pSrc, srcStep, srcRoiSize,
 pTpl, tplStep, tplRoiSize, pDst, dstStep);
}
Result:              pDst
          0.16  -0.14  -0.17  -0.10  -0.36
          0.31  -0.08  -0.12   0.55  -0.43
          0.42  -0.49  -0.34   0.46  -0.39
          0.63  -0.38   0.31   0.21  -0.53
```

CrossCorrValid_NormLevel

Computes normalized cross-correlation between an image and a template.

Syntax

Case 1: Operation with integer output

```
IppStatus ippiCrossCorrValid_NormLevel_<mod>(const Ipp8u* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrValid_NormLevel_<mod>(const Ipp<srcDatatype>*
    pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* ::

32f_C1R	8u32f_C1R	8s32f_C1R	16u32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R	16u32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R	16u32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R	16u32f_AC4R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image buffer.
tplStep	Distance in bytes between starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiCrossCorrValid_NormLevel` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function computes the normalized correlation coefficients $\gamma_{tx}(r,c)$ for the pixels of the source image without zero-padded edges. The size of the resulting matrix with normalized correlation coefficients is

$$(W_s - W_t + 1) * (H_s - H_t + 1),$$

where W_s, H_s denote the width and height of the source image, and W_t, H_t denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for $\gamma_{tx}(r,c)$ in the introduction section above.).

NOTE. For multi-channel images each channel is processed separately

Example 11-17 shows how to use the function `ippiCrossCorrValid_NormLevel_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition when *srcRoiSize* or *tplRoiSize* has a field with zero or negative value, or when *srcRoiSize* has a field with value smaller than value of the corresponding field of *tplRoiSize*.
`ippStsStepErr`	Indicates an error condition if at least one of *srcStep*, *tplStep*, or *dstStep* has a zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

Example 11-18 Using the function `ippiCrossCorrValid_NormLevel`

```
void func_ippiCrossCorrValid_NormLevel()
{
    Ipp32f pSrc[5*4] = { 1.0, 2.0, 1.5, 4.1, 3.6,
                         0.2, 3.2, 2.5, 1.5, 10.0,
                         5.0, 6.8, 0.5, 4.1, 1.1,
```

Example 11-18 Using the function ippiCrossCorrValid_NormLevel

```
                        7.1, 4.2, 2.2, 8.7, 10.0};

    Ipp32f pTpl[3*3] = {2.1, 3.5, 7.7,
                        0.4, 2.3, 5.5,
                        1.4, 2.8, 3.1};

    IppiSize srcRoiSize = {5, 4};
    IppiSize tplRoiSize = {3, 3};
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 3*sizeof(Ipp32f);
    int dstStep = 3*sizeof(Ipp32f);

// size of pDst is (srcRoiSize.width - tplRoiSize.width + 1)*(srcRoiSize.height
- tplRoiSize.height + 1) = 3*2
    Ipp32f pDst[3*2];

    ippiCrossCorrValid_NormLevel_32f_C1R(pSrc, srcStep, srcRoiSize, pTpl,
tplStep, tplRoiSize, pDst, dstStep);
}
Result:            pDst
 -0.08  -0.12  0.55
 -0.49  -0.34  0.46
```

Image Geometry Transforms

12

This chapter describes the Intel® IPP image processing functions that perform geometric operations of resizing, rotating, warping and remapping an image.

Table 12-1 lists the Intel IPP image geometric transform functions:

Table 12-1 Image Geometric Transform Functions

Function Base Name	Operation
Resize	Changes the size of an image.
ResizeCenter	Changes the size of an image and shifts it.
ResizeSqrPixel	Changes the size of an image using different interpolation algorithm.
ResizeGetBufSize	Computes the size of the external buffer for image resizing.
ResizeSqrPixelGetBufSize	Computes the size of the external buffer for image resizing.
GetResizeFract	Recalculates resize factors for a tiled image.
ResizeShift	Changes the size of an image tile.
ResizeYUV422	Changes the size of an 2-channel image in 4:2:2 sampling format.
ResizeFilterGetSize	Calculates the size of the state structurer for resizing filter.
ResizeFilterInit	Initializes the state structurer for resizing filter.
ResizeFilter	Changes the size of an image using a generic filter.
Mirror	Mirrors an image.
Remap	Performs arbitrary remapping of pixels in an image.
Rotate	Rotates an image around the origin (0,0) and shifts the result.

Table 12-1 Image Geometric Transform Functions (continued)

Function Base Name	Operation
GetRotateShift	Computes the shift values for ippiRotate, given the rotation center and angle.
AddRotateShift	Computes shift values for an image rotation around the specified center with specified shifts.
GetRotateQuad	Computes the quadrangle that results from an image ROI transformed by ippiRotate function.
GetRotateBound	Computes the bounding rectangle for an image ROI transformed by ippiRotate function.
RotateCenter	Rotates an image around an arbitrary center.
Shear	Performs a shearing transform of an image.
GetShearQuad	Computes the quadrangle that results from an image ROI transformed by ippiShear function.
GetShearBound	Computes the bounding rectangle for an image ROI transformed by ippiShear function.
WarpAffine	Warps an image with an affine transform.
WarpAffineBack	Performs an inverse affine transform of an image.
WarpAffineQuad	Performs an affine transform of an image from the source quadrangle to the destination quadrangle.
GetAffineQuad	Computes the quadrangle that results from an image ROI transformed by ippiWarpAffine function.
GetAffineBound	Computes the bounding rectangle for an image ROI transformed by ippiWarpAffine function.
GetAffineTransform	Computes the affine transform coefficients.
WarpPerspective	Warps an image with a perspective transform.
WarpPerspectiveBack	Performs an inverse perspective transform of an image.
WarpPerspectiveQuad	Performs a perspective transform of an image from the source quadrangle to the destination quadrangle.
GetPerspectiveQuad	Computes the quadrangle that results from an image ROI transformed by ippiWarpPerspective function.
GetPerspectiveBound	Computes the bounding rectangle for an image ROI transformed by ippiWarpPerspective function.
GetPerspectiveTransform	Computes the perspective transform coefficients.

Table 12-1 Image Geometric Transform Functions (continued)

Function Base Name	Operation
WarpBilinear	Warps an image with a bilinear transform.
WarpBilinearBack	Performs an inverse bilinear transform of an image.
WarpBilinearQuad	Performs a bilinear transform of an image from the source quadrangle to the destination quadrangle.
GetBilinearQuad	Computes the quadrangle that results from an image ROI transformed by `ippiWarpBilinear` function.
GetBilinearBound	Computes the bounding rectangle for an image ROI transformed by `ippiWarpBilinear` function.
GetBilinearTransform	Computes the bilinear transform coefficients.

Most functions performing geometric transform of an image uses some interpolation algorithm to resample the image. The type of interpolation method to be used is passed to the function in the *`interpolation`* parameter.

The following interpolation algorithms are used:

— nearest neighbor

— linear interpolation

— cubic convolution

— supersampling

— interpolation using Lanczos window function

— optional edge smoothing of the destination image.

The nearest neighbor algorithm is the fastest, while other methods yield higher quality results, but are slower.

Use one of the following constant identifiers for the applicable interpolation methods:

`IPPI_INTER_NN`	Nearest neighbor interpolation.
`IPPI_INTER_LINEAR`	Linear interpolation
`IPPI_INTER_CUBIC`	Cubic interpolation
`IPPI_INTER_LANCZOS`	Interpolation using 3-lobed Lanczos window function.
`IPPI_INTER_SUPER`	Supersampling interpolation.

For certain functions, you can combine the above interpolation algorithms with additional smoothing (antialiasing) of edges to which the original image borders are transformed. To use this edge smoothing, set the parameter *interpolation* to the bitwise OR of IPPI_SMOOTH_EDGE and the desired interpolation mode.

CAUTION. You can use interpolation with edge smoothing option only in those geometric transform functions where this option is explicitly listed in the parameters definition section.

See appendix B "Interpolation in Image Geometric Transform Functions" for more information on the interpolation algorithms that are used in the library.

Many solutions and hints for use of these functions can be found in Intel® IPP Samples. See *Intel IPP Image Processing and Media Processing Samples* downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

ROI Processing in Geometric Transforms

All the transform functions described in this chapter operate in rectangular regions of interest (ROIs) that are defined in both the source and destination images.
The procedures for handling ROIs in geometric transform functions differ from those used in other functions (see Regions of Interest in IPP in chapter 2). The main difference is that operations take place in the intersection of the *transformed* source ROI and the destination ROI. More specifically, all geometric transform functions (except those which perform inverse warping operations) handle ROIs with the following sequence of operations:

- transform the rectangular ROI of the source image to quadrangle in the destination image;
- find the intersection of this quadrangle and the rectangular ROI of the destination image;
- update the destination image in the intersection area.

The coordinates in the source and destination images must have the same origin.

Using functions with ROI allows every scanline of a source image to be padded with zeroes for alignment, so that actual size of a scanline in bytes can be greater than it may be assumed from the image width. In that case the size of each row of image data in bytes is determined by the value of *srcStep* parameter, which gives distance in bytes between the starts of consecutive lines of an image.

To fully describe a rectangular ROI, both its origin (coordinates of top left corner) and size must be referenced. For geometrical transform functions, the source image ROI is specified by *srcRoi* parameter of IppiRect type, meaning that all four values describing the rectangular ROI are given explicitly.
On the other hand, the destination image ROI for different functions can be specified either by *dstRoi* parameter of IppiRect type, or *dstRoiSize* parameter of IppiSize type. In the latter case, only the destination ROI size is passed, while its origin is referenced by *pDst* pointer.

Geometric Transform Functions

Resize

Changes an image size.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* pSrc, IppiSize
    srcSize,int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int
    dstStep, IppiSize dstRoiSize, double xFactor, double yFactor, int
    interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* const pSrc[3], IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[3],
    int dstStep, IppiSize dstRoiSize, double xFactor, double yFactor, int
    interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* const pSrc[4], IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const pDst[4],
    int dstStep, IppiSize dstRoiSize, double xFactor, double yFactor, int
    interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R     32f_P4R
```

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image ROI. An array of pointers to separate ROI in each planes for planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoiSize	Size of the destination ROI in pixels.
xFactor, yFactor	Factors by which the *x* and *y* dimensions of the source ROI are changed. The factor value greater than `1` corresponds to increasing the image size in that dimension.
interpolation	The type of interpolation to perform for image resampling. Use one of the following values:

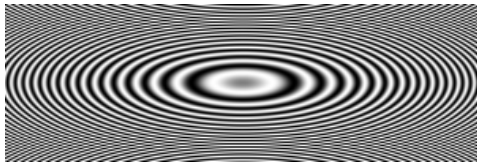
`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation
`IPPI_INTER_SUPER`	supersampling interpolation, can not be applied for image enlarging
`IPPI_INTER_LANCZOS`	interpolation with Lanczos window.

Description

The function `ippiResize` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor, yFactor*. The result is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI. Supersampling interpolation can be applied if both *xFactor* and *yFactor* are less than or equal to 1. Figure 12-1 shows the result of applying the `ippiResize` function to the sample image of size 256 by 256 pixels. The destination image is of 300 by 100 pixels size.

Figure 12-1 Sample Image Changed by the `ippiResize` Function



The following code Example 12-1 shows how to use the `ippiResize` function:

Example 12-1 Resizing an Image

```
IppStatus resize( void ) {
   Ipp8u x[8*8], y[8*8];
   IppiSize srcsz={8,8}, dstroi={6,6};
   IppiRect srcroi={1,1,6,6};
   IppiSize roi = {8,8};
 int i;
   for( i=0; i<8; ++i ) {
      ippiSet_8u_C1R( (Ipp8u)i, x+8*i+i, 8, roi );
      --roi.width;
      --roi.height;
   }
   return ippiResize_8u_C1R( x, srcsz, 8, srcroi, y, 8, dstroi, 2.9, 2,
IPPI_INTER_NN );
}
The image has the following contents after resizing:
  01 01 01 01 01 01 CC CC
  01 01 01 01 01 01 CC CC
  01 01 02 02 02 02 CC CC
  01 01 02 02 02 02 CC CC
  01 01 02 02 02 03 CC CC
  01 01 02 02 02 03 CC CC
  CC CC CC CC CC CC CC CC
  CC CC CC CC CC CC CC CC
```

This function is used in H.264 encoder included into Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *srcSize* or *dstRoiSize* has a field with zero or negative value.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.
ippStsResizeNoOperationErr	Indicates an error condition if one of the destination image dimensions is less than 1 pixel.
ippStsResizeFactorErr	Indicates an error condition if *xFactor* or *yFactor* is less than or equal to zero.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.

ResizeCenter

Changes an image size and shifts the destination image relative to the given point.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiSize dstRoiSize, double xFactor, double yFactor,
    double xCenter, double yCenter, int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[3], int dstStep, IppiSize dstRoiSize,
    double xFactor, double yFactor, double xCenter, double yCenter,
    int interpolation);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R     32f_P3R
```

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[4], int dstStep, IppiSize dstRoiSize,
    double xFactor, double yFactor, double xCenter, double yCenter,
    int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R     32f_P4R
```

Parameters

pSrc	Pointer to the source image origin. An array of 3 or 4 pointers to different color planes in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image ROI. An array of 3 or 4 pointers to destination ROI in different color planes for planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoiSize	Size of the destination ROI in pixels.
xFactor, yFactor	Factors by which the *x* and *y* dimensions of the source ROI are changed. The factor value greater than `1` corresponds to increasing the image size in that dimension.
xCenter, yCenter	Coordinates of the preset center.
interpolation	The type of interpolation to perform for image resampling. Use one of the following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation

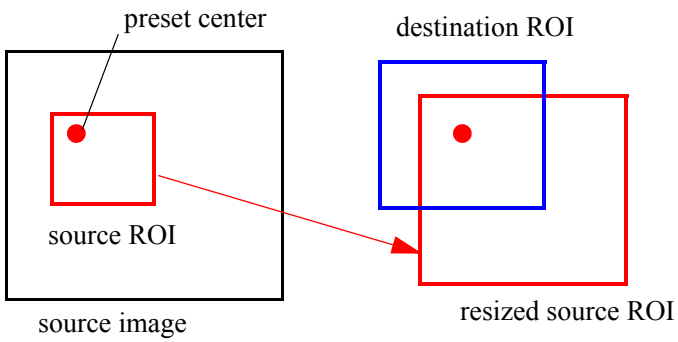
IPPI_INTER_CUBIC	cubic interpolation
IPPI_INTER_SUPER	supersampling interpolation, can not be applied for image enlarging
IPPI_INTER_LANCZOS	interpolation with Lanczos window.

Description

The function `ippiResizeCenter` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor, yFactor*.
Unlike `ippiResize`, this function also shifts the destination image in such a way that a given point with coordinate *xCenter*, *yCenter* - preset center - remains stationary after resizing, which creates an effect of changing the image size relative to that point. The preset center is always positioned in the center of the destination image ROI with coordinates *dstROI.width/2*; *dstROI.height/2* (see Figure 12-2). Note that the preset center may be specified in an arbitrary point including points outside the source image..

Figure 12-2 Resizing Image with the Function `ippiResizeCenter`



The result is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI. Supersampling interpolation can be applied if both *xFactor* and *yFactor* are less than or equal to 1.

The code Example 12-2 shows how to use the `ippiResizeCenter` function.

Example 12-2 Resizing and Shifting an Image

```
IppStatus resizeCenter( void ) {
   Ipp8u x[8*8], y[8*8];
   IppiSize sz = {8,8}, roi = {8,8};
   IppiRect rect = {0,0,8,8};
   int i;
   for( i=0; i<8; ++i ) {
      ippiSet_8u_C1R( (Ipp8u)i, x+8*i+i, 8, roi );
      --roi.width;
      --roi.height;
   }
   return ippiResizeCenter_8u_C1R( x, sz, 8, rect, y, 8, sz,
      2.9, 2.0, 4, 4, IPPI_INTER_NN );
}
```

The image has the following contents after applying the `ippiResizeCenter` function:

```
02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02
02 02 03 03 03 03 03 03
02 02 03 03 03 03 03 03
02 02 03 03 03 04 04 04
02 02 03 03 03 04 04 04
02 02 03 03 03 04 04 04
02 02 03 03 03 04 04 04
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
`ippStsSizeErr`	Indicates an error condition if *srcSize* or *dstRoiSize* has a field with zero or negative value.
`ippStsWrongIntersectROI`	Indicates a warning if *srcRoi* has no intersection with the source image.
`ippStsResizeNoOperationErr`	Indicates an error condition if one of the destination image dimensions is less than 1 pixel.

ippStsResizeFactorErr	Indicates an error condition if *xFactor* or *yFactor* is less than or equal to zero.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.

ResizeSqrPixel

Changes an image size using different interpolation algorithm.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiResizeSqrPixel_<mod>(const Ipp<datatype>* pSrc, IppiSize
srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep,
IppiRect dstRoi, double xFactor, double yFactor, double xShift, double
yShift, int interpolation, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiResizeSqrPixel_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, double xFactor, double
    yFactor, double xShift, double yShift, int interpolation, Ipp8u*
    pBuffer);
```

Supported values for *mod*:

8u_P3R	16u_P3R	16s_P3R	32f_P3R

```
IppStatus ippiResizeSqrPixel_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, double xFactor, double
    yFactor, double xShift, double yShift, int interpolation, Ipp8u*
    pBuffer);
```

Supported values for *mod*:

`8u_P4R` `16u_P4R` `16s_P4R` `32f_P4R`

Parameters

pSrc	Pointer to the source image origin. An array of 3 or 4 pointers to different color planes for planar image.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image. An array of 3 or 4 pointers to the separate color planes for planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
xFactor, yFactor	Factors by which the *x* and *y* dimensions of the source ROI are changed. The factor value greater than `1` corresponds to increasing the image size in that dimension.
xShift, yShift	Shift values in the *x* and *y* directions respectively.
interpolation	The type of interpolation to perform for image resampling. Use one of the following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation
`IPPI_INTER_LANCZOS`	interpolation with Lanczos window
`IPPI_INTER_CUBIC2P_BSPLINE`	B-spline filter
`IPPI_INTER_CUBIC2P_CATMULLROM`	Catmull-Rom spline
`IPPI_INTER_CUBIC2P_B05C03`	special two-parameters (1/2, 3/10)
`IPPI_INTER_SUPER`	supersampling interpolation.

pBuffer	Pointer to the external buffer.

Description

The function `ippiResizeSqrPixel` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor, yFactor*. The result is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI.

Unlike `ippiResize`, this function uses different algorithm for interpolation.

Pixel coordinates x' and y' in the resized image are obtained from the following equations:

$$x' = xFactor*x + xShift$$
$$y' = yFactor*y + yShift$$

where x and y denote the pixel coordinates in the source image.

The function requires the external buffer *pBuffer*, its size can be previously computed by calling the function `ResizeSqrPixelGetBufSize` or `ResizeGetBufSize`.

Example 12-3 shows how to use the function `ippiResizeSqrPixel_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if width or height of images has zero or negative value.
`ippStsResizeFactorErr`	Indicates an error condition if *xFactor* or *yFactor* is less than or equal to zero.
`ippStsInterpolationErr`	Indicates an error condition if *interpolation* has an illegal value.
`ippStsWrongIntersectROI`	Indicates a warning if *srcRoi* has no intersection with the source image.

Example of Using the Function ippiResizeSqrPixel.

The following Example 12-3 shows how the Intel IPP functions can be used to resample the source image of size 12x12 to the destination image of size 6x6 with and without using OMP.

Please note that this code is developed for demonstration purposes only. Do not use OMP for small image processing - it causes degradation of performance.

Example 12-3 Using the Function ippiResizeSqrPixel

```
#ifdef _OPENMP
#include <omp.h>
#endif

IppStatus ResizeSqrPixel( void )
{
    Ipp32f src[12*12], dst[12*12];
    IppiSize size  = {12,12};
    IppiRect srect = {0,0,12,12};
    IppiRect drect = {0,0,6,6};
    Ipp8u *buf;
    int bufsize, nt;
    IppStatus status = ippStsNoErr;

    /* source image */
    {
        IppiSize roi = {12,12};
        int i;
        for( i=0; i<12; ++i ) {
            ippiSet_32f_C1R( (Ipp32f)i, src+12*i+i, 12*sizeof(Ipp32f), roi );
            --roi.width;
            --roi.height;
        }
    }
    /* calculation of work buffer size */
    ippiResizeGetBufSize( srect, drect, 1, IPPI_INTER_SUPER, &bufsize );

#ifdef _OPENMP
/* parallel algorithm */
  #pragma omp parallel
    {
  #pragma omp master
      {
        /* number of threads */
        nt = omp_get_num_threads();
        /* memory allocate */
        buf = ippsMalloc_8u( nt*bufsize );
      }
  #pragma omp barrier
      {
        /* thread's number */
        int id = omp_get_thread_num();
        /* ROI for one slice */
        drect.y      = (drect.height/nt)*id;
        drect.height = (drect.height/nt)*(id+1);
        if( NULL != buf )
            status = ippiResizeSqrPixel_32f_C1R(
                src, size, 12*sizeof(Ipp32f), srect,
                dst, 6*sizeof(Ipp32f), drect,
```

Example 12-3 Using the Function `ippiResizeSqrPixel` (continued)

```
                0.5, 0.5, 0, 0, IPPI_INTER_SUPER, buf+id*bufsize );
        }
    }
#else
/* single algorithm */
    {
        /* memory allocate */
        buf = ippsMalloc_8u( bufsize );
        if( NULL != buf )
            status = ippiResizeSqrPixel_32f_C1R(
                src, size, 12*sizeof(Ipp32f), srect,
                dst, 6*sizeof(Ipp32f), drect,
                0.5, 0.5, 0, 0, IPPI_INTER_SUPER, buf );
    }
#endif
    /* memory free */
    if( NULL != buf ) ippsFree( buf );

    return status;
}


The source image

 0  0  0  0  0  0  0  0  0  0  0  0
 0  1  1  1  1  1  1  1  1  1  1  1
 0  1  2  2  2  2  2  2  2  2  2  2
 0  1  2  3  3  3  3  3  3  3  3  3
 0  1  2  3  4  4  4  4  4  4  4  4
 0  1  2  3  4  5  5  5  5  5  5  5
 0  1  2  3  4  5  6  6  6  6  6  6
 0  1  2  3  4  5  6  7  7  7  7  7
 0  1  2  3  4  5  6  7  8  8  8  8
 0  1  2  3  4  5  6  7  8  9  9  9
 0  1  2  3  4  5  6  7  8  9 10 10
 0  1  2  3  4  5  6  7  8  9 10 11

The image has the following contents after resizing with NEAREST NEIGHBOR
interpolation

 1.0000000  1.0000000  1.0000000  1.0000000  1.0000000  1.0000000
 1.0000000  3.0000000  3.0000000  3.0000000  3.0000000  3.0000000
 1.0000000  3.0000000  5.0000000  5.0000000  5.0000000  5.0000000
 1.0000000  3.0000000  5.0000000  7.0000000  7.0000000  7.0000000
 1.0000000  3.0000000  5.0000000  7.0000000  9.0000000  9.0000000
 1.0000000  3.0000000  5.0000000  7.0000000  9.0000000 11.0000000

The image has the following contents after resizing with LINEAR interpolation

0.25000000  0.5000000  0.5000000  0.5000000  0.5000000  0.5000000
0.50000000  2.2500000  2.5000000  2.5000000  2.5000000  2.5000000
```

Example 12-3 Using the Function `ippiResizeSqrPixel` (continued)

```
0.50000000  2.5000000  4.2500000  4.5000000  4.5000000  4.5000000
0.50000000  2.5000000  4.5000000  6.2500000  6.5000000  6.5000000
0.50000000  2.5000000  4.5000000  6.5000000  8.2500000  8.5000000
0.50000000  2.5000000  4.5000000  6.5000000  8.5000000 11.2500000

The image has the following contents after resizing with CUBIC interpolation

0.25390625  0.4335938  0.4375000  0.4375000  0.4375000  0.4375000
0.43359375  2.3828125  2.4960938  2.5000000  2.5000000  2.5000000
0.43750000  2.4960938  4.3828125  4.4960938  4.5000000  4.5000000
0.43750000  2.5000000  4.4960938  6.3828125  6.4960938  6.5000000
0.43750000  2.5000000  4.5000000  6.4960938  8.3828125  8.4960938
0.43750000  2.5000000  4.5000000  6.5000000  8.4960938 10.3789060

The image has the following contents after resizing with LANCZOS interpolation

0.26301101  0.3761742  0.4124454  0.4130435  0.4130435  0.4130435
0.37617415  2.4499352  2.4631310  2.4994020  2.5000002  2.5000002
0.41244543  2.4631310  4.4499354  4.4631310  4.4994025  4.5000005
0.41304356  2.4994023  4.4631310  6.4499359  6.4631314  6.4994025
0.41304356  2.5000005  4.4994020  6.4631314  8.4499359  8.4631310
0.41304356  2.5000005  4.5000000  6.4994025  8.4631310 10.4369250

The image has the following contents after resizing with SUPERSAMPLING
interpolation

0.25000000  0.5000000  0.5000000  0.5000000  0.5000000  0.5000000
0.50000000  2.2500000  2.5000000  2.5000000  2.5000000  2.5000000
0.50000000  2.5000000  4.2500000  4.5000000  4.5000000  4.5000000
0.50000000  2.5000000  4.5000000  6.2500000  6.5000000  6.5000000
0.50000000  2.5000000  4.5000000  6.5000000  8.2500000  8.5000000
0.50000000  2.5000000  4.5000000  6.5000000  8.5000000 10.2500000
```

ResizeGetBufSize

Computes the size of the external buffer for image resizing.

Syntax

```
IppStatus ippiResizeGetBufSize(IppiRect srcRoi, IppiRect dstRoi, int
    nChannel, int interpolation, int* pBufferSize);
```

Parameters

srcRoi	Region of interest in the source image (of the `IppiRect` type).
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
nChannel	Number of channels or planes, possible values are 1, 3, 4.
interpolation	The type of interpolation to perform for image resampling. Use one of the following values: `IPPI_INTER_NN` nearest neighbor interpolation `IPPI_INTER_LINEAR` linear interpolation `IPPI_INTER_CUBIC` cubic interpolation `IPPI_INTER_LANCZOS` interpolation with Lanczos window `IPPI_INTER_CUBIC2P_BSPLINE` B-spline filter `IPPI_INTER_CUBIC2P_CATMULLROM` Catmull-Rom spline `IPPI_INTER_CUBIC2P_B05C03` special two-parameters (1/2, 3/10) `IPPI_INTER_SUPER` supersampling interpolation
pBufferSize	Pointer to the size of the external buffer.

Description

The function `ippiResizeGetBufSize` is declared in the `ippi.h` file. This function computes the size (in bytes) of the external buffer that is required for the function `ippiResizeSqrPixel`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if width or height of the *srcRoi* or *dstRoi* is less than or equal to 0.
`ippStsNumChannelErr`	Indicates an error condition if *nChannel* has an illegal value.
`ippStsInterpolationErr`	Indicates an error condition if *interpolation* has an illegal value.

ResizeSqrPixelGetBufSize

Computes the size of the external buffer for image resizing.

Syntax

```
IppStatus ippiResizeSqrPixelGetBufSize(IppiSize dstSize, int nChannel, int
    interpolation, int* pBufferSize);
```

Parameters

dstSize	Size in pixels of the destination image.
nChannel	Number of channels or planes, possible values are 1, 3, 4.
interpolation	The type of interpolation to perform for image resampling. Use one of the following values: `IPPI_INTER_NN` nearest neighbor interpolation `IPPI_INTER_LINEAR` linear interpolation `IPPI_INTER_CUBIC` cubic interpolation `IPPI_INTER_LANCZOS` interpolation with Lanczos window `IPPI_INTER_CUBIC2P_BSPLINE` B-spline filter `IPPI_INTER_CUBIC2P_CATMULLROM` Catmull-Rom spline filter `IPPI_INTER_CUBIC2P_B05C03` special two-parameters (1/2, 3/10) filter
pBufferSize	Pointer to the size of the external buffer.

Description

The function `ippiResizeSqrPixelGetBufSize` is declared in the `ippi.h` file. This function computes the size of the external buffer that is required for the function `ippiResizeSqrPixel`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *dstSize* has a field with zero or negative value.
ippStsNumChannelErr	Indicates an error condition if *nChannel* has an illegal value.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.

GetResizeFract

Recalculates resize factors for a tiled image.

Syntax

```
IppStatus ippiGetResizeFract(IppiSize srcSize, IppiRect srcRoi,
    double xFactor, double yFactor, double* xFr, double* yFr,
    int interpolation);
```

Parameters

srcSize	Size of the source image in pixels.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
xFactor, yFactor	Factors by which the *x* and *y* dimensions of the source ROI are changed. The factor value greater than `1` corresponds to increasing the image size in that dimension.
xFr, yFr	Pointers to the recalculated resize factors. These values should be passed to `ippiResizeShift` function to bring about the desired resizing of image tiles.The factor value greater than `1` corresponds to increasing the image size in that dimension.
interpolation	The type of interpolation to perform for image resampling. Use one of the following values:

IPPI_INTER_NN	nearest neighbor interpolation
IPPI_INTER_LINEAR	linear interpolation
IPPI_INTER_CUBIC	cubic interpolation
IPPI_INTER_LANCZOS	interpolation with Lanczos window.

Description

The function `ippiResizeFract` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

Use this function if you need to resize the tiled source image by *xFactor* in the *x* direction and *yFactor* in the *y* direction. It calculates the new values of resize factors *xFr* and *yFr* that should be passed to the ippiResizeShift function, which performs resize operation on the individual tiles.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsSizeErr	Indicates an error condition if *srcSize* has a field with zero or negative value.
ippStsResizeFactorErr	Indicates an error condition if *xFactor* or *yFactor* is less than or equal to zero.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.

ResizeShift

Changes the size of an image tile.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int
    dstStep, IppiSize dstRoiSize, double xFr, double yFr, double xShift,
    double yShift, int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiSize dstRoiSize, double xFr, double yFr,
    double xShift, double yShift, int interpolation);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R     32f_P3R
```

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiSize dstRoiSize, double xFr, double yFr,
    double xShift, double yShift, int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R     32f_P4R
```

Parameters

pSrc	Pointer to the source image. An array of 3 or 4 pointers to different color planes in case of data in planar format.
srcSize	Size of the source image in pixels.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image ROI. An array of 3 or 4 pointers to destination ROI in different color planes for planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoiSize	Size of the destination ROI in pixels.
xFr, yFr	Factors, that are used to obtain the desired resizing of the source image ROI by the *xFactor*, *yFactor*.
xShift, yShift	Corrections of the position of the processing area (*srcRect*).
interpolation	The type of interpolation to perform for image resampling. Use one of the following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation
`IPPI_INTER_LANCZOS`	interpolation with Lanczos window.

Description

The function `ippiResizeShift` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used for resizing the tiled image by *xFactor* in the *x* direction and *yFactor* in the *y* direction. Prior to using it, the `ippiGetResizeFract` function should be called to calculate the values of the *xFr* and *yFr* factors. The image size can be either reduced or increased in each direction, depending on the values of *xFactor, yFactor*. The value of *xFr* or *yFr* factor greater than `1` corresponds to reducing the size, and the value less than 1 corresponds to increasing the image size in that dimension.
The result is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcSize* or *dstRoiSize* has a field with zero or negative value.
`ippStsResizeFactorErr`	Indicates an error condition if *xFr* or *yFr* is less than or equal to zero.
`ippStsInterpolationErr`	Indicates an error condition if *interpolation* has an illegal value.
`ippStsWrongIntersectROIErr`	Indicates an error condition if *srcRoi* has no intersection with the source image.

ResizeYUV422

Changes a size of two-channel image.

Syntax

```
IppStatus ippiResizeYUV422_8u_C2R(const Ipp8u* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp8u* pDst, int dstStep, IppiSize
    dstRoiSize, double xFactor, double yFactor, int interpolation);
```

Parameters

pSrc	Pointer to the source image data.

srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination ROI buffer.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoiSize	Size of the destination ROI in pixels.
xFactor, yFactor	Factors by which the *x* and *y* dimensions of the source ROI are changed. The factor value greater than `1` corresponds to increasing the image size in that dimension.
interpolation	The type of interpolation to perform for image resampling. Use one of the following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation
`IPPI_INTER_SUPER`	supersampling interpolation, can not be applied for image enlarging

Description

The function `ippiResizeYUV422` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor, yFactor*. The result is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI. Supersampling interpolation can be applied if both *xFactor* and *yFactor* are less than or equal to 1.

The source image is a two-channel image in 4:2:2 sampling format in color spaces with decoupled luminance and chrominance components, for example, YUV422 or YCrCb422.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.

ippStsSizeErr	Indicates an error condition if *srcSize* or *dstRoiSize* has a field with zero or negative value; or if *dstRoiSize.width* is less than 2; or if width of the source image or the source ROI is odd; or if horizontal offset of the source ROI is odd.
ippStsNullPtrErr	Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
ippStsResizeNoOperationErr	Indicates an error condition if one of the destination image dimensions is less than 1 pixel.
ippStsResizeFactorErr	Indicates an error condition if *xFactor* or *yFactor* is less than or equal to zero.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.

ResizeFilterGetSize

Calculates the size of the state structure for resizing filter.

Syntax

```
IppStatus ippiResizeFilterGetSize_8u_C1R(IppiSize srcRoiSize, IppiSize
    dstRoiSize, IppiResizeFilterType filter, Ipp32u* pSize);
```

Parameters

srcRoiSize	Size of the source image ROI in pixels.
dstRoiSize	Size of the destination image ROI in pixels.
filter	Type of filter used in resizing; possible values: ippResizeFilterHann ippResizeFilterLanczos
pSize	Pointer to the size (in bytes) of the state structure.

Description

The function ippiResizeFilterGetSize is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function calculates the size *pSize* of the state structure required for the function ippiResizeFilter. The type of filter is specified by the parameter *filter*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if the *pSize* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value.
ippStsNotSupportedModeErr	Indicates an error condition if *filter* has an invalid value.

ResizeFilterInit

Initializes the state structure for the resize filter.

Syntax

```
IppStatus ippiResizeFilterInit_8u_C1R(IppiResizeFilterState* pState,
    IppiSize srcRoiSize, IppiSize dstRoiSize, IppiResizeFilterType
    filter);
```

Parameters

pState	Pointer to the state structure for resize filter.
srcRoiSize	Size of the source image ROI in pixels.
dstRoiSize	Size of the destination image ROI in pixels.
filter	Type of filter used in resizing; possible values: ippResizeFilterHann ippResizeFilterLanczos

Description

The function ippiResizeFilterInit is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function initializes the state structure *pState* for resizing filter used by the function ippiResizeFilter. The size of the structure must be computed by the function ippiResizeFilterGetSize beforehand. The type of filter is specified by the parameter *filter* and it must be the same as in the function ippiResizeFilterGetSize.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if the *pState* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value.
ippStsNotSupportedModeErr	Indicates an error condition if *filter* has an invalid value.

ResizeFilter

Changes the size of an image using a generic filter.

Syntax

```
IppStatus ippiResizeFilter_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize,
    IppiResizeFilterState* pState);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoiSize	Size of the source image ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoiSize	Size of the destination image ROI in pixels.
pState	Pointer to the state structure for the resize filter.

Description

The function ippiResizeFilter is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function resizes the source image *pSrc* using the special generic filters. The state structure *pState* contains the parameters of filtering and must be initialized by the function ippiResizeFilterInit beforehand.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsSizeErr	Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid state structure is passed.

Mirror

Mirrors an image about a horizontal or vertical axis, or both.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiMirror_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
```

Supported values for *mod*:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

Case 2: In-place operation

```
IppStatus ippiMirror_<mod>(const Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR

Parameters

pSrc	Pointer to the source buffer.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination buffer.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSrcDst	Pointer to the source and destination buffer for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
flip	Specifies the axis to mirror the image about. Use the following values to specify the axes: `ippAxsHorizontal` for the horizontal axis `ippAxsVertical` for the vertical axis `ippAxsBoth` for both horizontal and vertical axes.

Description

The function `ippiMirror` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function mirrors the source image *pSrc* about a horizontal or vertical axis or both, depending on the *flip* value, and writes it to the destination image *pDst*.

Example 12-4 shows how to use the function `ippiMirror_8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsMirrorFlipErr`	Indicates an error condition if *flip* has an illegal value.

Example 12-4 Using the function `ippiMirror`

```
Ipp8u src[8*4] = {1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8};
Ipp8u dst[4*4];
IppiSize srcRoi = { 4, 4 };

ippiMirror_8u_C1R(src, 8, dst, 4, srcRoi, ippAxsBoth);

Result:

1 2 3 4 8 8 8 8
1 2 3 4 8 8 8 8    src
1 2 3 4 8 8 8 8
1 2 3 4 8 8 8 8

4 3 2 1
4 3 2 1
4 3 2 1   dst
4 3 2 1
```

Remap

Performs the look-up coordinate mapping of pixels of the source image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, const Ipp32f* pxMap, int xMapStep,
    const Ipp32f* pyMap, int yMapStep, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, const Ipp32f* pxMap,
    int xMapStep, const Ipp32f* pyMap, int yMapStep, Ipp<datatype>* const
    pDst[3], int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R     32f_P3R
```

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, const Ipp32f* pxMap,
    int xMapStep, const Ipp32f* pyMap, int yMapStep, Ipp<datatype>* const
    pDst[4], int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R     32f_P4R
```

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pxMap, pyMap	Pointers to the starts of 2D buffers, containing tables of the *x*- and *y*-coordinates. If the referenced coordinates correspond to a pixel outside of the source ROI, no mapping of the source pixel is done.
xMapStep, yMapStep	Steps in bytes through the buffers containing tables of the *x*- and *y*-coordinates.
pDst	Pointer to the destination image ROI. An array of separate pointers to ROI in each plane for planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoiSize	Size of the destination ROI in pixels.
interpolation	Type of the interpolation to perform for image resampling. The following values are possible:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation.

Description

The function `ippiRemap` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function transforms the source image by remapping its pixels. Pixel remapping is performed using *pxMap* and *pyMap* buffers to look-up the coordinates of the source image pixel that is written to the target destination image pixel. The application has to supply these look-up tables.
The remapping of the source pixels to the destination pixels is made according to the following formula:

$$dst_pixel[i,j] = src_pixel[pxMap[i,j], pyMap[i,j]]$$

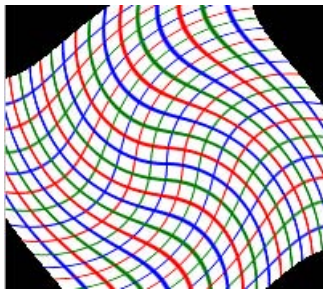
where *i,j* are the *x*- and *y*-coordinates of the target destination image pixel *dst_pixel*;

pxMap[*i,j*] contains the *x*- coordinates of the source image pixels *src_pixel* that are written to *dst_pixel*

pyMap[*i,j*] contains the *y*- coordinates of the source image pixels *src_pixel* that are written to *dst_pixel*

Figure 12-3 gives an example of applying the function `ippiRemap` to a sample image that is a square grid of alternating blue, red, and green lines

Figure 12-3 Remapping the Sample Image



The transformed part of the image is resampled using the interpolation method specified by the *interpolation* parameter, and is written to the destination image ROI.

Example 12-5 shows how to use the function `ippiRemap_8u_C1R`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *srcSize* or *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the *srcStep*, *dstStep*, *xMapStep*, or *yMapStep* has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.

Example 12-5 Using the function ippiRemap

```
IppiSize size = { 4, 4 };
IppiRect srcRect = { 0, 0, 4, 4 };
Ipp8u src[4*4] = {0, 1, 2, 3,
      4, 5, 6, 7,
      8, 9, 0, 1,
      2, 3, 4, 5};
Ipp8u dst[4*4];
Ipp32f pxMap[16] = {0, 1, 2, 3,
  0, 1, 2, 0,
  0, 1, 0, 0,
  0, 1, 2, 3};

Ipp32f pyMap[16] = {0, 0, 0, 0,
  1, 1, 1, 0,
  2, 2, 0, 0,
  3, 3, 3, 3};   // some order of pixels coordinates to write

ippiRemap_8u_C1R ( src, size, 4, srcRect, pxMap, 16, pyMap, 16, dst, 4,
                     size, IPPI_INTER_NN);

Result:
0 1 2 3
4 5 6 7
8 9 0 1     src
2 3 4 5

0 1 2 3
4 5 6 0
8 9 0 0    dst
2 3 4 5
```

Rotate

Rotates an image around the origin (0,0) and then shifts it.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep,
    IppiRect dstRoi, double angle, double xShift, double yShift, int
    interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, double angle, double xShift,
    double yShift, int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, double angle, double xShift,
    double yShift, int interpolation);
```

Supported values for *mod*:

8u_P4R	16u_P4R	32f_P4R

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.

srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
angle	The angle of rotation in degrees. The source image is rotated counterclockwise around the origin (0,0).
xShift, yShift	The shifts along horizontal and vertical axes to perform after the rotation.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following he following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation
`IPPI_SMOOTH_EDGE`	use edge smoothing option in addition to one of the above methods.

Description

The function `ippiRotate` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

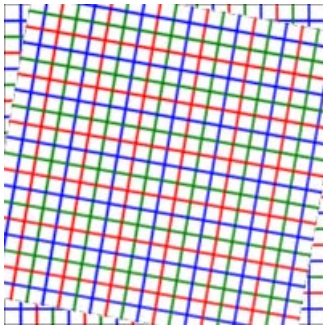
This function rotates the ROI of the source image by *angle* degrees (counterclockwise for positive *angle* values) around the origin (0,0) that is implied to be in the top left corner, and shifts it by *xShift* and *yShift* values along the *x*- and *y*- axes, respectively. The result is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI.

If you need to rotate an image about an arbitrary center (*xCenter, yCenter*), use the function `ippiGetRotateShift` to compute the appropriate *xShift, yShift* values, and then call `ippiRotate` with these values as input parameters.

Alternatively, you can use the `ippiRotateCenter` function instead.

Figure 12-4 shows the result of rotating the sample image by 10 degrees.

Figure 12-4 Rotation of an Image



The figures below illustrate how the SMOOTH_EDGE option affects the resulting destination image appearance. Both images are computed as a result of rotating the same source image by 9 degrees using the same interpolation method, with the SMOOTH_EDGE option turned off and on, respectively. The use of this option has an antialiasing effect seen on the right border of image b).

Figure 12-5 The Effect of Edge Smoothing in the Destination Image

a) SMOOTH_EDGE option turned off

b) SMOOTH_EDGE option turned on

Example 12-6 shows how to use the function ippiRotate_8u_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any image dimension has zero or negative value.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.
ippStsRectErr	Indicates an error condition if width or height of the intersection of the *srcRoi* and source image is less than or equal to 1.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

GetRotateShift

Computes shift values for rotation of an image around the specified center.

Syntax

```
IppStatus ippiGetRotateShift(double xCenter, double yCenter, double
    angle, double* xShift, double* yShift)
```

Parameters

xCenter, yCenter	Coordinates of the required center of rotation.
angle	The angle in degrees to rotate the image clockwise around the point with coordinates (*xCenter, yCenter*).
xShift, yShift	Pointers to computed shift values along horizontal and vertical axes. These shift values should be passed to ippiRotate function to bring about the desired rotation around (*xCenter, yCenter*).

Description

The function `ippiGetRotateShift` is declared in the `ippi.h` file. Use this function if you need to rotate an image about an arbitrary center (*xCenter, yCenter*) rather than the origin (0,0). The function helps compute shift values *xShift, yShift* that should be passed to `ippiRotate` function for the rotation around (*xCenter, yCenter*) to take place.

Example 12-6 shows how to use the function `ippiGetRotateShift`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *xShift* or *yShift* pointer is `NULL`.

AddRotateShift

Computes shift values for rotating an image around the specified center with specified shifts.

Syntax

```
IppStatus ippiAddRotateShift(double xCenter, double yCenter,
    double angle, double* xShift, double* yShift)
```

Parameters

xCenter, yCenter	Coordinates of the required center of rotation.
angle	The angle in degrees to rotate the image clockwise around the point with coordinates (*xCenter, yCenter*).
xShift, yShift	Pointers to modified shift values along horizontal and vertical axes. These new shift values should be passed to `ippiRotate` function to bring about the desired rotation around (*xCenter, yCenter*) and shifting.

Description

The function `ippiAddRotateShift` is declared in the `ippi.h` file. Use this function if you need to rotate an image about an arbitrary center (*xCenter, yCenter*) rather than the origin (0,0) with required shifts. The function helps compute shift values *xShift, yShift* that should be passed to the `ippiRotate` function to perform the rotation around

(*xCenter, yCenter*) and desired shifting. The shift values should be initialized. For example, to rotate an image around a point (*xCenter*, *yCenter*) by the *angle* with shift values (30.3, 26.2) the following code must be written:

```
xShift = 30.3
yShift = 26.2
ippiAddRotateShift(xCenter,yCenter,angle,&xShift,&yShift);
ippiRotate(angle,xShift,yShift);
```

Example 12-6 shows how to use the function `ippiAddRotateShift`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if *xShift* or *yShift* pointer is `NULL`.

GetRotateQuad

Computes vertex coordinates of the quadrangle, to which the source ROI is mapped by the `ippiRotate` *function.*

Syntax

```
IppStatus ippiGetRotateQuad(IppiRect srcRoi, double quad[4][2],
    double angle, double xShift, double yShift);
```

Parameters

srcRoi	Source image ROI.
quad	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by `ippiRotate` function.
angle	The angle of rotation in degrees.
xShift, yShift	The shifts along horizontal and vertical axes to perform after the rotation.

Description

The function `ippiGetRotateQuad` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for ippiRotate. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the ippiRotate function that rotates an image by *angle* degrees and shifts it by *xShift, yShift*.

The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

quad[0] corresponds to the transformed top-left corner of the source ROI,
quad[1] corresponds to the transformed top-right corner of the source ROI,
quad[2] corresponds to the transformed bottom-right corner of the source ROI,
quad[3] corresponds to the transformed bottom-left corner of the source ROI.

Example 12-6 shows how to use the function ippiGetRotateQuad.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsSizeErr	Indicates an error condition if *srcRoi* has a size field with zero or negative value.

GetRotateBound

Computes the bounding rectangle for the source ROI transformed by the ippiRotate *function.*

Syntax

```
IppStatus ippiGetRotateBound(IppiRect srcRoi, double bound[2][2],
    double angle, double xShift, double yShift);
```

Parameters

srcRoi	Source image ROI.
bound	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
angle	The angle of rotation in degrees.
xShift, yShift	The shifts along horizontal and vertical axes to perform after the rotation.

Description

The function ippiGetRotateBound is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for ippiRotate. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the ippiRotate function that rotates an image by *angle* degrees and shifts it by *xShift, yShift.*
bound[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

Example 12-6 shows how to use the function ippiGetRotateBound.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsSizeErr	Indicates an error condition if *srcRoi* has a size field with zero or negative value.

Example 12-6 Using Intel IPP Functions for Image Rotation

```
Ipp8u src[4*4] = {4, 4, 4, 4,
                  3, 3, 3, 3,
                  2, 2, 2, 2,
                  1, 1, 1, 1};
Ipp8u dst[6*6];
IppiSize srcSize = { 4, 4 };
IppiSize dstSize = { 6, 6 };
IppiRect srcRect = { 0, 0, 4, 4 };
double xShift;
double yShift;
double xCenterSrc = 2.0;
double yCenterSrc = 2.0;
double xCenterDst = 3.5;
double yCenterDst = 2.5;
double quad[4][2];
double bound[2][2];
double angle = 35.0;

ippiGetRotateShift ( xCenterSrc, yCenterSrc, angle, &xShift, &yShift );
// xShift = -0.7 , yShift = 1.5

ippiAddRotateShift ( 1.0, 1.0, angle, &xShift, &yShift);
// xShift = -1.1, yShift = 2.2

xShift += xCenterDst - xCenterSrc;
yShift += yCenterDst - yCenterSrc;
// xShift = 0.3, yShift = 2.7
```

Example 12-6 Using Intel IPP Functions for Image Rotation

```
    ippiGetRotateQuad ( srcRect, quad, angle, xShift, yShift);
    // quad[0] = { 0.3, 2.7 }
    // quad[1] = { 2.7, 1.0 }
    // quad[2] = { 4.5, 3.5 }
    // quad[3] = { 2.0, 5.2 }

    ippiGetRotateBound ( srcRect, bound, angle, xShift, yShift);
    // bound[0] = { 0.3, 1.0 }
    // bound[1] = { 4.5, 5.2 }

    IppiRect dstRect = { (int)quad[1][0], (int)quad[1][1], (int)quad[3][0],
(int)quad[3][1] };

    ippiSet_8u_C1R ( 0, dst, 6, dstSize );
    ippiRotate_8u_C1R ( src, srcSize, 4, srcRect, dst, 6, dstRect, angle,
xShift, yShift, IPPI_INTER_NN);

Result:
4 4 4 4
3 3 3 3
2 2 2 2        src
1 1 1 1

0 0 0 0 0 0
0 0 0 0 0 0
0 0 4 3 0 0
0 0 3 2 0 0    dst
0 0 2 1 0 0
0 0 1 0 0 0
```

RotateCenter

Rotates an image about an arbitrary center.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRotateCenter_<mod>(const Ipp<datatype>* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int
    dstStep, IppiRect dstRoi, double angle, double xCenter, double
    yCenter, int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiRotateCenter_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, double angle, double xCenter,
    double yCenter, int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiRotateCenter_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, double angle, double xCenter,
    double yCenter, int interpolation);
```

Supported values for *mod*:

8u_P4R	16u_P4R	32f_P4R

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
angle	The angle of rotation in degrees.
xCenter, yCenter	Center of rotation coordinates.

interpolation The type of interpolation to perform for resampling the image. Use one of the following he following values:

IPPI_INTER_NN	nearest neighbor interpolation
IPPI_INTER_LINEAR	linear interpolation
IPPI_INTER_CUBIC	cubic interpolation
IPPI_SMOOTH_EDGE	use edge smoothing option in addition to one of the above methods.

Description

The function ippiRotateCenter is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function rotates the ROI of the source image by *angle* degrees (counterclockwise for positive *angle* values) around the point with coordinates *xCenter, yCenter*. The origin of the source image is implied to be in the top left corner. The result is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI.

Example 12-7 shows how to use the function ippiRotateCenter_8u_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any image dimension has zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.
ippStsRectErr	Indicates an error condition if width or height of the intersection of the *srcRoi* and source image is less than or equal to 1.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

Example 12-7 Using the function `ippiRotateCenter`

```
IppiSize srcSize = { 4, 4 }, dstSize = { 6, 6 };
IppiRect srcRect = { 0, 0, 4, 4};
IppiRect dstRect = { 0, 0, 6, 6};
Ipp8u src[4*4] = {4, 4, 4, 4,
      3, 3, 3, 3,
      2, 2, 2, 2,
      1, 1, 1, 1};
Ipp8u dst[6*6];
double angle = -45.0;
double xCenter = 1.0;
double yCenter = 4.0;

ippiSet_8u_C1R ( 0, dst, 6, dstSize);

ippiRotateCenter_8u_C1R (src, srcSize, 4, srcRect, dst, 6, dstRect, angle,
                         xCenter, yCenter, IPPI_INTER_NN);

Result:

4 4 4 4
3 3 3 3
2 2 2 2    src
1 1 1 1

0 0 0 0 0 0
0 0 0 4 0 0
0 0 2 3 4 0
0 0 1 2 3 0    dst
0 0 0 1 0 0
0 0 0 0 0 0
```

Shear

Performs shearing transform of the source image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep,
    IppiRect dstRoi double xShear, double yShear, double xShift,
    double yShift, int interpolation);
```

Supported values for *mod*:

8u_P4R	16u_P4R	32f_C1R
8u_P4R	16u_P4R	32f_C3R
8u_P4R	16u_P4R	32f_C4R
8u_P4R	16u_P4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, double xShear, double yShear,
    double xShift, double yShift, int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, double xShear, double yShear,
    double xShift, double yShift, int interpolation);
```

Supported values for *mod*:

8u_P4R	16u_P4R	32f_P4R

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
xShear, yShear	Coefficients of the shearing transform.
xShift, yShift	Additional shift values along the horizontal and vertical axes.

interpolation	The type of interpolation to perform for resampling the image. Use one of the following he following values:
	`IPPI_INTER_NN` — nearest neighbor interpolation
	`IPPI_INTER_LINEAR` — linear interpolation
	`IPPI_INTER_CUBIC` — cubic interpolation
	`IPPI_SMOOTH_EDGE` — use edge smoothing option in addition to one of the above methods.

Description

The function `ippiShear` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

The shearing transform, performed by this function, maps the source image pixel coordinates (*x,y*) according to the following formulas:

$$x' = x + xShear * y + xShift$$

$$y' = yShear * x + y + yShift$$

where *x*' and *y*' denote the pixel coordinates in the sheared image. The shearing transform is a special case of an affine transform, performed by the `ippiWarpAffine` function.

The transformed part of the image is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI.

Figure 12-6 gives an example of applying the shearing transform function `ippiShear` to a sample image.

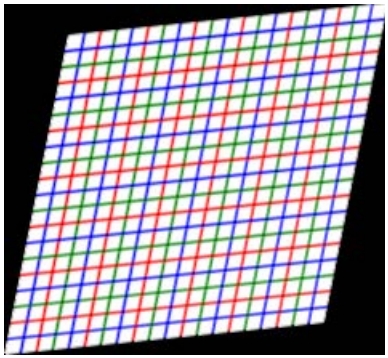
Example 12-8 shows how to use the function `ippiShear_8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any image dimension has zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsInterpolationErr`	Indicates an error condition if *interpolation* has an illegal value.

ippStsCoeffErr	Indicates an error condition if *xShear*yShear = 1*.
ippStsRectErr	Indicates an error condition if width or height of the intersection of the *srcRoi* and source image is less than or equal to 1.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

Figure 12-6 Shearing a Sample Image



Example 12-8 Using the function ippiShear

```
IppiSize srcSize = { 4, 4 };
IppiSize dstSize = { 7, 4 };
IppiRect srcRect = { 0, 0, 4, 4};
IppiRect dstRect = { 0, 0, 7, 4};
Ipp8u    src[4*4] = { 4, 4, 4, 4,
          3, 3, 3, 3,
          2, 2, 2, 2,
          1, 1, 1, 1 };
Ipp8u  dst[7*4];
double xShear = -1.0;
double yShear = 0.0;
double xShift = 3.0;
double yShift = 0.0;

ippiSet_8u_C1R (0, dst, 7, dstSize);
```

Example 12-8 Using the function `ippiShear`

```
ippiShear_8u_C1R (src, srcSize, 4, srcRect, dst, 7, dstRect,  xShear,
                    yShear,  xShift,  yShift,  IPPI_INTER_NN );

result:
4 4 4 4
3 3 3 3   src
2 2 2 2
1 1 1 1

0 0 0 4 4 4 4
0 0 3 3 3 3 0
0 2 2 2 2 0 0   dst
1 1 1 1 0 0 0
```

GetShearQuad

Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the shearing transform.

Syntax

```
IppStatus ippiGetShearQuad(IppiRect srcRoi, double quad[4][2],
   double xShear, double yShear, double xShift, double yShift);
```

Parameters

srcRoi	Region of interest in the source image (of the `IppiRect` type).
quad	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the shearing transform function.
xShear, yShear	Shearing transform coefficients.
xShift, yShift	Additional shift values along the horizontal and vertical axes.

Description

The function `ippiGetShearQuad` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for ippiShear. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the shearing transform function ippiShear using coefficients *xShear, yShear* and shift values *xShift, yShift*.

The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

quad[0] corresponds to the transformed top-left corner of the source ROI,
quad[1] corresponds to the transformed top-right corner of the source ROI,
quad[2] corresponds to the transformed bottom-right corner of the source ROI,
quad[3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsCoeffErr	Indicates an error condition if *xShear*yShear = 1*.
ippStsSizeErr	Indicates an error condition if *srcRoi* has a size field with zero or negative value.

GetShearBound

Computes the bounding rectangle for the source ROI transformed by the ippiShear function.

Syntax

```
IppStatus ippiGetShearBound(IppiRect srcRoi, double bound[2][2],
    double xShear, double yShear, double xShift, double yShift);
```

Parameters

srcRoi	Region of interest in the source image (of the IppiRect type).
bound	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
xShear, yShear	Shearing transform coefficients.
xShift, yShift	Additional shift values along the horizontal and vertical axes.

Description

The function `ippiGetShearBound` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for `ippiShear`. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the shearing transform function `ippiShear` using coefficients *xShear, yShear* and shift values *xShift, yShift.*

bound[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsCoeffErr	Indicates an error condition if *xShear*yShear = 1*.
ippStsSizeErr	Indicates an error condition if *srcRoi* has a size field with zero or negative value.

WarpAffine

Performs general affine transform of the source image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, const double coeffs[2][3],
    int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
    const double coeffs[2][3], int interpolation);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R     32f_P3R
```

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
    const double coeffs[2][3], int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R     32f_P4R
```

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
coeffs	The affine transform coefficients.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following he following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation
`IPPI_SMOOTH_EDGE`	use edge smoothing option in addition to one of the above methods.

Description

The function `ippiWarpAffine` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This affine warp function transforms the source image pixel coordinates (*x,y*) according to the following formulas:

$$x' = c_{00}*x + c_{01}*y + c_{02}$$
$$y' = c_{10}*x + c_{11}*y + c_{12}$$

where x' and y' denote the pixel coordinates in the transformed image, and c_{ij} are the affine transform coefficients passed in the array *coeffs*.

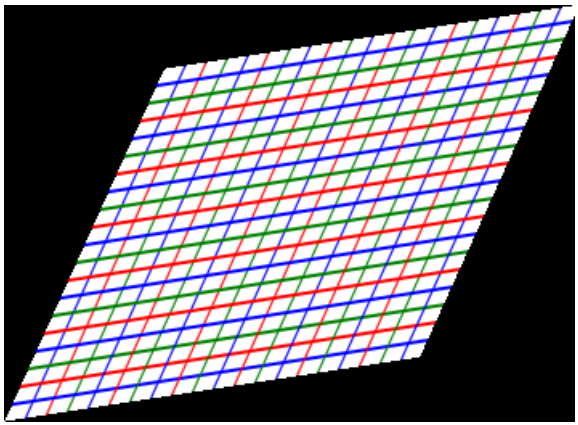
The affine warping is a general linear transform which incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The transformed part of the image is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI.

Figure 12-7 gives an example of applying the affine warping function `ippiWarpAffine` to a sample image.

To compute the affine transform parameters, use the `GetAffineQuad`, `GetAffineBound`, and `GetAffineTransform` functions described later in this chapter.

Figure 12-7 Affine Transform of an Image



Example 12-8 shows how to use the function `ippiWarpAffine_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any image dimension has zero or negative value.
`ippStsInterpolationErr`	Indicates an error condition if *interpolation* has an illegal value.
`ippStsCoeffErr`	Indicates an error condition if $c_{00}*c_{11} - c_{01}*c_{10} = 0$.
`ippStsRectErr`	Indicates an error condition if width or height of the intersection of the *srcRoi* and source image is less than or equal to 1.
`ippStsWrongIntersectROIErr`	Indicates an error condition if *srcRoi* has no intersection with the source image.
`ippStsWrongIntersectQuad`	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

WarpAffineBack

Performs an inverse affine transform of an image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiWarpAffineBack_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, const double coeffs[2][3],
    int interpolation);
```

Supported values for *mod*:

`8u_C1R`	`16u_C1R`	`32f_C1R`
`8u_C3R`	`16u_C3R`	`32f_C3R`
`8u_C4R`	`16u_C4R`	`32f_C4R`
`8u_AC4R`	`16u_AC4R`	`32f_AC4R`

Case 2: Operation on planar-order data

```
IppStatus ippiWarpAffineBack_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[2][3],
    int interpolation);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R     32f_P3R
```

```
IppStatus ippiWarpAffineBack_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[2][3],
    int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R     32f_P4R
```

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
coeffs	The affine transform coefficients.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following he following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation

Description

The function `ippiWarpAffineBack` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function performs the inverse transform to that defined by `ippiWarpAffine` function. Pixel coordinates x' and y' in the transformed image are obtained from the following equations:

$$c_{00}*x' + c_{01}*y' + c_{02} = x$$
$$c_{10}*x' + c_{11}*y' + c_{12} = y$$

where x and y denote the pixel coordinates in the source image, and coefficients c_{ij} are given in the array *coeffs*. Thus, you do not need to invert transform coefficients in your application program before calling `ippiWarpAffineBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. Their implementation include the following logic:

- backward transform is applied to coordinates of each pixel in the destination ROI, which gives as a result coordinates of some pixel in the source image
- if the obtained source pixel is inside source ROI, the corresponding pixel in destination ROI is modified accordingly; otherwise no change is made.

The above algorithm can be represented in pseudocode as follows:

```
for (j = dstRoi.y; j < dstRoi.y + dstRoi.height; j ++) {
    for (i = dstRoi.x; i < dstRoi.x + dstRoi.width; i ++) {
        sx = TransformX(i, j);
        sy = TransformY(i, j);
        if (sx >= srcRoi.x && sx < srcRoi.x + srcRoi.width && sy >=
srcRoi.y && sy < srcRoi.y + srcRoi.height) {
            dst[i, j] = Interpolate(sx, sy);}
    }
}
```

Example 12-8 shows how to use the function `ippiWarpAffineBack_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if any image dimension has zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.
ippStsCoeffErr	Indicates an error condition if $c_{00}*c_{11} - c_{01}*c_{10} = 0$.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

WarpAffineQuad

Performs image affine warping that transforms the given source quadrangle to the specified destination quadrangle.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiWarpAffineQuad_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, const double
    srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi,
    const double dstQuad[4][2], int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiWarpAffineQuad_<mod>(const Ipp<datatype>*
    const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcRoi,
    const double srcQuad[4][2], Ipp<datatype>* const pDst[3],
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],
    int interpolation);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R     32f_P3R
```

```
IppStatus ippiWarpAffineQuad_<mod>(const Ipp<datatype>*
    const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcRoi,
    const double srcQuad[4][2], Ipp<datatype>* const pDst[4],
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],
    int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R     32f_P4R
```

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
srcQuad	A given quadrangle in the source image.
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
dstQuad	A given quadrangle in the destination image.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following he following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation

IPPI_SMOOTH_EDGE	use edge smoothing option in addition to one of the above methods.

Description

The function ippiWarpAffineQuad is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function applies the affine transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only inside the intersection of the source image ROI *srcRoi* and the source quadrangle. The function ippiWarpAffineQuad uses the same formulas for pixel mapping as in the case of the ippiWarpAffine function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*. The function computes the coordinates of the 4th vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the ones specified in *dstQuad*, the function returns the warning message and continues operation with the computed values.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] holds x and y coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any image dimension has zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.

ippStsQuadErr	Indicates an error condition if *srcQuad* or *dstQuad* degenerates into triangle, line, or point, or if *dstQuad* has wrong vertex coordinates.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the *srcRoi* has no intersection with the *srcQuad*, or *dstRoi* has no intersection with the *dstQuad*.
ippStsAffineQuadChanged	Indicates a warning that coordinates of the 4th vertex of the destination quadrangle *dstQuad* are corrected by the function.

GetAffineQuad

Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.

Syntax

```
IppStatus ippiGetAffineQuad(IppiRect srcRoi, double quad[4][2],
    const double coeffs[2][3]);
```

Parameters

srcRoi	Region of interest in the source image (of the IppiRect type).
quad	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function.
coeffs	The given affine transform coefficients.

Description

The function ippiGetAffineQuad is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for ippiWarpAffine. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the affine transform function ippiWarpAffine using the given coefficients *coeffs*.

The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

quad[0] corresponds to the transformed top-left corner of the source ROI,
quad[1] corresponds to the transformed top-right corner of the source ROI,
quad[2] corresponds to the transformed bottom-right corner of the source ROI,
quad[3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsCoeffErr	Indicates an error condition if $c_{00}*c_{11} - c_{01}*c_{10} = 0$.
ippStsSizeErr	Indicates an error condition if *srcRoi* has a size field with zero or negative value.

GetAffineBound

Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.

Syntax

```
IppStatus ippiGetAffineBound(IppiRect srcRoi, double bound[2][2],
    const double coeffs[2][3]);
```

Parameters

srcRoi	Region of interest in the source image (of the IppiRect type).
bound	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
coeffs	The given affine transform coefficients.

Description

The function ippiGetAffineBound is declared in the ippi.h file. This function is used as a support function for ippiWarpAffine. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the affine transform function ippiWarpAffine using coefficients *coeffs*.

bound[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsCoeffErr	Indicates an error condition if $c_{00}*c_{11} - c_{01}*c_{10} = 0$.
ippStsSizeErr	Indicates an error condition if *srcRoi* has a size field with zero or negative value.

GetAffineTransform

Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

Syntax

```
IppStatus ippiGetAffineTransform(IppiRect srcRoi, const double quad[4][2],
    double coeffs[2][3]);
```

Parameters

srcRoi	Region of interest in the source image (of the IppiRect type).
quad	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function.
coeffs	Output array. Contains the target affine transform coefficients.

Description

The function ippiGetAffineTransform is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for ippiWarpAffine. It computes the coefficients *coeffs* of the affine transform that must be used by the function ippiWarpAffine to map the source rectangular ROI to the quadrangle with the specified vertex coordinates *quad*.
The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

quad[0] corresponds to the transformed top-left corner of the source ROI,
quad[1] corresponds to the transformed top-right corner of the source ROI,
quad[2] corresponds to the transformed bottom-right corner of the source ROI,
quad[3] corresponds to the transformed bottom-left corner of the source ROI.

The function computes the coordinates of the 4th vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the ones specified in *quad*, the function returns the warning message and continues operation with the computed values.

Example 12-9 shows how to use the function `ippiGetAffineTransform`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or warning.
`ippStsRectErr`	Indicates an error condition if width or height of the *srcRoi* is less than or equal to 1.
`ippStsCoeffErr`	Indicates an error condition if $c_{00}*c_{11} - c_{01}*c_{10} = 0$.
`ippStsSizeErr`	Indicates an error condition if *srcRoi* has a size field with zero or negative value.
`ippStsAffineQuadChanged`	Indicates a warning that coordinates of the 4th vertex of the specified quadrangle *quad* are not correct.

Example 12-9 Using Intel IPP Functions for General Affine Transform

```
void func_warp_affine()
{
    int step;
    Ipp32f pSrc[8*8] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                        1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                        1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                        1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
    int srcStep = 8*sizeof(Ipp32f);

    IppiSize srcSize = { 8, 8};
    IppiRect srcRoi = {0, 0, 8, 8};

    Ipp32f pDst[8*8];
    Ipp32f pDstB[8*8];
    int dstStep = 8*sizeof(ipp32f);
    int dstbStep = 8*sizeof(ipp32f);
```

Example 12-9 Using Intel IPP Functions for General Affine Transform

```
    IppiRect dstRoi = {0, 0, 8, 8};
    double quad[4][2] = {{0.0, 0.0},
                         {7.0, 1.0},
                         {8.0, 8.0},
                         {1.0, 7.0}};
    double coeffs[2][3]; // affine transform coefficients

    ippiGetAffineTransform(srcRoi, quad, coeffs); //computes the affine
transform coefficients

    ippiSet_32f_C1R(2,pDst,srcStep,srcSize);
    ippiWarpAffine_32f_C1R(pSrc, srcSize, srcStep, srcRoi, pDst, dstStep,
                           dstRoi, coeffs, IPPI_INTER_NN);

    ippiSet_32f_C1R(3,pDstB,srcStep,srcSize);
    ippiWarpAffineBack_32f_C1R( pDst, srcSize, dstStep, dstRoi, pDstB, dstbStep,
                                dstRoi, coeffs, IPPI_INTER_NN);
}

Result:
            pSrc
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0

pDst
1.0  2.0  2.0  2.0  2.0  2.0  2.0  2.0
2.0  1.0  1.0  5.0  1.0  1.0  1.0  1.0
2.0  1.0  5.0  5.0  5.0  1.0  1.0  1.0
2.0  5.0  5.0  5.0  5.0  5.0  1.0  1.0
2.0  1.0  5.0  5.0  5.0  5.0  5.0  1.0
2.0  1.0  1.0  5.0  5.0  5.0  5.0  5.0
2.0  1.0  1.0  1.0  5.0  5.0  5.0  1.0
2.0  1.0  1.0  1.0  1.0  5.0  1.0  1.0


pDstB
1.0  2.0  2.0  2.0  1.0  1.0  1.0  1.0
2.0  1.0  1.0  5.0  5.0  1.0  1.0  3.0
2.0  1.0  5.0  5.0  5.0  5.0  1.0  3.0
2.0  5.0  5.0  5.0  5.0  5.0  5.0  3.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  3.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  3.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  3.0
1.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0
```

WarpPerspective

Performs perspective warping of the source image using the given transform coefficients.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u_P4R	16u_P4R	32f_P4R

Parameters

pSrc	Pointer to the source image origin.An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.

srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
coeffs	The perspective transform coefficients.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following he following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation
`IPPI_SMOOTH_EDGE`	use edge smoothing option in addition to one of the above methods.

Description

The function `ippiWarpPerspective` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This perspective warp function transforms the source image pixel coordinates (*x*,*y*) according to the following formulas:

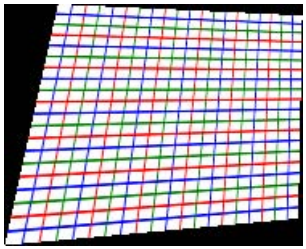
$$x' = (c_{00}*x + c_{01}*y + c_{02})/(c_{20}*x + c_{21}*y + c_{22})$$
$$y' = (c_{10}*x + c_{11}*y + c_{12})/(c_{20}*x + c_{21}*y + c_{22})$$

where *x*' and *y*' denote the pixel coordinates in the transformed image, and c_{ij} are the perspective transform coefficients passed in the array *coeffs*.

The transformed part of the image is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI.

Figure 12-8 gives an example of applying the perspective transform function `ippiWarpPerspective` to a sample image.

Figure 12-8 Perspective Transform of an Image



To estimate how the source image ROI will be transformed by the `ippiWarpPerspective` function, use functions `ippiWarpPerspectiveQuad` and `ippiGetPerspectiveBound` described in the sections that follow. To calculate coefficients of the perspective transform which maps source ROI to a given quadrangle, use `ippiGetPerspectiveTransform` function.

Example 12-10 shows how to use the function `ippiWarpPerspective_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any image dimension has zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsRectErr`	Indicates an error condition if width or height of the intersection of the *srcRoi* and source image is less than or equal to 1.
`ippStsInterpolationErr`	Indicates an error condition if *interpolation* has an illegal value.
`ippStsCoeffErr`	Indicates an error condition if coefficient values are invalid.

ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

WarpPerspectiveBack

Performs an inverse perspective warping of the source image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u_P4R	16u_P4R	32f_P4R

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image (of the `IppiRect` type).
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image.
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
coeffs	The perspective transform coefficients.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following he following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation

Description

The function `ippiWarpPerspectiveBack` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function performs the inverse transform to that defined by `ippiWarpPerspective` function. Pixel coordinates *x*' and *y*' in the transformed image are obtained from the following equations

$$(c_{00}*x' + c_{01}*y' + c_{02})/(c_{20}*x' + c_{21}*y' + c_{22}) = x$$
$$(c_{10}*x' + c_{11}*y' + c_{12})/(c_{20}*x' + c_{21}*y' + c_{22}) = y$$

where *x* and *y* denote the pixel coordinates in the source image, and coefficients c_{ij} are given in the array *coeffs*. Thus, you don't need to invert transform coefficients in your application program before calling `ippiWarpPerspectiveBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. See implementation details in the description of `ippiWarpAffineBack` function.

Example 12-10 shows how to use the function `ippiWarpPerspectiveBack_32f_C1R`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any image dimension has zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

WarpPerspectiveQuad

Performs perspective warping of the given source quadrangle to the specified destination quadrangle.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, const double
    srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi,
    const double dstQuad[4][2], int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>*
    const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcRoi,
    const double srcQuad[4][2], Ipp<datatype>* const pDst[3],
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],
    int interpolation);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R     32f_P3R
```

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>*
    const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcRoi,
    const double srcQuad[4][2], Ipp<datatype>* const pDst[4],
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],
    int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R     32f_P4R
```

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
srcQuad	A given quadrangle in the source image.
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
dstQuad	A given quadrangle in the destination image.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation

Description

The function `ippiWarpPerspectiveQuad` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function applies a perspective transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only in the intersection of the source image ROI *srcRoi* and the source quadrangle. The function `ippiWarpPerspectiveQuad` uses the same formulas for pixel mapping as in the case of the `ippiWarpPerspective` function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] holds `x` and `y` coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any image dimension has zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsInterpolationErr`	Indicates an error condition if *interpolation* has an illegal value.
`ippStsQuadErr`	Indicates an error condition if *srcQuad* or *dstQuad* degenerates into triangle.
`ippStsWrongIntersectROIErr`	Indicates an error condition if *srcRoi* has no intersection with the source image.
`ippStsWrongIntersectQuad`	Indicates a warning that no operation is performed if the *srcRoi* has no intersection with the *srcQuad*, or *dstRoi* has no intersection with the *dstQuad*.

GetPerspectiveQuad

Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

Syntax

```
IppStatus ippiGetPerspectiveQuad(IppiRect srcRoi, double quad[4][2],
    const double coeffs[3][3]);
```

Parameters

srcRoi	Region of interest in the source image (of the `IppiRect` type).
quad	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.
coeffs	The given perspective transform coefficients.

Description

The function `ippiGetPerspectiveQuad` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for `ippiWarpPerspective`. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the perspective transform function `ippiWarpPerspective` using the given coefficients *coeffs*.

The first dimension [4] of the array *quad*`[4][2]` is equal to the number of vertices, and the second dimension `[2]` means `x` and `y` coordinates of the vertex. Quadrangle vertices have the following meaning:

quad`[0]` corresponds to the transformed top-left corner of the source ROI,
quad`[1]` corresponds to the transformed top-right corner of the source ROI,
quad`[2]` corresponds to the transformed bottom-right corner of the source ROI,
quad`[3]` corresponds to the transformed bottom-left corner of the source ROI.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.

ippStsSizeErr	Indicates an error condition if *srcRoi* has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.

GetPerspectiveBound

Computes the bounding rectangle for the source ROI transformed by the ippiWarpPerspective *function.*

Syntax

```
IppStatus ippiGetPerspectiveBound(IppiRect srcRoi, double bound[2][2],
    const double coeffs[3][3]);
```

Parameters

srcRoi	Region of interest in the source image (of the IppiRect type).
bound	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
coeffs	The given perspective transform coefficients.

Description

The function ippiGetPerspectiveBound is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for ippiWarpPerspective. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the perspective transform function ippiWarpPerspective using the given coefficients *coeffs*.

bound[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsSizeErr	Indicates an error condition if *srcRoi* has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.

GetPerspectiveTransform

Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

Syntax

```
IppStatus ippiGetPerspectiveTransform(IppiRect srcRoi,
    const double quad[4][2], double coeffs[3][3]);
```

Parameters

srcRoi	Region of interest in the source image (of the `IppiRect` type).
quad	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.
coeffs	Output array. Contains the target perspective transform coefficients.

Description

The function `ippiGetPerspectiveTransform` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for `ippiWarpPerspective`. It computes the coefficients *coeffs* that should be used by the function `ippiWarpPerspective` to map the source rectangular ROI to the quadrangle with the given vertex coordinates *quad*.

The first dimension [4] of the array *quad*`[4][2]` is equal to the number of vertices, and the second dimension `[2]` means `x` and `y` coordinates of the vertex. Quadrangle vertices have the following meaning:

quad`[0]` corresponds to the transformed top-left corner of the source ROI,
quad`[1]` corresponds to the transformed top-right corner of the source ROI,
quad`[2]` corresponds to the transformed bottom-right corner of the source ROI,
quad`[3]` corresponds to the transformed bottom-left corner of the source ROI.

Example 12-10 shows how to use the function `ippiGetPerspectiveTransform`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsSizeErr`	Indicates an error condition if *srcRoi* has a size field with zero or negative value.

ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.
ippStsRectErr	Indicates an error condition if width or height of the *srcRoi* is less than or equal to 1.

Example 12-10 Using Intel IPP Functions for Perspective Warping

```
void func_WarpPerspective()
{
    int step;
    Ipp32f pSrc[8*8] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                        1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                        1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                        1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};

    int srcStep = 8*sizeof(Ipp32f);
    IppiSize srcSize = { 8, 8};
    IppiRect srcRoi = {0, 0, 8, 8};

    Ipp32f pDst[8*8];
    Ipp32f pDstB[8*8];
    int dstStep = 8*sizeof(ipp32f);
    IppiRect dstRoi = {0, 0, 8, 8};
    IppiSize dstSize = {8, 8};
    int dstbStep = 8*sizeof(ipp32f);
    IppiRect dstbRoi = {0, 0, 8, 8};
    double quad[4][2] = {{0.0, 0.0},
                         {8.0, 0.0},
                         {6.0, 8.0},
                         {2.0, 8.0}};
    double coeffs[3][3];

    ippiSet_32f_C1R(2,pDst,srcStep,srcSize);
    ippiSet_32f_C1R(3,pDstB,srcStep,srcSize);

    ippiGetPerspectiveTransform(srcRoi, quad, coeffs);
    ippiWarpPerspective_32f_C1R( pSrc, srcSize, srcStep, srcRoi, pDst, dstStep,
dstRoi, coeffs, IPPI_INTER_NN);
    ippiWarpPerspectiveBack_32f_C1R( pDst, dstSize, dstStep, dstRoi, pDstB,
dstbStep, dstbRoi, coeffs, IPPI_INTER_NN);
}
Result:
   pSrc
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
```

Example 12-10 Using Intel IPP Functions for Perspective Warping

```
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0

pDst
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
2.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
2.0  1.0  1.0  1.0  5.0  5.0  1.0  1.0
2.0  1.0  1.0  5.0  5.0  5.0  1.0  1.0
2.0  1.0  1.0  5.0  5.0  5.0  1.0  1.0
2.0  2.0  5.0  5.0  5.0  5.0  5.0  2.0
2.0  2.0  5.0  5.0  5.0  5.0  5.0  2.0
2.0  2.0  1.0  5.0  5.0  5.0  1.0  2.0

pDstB
1.0  1.0  1.0  1.0  1.0  1.0  1.0  3.0
2.0  1.0  1.0  5.0  5.0  1.0  1.0  3.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  3.0
2.0  5.0  5.0  5.0  5.0  5.0  5.0  2.0
2.0  5.0  5.0  5.0  5.0  5.0  5.0  2.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0
3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0
```

WarpBilinear

Performs bilinear warping of the source image using the specified transform coefficients.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* pSrc, IppiSize
    srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int
    dstStep, IppiRect dstRoi, const double coeffs[2][4], int
    interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[2][4],
    int interpolation);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R     32f_P3R
```

```
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[2][4],
    int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R     32f_P4R
```

Parameters

pSrc	Pointer to the source image origin.An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
coeffs	The bilinear transform coefficients.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation
`IPPI_SMOOTH_EDGE`	use edge smoothing option in addition to one of the above methods.

Description

The function `ippiWarpBilinear` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This bilinear warp function transforms the source image pixel coordinates (*x,y*) according to the following formulas:

$$x' = c_{00}*x*y + c_{01}*x + c_{02}*y + c_{03}$$
$$y' = c_{10}*x*y + c_{11}*x + c_{12}*y + c_{13}$$

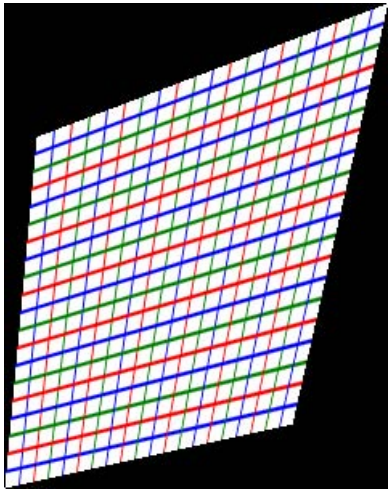
where *x*' and *y*' denote the pixel coordinates in the transformed image, and c_{ij} are the bilinear transform coefficients passed in the array *coeffs*.

The bilinear transform preserves equal distances between points on a line.

The transformed part of the source image is resampled using the interpolation method specified by the *interpolation* parameter, and written to the destination image ROI.

Figure 12-9 gives an example of applying the bilinear warping function `ippiWarpBilinear` to a sample image.

Figure 12-9 Bilinear Transform of an Image



To estimate how the source image ROI will be transformed by the `ippiWarpBilinear` function, use functions `ippiWarpBilinearQuad` and `ippiGetBilinearBound` described in the sections that follow. To calculate coefficients of the bilinear transform which maps source ROI to a given quadrangle, use `ippiGetBilinearTransform` function.

Example 12-11 shows how to use the function `ippiWarpBilinear_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any image dimension has zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
`ippStsInterpolationErr`	Indicates an error condition if *interpolation* has an illegal value.
`ippStsRectErr`	Indicates an error condition if width or height of the intersection of the *srcRoi* and source image is less than or equal to 1.
`ippStsCoeffErr`	Indicates an error condition if coefficient values are invalid.
`ippStsWrongIntersectROIErr`	Indicates an error condition if *srcRoi* has no intersection with the source image.
`ippStsWrongIntersectQuad`	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

WarpBilinearBack

Performs an inverse bilinear warping of the source image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, const double coeffs[2][4], int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[2][4],
    int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[2][4],
    int interpolation);
```

Supported values for *mod*:

8u_P4R	16u_P4R	32f_P4R

Parameters

pSrc	Pointer to the source image origin.An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
coeffs	The bilinear transform coefficients.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following values:

IPPI_INTER_NN	nearest neighbor interpolation
IPPI_INTER_LINEAR	linear interpolation
IPPI_INTER_CUBIC	cubic interpolation

Description

The function ippiWarpBilinearBack is declared in the ippi.h file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function performs the inverse transform to that defined by ippiWarpBilinear function. Pixel coordinates x' and y' in the transformed image are obtained from the following equations

$$c_{00}*x'*y' + c_{01}*x' + c_{02}*y' + c_{03} = x$$
$$c_{10}*x'*y' + c_{11}*x' + c_{12}*y' + c_{13} = y$$

where x and y denote the pixel coordinates in the source image, and coefficients c_{ij} are given in the array *coeffs*. Thus, you don't need to invert transform coefficients in your application program before calling ippiWarpBilinearBack.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. See implementation details in the description of ippiWarpAffineBack function.

Example 12-11 shows how to use the function ippiWarpBilinearBack_32f_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any image dimension has zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.

`ippStsWrongIntersectQuad`	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

WarpBilinearQuad

Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    const double srcQuad[4][2], Ipp<datatype>* pDst, int dstStep,
    IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

Case 2: Operation on planar-order data

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    const double srcQuad[4][2], Ipp<datatype>* const pDst[3],
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],
    int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    const double srcQuad[4][2], Ipp<datatype>* const pDst[4],
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],
    int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R      32f_P4R
```

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the `IppiRect` type).
srcQuad	A given quadrangle in the source image.
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the `IppiRect` type).
dstQuad	A given quadrangle in the destination image.
interpolation	The type of interpolation to perform for resampling the image. Use one of the following values:

`IPPI_INTER_NN`	nearest neighbor interpolation
`IPPI_INTER_LINEAR`	linear interpolation
`IPPI_INTER_CUBIC`	cubic interpolation
`IPPI_SMOOTH_EDGE`	use edge smoothing option in addition to one of the above methods.

Description

The function `ippiWarpBilinearQuad` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function applies a bilinear transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only in the intersection of the source image ROI *srcRoi* and the source quadrangle *srcQuad*. The function `ippiWarpBilinearQuad` uses the same formulas for pixel mapping as in the case of the `ippiWarpBilinear` function.

Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] holds x and y coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any image dimension has zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if *interpolation* has an illegal value.
ippStsQuadErr	Indicates an error condition if *srcQuad* or *dstQuad* degenerates into triangle.
ippStsWrongIntersectROIErr	Indicates an error condition if *srcRoi* has no intersection with the source image.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the *srcRoi* has no intersection with the *srcQuad*, or *dstRoi* has no intersection with the *dstQuad*.

GetBilinearQuad

Computes the vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform.

Syntax

```
IppStatus ippiGetBilinearQuad(IppiRect srcRoi, double quad[4][2],
    const double coeffs[2][4]);
```

Parameters

srcRoi	Region of interest in the source image (of the `IppiRect` type).
quad	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function.
coeffs	The given bilinear transform coefficients.

Description

The function `ippiGetBilinearQuad` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for `ippiWarpBilinear`. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform function `ippiWarpBilinear` using coefficients *coeffs*.

The first dimension [4] of the array *quad*`[4][2]` is equal to the number of vertices, and the second dimension [2] means `x` and `y` coordinates of the vertex. Quadrangle vertices have the following meaning:

quad`[0]` corresponds to the transformed top-left corner of the source ROI,
quad`[1]` corresponds to the transformed top-right corner of the source ROI,
quad`[2]` corresponds to the transformed bottom-right corner of the source ROI,
quad`[3]` corresponds to the transformed bottom-left corner of the source ROI.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsSizeErr`	Indicates an error condition if *srcRoi* has a size field with zero or negative value.

ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.

GetBilinearBound

Computes the bounding rectangle for the source ROI transformed by the `ippiWarpBilinear` *function.*

Syntax

```
IppStatus ippiGetBilinearBound(IppiRect srcRoi, double bound[2][2],
    const double coeffs[2][4]);
```

Parameters

srcRoi	Region of interest in the source image (of the `IppiRect` type).
bound	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
coeffs	The given bilinear transform coefficients.

Description

The function `ippiGetBilinearBound` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for `ippiWarpBilinear`. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI is mapped by the bilinear transform function `ippiWarpBilinear` using coefficients *coeffs*.

bound`[0]` specifies `x`, `y` coordinates of the top-left corner, *bound*`[1]` specifies `x`, `y` coordinates of the bottom-right corner.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsSizeErr	Indicates an error condition if *srcRoi* has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.

GetBilinearTransform

Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

Syntax

```
IppStatus ippiGetBilinearTransform(IppiRect srcRoi, const double quad[4][2],
    double coeffs[2][4]);
```

Parameters

srcRoi	Region of interest in the source image (of the `IppiRect` type).
quad	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function.
coeffs	Output array. Contains the target bilinear transform coefficients.

Description

The function `ippiGetBilinearTransform` is declared in the `ippi.h` file. It operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for `ippiWarpBilinear`. It computes the coefficients *coeffs* of the bilinear transform that maps the source rectangular ROI to the quadrangle with the specified vertex coordinates *quad*.

The first dimension [4] of the array *quad*`[4][2]` is equal to the number of vertices, and the second dimension `[2]` means `x` and `y` coordinates of the vertex. Quadrangle vertices have the following meaning:

quad`[0]` corresponds to the transformed top-left corner of the source ROI,
quad`[1]` corresponds to the transformed top-right corner of the source ROI,
quad`[2]` corresponds to the transformed bottom-right corner of the source ROI,
quad`[3]` corresponds to the transformed bottom-left corner of the source ROI.

Example 12-11 shows how to use the function `ippiGetBilinearTransform`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsSizeErr`	Indicates an error condition if *srcRoi* has a size field with zero or negative value.

ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.
ippStsRectErr	Indicates an error condition if width or height of the *srcRoi* is less than or equal to 1.

Example 12-11 Using Intel IPP Functions for Bilinear Transform

```
void func_WarpBilinear()
{
    int step;
    Ipp32f pSrc[8*8] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                        1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                        1.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 1.0,
                        1.0, 1.0, 5.0, 5.0, 5.0, 5.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 5.0, 5.0, 1.0, 1.0, 1.0,
                        1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};

    int srcStep = 8*sizeof(Ipp32f);
    IppiSize srcSize = { 8, 8};
    IppiRect srcRoi = {0, 0, 8, 8};

    Ipp32f pDst[8*8];
    Ipp32f pDstB[8*8];
    int dstStep = 8*sizeof(ipp32f);
    IppiRect dstRoi = {0, 0, 8, 8};
    IppiSize dstSize = {8, 8};
    int dstbStep = 8*sizeof(ipp32f);
    IppiRect dstbRoi = {0, 0, 8, 8};

    double quad[4][2] = {{1.0, 1.0},
                         {8.0, 0.0},
                        {6.0, 8.0},
                        {2.0, 6.0}};
    double coeffs[2][4];

    ippiSet_32f_C1R(2,pDst,srcStep,srcSize);
    ippiSet_32f_C1R(3,pDstB,srcStep,srcSize);

    ippiGetBilinearTransform(srcRoi, quad, coeffs);
    ippiWarpBilinear_32f_C1R( pSrc, srcSize, srcStep, srcRoi, pDst, dstStep,
dstRoi, coeffs, IPPI_INTER_NN);
    ippiWarpBilinearBack_32f_C1R( pDst, dstSize, dstStep, dstRoi, pDstB,
dstbStep, dstbRoi, coeffs, IPPI_INTER_NN);
}

Result:
pSrc
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
```

Example 12-11 Using Intel IPP Functions for Bilinear Transform

```
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0

pDst
2.0  2.0  2.0  2.0  2.0  2.0  2.0  2.0
2.0  1.0  1.0  1.0  1.0  5.0  1.0  1.0
2.0  2.0  1.0  5.0  5.0  5.0  5.0  1.0
2.0  2.0  5.0  5.0  5.0  5.0  5.0  1.0
2.0  2.0  5.0  5.0  5.0  5.0  5.0  1.0
2.0  2.0  1.0  5.0  5.0  5.0  5.0  2.0
2.0  2.0  1.0  1.0  5.0  1.0  1.0  2.0
2.0  2.0  2.0  2.0  1.0  1.0  1.0  2.0

pDstB
1.0  1.0  1.0  1.0  2.0  2.0  2.0  3.0
2.0  1.0  5.0  1.0  5.0  1.0  1.0  3.0
2.0  1.0  5.0  5.0  5.0  5.0  1.0  3.0
2.0  5.0  5.0  5.0  5.0  5.0  5.0  3.0
5.0  5.0  5.0  5.0  5.0  5.0  5.0  2.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  2.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  2.0  1.0  3.0  3.0  3.0  3.0
```

Wavelet Transforms

13

This chapter describes the Intel® IPP image processing functions that perform two-dimensional discrete wavelet transform (DWT).

In many applications the multiresolution analysis by discrete wavelet transforms is a better alternative to windowing and discrete Fourier analysis techniques. On the one hand, the forward two-dimensional wavelet transform may be considered as a decomposition of an image on the base of functions bounded or localized in space; and on the other, the wavelet transforms are related to subband filtering and resampling.

Intel IPP for image processing contains one-level discrete wavelet decomposition and reconstruction functions. It also provides the necessary interface for initialization and deallocation of the transform context structure. Table 13-1 lists all functions of this group:

Table 13-1 Image Wavelet Transform Functions

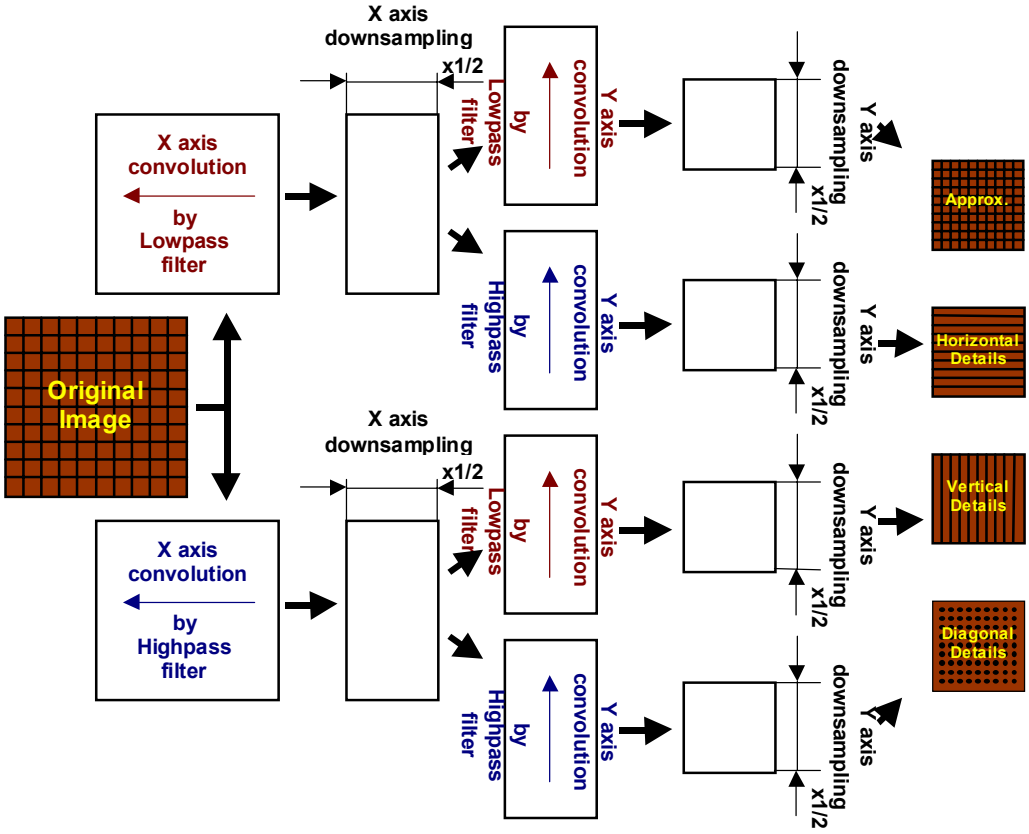
Function Base Name	Operation
WTFwdInitAlloc	Allocates memory and initializes the forward wavelet transform context structure
WTFwdFree	Deallocates memory allocated for a forward wavelet transform context structure
WTFwdGetBufSize	Determines the size of external work buffer for a forward wavelet transform
WTFwd	Performs one-level wavelet decomposition of an image
WTInvInitAlloc	Allocates memory and initializes the inverse wavelet transform context structure
WTInvFree	Deallocates memory allocated for an inverse wavelet transform context structure
WTInvGetBufSize	Determines the size of external work buffer for an inverse wavelet transform
WTInv	Performs one-level wavelet reconstruction of an image

The wavelet transform type can be set by specifying the appropriate filter taps in the initialization function.

Note that Intel IPP supports only one-dimensional finite impulse response filters for separable convolution.

The Intel IPP functions for wavelet decomposition and reconstruction use fast polyphase algorithm, which is equivalent to traditional application of separable convolution and dyadic resampling in different order. The Figure 13-1 shows the equivalent algorithm of wavelet-based image decomposition:

Figure 13-1 Equivalent Scheme of Wavelet Decomposition Algorithm



Decomposition operation applied to a source image produces four output images of equal size: approximation image, horizontal detail image, vertical detail image, and diagonal detail image.

These decomposition components have the following meaning:

- The 'approximation' image is obtained by vertical and horizontal lowpass filtering.
- The 'horizontal detail' image is obtained by vertical highpass and horizontal lowpass filtering.
- The 'vertical detail' image is obtained by vertical lowpass and horizontal highpass filtering.
- The 'diagonal detail' image is obtained by vertical and horizontal highpass filtering.

The above image names are used in this manual for identification convenience only.

The wavelet-based image reconstruction can be represented by a sequence of separate convolution and dyadic upsampling.

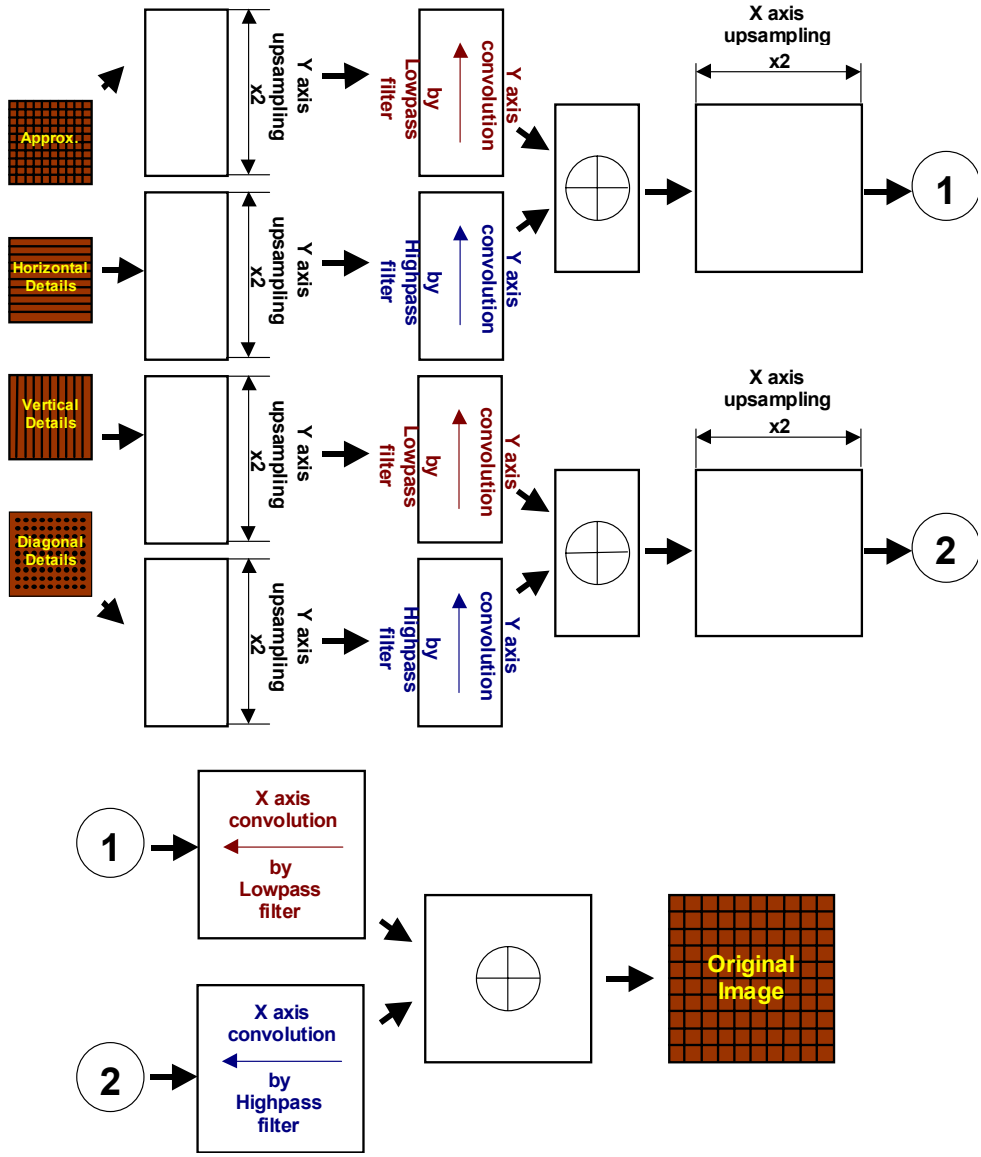
The reconstruction function uses four input images that are the same as those resulting from the decomposition operation.

Figure 13-2 shows the equivalent algorithm of wavelet reconstruction of an image.

Wavelet transform functions support regions of interest (ROI, see Regions of Interest in IPP in chapter 2) in the images. However, these functions do not perform internally any border extensions of image ROI data. It means that source images must already contain all border data that are necessary for convolution operations. See descriptions of the functions `ippiWTFwd` and `ippiWTInv` for detailed information on how to calculate extended image border sizes.

The use of Intel IPP for wavelet transform is demonstrated in the *Wavelet Transform Sample.* In addition to this sample many solutions and hints for use of Intel IPP in wavelet applications can be found in the *JPEG 2000 Encode-Decode Sample*. Both samples can be downloaded from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

Figure 13-2 Equivalent Scheme of Wavelet Reconstruction Algorithm



WTFwdInitAlloc

Allocates memory and initializes the forward wavelet transform context structure.

Syntax

```
IppStatus ippiWTFwdInitAlloc_32f_C1R(IppiWTFwdSpec_32f_C1R** ppSpec,
    const Ipp32f* pTapsLow, int lenLow, int anchorLow,
    const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
IppStatus ippiWTFwdInitAlloc_32f_C3R(IppiWTFwdSpec_32f_C3R** ppSpec,
    const Ipp32f* pTapsLow, int lenLow, int anchorLow,
    const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

Parameters

ppSpec	Double pointer to the forward DWT context structure.
pTapsLow	Pointer to lowpass filter taps.
lenLow	Length of the lowpass filter.
anchorLow	Anchor position of the lowpass filter.
pTapsHigh	Pointer to highpass filter taps.
lenHigh	Length of the highpass filter.
anchorHigh	Anchor position of the highpass filter.

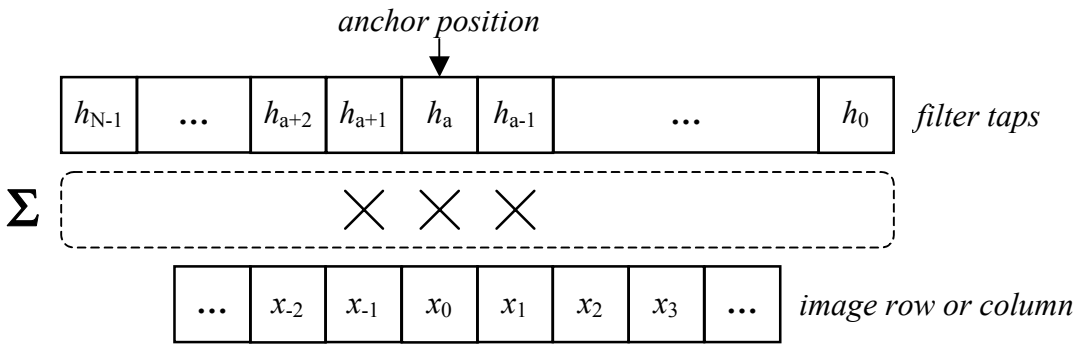
Description

The function `ippiWTFwdInitAlloc` is declared in the `ippi.h` file. This function allocates memory for the context structure *pSpec* of a one-level wavelet decomposition and initializes this structure.

The forward wavelet transform context structure contains parameters of a wavelet filter bank used for image decomposition. The filter bank consists of two analysis filters and includes the lowpass decomposition filter (or *coarse* filter) and the highpass decomposition filter (or *detail* filter).

The parameters *pTapsLow* and *pTapsHigh* specify coefficients, and *anchorLow* and *anchorHigh* - anchor positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the Figure 13-3:

Figure 13-3 Anchor Value and Initial Filter Position for Wavelet Decomposition



Here *a* stands for anchor value, N is filter length, x_0 is the starting pixel of the processed row or column, and x_{-1}, x_{-2} ,... are the additional border pixels that are needed for calculations. The anchor value and filter length completely determine right, left, top, and bottom border sizes for the source image used in decomposition. The corresponding C-language expressions to calculate border sizes are given in the description of `ippiWTFwd` function.

Once allocated and initialized the context structure can be used in several threads of calculations as well as in a number of wavelet decomposition levels.

Example 13-1 shows how to use the function `ippiWTFwdInitAlloc_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pTapsLow, pTapsHigh,* or *pSpec* pointer is NULL.
`ippStsSizeErr`	Indicates an error condition if filter length *lenLow* or *lenHigh* is less then 2.
`ippStsAnchorErr`	Indicates an error condition if anchor position *anchorLow* or *anchorHigh* is less then zero.

ippStsMemAllocErr	Indicates an error condition in case of memory allocation failure for the context structure.

WTFwdFree

Frees memory allocated for a forward wavelet transform context structure.

Syntax

```
IppStatus ippiWTFwdFree_32f_C1R (IppiWTFwdSpec_32f_C1R* pSpec);
IppStatus ippiWTFwdFree_32f_C3R (IppiWTFwdSpec_32f_C3R* pSpec);
```

Parameters

pSpec	Pointer to an allocated and initialized forward DWT context structure.

Description

The function `ippiWTFwdFree` is declared in the `ippi.h` file. This function frees memory previously allocated by the function `ippiWTFwdInitAlloc` for a forward wavelet transform context structure *pSpec*.

Example 13-1 shows how to use the function `ippiWTFwdFree_32f_C1R`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSpec* pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid context structure is passed.

WTFwdGetBufSize

Calculates the size of external work buffer for a forward wavelet transform.

Syntax

```
IppStatus ippiWTFwdGetBufSize_C1R(const IppiWTFwdSpec_32f_C1R* pSpec,
    int* pSize);
IppStatus ippiWTFwdGetBufSize_C3R(const IppiWTFwdSpec_32f_C3R* pSpec,
    int* pSize);
```

Parameters

pSpec	Pointer to an allocated and initialized forward DWT context structure.
pSize	Pointer to a variable that will receive the size of work buffer required for forward wavelet transform.

Description

The function `ippiWTFwdGetBufSize` is declared in the `ippi.h` file. This function computes the size in bytes of the work buffer required for the forward wavelet transform function `ippiWTFwd`.

Example 13-1 shows how to use the function `ippiWTInvInitAlloc_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid context structure is passed.

WTFwd

Performs one-level wavelet decomposition of an image.

Syntax

```
IppStatus ippiWTFwd_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp32f*
    pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep,
    Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int
    detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C1R*
    pSpec, Ipp8u* pBuffer);
IppStatus ippiWTFwd_32f_C3R (const Ipp32f* pSrc, int srcStep, Ipp32f*
    pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep,
    Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int
    detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C3R*
    pSpec, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pApproxDst	Pointer to ROI of the destination approximation image.
approxStep	Distance in bytes between starts of consecutive lines in the approximation image buffer.
pDetailXDst	Pointer to ROI of the destination horizontal detail image.
detailXStep	Distance in bytes between starts of consecutive lines in the horizontal detail image buffer.
pDetailYDst	Pointer to ROI of the destination vertical detail image.
detailYStep	Distance in bytes between starts of consecutive lines in the vertical detail image buffer.
pDetailXYDst	Pointer to ROI of the destination diagonal detail image.
detailXYStep	Distance in bytes between starts of consecutive lines in the diagonal detail image buffer.
dstRoiSize	Size of the ROI in pixels for all destination images.
pSpec	Pointer to the allocated and initialized forward DWT context structure.

pBuffer	Pointer to the allocated buffer for intermediate operations

Description

The function `ippiWTFwd` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function performs one-level wavelet decomposition of a source image pointed to by *pSrc* into four destination subimages pointed to by *pApproxDst*, *pDetailXDst*, *pDetailYDst*, and *pDetailXYDst*. See Figure 13-1 for the equivalent algorithm of `ippiWTFwd` function operation.

Wavelet parameters are contained in the forward transform context structure *pSpec*. It must be allocated and initialized by the function `ippiWTFwdInitAlloc` beforehand.

The decomposition function needs a buffer *pBuffer* for temporary calculations. Its size can be computed by calling the function `ippiWTFwdGetBufSize`. The buffer size does not depend on the size of processed images, and the same buffer can be used for decomposing images of different size. However, it is not recommended to use the same buffer in different threads in a multithreaded mode. For better performance, use an aligned memory location for this buffer, which can be allocated by the function `ippiMalloc`.

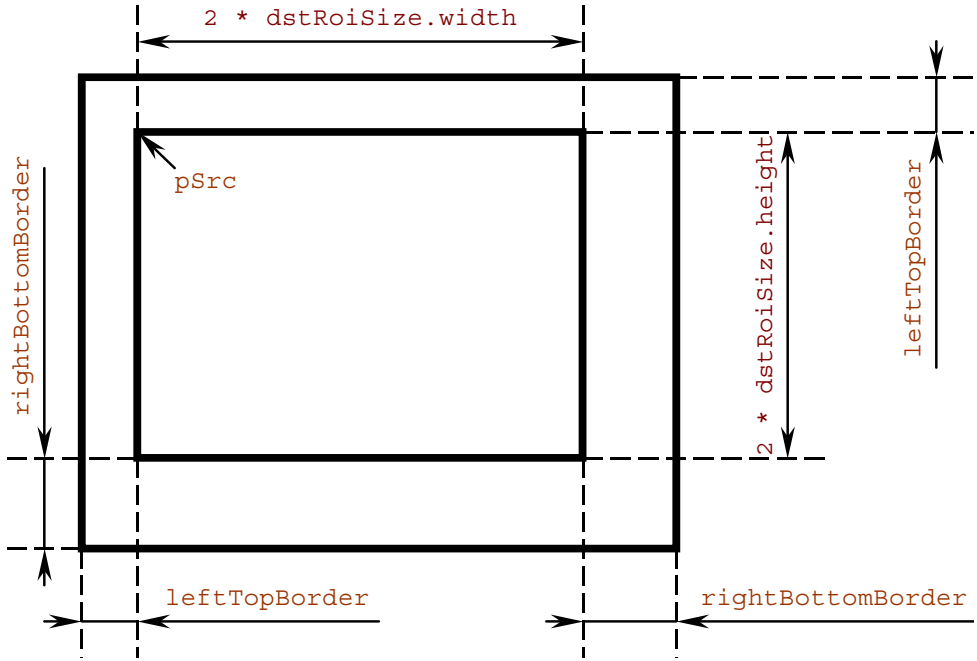
The pointer *pSrc* points to memory location of the source image rectangular ROI of size `srcWidth` by `srcHeight` which is uniquely determined by the size of destination ROI as:

```
srcWidth  = 2 * dstRoiSize.width
srcHeight = 2 * dstRoiSize.height
```

Note the source image ROI does not include border pixels necessary to compute some destination pixels. It means that prior to using `ippiWTFwd` function the application program must apply some border extension model (symmetrical, wraparound or another) to the source image ROI through filling of neighboring memory locations. As a result, the size of memory block allocated for the source image must be extended to accommodate for added border pixels outside ROI formal boundaries.

Figure 13-4 schematically shows the source image ROI and extended image area.

Figure 13-4 Extended Source Image for Wavelet Decomposition



Use the following C-language expressions to calculate extended image border sizes:

```
int leftBorderLow    = lenLow    - 1 - anchorLow;
int leftBorderHigh   = lenHigh   - 1 - anchorHigh;

int rightBorderLow   = lenLow    - 2 - leftBorderLow;
int rightBorderHigh  = lenHigh   - 2 - leftBorderHigh;

int leftTopBorder     = IPP_MAX(leftBorderLow,  leftBorderHigh);
int rightBottomBorder = IPP_MAX(rightBorderLow, rightBorderHigh);
```

See the description of the function ippiWTFwdInitAlloc for the explanation of the parameters.

Note that the left and top borders have equal size. The same holds for the right and bottom borders which have equal size too.

The size of the source image area extended by border pixels can be defined as

```
srcWidthWithBorders  = srcWidth  + leftTopBorder + rightBottomBorder;
srcHeightWithBorders = srcHeight + leftTopBorder + rightBottomBorder;
```

All destination images have equal size specified by the parameter *dstRoiSize*.

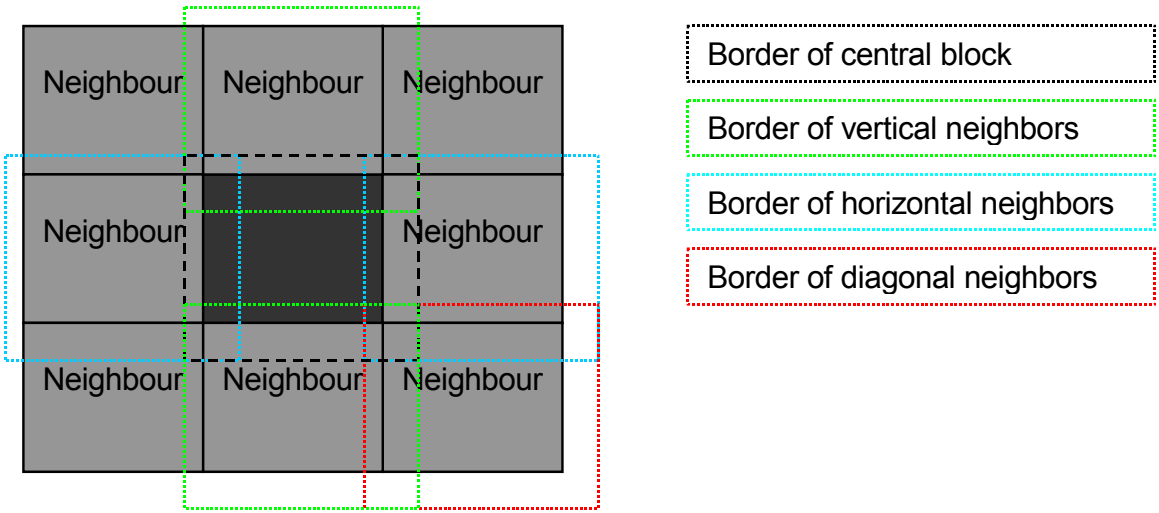
Conversely, to perform a wavelet reconstruction of the full size source image from the component images obtained by decomposition, use extended component images for the reconstruction pass. See the description of the function ippiWTInv for more details.

The ROI concept used in wavelet transform functions can be applied to processing large images by blocks, or 'tiles'. To accomplish this, the source image should be subdivided into overlapping blocks in the following way:

- Main part (ROI) of each block is adjacent to neighboring blocks and has no intersection with neighbor's ROIs;
- Extended borders of each block overlap with ROIs of neighboring blocks.

This subdivision scheme is illustrated in the figure below.

Figure 13-5 Image Division into Blocks with Overlapping Borders



Example 13-1 shows how to use the function ippiWTFwd_32f_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if step through any buffer is less than or equal to zero.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid context structure is passed.

WTInvInitAlloc

Allocates memory and initializes the inverse wavelet transform context structure.

Syntax

```
IppStatus ippiWTInvInitAlloc_32f_C1R (IppiWTInvSpec_32f_C1R**
    ppSpec,const Ipp32f* pTapsLow,  int lenLow,  int anchorLow,const
    Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
IppStatus ippiWTInvInitAlloc_32f_C3R (IppiWTInvSpec_32f_C3R**
    ppSpec,const Ipp32f* pTapsLow,  int lenLow,  int anchorLow,const
    Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

Arguments:

ppSpec	Double pointer to a new allocated and initialized inverse DWT context structure.
pTapsLow	Pointer to lowpass filter taps.
lenLow	Length of the lowpass filter.
anchorLow	Anchor position of the lowpass filter.
pTapsHigh	Pointer to highpass filter taps.
lenHigh	Length of the highpass filter.
anchorHigh	Anchor position of the highpass filter.

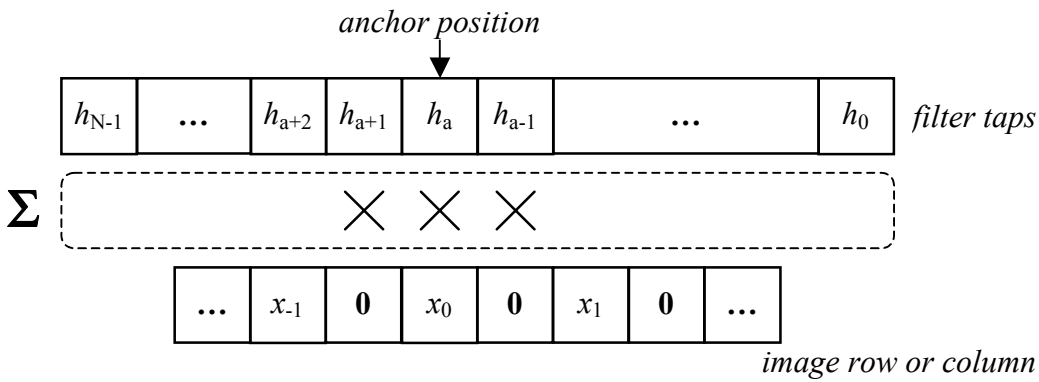
Description

The function `ippiWTInvInitAlloc` is declared in the `ippi.h` file. This function allocates memory for the context structure *pSpec* of a one-level wavelet reconstruction and initializes this structure.

The inverse wavelet transform context structure contains parameters of a wavelet filter bank used for image reconstruction. The filter bank consists of two synthesis filters and includes the lowpass reconstruction filter (or *coarse* filter) and the highpass reconstruction filter (or *detail* filter).

The parameters *pTapsLow* and *pTapsHigh* specify coefficients, and *anchorLow* and *anchorHigh* - anchor positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

Figure 13-6 Anchor Value and Initial Filter Position for Wavelet Reconstruction



Here *a* stands for anchor value, N is filter length, x_0 is the starting pixel of the processed row or column, and x_{-1}, x_{-2} ,... are the additional border pixels that are needed for calculations. As seen from this figure, anchor position is specified relative to upsampled source data. The anchor value and filter length completely determine right, left, top, and bottom border sizes for source images used in reconstruction. The corresponding C-language expressions to calculate border sizes are given in the description of the function `ippiWTInv`.

Once allocated and initialized the context structure can be used in several threads of calculations as well as in a number of wavelet reconstruction levels.

Example 13-1 shows how to use the function `ippiWTInvInitAlloc_32f_C1R`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pTapsLow*, *pTapsHigh*, or *pSpec* pointer is NULL.
ippStsSizeErr	Indicates an error condition if filter length *lenLow* or *lenHigh* is less then 2.
ippStsAnchorErr	Indicates an error condition if anchor position *anchorLow* or *anchorHigh* is less then zero.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

WTInvFree

Frees memory allocated for an inverse wavelet transform context structure.

Syntax

```
IppStatus ippiWTInvFree_32f_C1R (IppiWTInvSpec_32f_C1R* pSpec);
IppStatus ippiWTInvFree_32f_C3R (IppiWTInvSpec_32f_C3R* pSpec);
```

Parameters

pSpec	Pointer to an allocated and initialized inverse DWT context structure.

Description

The function ippiWTInvFree is declared in the ippi.h file. This function frees memory that is previously allocated by the function ippiWTInvInitAlloc for the inverse wavelet transform context structure *pSpec*.

Example 13-1 shows how to use the function ippiWTInvFree_32f_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pSpec* pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid context structure is passed.

WTInvGetBufSize

Computes the size of external work buffer for an inverse wavelet transform.

Syntax

```
IppStatus ippiWTInvGetBufSize_C1R( const IppiWTInvSpec_32f_C1R* pSpec,
    int* pSize );
IppStatus ippiWTInvGetBufSize_C3R( const IppiWTInvSpec_32f_C3R* pSpec,
    int* pSize );
```

Parameters

pSpec	Pointer to an allocated and initialized inverse DWT context structure.
pSize	Pointer to a variable that will receive the size of work buffer for inverse wavelet transform.

Description

The function `ippiWTInvGetBufSize` is declared in the `ippi.h` file. This function computes the size in bytes of the work buffer that is required for the inverse wavelet transform function `ippiWTInv` to operate.

Example 13-1 shows how to use the function `ippiWTInvGetBufSize_32f_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid context structure is passed.

WTInv

Performs one-level wavelet reconstruction of an image.

Syntax

```
IppStatus ippiWTInv_32f_C1R(const Ipp32f* pApproxSrc, int approxStep, const
    Ipp32f* pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int
    detailYStep, const Ipp32f* pDetailXYSrc, int detailXYStep, IppiSize
    srcRoiSize, Ipp32f* pDst, int dstStep, const IppiWTInvSpec_32f_C1R* pSpec,
    Ipp8u* pBuffer);
```

```
IppStatus ippiWTInv_32f_C3R(const Ipp32f* pApproxSrc, int approxStep, const
    Ipp32f* pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int
    detailYStep, const Ipp32f* pDetailXYSrc, int detailXYStep, IppiSize
    srcRoiSize, Ipp32f* pDst, int dstStep, const IppiWTInvSpec_32f_C3R* pSpec,
    Ipp8u* pBuffer);
```

Parameters

pApproxSrc	Pointer to ROI of the source approximation image.
approxStep	Distance in bytes between starts of consecutive lines in the approximation image buffer.
pDetailXSrc	Pointer to ROI of the source horizontal detail image.
detailXStep	Distance in bytes between starts of consecutive lines in the horizontal detail image buffer.
pDetailYSrc	Pointer to ROI of the source vertical detail image.
detailYStep	Distance in bytes between starts of consecutive lines in the vertical detail image buffer.
pDetailXYSrc	Pointer to ROI of the source diagonal detail image.
detailXYStep	Distance in bytes between starts of consecutive lines in the diagonal detail image buffer.
srcRoiSize	Size of ROI in pixels for all source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
pSpec	Pointer to the allocated and initialized inverse DWT context structure.

pBuffer	Pointer to the allocated buffer for intermediate operations.

Description

The function ippiWTInv is declared in the ippi.h file. It operates with ROI (see Regions of Interest in Intel IPP). This function performs wavelet reconstruction of the output image *pDst* from the four component images.
See Figure 13-1 for the equivalent algorithm of ippiWTInv function operation. Wavelet parameters are contained in the inverse transform context structure *pSpec*. It must be allocated and initialized by the function ippiWTInvInitAlloc beforehand.

The reconstruction function needs a buffer *pBuffer* for temporary calculations. Its size can be computed by calling the function ippiWTInvGetBufSize. The buffer size does not depend on the size of processed images, and the same buffer can be used for reconstructing images of different size. However, it is not recommended to use the same buffer in different threads in a multithreaded mode. For better performance, use an aligned memory location for this buffer, which can be allocated by the function ippiMalloc.

The pointers *pApproxSrc, pDetailXSrc, pDetailYSrc,* and *pDetailXYSrc* point to ROIs of source images excluding borders. All source ROIs have the same size *srcRoiSize*, while the destination image size is uniquely determined from the following relations:

```
dstWidth  = 2 * srcRoiSize.width;
dstHeight = 2 * srcRoiSize.height;
```

As source ROIs do not include border pixels required to computations, the application program have to apply a border extension model (symmetrical, wraparound or another) to ROIs of all source images filling the neighboring memory locations.
Note the border sizes may be different for different source images. The following C-language expressions can be used to calculate extended image border sizes:

```
int leftBorderLow    = (lenLow  - 1 - anchorLow) / 2;
int leftBorderHigh   = (lenHigh - 1 - anchorHigh) / 2;

int rightBorderLow   = (anchorLow  + 1) / 2;
int rightBorderHigh  = (anchorHigh + 1) / 2;

int apprLeftBorder   = leftBorderLow;
int apprRightBorder  = rightBorderLow;
int apprTopBorder    = leftBorderLow;
int apprBottomBorder = rightBorderLow;

int detxLeftBorder   = leftBorderLow;
int detxRightBorder  = rightBorderLow;
int detxTopBorder    = leftBorderHigh;
int detxBottomBorder = rightBorderHigh;
```

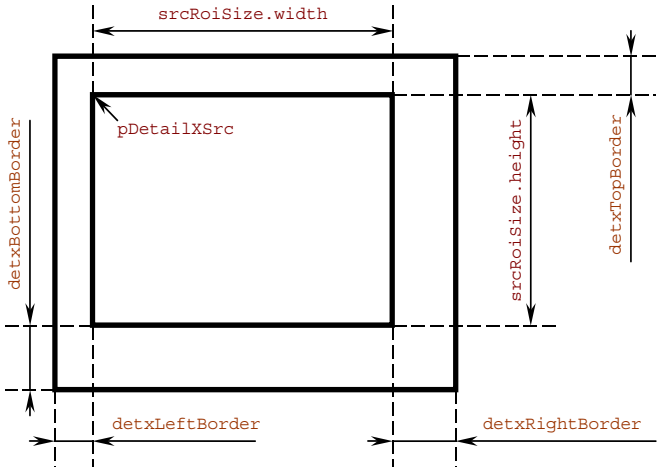
```
int detyLeftBorder   = leftBorderHigh;
int detyRightBorder  = rightBorderHigh;
int detyTopBorder    = leftBorderLow;
int detyBottomBorder = rightBorderLow;

int detxyLeftBorder   = leftBorderHigh;
int detxyRightBorder  = rightBorderHigh;
int detxyTopBorder    = leftBorderHigh;
int detxyBottomBorder = rightBorderHigh;
```

See the description of the function ippiWTInvInitAlloc for the explanation of the used parameters.

The above relations show that left and top borders always have equal size only for approximation and diagonal detail images. Right and bottom borders also have equal size only for approximation and diagonal detail images. Thus, the size of memory block allocated for each source image must be extended to accommodate for added border pixels outside ROI.

Figure 13-7 shows ROI and extended image area for the horizontal detail source image.

Figure 13-7 Extended Horizontal Detail Source Image for Wavelet Reconstruction



Sizes of source images extended by border pixels can be calculated as follows:

```
apprWidthWithBorders  = srcWidth  + apprLeftBorder + apprRightBorder;
apprHeightWithBorders = srcHeight + apprTopBorder  + apprBottomBorder;
```

```
detxWidthWithBorders  = srcWidth  + detxLeftBorder + detxRightBorder;
detxHeightWithBorders = srcHeight + detxTopBorder  + detxBottomBorder;

detyWidthWithBorders  = srcWidth  + detyLeftBorder + detyRightBorder;
detyHeightWithBorders = srcHeight + detyTopBorder  + detyBottomBorder;

detxyWidthWithBorders  = srcWidth  + detxyLeftBorder + detxyRightBorder;
detxyHeightWithBorders = srcHeight + detxyTopBorder  + detxyBottomBorder;
```

Example 13-1 shows how to use the function `ippiWTInv_32f_C1R`.

The ROI concept can be used to reconstruct large images by blocks or 'tiles'.

To accomplish this, each the source images into blocks with overlapping borders, similar to what is considered in the description of the function `ippiWTFwd`. Each component must be subdivided into the same pattern of rectangular blocks.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`srcRoiSize`* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if step through any buffer is less than or equal to zero.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid context structure is passed.

Example 13-1 Using Intel IPP Functions for Wavelet Transforms

```
void func_wavelet()
{
    IppiWTFwdSpec_32f_C1R* pSpec;
    IppiWTInvSpec_32f_C1R* pSpecInv;

    Ipp32f pTapsLow[3] = {0.25, 0.5, 0.25};
    int lenLow = 3;
    int anchorLow = 1;

    Ipp32f pTapsHigh[3] = { 0.75, -0.25, -0.125};
    int lenHigh = 3;
    int anchorHigh = 1;

    Ipp32f pSrc[8*8] = { 0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
```

Example 13-1 Using Intel IPP Functions for Wavelet Transforms

```
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                        11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
                        11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0};
    Ipp32f pSrcB[9*9];
    int srcStepB = 9*sizeof(Ipp32f);
    IppiSize roiSizeB = {9, 9};
    int srcStep = 8*sizeof(Ipp32f);
    IppiSize roiSize = {8, 8};

    Ipp32f pDetailXDst[4*4];
    Ipp32f pDetailYDst[4*4];
    Ipp32f pDetailXYDst[4*4];
    Ipp32f pApproxDst[4*4];
    IppiSize dstRoiSize = {4, 4};

    int bufSize, bufSizeInv;
    Ipp8u* pBuffer;
    Ipp8u* pBufferInv;
    Ipp32f pDstInv[8*8];
    Ipp32f pAppB[5*5];
    Ipp32f pXB[5*5];
    Ipp32f pYB[5*5];
    Ipp32f pXYB[5*5];
    int StepB =  5*sizeof(Ipp32f);
    IppiSize roiInvSize = {4, 4};
    IppiSize roiInvSizeB = {5, 5};
    int stepDstInv = 8*sizeof(Ipp32f);
    int approxStep, detailXStep, detailYStep, detailXYStep;
    approxStep = detailXStep = detailYStep = detailXYStep = 4*sizeof(Ipp32f);

    //adds border to the source image
    ippiCopyWrapBorder_32s_C1R((Ipp32s*)pSrc, srcStep, roiSize, (Ipp32s*)pSrcB,
srcStepB, roiSizeB, 1, 1);
    //performs forward  wavelet transform
    ippiWTFwdInitAlloc_32f_C1R ( &pSpec, pTapsLow, lenLow, anchorLow, pTapsHigh,
lenHigh, anchorHigh);
    ippiWTFwdGetBufSize_C1R(pSpec, &bufSize);
    pBuffer = ippsMalloc_8u(bufSize);
    ippiWTFwd_32f_C1R (pSrcB + roiSizeB.width + 1, srcStepB, pApproxDst,
approxStep, pDetailXDst, detailXStep, pDetailYDst, detailYStep, pDetailXYDst,
detailXYStep, dstRoiSize, pSpec, pBuffer);

//-------------------------------------
    ippiWTInvInitAlloc_32f_C1R (&pSpecInv, pTapsLow, lenLow, anchorLow,
pTapsHigh, lenHigh, anchorHigh);
    ippiWTInvGetBufSize_C1R(pSpecInv, &bufSizeInv);
    pBufferInv = ippsMalloc_8u(bufSizeInv);
```

Example 13-1 Using Intel IPP Functions for Wavelet Transforms

```
//adds border to four images obtained after ippiWTFwd
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pApproxDst, approxStep, dstRoiSize,
(Ipp32s*)pAppB, StepB, roiInvSizeB, 0, 0);
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailXDst, detailXStep, dstRoiSize,
(Ipp32s*)pXB, StepB, roiInvSizeB, 0, 0);
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailYDst, detailYStep, dstRoiSize,
(Ipp32s*)pYB, StepB, roiInvSizeB, 0, 0);
ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailXYDst, detailXYStep, dstRoiSize,
(Ipp32s*)pXYB, StepB, roiInvSizeB, 0, 0);
//performs inverse  wavelet transform
ippiWTInv_32f_C1R( pAppB, StepB, pXB, StepB, pYB, StepB, pXYB, StepB,
roiInvSize, pDstInv, stepDstInv, pSpecInv, pBufferInv);
ippiWTInvFree_32f_C1R (pSpecInv);
ippiWTFwdFree_32f_C1R (pSpec);
}

Result:
After WTFwd ->

  pApproxDst
0.0  2.8   8.3  0.0
2.8  4.9   9.0  2.8
8.3  9.0  10.4 8.3
0.0  2.8   8.3  0.0

   pDetailXDst
  0.0   1.0  3.1   0.0
  8.3   7.3  5.2   8.3
 -4.2  -2.1  2.1  -4.2
  0.0   1.0  3.1   0.0

   pDetailYDst
  0.0   8.3   -4.2   0.0
  1.0   7.3   -2.1   1.0
  3.1   5.2    2.1   3.1
  0.0   8.3   -4.2   0.0

   pDetailXYDst
  0.0   3.1   -1.6   0.0
  3.1   0.0    4.7   3.1
 -1.6   4.7   -4.7  -1.6
  0.0   3.1   -1.6   0.0

After WTFinv ->
        pDstInv
 0.0    2.8    -0.3   -0.6    2.1    1.1    0.0    0.0
 2.8    5.9     2.7    4.9    3.4    5.4    2.8    5.1
-0.3    2.7    -0.6   -1.2    2.2    0.7   -0.3   -0.5
-0.6    4.9    -1.2   -2.2    3.9    1.2   -0.6   -1.0
```

Example 13-1 Using Intel IPP Functions for Wavelet Transforms

```
 2.1     3.4      2.2     3.9     1.8     3.7     2.1     3.8
 1.1     5.4      0.7     1.2     3.7     3.2     1.1     1.9
 0.0     2.8     -0.3    -0.6     2.1     1.1     0.0     0.0
 0.0     5.1     -0.5    -1.0     3.8     1.9     0.0     0.0
```

Computer Vision

14

This chapter provides a background for the computer vision concepts used in the Intel® IPP library as well as detailed descriptions of the Intel IPP image processing functions for computer vision. These functions are combined in groups by their functionality. Each group of functions is described in a separate section.

Table 14-1 Intel IPP Functions for Computer Vision Applications

Function Name	Description
Feature Detection Functions	
CannyGetSize	Pre-calculates the size of the temporary buffer for the function ippiCanny.
Canny	Finds edges in the image.
EigenValsVecsGetBufferSize	Pre-calculates the size of the temporary buffer for the function ippiEigenValsVecs.
EigenValsVecs	Calculates eigen values and eigen vectors of image blocks for corner detection.
MinEigenValGetBufferSize	Pre-calculates the size of the temporary buffer for the function ippiCornerMinEigenVal.
MinEigenVal	Calculates the minimal eigen value of image blocks for corner detection.
HoughLineGetSize	Computes the size of the working buffer straight lines detection.
HoughLine	Detects straight lines in the source image.
HoughLine_Region	Detects straight lines with the specified range of parameters in the source image
Distance Transform Functions	
DistanceTransform	Performs the distance transform operation.
GetDistanceTransformMask	Calculates metrics for distance transform.

Table 14-1 Intel IPP Functions for Computer Vision Applications (continued)

Function Name	Description
TrueDistanceTransformGetBufferSize	Calculates the size of the temporary working buffer for the function ippiTrueDistanceTransform.
TrueDistanceTransform	Calculates the Eucledian distance to the closest zero pixel for all non-zero pixels of the source image.
FastMarchingGetBufferSize	Computes the size of the working buffer for the peak search.
FastMarching	Calculates distance transform to closest zero pixel for all non-zero pixels of source image using fast marching method.
Image Gradients	
GradientColorToGray	Converts a color gradient image to grayscale.
Flood Fill Functions	
FloodFillGetSize	Calculates size of temporary buffer for flood filling operation.
FloodFillGetSize_Grad	Calculates size of temporary buffer for the gradient flood filling.
FloodFill	Fills the connected area of the same pixel values on the image with a new value, considering 4 or 8 neighbor values of the pixel.
FloodFill_Grad	Fills the connected area of the pixel values within a small threshold on the image with a new value, considering 4 or 8 neighbor values of the pixel.
FloodFill_Range	Fills the connected region of the pixel values within a specified range on the image with a new value, considering 4 neighbor values of the pixel
Motion Templates Functions	
UpdateMotionHistory	Updates the motion history image using the motion silhouette at a given timestamp.
OpticalFlowPyrLKInitAlloc	Allocates memory and initializes a structure for optical flow calculation.
OpticalFlowPyrLKFree	Frees memory allocated for the optical flow structure.
OpticalFlowPyrLK	Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

Table 14-1 Intel IPP Functions for Computer Vision Applications (continued)

Function Name	Description
Pyramids Functions	
PyrDownGetBufSize	Calculates the size of a temporary buffer used by the function PyrDown.
PyrUpGetBufSize	Calculates the size of a temporary buffer used by the function PyrUp.
PyrDown	Applies the Gaussian to an image, then performs down-sampling.
PyrUp	Performs up-sampling of an image, then applies the Gaussian multiplied by 4.
Universal Pyramids Functions	
PyramidInitAlloc	Allocates memory and initializes a pyramid structure.
PyramidFree	Frees memory allocated for the pyramid structure.
PyramidLayerDownInitAlloc	Allocates memory and initializes a structure for creating a lower pyramid layer.
PyramidLayerUpInitAlloc	Allocates memory and initializes a structure for creating an upper pyramid layer.
PyramidLayerDownFree	Frees memory allocated for the lower pyramid layer structure.
PyramidLayerUpFree	Frees memory allocated for the upper pyramid layer structure.
GetPyramidDownROI	Computes the size of the lower pyramid layer.
GetPyramidUpROI	Computes the size of the upper pyramid layer.
PyramidLayerDown	Creates a lower pyramid layer.
PyramidLayerUp	Creates an upper pyramid layer.
Image Inpainting Functions	
InpaintInitAlloc	Allocates memory and initializes a structure for image inpainting.
InpaintFree	Frees memory allocated for the structure for image inpainting.
Inpaint	Restorse unknown image pixels.
Image Segmentation Functions	
LabelMarkersGetBufferSize	Computes the size of the working buffer for the marker labeling.

Table 14-1 Intel IPP Functions for Computer Vision Applications (continued)

Function Name	Description
LabelMarkers	Labels markers in image with different values.
SegmentWatershedGetBufferSize	Computes the size of the working buffer for the watershed segmentation.
SegmentWatershed	Performs watershed image segmentation using markers.
SegmentGradientGetBufferSize	Computes the size of the working buffer for the gradient segmentation.
SegmentGradient	Performs image segmentation by region growing to the least gradient direction.
BoundSegments	Marks pixels belonging to segment boundaries.
ForegroundHistogramInitAlloc	Allocates memory and initializes a state structure for foreground and background segmentation using histograms.
ForegroundHistogramFree	Frees memory allocated for the foreground and background segmentation structure.
ForegroundHistogram	Calculates foreground mask using histograms.
ForegroundHistogramUpdate	Updates histogram statistical model for foreground segmentation.
ForegroundGaussianInitAlloc	Allocates memory and initializes a state structure for foreground/background segmentation using Gaussian mixtures.
Pattern Recognition Functions	
HaarClassifierInitAlloc	Allocates memory and initializes the structure for standard Haar classifiers.
TiltedHaarClassifierInitAlloc	Allocates memory and initializes the structure for tilted Haar classifiers.
HaarClassifierFree	Frees memory allocated for the Haar classifier structure.
GetHaarClassifierSize	Returns the size of the Haar classifier.
TiltHaarFeatures	Modifies a Haar classifier by tilting specified features.
ApplyHaarClassifier	Applies a Haar classifier to an image.
ApplyMixedHaarClassifier	Applies a mixed Haar classifier to an image.
Camera Calibration and 3D Reconstruction	
UndistortGetSize	Computes the size of the external buffer.

Table 14-1 Intel IPP Functions for Computer Vision Applications (continued)

Function Name	Description
UndistortRadial	Corrects radial distortions of the single image.
CreateMapCameraUndistort	Creates look-up tables of coordinates of corrected image.
Image Enchancement Functions	
SRHNInitAlloc_PSF3x3, SRHNInitAlloc_PSF2x2	Allocates memory and initializes the internal representation of the PSF table.
SRHNFree_PSF3x3, SRHNFree_PSF3x3	Deallocates the internal representation of the PSF table.
SRHNCalcResidual_PSF3x3, SRHNCalcResidual_PSF2x2	Computes residuals for the likelihood part of the target function or its gradient.
SRHNUpdateGradient_PSF3x3, SRHNUpdateGradient_PSF2x2	Updates the likelihood part of the gradient.

Using ippiAdd for Background Differencing

This section describes functions that help build a statistical model of a background. This model can be used to subtract the background from an image.

Here, the term “background” stands for a set of motionless image pixels –that is, pixels that do not belong to any object, moving in front of the camera. The definition of background can vary if considered in other techniques of object extraction. For example, if the depth map of the scene can be obtained, e.g., with the help of stereo, background can be determined as static parts of the scene that are located far enough from the camera.

The simplest background model assumes that every background pixel brightness varies independently, according to normal distribution. To calculate the characteristics of the background, several dozens of frames, as well as their squares, can be accumulated. That is, for every pixel location we find the sum of pixel values in this location *S(x, y)*, using the function ippiAdd, and the sum of squares of the values *Sq(x, y)*, using the function ippiAddSquare. Then mean is calculated as

$m(x, y) = \frac{S(x, y)}{N}$, where *N* is the number of collected frames, and standard deviation as

$$stddev(x, y) = sqrt\left(\frac{Sq(x, y)}{N} - \left(\frac{S(x, y)}{N}\right)^2\right).$$

After that, the pixel in a certain pixel location within a certain frame is considered as belonging to a moving object, if the condition $abs(p(x, y) - m(x, y)) < C * stddev(x, y)$, where C is a constant, is met. If C is equal to 3, it satisfies the "three sigmas" rule. To obtain such background model, objects should be put away from the camera for a few seconds, so that the whole image from the camera represents the subsequent background observation.

Adapting the background differencing model to changes in lighting conditions and background scenes, for example, when the camera moves or an object passes behind the front object, can improve the described technique.

The mean brightness can be calculated through replacing the simple average with the running average found by using the function `ippiAddWeighted`. Also, several techniques can be used to identify moving scene parts and exclude them while accumulating background information. These techniques include change detection (see the functions `ippiAbsDiff` in chapter 5 and `ippiThreshold` in chapter 7), optical flow, and some other operations.

Relevant addition functions used for background differencing include:

`Add_8u32f_C1IR, Add_8s32f_C1IR, Add_32f_C1IR,`
`Add_8u32f_C1IMR, Add_8s32f_C1IMR, Add_32f_C1IMR` (see `Add`),

and also all flavors of `AddSquare`, `AddProduct`, and `AddWeighted`.

Feature Detection Functions

This section describes feature detection functions.

The set of the Sobel derivative filters is generally used to find edges, ridges, and blobs, especially in case of scale-space images, for example, pyramids.

The following naming conventions are used in the equations described below:

- D_x and D_y are the first x and y derivatives, respectively.
- D_{xx} are D_{yy} are the second x and y derivatives, respectively.
- D_{xy} is the partial x and y derivative.
- D_{xxx} and D_{yyy} are the third x and y derivatives, respectively.
- D_{xxy} and D_{xyy} are the third partial x and y derivatives.

Corner Detection

The Sobel and Sharr first derivative operators are to be used to take the *x* and *y* derivatives of an image. Then a small region of interest (ROI) is to be defined to detect corners in.

A 2x2 matrix of sums of the *x* and *y* derivatives is created as follows:

$$C = \begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix} .$$

Solving $det(C - \lambda I) = 0$, where λ is a column vector of the eigen values and I is the identity matrix, gives the eigen values. For the 2×2 matrix of the equation above, the solutions may be written in closed form:

$$\lambda = \frac{\Sigma D_x^2 + \Sigma D_y^2 \pm \sqrt{(\Sigma D_x^2 + \Sigma D_y^2)^2 - 4 \cdot (\Sigma D_x^2 \Sigma D_y^2 - (\Sigma D_x D_y)^2)}}{2} .$$

If $\lambda_1, \lambda_2 > t$, where *t* is some threshold, then a corner is considered to be found at that location. This can be very useful for object or shape recognition.

Canny Edge Detector

This subsection describes a classic edge detector proposed by J.Canny, see [Canny86]. The detector uses a grayscale image as an input and outputs a black-and-white image, where non-zero pixels mark detected edges. The algorithm consists of three stages described below.

Stage 1: Differentiation

Assuming two-dimensional convolution, the image data are differentiated with respect to the directions *x* and *y*. The gradient of the surface of the convoluted image function in any direction is possible to compute from the known gradient in any two directions.

From the computed *x* and *y* gradient values, the magnitude and angle of the slope can be calculated from the hypotenuse and arctangent.

NOTE. The `ippiSobel` functions perform the first stage and Canny edge detector functions use their output.

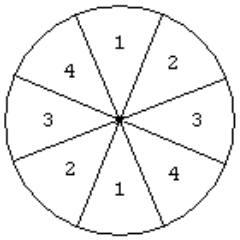
Stage 2: Non-Maximum Suppression

With the rate of intensity change found at each point in the image, edges must now be placed at the points of maximum, or rather non-maximum must be suppressed. A local maximum occurs at a peak in the gradient function, or alternatively where the derivative of the gradient function is set to zero. However, in this case it is preferable to suppress non-maximum perpendicular to the edge direction, rather than parallel to the edge direction, since the edge strength is expected to continue along an extended contour.

The algorithm starts off by reducing the angle of gradient to one of 4 sectors shown in Figure 14-1. The algorithm passes 3×3 neighborhood across the magnitude array. At each point, the center element of neighborhood is compared with its two neighbors along line of the gradient given by the sector value.

If the central value is non-maximum, i.e., not greater than the neighbors, it is suppressed.

Figure 14-1Gradient Sectors



Stage 3: Edge Thresholding

The Canny operator uses the so-called “hysteresis” thresholding. Most thresholders use a single threshold limit, which means that if the edge values fluctuate above and below this value, the line appears broken. This phenomenon is commonly referred to as “streaking”. Hysteresis counters streaking by setting an upper and lower edge value limit. Considering a line segment, if a value lies above the upper threshold limit it is immediately accepted. If the value lies below the low threshold it is immediately rejected. Points which lie between the two limits are accepted if they are connected to pixels which exhibit strong response. The likelihood of streaking is reduced drastically since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur.

J.Canny recommends the ratio of high to low limit be in the range two or three to one, based on predicted signal-to-noise ratios.

Example 14-1 shows how to use the Intel IPP functions for the Canny edge detector.

CannyGetSize

Calculates size of temporary buffer for function ippiCanny.

Syntax

```
IppStatus ippiCannyGetSize(IppiSize roiSize, int* pBufSize);
```

Parameters

roiSize	Size of the image ROI in pixels.
pBufSize	Pointer to the computed size of the temporary buffer.

Description

The function `ippiCannyGetSize` is declared in the `ippcv.h` file. This function calculates the size of a temporary buffer for the function `ippiCanny`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufSize* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if *pRoiSize* has a field with zero or negative value.

Canny

Implements Canny algorithm for edge detection.

Syntax

```
IppStatus ippiCanny_16s8u_C1R(Ipp16s* pSrcDx, int srcDxStep, Ipp16s* pSrcDy,
    int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f
    lowThresh, Ipp32f highThresh, Ipp8u* pBuffer);
IppStatus ippiCanny_32f8u_C1R(Ipp32f* pSrcDx, int srcDxStep, Ipp32f* pSrcDy,
    int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f
    lowThresh, Ipp32f highThresh, Ipp8u* pBuffer);
```

Parameters

pSrcDx	Pointer to the source image ROI *x*-derivative.
srcDxStep	Distance in bytes between starts of consecutive lines in the source image *pSrcDx*.
pSrcDy	Pointer to the source image ROI *y*-derivative.
srcDyStep	Distance in bytes between starts of consecutive lines in the source image *pSrcDy*.
pDstEdges	Pointer to the output array of the detected edges.
dstEdgeStep	Distance in bytes between starts of consecutive lines in the output image.
roiSize	Size of the source image ROI in pixels.
lowThresh	Lower threshold for edges detection.
highThresh	Upper threshold for edges detection.
pBuffer	Pointer to the pre-allocated temporary buffer.

Description

The function `ippiCanny` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function finds edges in the source image ROI and stores them into the output image *pDstEdges* using the Canny algorithm. The function requires a temporary working buffer; its size should be computed previously by calling the function `ippiCannyGetSize`.

Example 14-1 shows how to use the function `ippiCanny_16s8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *pRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcDxStep, srcDyStep* or *dstEdgeStep* is less than *roi.width* * *<pixelSize>*.
`ippStsBadArgErr`	Indicates an error when *lowThresh* is negative or *highThresh* is less than *lowThresh*.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 2 for `16s` images, and by 4 for `32f` images.

Example 14-1 Edge detection using the `ippiCanny` function

```
IppStatus sts;
Ipp32f low=8.0f, high=12.0f;
int size, size1, srcStep, dxStep, dyStep, dstStep;
IppiSize roi;
Ipp8u *src, *dst, *buffer;
Ipp16s *dx, *dy;
…
sts = ippiFilterSobelNegVertGetBufferSize_8u16s_C1R(roi, ippMskSize3x3, &size);
sts = ippiFilterSobelHorizGetBufferSize_8u16s_C1R(roi, ippMskSize3x3, &size1);
if (size<size1) size=size1;
ippiCannyGetSize(roi, &size1);
if (size<size1) size=size1;
buffer = ippsMalloc_8u(size);
sts = ippiFilterSobelNegVertBorder_8u16s_C1R (src, srcStep, dx, dxStep, roi,
                                        ippMskSize3x3, ippBorderRepl, 0, buffer);
sts = ippiFilterSobelHorizBorder_8u16s_C1R(src, srcStep, dy, dyStep, roi,
                                        ippMskSize3x3, ippBorderRepl, 0, buffer);
sts = ippiCanny_16s8u_C1R(dx, dxStep, dy, dyStep, dst, dstStep, roi, low,
                                          high, buffer);
ippsFree(buffer);
```

EigenValsVecsGetBufferSize

Calculates size of temporary buffer for the function `ippiEigenValsVecs`.

Syntax

```
IppStatus ippiEigenValsVecsGetBufferSize_32f_C1R(IppiSize roiSize,int
    apertureSize, int avgWindow, int* pBufferSize);
IppStatus ippiEigenValsVecsGetBufferSize_8u32f_C1R(IppiSize roiSize, int
    apertureSize,  int avgWindow, int* pBufferSize);
```

Parameters

roiSize	Size of the source image ROI in pixels.
apertureSize	Size (pixels) of the derivative operator used by the function, possible values are 3 or 5.
avgWindow	Size of the blurring window in pixels, possible values are 3 or 5.
pBufSize	Pointer to the variable that returns the size of the temporary buffer.

Description

The function `ippiEigenValsVecsGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the size of a temporary buffer to be used by the function `ippiEigenValsVecs`.

CAUTION. The parameters *apertureSize* and *avgWindow* must be the same for both functions `ippiEigenValsVecsGetBufferSize` and `ippiEigenValsVecs`.

Example 14-2 shows how to use the function `ippiEigenValsVecsGetBufferSizee`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiWidth* has zero or negative value, or if *apertureSize* or *avgWindow* has an illegal value.

EigenValsVecs

Calculates eigen values and eigen vectors of image blocks for corner detection.

Syntax

```
IppStatus ippiEigenValsVecs_8u32f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType,
    int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

```
IppStatus ippiEigenValsVecs_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType, int
    apertureSize, int avgWindow, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pEigenVV	Image to store the results.
eigStep	Distance in bytes between starts of consecutive lines in the output image.
roiSize	Size of the source image ROI in pixels.
kernType	Specifies the type of kernel used to compute derivatives, possible values are: ippKernelSobel Sobel kernel 3x3 or 5x5; ippKernelScharr Scharr kernel 3x3.
apertureSize	Size of the derivative operator in pixels, possible values are 3 or 5.
avgWindow	Size of the blurring window in pixels, possible values are 3 or 5.
pBuffer	Pointer to the temporary buffer.

Description

The function ippiEigenValsVecs is declared in the ippcv.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function takes a block around the pixel and computes the first derivatives D_x and D_y. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the *kernType* parameter. The size of the Sobel kernel may be specified the parameter *apertureSize*. If this parameter is set to 3 - the function used 3x3 kernel, if it set to 5 - the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter *apertureSize* must be set to 3 if the Scharr kernel is used.

CAUTION. If the parameter *apertureSize* is set to 5 for operation with the Scharr kernel, the function returns error status.

Then, the function computes eigen values and vectors of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix} .$$

The summation is performed over the full block with averaging over the blurring window with size *avgWindow*.

The image *eigenVV* has the following format. For every pixel of the source image it contains six floating-point values – $\lambda 1$, $\lambda 2$, $x1$, $y1$, $x2$, $y2$. These values are defined as follows:

$\lambda 1$, $\lambda 2$	Eigen values of the above matrix ($\lambda 1 \geq \lambda 2 \geq 0$).
$x1$, $y1$	Coordinates of the normalized eigen vector corresponding to $\lambda 1$.
$x2$, $y2$	Coordinates of the normalized eigen vector corresponding to $\lambda 2$.

In case of a singular matrix or when one eigen value is much smaller than the second one, all these six values are set to 0.

The function requires a temporary working buffer; its size should be computed previously by calling the function ippiEigenValsVecsGetBufferSize.

CAUTION. The parameters *apertureSize* and *avgWindow* must be the same for both functions ippiEigenValsVecsGetBufferSize and ippiEigenValsVecs.

Example 14-2 shows how to use the function ippiEigenValsVecs_8u32f_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *pRoiSize* has a field with zero or negative value, or if *apertureSize* or *avgWindow* has an illegal value; or if *kernType* has wrong value.
ippStsStepErr	Indicates an error condition if *srcStep* is less than *roiSize.width*<pixelSize>*, or *eigStep* is less than *roiSize.width*sizeof(Ipp32f)*6*.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.

Example 14-2 Using the function ippiEigenValsVecs

```
Ipp8u pSrc [4*3] = { 2, 17, 2, 21,
                     9,  4, 11,  5,
                     2, 32, 7,  2};
Ipp32f pEigenVV [24*3];
IppiSize roiSize = { 4, 3 };
int apertureSize = 3;
int avgWindow = 3;
int pBufferSize;

ippiEigenValsVecsGetBufferSize_8u32f_C1R ( roiSize, apertureSize, avgWindow,
&pBufferSize );

Ipp8u* pBuffer = new Ipp8u [ pBufferSize ];

ippiEigenValsVecs_8u32f_C1R ( pSrc, 4, pEigenVV, 24*sizeof(Ipp32f), roiSize,
ippKernelSobel, apertureSize, avgWindow, pBuffer );

result:

2 17  2 21
9  4 11  5    pSrc
2 32  7  2

        pEigenVV
[0.3, 0.0, 0.9, 0.3, 0.3, -0.9] [0.2, 0.0, 1.0, 0.3, 0.3, -1.0]  [0.3, 0.1,
-0.7, 0.8, -0.8, -0.7] [0.5, 0.1, -0.7, 0.7, -0.7, -0.7]
[0.6, 0.1, 1.0, 0.2, 0.2, -1.0]   [0.5, 0.1, 1.0, 0.0, 0.0, -1.0]  [0.4, 0.2,
-0.9, 0.4, -0.4, -0.9] [0.5, 0.2, -0.9, 0.5, -0.5, -0.9]
[0.9, 0.1, 1.0, 0.2, 0.2, -1.0]   [0.9, 0.2, 1.0, 0.0, 0.0, -1.0]  [0.5, 0.2,
-1.0, 0.1, -0.1, -1.0]  [0.5, 0.2, -1.0, 0.2, -0.2, -1.0]
```

MinEigenValGetBufferSize

Calculates size of temporary buffer for the function `ippiMinEigenVal`.

Syntax

```
IppStatus ippiMinEigenValGetBufferSize_32f_C1R(IppiSize roiSize, int
    apertureSize, int avgWindow, int* pBufferSize);
IppStatus ippiMinEigenValGetBufferSize_8u32f_C1R(IppiSize roiSize, int
    apertureSize, int avgWindow, int* pBufferSize);
```

Parameters

roiSize	Size of the source image ROI in pixels.
apertureSize	Size (in pixels) of the derivative operator used by the function, possible values are 3 or 5.
avgWindow	Size of the blurring window in pixels, possible values are 3 or 5.
pBufSize	Pointer to the variable that returns the size of the temporary buffer.

Description

The function `ippiMinEigenValGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function calculates the size of a temporary buffer to be used by the function `ippiMinEigenVal`.

CAUTION. The parameters *apertureSize* and *avgWindow* must be the same for both functions `ippiMinEigenValGetBufferSize` and `ippiMinEigenVal`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiWidth* has a zero or negative value, or if *apertureSize* or *avgWindow* has an illegal value.

MinEigenVal

Calculates the minimal eigen value of image blocks for corner detection.

Syntax

```
IppStatus ippiMinEigenVal_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType,
    int apertureSize, int avgWindow, Ipp8u* pBuffer);
IppStatus ippiMinEigenVal_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType,
    int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMinEigenVal	Pointer to the image that is filled with the minimal eigen values.
minValStep	Distance in bytes between starts of consecutive lines in the output image.
roiSize	Size of the source image ROI in pixels.
kernType	Specifies the type of kernel used to compute derivatives, possible values are: `ippKernelSobel` Sobel kernel 3x3 or 5x5; `ippKernelScharr` Scharr kernel 3x3.
apertureSize	Size of the derivative operator in pixels, possible values are 3 or 5.
avgWindow	Size of the averaging window in pixels, possible values are 3 or 5.
pBuffer	Pointer to the temporary buffer.

Description

The function `ippiMinEigenVal` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function takes a block around the pixel and computes the first derivatives D_x and D_y. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the *kernType* parameter. The size of the Sobel kernel may be specified the parameter *apertureSize*. If this parameter is set to

3 - the function used 3x3 kernel, if it set to 5 - the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter *apertureSize* must be set to 3 if the Scharr kernel is used.

CAUTION. If the parameter *apertureSize* is set to 5 for operation with the Scharr kernel, the function returns error status.

Then, the function computes the minimal eigen value λ ($\lambda \geq 0$) of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix} .$$

The summation is performed over the full block with averaging over the blurring window with size *avgWindow*.

The function requires a temporary working buffer; its size should be computed previously by calling the function ippiMinEigenValGetBufferSize.

CAUTION. The parameters *apertureSize* and *avgWindow* must be the same for both functions ippiMinEigenValGetBufferSize and ippiMinEigenVal.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *pRoiSize* has a field with zero or negative value, or if *apertureSize* or *avgWindow* has an illegal value; or if *kernType* has wrong value.
ippStsStepErr	Indicates an error condition if *srcStep* is less than *roiSize.width*<pixelSize>*, or *eigenvvStep* is less than *roiSize.width*sizeof(Ipp32f)*.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.

Hough Transform

The Hough transform is a general technique that allows to detect the flat curves in the binarised images [Gon93]. The current version of Intel IPP implements the detecton of the straight lines that are defined by the parametric equation $r = x*\cos(\theta) + y*\sin(\theta)$, where r and θ are the length and angle from the origin of a normal to the line respectively.

HoughLineGetSize

Computes the size of the working buffer for the straight lines detection.

Syntax

```
IppStatus ippiHoughLineGetSize_8u_C1R(IppiSize roiSize, IppPointPolar
    delta, int maxLineCount, int* pBufSize);
```

Parameters

roiSize	Size of the source image ROI in pixels.
delta	Step of discretization (*delta.rho* -radial increment, *delta.theta* - angular increment).
maxLineCount	Number of elements of the destination buffer.
pBufSize	Pointer to the size of the working buffer.

Description

The function `ippiHoughLineGetSize` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the temporary working buffer that is required for the function `ippiHoughLine`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pBufSize* is NULL.
`ippStsSizeErr`	Indicates an error condition if *roiSize* or *delta* has a field with zero or negative value.

HoughLine

Detects straight lines in the source image.

Syntax

```
IppStatus ippiHoughLine_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
    roiSize, IppPointPolar delta, int threshold,    IppPointPolar* pLine,
    int maxLineCount, int* pLineCount, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of source image ROI in pixels.
delta	Step of discretization (*delta.rho* -radial increment, *delta.theta* - angular increment).
threshold	Minimum number of points that are required to detect the line.
pLine	Pointer to the destination buffer for lines.
maxLineCount	Number of elements of the destination buffer.
pBuffer	Pointer to the working buffer.

Description

The function `ippiHoughLine` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs detection of the straight line defined by the above equation in the binarised source image *pSrc*. The level of discretization *delta* is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter.
The function requires the external working buffer *pBuffer* whose size should be computed previously using the function `ippiHoughLineGetSize`.

CAUTION. The value of the parameter *delta* must not be greater than the value of the parameter *delta* set when the size of the working buffer is computed.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value; or if *maxLineCount* is less than or equal to 0.
ippStsStepErr	Indicates an error condition if *srcStep* has an illegal value.
ippStsBadArgErr	Indicates an error condition if *threshold* is less than or equal to 0; or *delta.rho* is less than 0, or greater than sum of the width and height of the ROI; or *delta.theta* is less than 0, or greater than π.
ippStsDstSizeLessExpected	Indicates a warning if number of the detected lines is greater thean the size of the destination buffer *maxLineCount*.

HoughLine_Region

Detects straight lines with the specified range of parameters in the source image.

Syntax

```
IppStatus ippiHoughLine_Region_8u32f_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize roiSize, IppPointPolar* pLine, IppPointPolar dstRoi[2], int
    maxLineCount, int* pLineCount, IppPointPolar delta, int threshold, Ipp8u*
    pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of source image ROI in pixels.
pLine	Pointer to the destination array of detected lines.
dstRoi	Specifies the range of parameters of straight lines to be detected.
pLineCount	Pointer to the number of detected lines.

delta	Step of discretization (*delta.rho* -radial increment, *delta.theta* - angular increment).
threshold	Minimum number of points that are required to detect the line.
maxLineCount	Maximum number of lines in the destination buffer.
pBuffer	Pointer to the working buffer.

Description

The function `ippiHoughLine_Region` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs detection of the straight line defined by the above equation in the binarised source image *pSrc*. Only lines *line*[n] with the parameters satisfying the following conditions are detected:

$$dstRoi[0].rho \le line[n].rho \le dstRoi[1].rho;$$

$$dstRoi[0].theta \le line[n].theta \le dstRoi[1].theta;$$

where `n = 0... pLineCount`.

The level of discretization *delta* is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter.
The function requires the external working buffer *pBuffer* whose size should be computed previously using the function `ippiHoughLineGetSize`.

CAUTION. The value of the parameter *delta* must not be greater than the value of the parameter *delta* set when the size of the working buffer is computed.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value; or if *maxLineCount* is less than or equal to 0.
`ippStsStepErr`	Indicates an error condition if *srcStep* has an illegal value.

`ippStsBadArgErr`	Indicates an error condition if *`threshold`* is less than or equal to 0; or *`delta.rho`* is less than 0, or greater than sum of the width and height of the ROI; or *`delta.theta`* is less than 0, or greater than π; or some filed of *`dstRoi`*`[0]` is greater than the corresponding filed of *`dstRoi`*`[1]`.
`ippStsDstSizeLessExpected`	Indicates a warning if number of the detected lines is greater thean the size of the destination buffer *`maxLineCount`*.

Distance Transform Functions

This section describes the distance transform functions.

Distance transform is used for calculating the distance to an object. The input is an image with feature and non-feature pixels. The function labels every non-feature pixel in the output image with a distance to the closest feature pixel. Feature pixels are marked with zero.

Distance transform is used for a wide variety of subjects including skeleton finding and shape analysis.

DistanceTransform

Calculates distance to the closest zero pixel for all non-zero pixels of source image.

Syntax

Case 1: Not-in-place operations

```
IppStatus ippiDistanceTransform_3x3_<mod>(const Ipp8u* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32s* pMetrics);
IppStatus ippiDistanceTransform_5x5_<mod>(const Ipp8u* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32s* pMetrics);
```

Supported values for *mod*:

```
8u_C1R          8u16u_C1R
```

```
IppStatus ippiDistanceTransform_3x3_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);
IppStatus ippiDistanceTransform_5x5_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);
```

Case 2: In-place operations

```
IppStatus ippiDistanceTransform_3x3_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32s* pMetrics);
IppStatus ippiDistanceTransform_5x5_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32s* pMetrics);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination distance image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
roiSize	Size of the image ROI in pixels.
pMetrics	Pointer to the array that specifies used metrics.

Description

The distance transform function `ippiDistanceTransform` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function approximates the actual distance from the closest zero pixel to each certain pixel with the sum of atomic distances from the fixed set. The set consists of two values for a 3×3 mask and three values for a 5×5 mask.

Figure 14-2 shows the result of the distance transform of a 7×7 image with zero point in the center. This example corresponds to a 3×3 mask. Two numbers specify metrics in case of the 3×3 mask:

- distance between two pixels that share an edge,
- distance between the pixels that share a corner.

In this case the values are 1 and 1.5 correspondingly.

Figure 14-2 3x3 Mask

4.5	4	3.5	3	3.5	4	4.5
4	3	2.5	2	2.5	3	4
3.5	2.5	1.5	1	1.5	2.5	3.5
3	2	1	0	1	2	3
3.5	2.5	1.5	1	1.5	2.5	3.5
4	3	2.5	2	2.5	3	4
4.5	4	3.5	3	3.5	4	4.5

Figure 14-3 shows the distance transform for the same image, but for a 5×5 mask.

For this mask yet another number is added to specify metrics - the additional distance, i.e., the distance between pixels corresponding to the chess knight move.

In this example, the additional distance is equal to 2.

Figure 14-3 5x5 Mask

4	3.5	3	3	3	3.5	4
3.5	3	2	2	2	3	3.5
3	2	1.5	1	1.5	2	3
3	2	1	0	1	2	3
3	2	1.5	1	1.5	2	3
3.5	3	2	2	2	3	3.5
4	3.5	3	3	3	3.5	4

Example 14-3 shows how to use the function `ippiDistanceTransform_3x3_8u_C1R`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width*<pixelSize>*.
`ippStsNotEvenStepErr`	Indicates an error condition if step value is not divisible by 2 for `16u` images, and by 4 for `32f` images.
`ippStsCoeffErr`	Indicates an error condition if at least one element of *pMetrics* array has zero or negative value.

Example 14-3 Using the function `ippiDistanceTransform_3x3`

```
Ipp8u src[7*7] = {
      1, 2, 3, 4, 5, 6, 7,
      1, 0, 3, 4, 5, 6, 7,
      1, 2, 3, 4, 5, 6, 7,
      1, 2, 3, 0, 5, 6, 7,
      1, 2, 3, 4, 5, 6, 7,
      1, 2, 3, 4, 5, 0, 7,
      1, 2, 3, 4, 5, 6, 7
    };
Ipp8u dst[7*7];
```

Example 14-3 Using the function `ippiDistanceTransform_3x3`

```
IppiSize roiSize = { 7, 7 };
Ipp32s pMetrics[2] = { 2, 2 };

ippiDistanceTransform_3x3_8u_C1R ( src, 7, dst, 7, roiSize, pMetrics );
result:
1 2 3 4 5 6 7
1 0 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 0 5 6 7     src
1 2 3 4 5 6 7
1 2 3 4 5 0 7
1 2 3 4 5 6 7

2 2 2 4 6 6 6
2 0 2 4 4 4 6
2 2 2 2 2 4 6
4 4 2 0 2 4 4     dst
6 4 2 2 2 2 2
6 4 4 4 2 0 2
6 6 6 4 2 2 2
```

GetDistanceTransformMask

Returns an optimal mask for a given type of metrics and given mask size.

Syntax

```
IppStatus ippiGetDistanceTransformMask_<mod>(int kerSize, IppiNorm norm,
    Ipp<datatype>* pMetrics);
```

Supported values for *mod*:

`32s` `32f`

Parameters

kerSize	Specifies the mask size as follows: 3 for 3x3 mask, 5 for 5x5 mask.
norm	Specifies the type of metrics. Possible values are:
	`ippiNormInf(0)` L_∞, $\Delta = max(\lvert x_1 - x_2\rvert, \lvert y_1 - y_2\rvert)$,
	`ippiNormL1(1)` L_1, $\Delta = \lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert$,
	`ippiNormL2(2)` L_2, $\Delta = sqrt((x_1 - x_2)^2 + (y_1 - y_2)^2)$

pMetrics	Pointer to the output array to store metrics parameters. The array contains the following number of elements: 2 for 3x3 mask, 3 for 5×5 mask.

Description

The function `ippiGetDistanceTransformMask` is declared in the `ippcv.h` file. This function fills up the output array with metrics parameters for the given type of metrics and size of mask. The function returns the following results:

(1, 1)	L_∞, 3×3 mask,
(1, 2)	L_1, 3×3 mask,
(2, 3)	L_2, 3×3 mask, `32s` data type,
(0.955, 1.3693)	L_2, 3×3 mask, `32f` data type,
(1, 1, 2)	L_∞, 5×5 mask,
(1, 2, 3)	L_1, 5×5 mask,
(4, 6, 9)	L_2, 5×5 mask, `32s` data type
(1.0, 1.4, 2.1969)	L_2, 5×5 mask, `32f` data type.

L_∞ and L_1 metrics for 5x5 mask are calculated using 3×3 mask,

For more information, see [Bor86].

NOTE. For compatibility with the previous versions of the library the earlier function `ippiGetDistanceTransformMask` replaced by the function `ippiGetDistanceTransformMask_32f` in the current version is also supported.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pMetrics* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *kerSize* has a wrong value.
`ippStsBadArgErr`	Indicates an error condition if *kerSize* or *norm* has a wrong value.

TrueDistanceTransformGetBufferSize

Calculates the size of the temporary working buffer for the function `ippiTrueDistanceTransform`.

Syntax

```
IppStatus ippiTrueDistanceTransformGetBufferSize_8u16u_C1RSfs(IppiSize
    roiSize, int* pBufferSize);
IppStatus ippiTrueDistanceTransformGetBufferSize_8u32f_C1R(IppiSize
    roiSize, int* pBufferSize);
```

Parameters

roiSize	Size of the source image ROI (in pixels).
pBufferSize	Pointer to the computed size of the buffer.

Description

The function `ippiTrueDistanceTransformGetBufferSiz` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the function `ippiTrueDistanceTransform`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufferSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has field with zero or negative value.

TrueDistanceTransform

Calculates the Eucledian distance to the closest zero pixel for all non-zero pixels of the source image.

Syntax

```
IppStatus ippiTrueDistanceTransform_8u32f_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
IppStatus ippiTrueDistanceTransform_8u16u_C1RSfs(const Ipp8u* pSrc, int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize, int scaleFactor,
    Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination distance image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the image ROI in pixels.
pBuffer	Pointer to the temporary working buffer.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiTrueDistanceTransform` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the Eucledian distance to the closest zero pixel for all non-zero pixels of the source image [Felz04].

Figure 14-2 shows the result of the integer version of the true distance transform of a 7x7 image with zero point in the center and with the scale factor -5.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width*<pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if step value is not divisible by 2 for 16u images, and by 4 for 32f images.

Figure 14-4 True Distance Transform, *scaleFactor*=-5

136	136	101	96	101	115	136
115	91	72	64	72	91	115
101	72	45	36	45	72	101
96	64	36	0	36	64	96
101	72	45	36	45	72	101
115	91	72	64	72	91	115
136	115	101	96	101	115	136

FastMarchingGetBufferSize

Computes the size of the working buffer for the peak search.

Syntax

```
IppStatus ippiFastMarchingGetBufferSize_8u32f_C1R(int roiWidth,
    int* pBufferSize);
```

Parameters

roiWidth	Maximum image size (in pixels).
pBufferSize	Pointer to the computed size of the buffer.

Description

The function ippiFastMarchingGetBufferSize is declared in the ippcv.h It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the ippiFastMarching function. The buffer with the length *pBufferSize*[0] can be used to filter images with width that is equal to or less than the parameter *roiWidth* specified for the function ippiFastMarching.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if *pBufferSize* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiWidth* is lessthan 1.

FastMarching

Calculates distance transform to closest zero pixel for all non-zero pixels of source image using fast marching method.

Syntax

```
IppStatus ippiFastMarching_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f radius, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
raduis	Radius of the neighborhod of the marked area.
pBuffer	Pointer to the working buffer.

Description

The function ippiFastMarching is declared in the ippcv.h It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the distance from the closest zero pixel to each image pixel according to the Fast Marching Method (FMM) [Telea04]. The FMM distance for area Ω with the border $\partial\Omega$ is a solution of the equations:

$$(|\nabla T(x,\ y)| = 1),\ \{x,\ y\} \in \Omega$$

$$(T(x,\ y) = 0),\ \{x,\ y\} \in \partial\Omega$$

The resulting distance complies with the equation

$$T(x,\ y) = min\left(\frac{T(u_1,\ v_1) + T(u_2,\ v_2) + \sqrt{2 - (T(u_1,\ v_1) - T(u_2,\ v_2))^2}}{2},\ min(T(u_1,\ v_1),\ T(u_2,\ v_2)) + 1\right)$$

Here $\{u_1,\ v_1\}$ and $\{u_2,\ v_2\}$ are coordinates for pair of pixels adjacent to the pixel with $\{x,\ y\}$ coordinates.

The area Ω is defined by the non-zero pixel of the image *pSrc*. If *raduis* is positive, then the FMM distance with the negative sign is calculated in Euclidean *raduis*-neighborhood of Ω.

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiFastMarchingGetBufferSize` beforehand.

Figure 14-5 shows the result of the fast marching method for the 7x9 image with centered 3x5 non-zero mask and *raduis*=1.

Figure 14-5 Result of the FFM Method

0.0000	0.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	0.0000	0.0000
0.0000	0.7071	-1.e-10	-1.e-10	-1.e-10	-1.e-10	-1.e-10	0.7071	-1.0000
-1.0000	-1.e-10	0.7071	0.9659	0.9994	0.9659	0.7071	-1.e-10	-1.0000
-1.0000	-1.e-10	0.9659	1.6730	1.9579	1.6730	0.9659	-1.e-10	-1.0000
-1.0000	-1.e-10	0.7071	0.9659	0.9994	0.9659	0.7071	-1.e-10	-1.0000
0.0000	0.7071	-1.e-10	-1.e-10	-1.e-10	-1.e-10	-1.e-10	0.7071	-1.0000
0.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	0.0000

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
ippStsNotEvenStepErr	Indicates an error condition if the step value is not divisible by 4 for floating-point images.
ippStsBadArgErr	Indicates an error condition if *raduis* is negative.

Image Gradients

GradientColorToGray

Converts a color gradient image to grayscale.

Syntax

```
IppStatus ippiGradientColorToGray_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiNorm
    norm);
```

Supported values for *mod* :

8u_C3C1R 16u_C3C1R 32f_C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
norm	Type of norm to form the mask for dilation; following values are possible: ippiNormInf Infinity norm. ippiNormL1 L1 norm. ippiNormL2 L2 norm.

Description

The function `ippiGradientColorToGray` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function creates the grayscale gradient image *pDst* from the source three-channel gradient image *pSrc*. The type of norm is specified by the parameter. Pixel values for destination image are computed for different type of norm in accordance with the following formula:

$$dst_{i,\ j} = \begin{cases} \max\{|src_{i,\ j,\ 0}|, |src_{i,\ j,\ 1}|, |src_{i,\ j,\ 2}|\} & norm = ippiNormInf \\ |src_{i,\ j,\ 0}| + |src_{i,\ j,\ 1}| + |src_{i,\ j,\ 2}| & norm = ippiNormL1 \\ \sqrt{src_{i,\ j,\ 0}^{2} + src_{i,\ j,\ 1}^{2} + src_{i,\ j,\ 2}^{2}} & norm = ippiNormL2 \end{cases}$$

For integer flavors the result is scaled to the full range of the destination data type.

Return Values

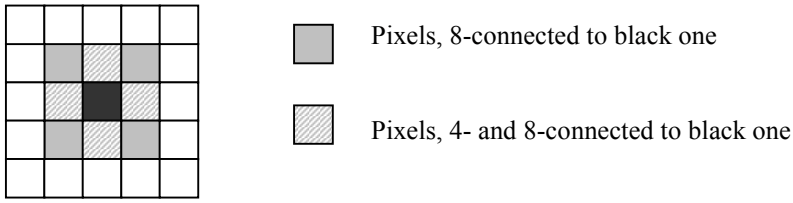
`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 2 for integer images, or by 4 for floating-point images.
`ippStsBadArgErr`	Indicates an error condition if *norm* has an illegal value.

Flood Fill Functions

This section describes functions performing flood filling of connected areas. *Flood filling* means that a group of connected pixels with close values is filled with, or is set to, a certain value. The flood filling process starts with a specified point ("seed") and continues until it reaches the image ROI boundary or cannot find any new pixels to fill due to a large difference in pixel values. For every pixel filled, the functions analyze neighbor pixels:

- 4 neighbors (except diagonal neighbors); this kind of connectivity is called 4-connectivity and the corresponding function name includes `4Con`, or
- 8 neighbors (diagonal neighbors included); this kind of connectivity is called 8-connectivity and the corresponding function name includes `8Con`.

Figure 14-6 Pixels Connectivity Patterns



These functions can be used for:

- segmenting a grayscale image into a set of uni-color areas,
- marking each connected component with individual color for bi-level images.

FloodFillGetSize

Calculates size of temporary buffer for flood filling operation.

Syntax

```
IppStatus ippiFloodFillGetSize(IppiSize roiSize, int* pBufSize);
```

Parameters

roiSize	Size of the source image ROI in pixels.

pBufSize	Pointer to the variable that returns the size of the temporary buffer.

Description

The function `ippiFloodFillGetSize` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function `ippiFloodFill`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

FloodFillGetSize_Grad

Calculates size of temporary buffer for the gradient flood filling.

Syntax

```
IppStatus ippiFloodFillGetSize_Grad(IppiSize roiSize, int* pBufSize);
```

Parameters

roiSize	Size of the source image ROI in pixels.
pBufSize	Pointer to the variable that returns the size of the temporary buffer.

Description

The function `ippiFloodFillGetSize_Grad` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function `ippiFloodFill_Grad`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pBufSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

FloodFill

Performs flood filling of connected area.

Syntax

Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u_C1IR        16u_C1IR       32f_C1IR
```

Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u_C3IR        32f_C3IR
```

Parameters

pImage	Pointer to the ROI in the source and destination image (for the in-place operation).
imageStep	Distance in bytes between starts of consecutive lines in the image buffer.
roiSize	Size of the image ROI in pixels.
seed	Initial point.
newVal	Value to fill with for one-channel data.
pNewVal	Pointer to the vector containing values to fill with for three-channel data.

pRegion	Pointer to the connected components structure that stores information about the refilled area.
pBuffer	Pointer to the temporary buffer.

Description

The function `ippiFloodFill` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs flood filling of the group of connected pixels whose pixel values are equal to the value in the *seed* point. Values of these pixel is set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function `ippiFloodFillGetSize` beforehand.

The functions with the "`_4con`" suffixes check 4-connected neighborhood of each pixel, that is, side neighbors. The functions with the "`_8con`" suffixes check 8-connected neighborhood of each pixel, that is, side and corner neighbors. See Figure 14-6.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *imageStep* is less than *pRoiSize.width* * *<pixelSize>*.
`ippStsNotEvenStepErr`	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
`ippStsOutOfRangeErr`	Indicates an error condition if the *seed* point is out of ROI.

FloodFill_Grad

Performs gradient flood filling of connected area on an image.

Syntax

Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal,
    Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal,
    Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u_C1IR        16u_C1IR       32f_C1IR
```

Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal,
    Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal,
    Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u_C3IR        32f_C3IR
```

Parameters

pImage	Pointer to the ROI in the source and destination image (in-place operation).
imageStep	Distance in bytes between starts of consecutive lines in the image buffer.
roiSize	Size of the image ROI in pixels.

seed	Initial point.
minDelta	Minimum difference between neighbor pixels for one-channel data.
maxDelta	Maximum difference between neighbor pixels for one-channel data.
newVal	Value to fill with for one-channel data.
pMinDelta	Pointer to the minimum differences between neighbor pixels for three-channel images.
pMaxDelta	Pointer to the maximum differences between neighbor pixels for three-channel images.
pNewVal	Pointer to the vector containing values to fill with for three-channel data.
pRegion	Pointer to the connected components structure that stores information about the refilled area.
pBuffer	Pointer to the temporary buffer.

Description

The function `ippiFloodFill_Grad` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs flood filling of the group of connected pixels in the *seed* pixel neighborhoods whose pixel values *v* satisfies the following conditions:

$$v_0 - d_{lw} \le v \le v_0 + d_{up},$$

where v_0 is the value of at least one of the current pixel neighbors, which already belongs to the refilled area, and d_{lw}, d_{up} are *minDelta*, *maxDelta*, respectively. Values of these pixel is set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function `ippiFloodFillGetSize_Grad` beforehand.

The functions with the "`_4con`" suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the "`_8con`" suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See Figure 14-6.

Example 14-4 shows how to use the function `ippiFloodFill_Grad4Con_8u_C1IR`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *imageStep* is less than *pRoiSize.width* * *<pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
ippStsOutOfRangeErr	Indicates an error condition if the *seed* point is out of ROI.

Example 14-4 Using the function ippiFloodFill_Grad

```
Ipp8u pImage[8*8] = { 0, 1, 2, 3, 4, 5, 6, 7,
                      0, 1, 2, 3, 4, 5, 6, 7,
                      0, 1, 2, 3, 4, 5, 6, 7,
                      0, 1, 2, 3, 4, 5, 6, 7,
                      0, 1, 2, 3, 4, 5, 6, 7,
                      0, 1, 2, 3, 4, 5, 6, 7,
                      0, 1, 2, 3, 4, 5, 6, 7,
                      0, 1, 2, 3, 4, 5, 6, 7};
IppiSize roiSize = { 4, 4 };
IppiPoint seed = { 2, 2 };
Ipp8u newVal = 9;
Ipp8u minDelta = 1;
Ipp8u maxDelta = 1;
int pBufSize;
IppiConnectedComp pRegion;

ippiFloodFillGetSize_Grad ( roiSize, &pBufSize );

Ipp8u* pBuffer = ippiMalloc_8u_C1 ( 8, 8, &pBufSize );

ippiFloodFill_Grad4Con_8u_C1IR ( pImage, 8, roiSize, seed, newVal, minDelta,
                                 maxDelta, &pRegion, pBuffer );

result:
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7        pImage (source)
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

9 9 9 9 4 5 6 7
9 9 9 9 4 5 6 7
9 9 9 9 4 5 6 7
9 9 9 9 4 5 6 7
0 1 2 3 4 5 6 7        pImage (destination)
0 1 2 3 4 5 6 7
```

Example 14-4 Using the function `ippiFloodFill_Grad`

```
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

pRegion.area = 16
pRegion.value = 9.0
pRegion.rect.x = 0.0        pRegion (after FloodFill_Grad)
pRegion.rect.y = 0.0
pRegion.rect.width = 4.0
pRegion.rect.height = 4.0
```

FloodFill_Range

Performs flood filling of pixels with values in the specified range in the connected area on an image.

Syntax

Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal,
    Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Range8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal,
    Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

Supported values for *mod* :

```
8u_C1IR        16u_C1IR       32f_C1IR
```

Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal,
    Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Range8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal,
    Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp*
    pRegion, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
8u_C3IR        32f_C3IR
```

Parameters

pImage	Pointer to the ROI in the source and destination image (in-place operation).
imageStep	Distance in bytes between starts of consecutive lines in the image buffer.
roiSize	Size of the image ROI in pixels.
seed	Initial point.
minDelta	Minimum difference between neighbor pixels for one-channel data.
maxDelta	Maximum difference between neighbor pixels for one-channel data.
newVal	Value to fill with for one-channel data.
pMinDelta	Pointer to the minimum differences between neighbor pixels for three-channel images.
pMaxDelta	Pointer to the maximum differences between neighbor pixels for three-channel images.
pNewVal	Pointer to the vector containing values to fill with for three-channel data.
pRegion	Pointer to the connected components structure that stores information about the refilled area.
pBuffer	Pointer to the temporary buffer.

Description

The function `ippiFloodFill_Grad` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs flood filling of the group of connected pixels in the *seed* pixel neighborhoods whose pixel values *v* satisfies the following conditions:

$$v_0 - d_{lw} \le v \le v_0 + d_{up},$$

where v_0 is the pixel value of the *seed* point, and d_{lw}, d_{up} are *minDelta*, *maxDelta*, respectively. Values of these pixel is set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function `ippiFloodFillGetSize` beforehand.

The functions with the "_4con" suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the "_8con" suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See Figure 14-6.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *imageStep* is less than *pRoiSize.width* * *<pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
ippStsOutOfRangeErr	Indicates an error condition if the *seed* point is out of ROI.

Motion Analysis and Object Tracking

Motion Template Functions

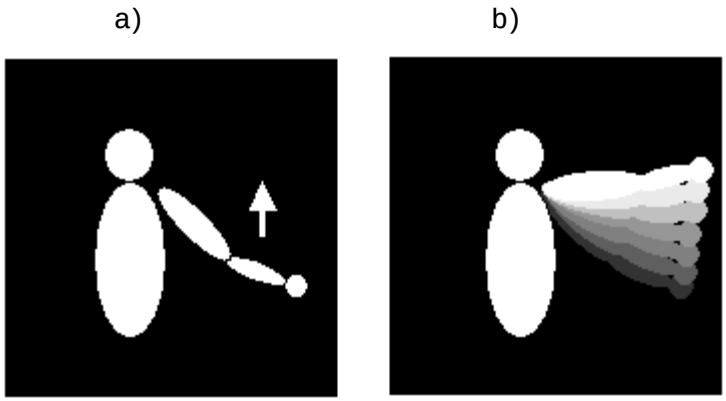
This section describes a motion templates function.
This function generates motion templates images to rapidly determine where, how, and in which direction the motion occurred. The algorithms are based on [Davis97], and [Davis99]. The function operates on images that are the output of frame or background differencing, or other image segmentation operations. Thus, the input and output image types are all grayscale, that is, one color channel. The pixel types can be 8u, 8s, or 32f.

Motion Representation

Figure 14-7 (a) shows capturing a foreground silhouette of the moving object or person. As the person or object moves, copying the most recent foreground silhouette as the highest values in the motion history image creates a "layered history" of the resulting motion. Typically, this "highest value" is just a floating-point timestamp of time since the code has been running in milliseconds. Figure 14-7 (b) shows the result that may be called the

Motion History Image (MHI). The MHI in Figure 14-7 represents how the motion took place. A pixel level or a time delta threshold, as appropriate, is set such that pixel values in the MHI that fall below that threshold are set to zero.

Figure 14-7Motion Image History



The most recent motion has the highest value, earlier motions have decreasing values subject to a threshold below which the value is set to zero.

Updating MHI Images

Generally, floating point images are used because system time differences, that is, time elapsing since the application was launched, are read in milliseconds to be further converted into a floating point number which is the value of the most recent silhouette. Then follows writing this current silhouette over the past silhouettes with subsequent thresholding away pixels that are too old to create the MHI.

UpdateMotionHistory

Updates motion history image using motion silhouette at given timestamp.

Syntax

```
IppStatus ippiUpdateMotionHistory_<mod>(const Ipp<srcDatatype>*
    pSilhouette, int silhStep, Ipp32f* pMhi, int mhiStep, IppiSize roiSize,
    Ipp32f timeStamp, Ipp32f mhiDuration);
```

Supported values for *mod*:

```
8u32f_C1IR     16u32f_C1IR    32f_C1IR
```

Parameters

pSilhouette	Pointer to the silhouette image ROI that has non-zero values for those pixels where the motion occurs.
roiSize	Size of the image ROI in pixels.
silhStep	Distance in bytes between starts of consecutive lines in the silhouette image.
pMhi	Pointer to the motion history image which is both an input and output parameter.
mhiStep	Distance in bytes between starts of consecutive lines in the motion history image.
timeStamp	Timestamp in milliseconds.
mhiDuration	Threshold for MHI pixels. MHI motions older than this threshold are deleted.

Description

The function `ippiUpdateMotionHistory` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function updates the motion history image. It sets MHI pixels to the current *timestamp* value, if their values are non-zero.

The function deletes MHI pixels, if their values are less than the *mhiDuration* timestamp, that is, the pixels are “old.”

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *mhiStep* or *silhStep* is less than *roiSize.width * <pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if the step value is not divisible by 2 for 16u images, and by 4 for 32f images.
ippStsOutOfRangeErr	Indicates an error when *mhiDuration* is negative.

Optical Flow

This section describes the functions that calculate the optical flow using the pyramidal Lucas-Kanade algorithm [Bou99].

The optical flow between two images is generally defined as an apparent motion of image brightness. Let *I(x, y, t)* be the image brightness that changes in time to provide an image sequence.

Optical flow coordinates $u = \frac{\partial x}{\partial t}$, $\nu = \frac{\partial y}{\partial t}$ can be found from so called *optical flow constraint equation*:

$$-\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}\nu$$

The Lucas-Kanade algorithm assumes that the group of adjacent pixels has the same velocity and founds the approximate solution of the above equation using the least square method.

OpticalFlowPyrLKInitAlloc

Allocates memory and initializes a structure for optical flow calculation.

Syntax

```
IppStatus ippiOpticalFlowPyrLKInitAlloc_<mod>(IppiOptFlowPyrLK_<mod>** ppState,
    IppiSize roiSize, int winSize, IppHintAlgorithm hint);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R

Parameters

ppState	Pointer to the pointer to the optical flow structure being initialized.
roiSize	Size of the source image (zero level of the pyramid) ROI in pixels.
winSize	Size of the search window of each pyramid level.
hint	Option to select the algorithmic implementation of the transform function

Description

The function `ippiOpticalFlowPyrLKInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function allocates memory and initializes a structure *pState* for calculation the optical flow between two images using the pyramidal Lucas-Kanade algorithm in the centered window of size *winSize* * *winSize*. Computation algorithm is specified by the *hint* argument. This structure is used by the function `ippiOpticalFlowPyrLK`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if *ppState* is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value or if *winSize* is equal to or less than 0.
ippStsMemAllocErr	Memory allocation error.

OpticalFlowPyrLKFree

Frees memory allocated for the optical flow structure.

Syntax

```
IppStatus ippiOpticalFlowPyrLKFree_<mod>(IppiOptFlowPyrLK_<mod>* pState);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R

Parameters

pState	Pointer to the optical flow structure.

Description

The function `ippiOpticalFlowPyrLKFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function `ippiOpticalFlowPyrLKInitAlloc` for the optical flow structure.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error if `pState` is `NULL`.

OpticalFlowPyrLK

Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

Syntax

```
IppStatus ippiOpticalFlowPyrLK_<mod>(IppiPyramid* pPyr1, IppiPyramid* pPyr2,
    const IppiPoint2D32f* pPrev, IppiPoint_32f* pNext, Ipp8s* pStatus, Ipp32f*
    pError, int numFeat, int winSize, int maxLev, int maxIter, Ipp32f threshold,
    IppiOptFlowPyrLK_<mod>* pState);
```

Supported values for *mod* :

`8u_C1R` `16u_C1R` `32f_C1R`

Parameters

pPyr1	Pointer to the ROI in the first image pyramid structure.
pPyr2	Pointer to the ROI in the second image pyramid structure.
pPrev	Pointer to the array of initial coordinates of the feature points.
pNext	Pointer to the array of new coordinates of feature point; as input it contains hints for new coordinates.
pStatus	Pointer to the array of result indicators.
pError	Pointer to the array of differences between neighborhoods of old and new point positions.

numFeat	Number of feature points.
winSize	Size of the square search window.
maxLev	Pyramid level to start the operation.
maxIter	Maximum number of algorithm iterations for each pyramid level.
threshold	Threshold value.
pState	Pointer to the pyramidal optical flow structure.

Description

The function `ippiOpticalFlowPyrLK` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function implements the iterative version of the Lucas-Kanade algorithms [Bou99]. It computes with sub-pixel accuracy new coordinates of the *numFeat* feature points of two images at time *t* and *t+dt* . Their initial coordinates are places in the *pPrev* array. Computed values of new coordinates of the feature points are stored in the array *pNext*, that initially contains estimations of them (or hints), for example, based on the flow values for the previous image in sequence. If there are not such hints, the *pNext* array contains the same initial coordinates as the *pPrev* array.

The images are presented by the pyramid structures *pPyr1* and *pPyr2* respectively (see description of the `ippiPyramidInitAlloc` function for more details).These structures should be initialized by calling the function `ippiPyramidInitAlloc` beforehand. Furthermore the function uses the pyramidal optical flow structure *pState* that also should be previously initialized by the function `ippiOpticalFlowPyrLKInitAlloc`.

The function starts operation on the highest pyramid level (smallest image) that is specified by the *maxLev* parameter in the centered search window whose size *winSize* could not exceed the corresponding value *winSize* that is specified in the function `ippiOpticalFlowPyrLKInitAlloc`. The operation for *i*-th feature point on the given pyramid level finishes if:

- new position of the point is found ($\sqrt{dx^2 + dy^2} < threshold$),
- specified number of iteration *maxIter* is performed,
- the intersection between the pyramid layer and the search window became empty.

In first two cases for non-zero levels the new position coordinates are scaled to the next pyramid level and the operation continues on the next level.
For zero level or for third case the operation stops, the number of the corresponding level

is written to the *pStatus*[i] element, the new coordinates are scaled to zero level and are written to pNext[i]. The square root of the average squared difference between neighborhoods of old and new positions is written to pError[i].

Example 14-5 shows how to build pyramids and calculate the optical flow for two images.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if numFeat or winSize has zero or negative value.
ippStsBadArgErr	Indicates an error condition if maxLev or threshold has negative value, or maxIter has zero or negative value.

Example 14-5 Pyramid Building and Optical Flow Calculation

```
void pyroptflow(
   const Ipp8u   *prevFrame, // previous frame
   int            prevStep,  // its row step
   const Ipp8u   *nextFrame, // next frame
   int            nextStep,  // its row step
   IppiSize       roiSize,   // frame size
   int            numLevel,  // pyramid level number (5)
   float          rate,      // pyramid rate (2.0f)
   Ipp16s        *pKernel,   // pyramid kernel
   int            kerSize,   // pyramid kernel size (5)
   IppiPoint_32f *prevPt,    // coordinates on previous frame
   IppiPoint_32f *nextPt,    // hint to coordinates on next frame
   Ipp8s         *pStatus,   // result indicators
   Ipp32f        *pError,    // differences
   int            numFeat,   // point number
   int            winSize,   // search window size (41)
   int            numIter,   // iteration number (5)
   float          threshold) // threshold (0.0001f)
{
   IppiPyramid *pPyr1, *pPyr2;
   IppiOptFlowPyrLK *pOF;

   ippiPyramidInitAlloc (&pPyr1, numLevel, roiSize, rate);
   ippiPyramidInitAlloc (&pPyr2, numLevel, roiSize, rate);
   {
     IppiPyramidDownState_8u_C1R **pState1 =
        (IppiPyramidDownState_8u_C1R**)&(pPyr1->pState);
     IppiPyramidDownState_8u_C1R **pState2 =
        (IppiPyramidDownState_8u_C1R**)&(pPyr2->pState);
     Ipp8u **pImg1 = pPyr1->pImage;
     Ipp8u **pImg2 = pPyr2->pImage;
     int *pStep1 = pPyr1->pStep;
     int *pStep2 = pPyr2->pStep;
```

Example 14-5 Pyramid Building and Optical Flow Calculation (continued)

```
      IppiSize *pRoi1 = pPyr1->pRoi;
      IppiSize *pRoi2 = pPyr2->pRoi;
      IppHintAlgorithm hint=ippAlgHintFast;
      int i,level = pPyr1->level;

      ippiPyramidLayerDownInitAlloc_8u_C1R(pState1,roiSize,rate,pKernel,kerSize,
         IPPI_INTER_LINEAR);
      ippiPyramidLayerDownInitAlloc_8u_C1R(pState2,roiSize,rate,pKernel,kerSize,
         IPPI_INTER_LINEAR);

      pImg1[0]  = (Ipp8u*)prevFrame;
      pImg2[0]  = (Ipp8u*)nextFrame;
      pStep1[0] = prevStep;
      pStep2[0] = nextStep;
      pRoi1[0]  = pRoi2[0] = roiSize;
      for (i=1; i<=level; i++) {
         pPyr1->pImage[i] = ippiMalloc_8u_C1(pRoi1[i].width,pRoi1[i].height,
            pStep1+i);
         pPyr2->pImage[i] = ippiMalloc_8u_C1(pRoi2[i].width,pRoi2[i].height,
            pStep2+i);
         ippiPyramidLayerDown_8u_C1R(pImg1[i-1],pStep1[i-1],pRoi1[i-1],
                 pImg1[i],pStep1[i],pRoi1[i],*pState1);
         ippiPyramidLayerDown_8u_C1R(pImg2[i-1],pStep2[i-1],pRoi2[i-1],
                 pImg2[i],pStep2[i],pRoi2[i],*pState2);
      }

      ippiOpticalFlowPyrLKInitAlloc_8u_C1R (&pOF,roiSize,winSize,hint);
      ippiOpticalFlowPyrLK_8u_C1R (pPyr1,pPyr2,prevPt,nextPt,pStatus,pError,
          numFeat,winSize,numLevel,numIter,threshold,pOF);

      ippiOpticalFlowPyrLKFree_8u_C1R(pOF);
      for (i=level; i>0; i--) {
         if (pImg2[i]) ippiFree(pImg2[i]);
         if (pImg1[i]) ippiFree(pImg1[i]);
      }
      ippiPyramidLayerDownFree_8u_C1R(*pState1);
      ippiPyramidLayerDownFree_8u_C1R(*pState2);
   }
   ippiPyramidFree (pPyr2);
   ippiPyramidFree (pPyr1);
}
```

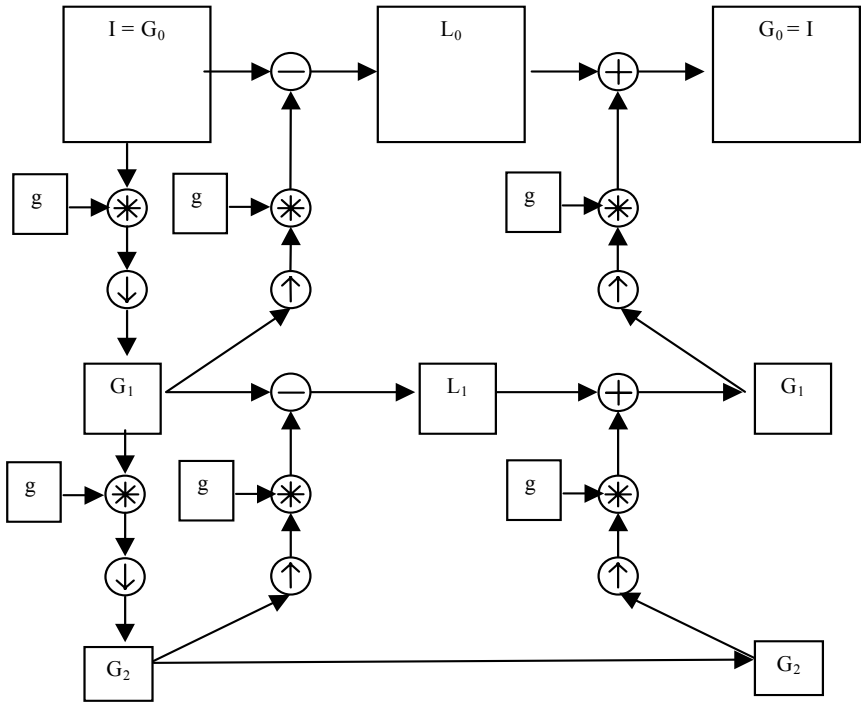
Pyramids Functions

The functions described in this section generate and reconstruct Gaussian and Laplacian image pyramids.

Figure 14-8 shows the basics of creating Gaussian or Laplacian pyramids. The original image G_0 is convolved with a Gaussian, then down-sampled to get the reduced image G_1. This process can be continued as far as desired or until the image size is one pixel.

The Laplacian pyramid can be built from a Gaussian pyramid as follows: the Laplacian level k, that is, L_k can be built by up-sampling the lower level image G_{k+1}. Convolving the image with a Gaussian kernel g interpolates the pixels "missing" after up-sampling. The resulting image is subtracted from the Gaussian smoothed image G_k. To rebuild the original image, the process is reversed as Figure 14-8 shows.

Figure 14-8Three-level Gaussian and Laplacian Pyramid



The Gaussian image pyramid on the left is used to create the Laplacian pyramid in the center, which is used to reconstruct the Gaussian pyramid and original image on the right. In Figure 14-8, I is the original image, G_k is the Gaussian image, L_k is the Laplacian image, subscripts k denote level of the pyramid, g is a Gaussian kernel used to convolve the image *before* down-sampling or *after* up-sampling.

The Intel IPP implement only Gaussian pyramids. To build Laplacian pyramids, the following general rule for Intel IPP pyramids functions is to be followed:

$$L_i = G_i - \mathrm{PyrUp}(\mathrm{PyrDown}(G_i)),$$

where L_i is an image on the i-th level in the Laplacian pyramid, G_i is an image on the i-th level in the Gaussian pyramid.

PyrDownGetBufSize

Computes the size of temporary buffer for the function `ippiPyrDown`.

Syntax

```
IppStatus ippiPyrDownGetBufSize_Gauss5x5(int roiWidth, IppDataType
    dataType, int channels, int* pBufSize);
```

Parameters

roiWidth	Width of the source image ROI in pixels.
dataType	Data type of the source image.
channels	Number of channels.
pBufSize	Pointer to the computed size of the temporary buffer.

Description

The function `ippiPyrDownGetBufSize` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function `ippiPyrDown`. The buffer of the calculated size can be used to process smaller images.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtr`	Indicates an error condition if the *pBufSize* pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if the value of *roiWidth* is zero or negative.
ippStsNumChannelsErr	Indicates an error condition if *channels* is not 1 or 3.
ippStsDataTypeErr	Indicates an error condition if *dataType* is not Ipp8u, Ipp8s, or Ipp32f.

PyrUpGetBufSize

Calculates size of temporary buffer for the function ippiPyrUp.

Syntax

```
IppStatus ippiPyrUpGetBufSize_Gauss5x5(int roiWidth, IppDataType dataType,
    int channels, int* pBufSize);
```

Parameters

roiWidth	Width of the source image ROI in pixels.
dataType	Data type of the source image.
channels	Number of channels.
pBufSize	Pointer to the computed size of the temporary buffer.

Description

The function ippiPyrUpGetBufSize is declared in the ippcv.h file. This function calculates the size of the temporary buffer to be used by the function ippiPyrUp. The buffer of the calculated size can be used to process smaller images.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtr	Indicates an error condition if the *pBufSize* pointer is NULL.
ippStsSizeErr	Indicates an error condition if the value of *roiWidth* is zero or negative.
ippStsNumChannelsErr	Indicates an error condition if *channels* is not 1 or 3.

ippStsDataTypeErr	Indicates an error condition if *dataType* is not Ipp8u, Ipp8s, or Ipp32f.

PyrDown

Applies the Gaussian to image and then performs down-sampling.

Syntax

```
IppStatus ippiPyrDown_Gauss5x5_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u*
    pBuffer);
```

Supported values for *mod* :

8u_C1R	8s_C1R	32f_C1R
8u_C3R	8s_C3R	32f_C3R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source image ROI in pixels.
pBuffer	Pointer to the temporary buffer.

Description

The function ippiPyrDown is declared in the ippcv.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function applies the 5×5 Gaussian to the source image *pSrc* and then down-samples it, that is, removes odd rows and columns from the image. The size of the destination image is (*roiSize.height*+1)/2 * (*roiSize.width*+1)/2. The following Gaussian mask is used:

$$\frac{1}{16}\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix} \times \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix} = \frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\4 & 16 & 24 & 16 & 4\\6 & 24 & 36 & 24 & 6\\4 & 16 & 24 & 16 & 4\\1 & 4 & 6 & 4 & 1\end{bmatrix}$$

The function uses the temporary buffer *pBuffer* - its size should be computed using the function ippiPyrDownGetBufSize beforehand.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* is less than *roiSize.width * <pixelSize>* , or *dstStep* is less than *(roiSize.width * <pixelSize>)/2*.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.

PyrUp

Up-samples image and then applies the Gaussian.

Syntax

```
IppStatus ippiPyrUp_Gauss5x5_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u_C1R	8s_C1R	32f_C1R
8u_C3R	8s_C3R	32f_C3R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source image ROI in pixels.
pBuffer	Pointer to the temporary buffer.

Description

The function `ippiPyrUp` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function up-samples the source image *pSrc*, that is, inserts odd zero rows and columns, and then applies the 5×5 Gaussian multiplied by 4 to it. The following mask is used:

$$4 \times \frac{1}{16}\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix} \times \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix} = \frac{1}{64}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\4 & 16 & 24 & 16 & 4\\6 & 24 & 36 & 24 & 6\\4 & 16 & 24 & 16 & 4\\1 & 4 & 6 & 4 & 1\end{bmatrix}$$

The function uses the temporary buffer *pBuffer* - its size should be computed using the function `ippiPyrUpGetBufSize` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* is less than *roiSize.width* * *<pixelSize>*, or *dstStep* is less than *2* * *roiSize.width* * *<pixelSize>*.

`ippStsNotEvenStepErr`	Indicates an error condition if when steps for floating-point images are not divisible by 4.

Universal Pyramids

The functions described in this section operate with universal image pyramids. These pyramids use separable symmetric kernel (not only Gaussian type) and downsampling/upsampling with arbitrary factor (not only 2). The next pyramid layer can be built for an image of an arbitrary size. These pyramids are used in some computer vision algorithms, for example, in optical flow calculations.

NOTE. All universal pyramid functions use the mirrored border.

Example 14-5 shows how to build pyramids and calculate the optical flow for two images.

PyramidInitAlloc

Allocates memory and initializes a pyramid structure.

Syntax

```
IppStatus ippiPyramidInitAlloc(IppiPyramid** ppPyr, int level, IppiSize roiSize,
    Ipp32f rate);
```

Parameters

ppPyr	Pointer to the pointer to the pyramid structure.
level	Maximum number of pyramid levels.
roiSize	Size of zero level image ROI in pixels.
rate	Ratio between neighbouring levels ($1 < rate \le 10$).

Description

The function `ippiPyramidInitAlloc` is declared in the `ippcv.h` file. This function allocates memory and initializes the structure for the pyramid with `level+1` levels.This structure is used by the `ippiOpticalFlowPyrLK` function for optical flow calculations.

The `IppiPyramid` structure contains the following fields:

pImage	Pointer to the array of `(level+1)` layer images.

pStep	Pointer to the array of (level+1) image row step values.
pRoi	Pointer to the array of (level+1) layer image ROIs.
pRate	Pointer to the array of (level+1) ratios of *i*-th levels to the zero level ($rate^{-i}$).
pState	Pointer to the structure to compute the next pyramid layer.
level	Number of levels in the pyramid.

The ippiPyramidInitAlloc function fills *pRoi* and *pRate* arrays and *level* field. The value of *level* is equal to the minimum of the input value of the *level* parameter and the maximum possible number of layers of the pyramid with given *rate* and zero level size.

Other fields should be specified by the user. The pyramid layer structure *pState* should be initialized by the functions ippiPyramidLayerDownInitAlloc or ippiPyramidLayerUpInitAlloc. Pyramid layer images can be obtained in many different ways, for example using the Intel IPP functions ippiPyramidLayerDown and ippiPyramidLayerUp.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if ppPyr pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsBadArgErr	Indicates an error condition if level is equal to or less than 0, or if rate is out of the range.
ippStsMemAllocErr	Memory allocation error.

PyramidFree

Frees memory allocated for the pyramid structure.

Syntax

```
IppStatus ippiPyramidFree(IppiPyramid* pPyr);
```

Parameters

pPyr	Pointer to the pyramid structure.

Description

The function `ippiPyramidFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function `ippiPyramidInitAlloc` for the pyramid structure *pPyr*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pPyr` pointer is `NULL`.

PyramidLayerDownInitAlloc

Allocates memory and initializes a structure for creating a lower pyramid layer.

Syntax

Case 1: Operation on integer data

```
IppStatus ippiPyramidLayerDownInitAlloc_<mod>(IppiPyramidDownState_<mod>**
    ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode);
```

Supported values for *mod* :

`8u_C1R`	`16u_C1R`
`8u_C3R`	`16u_C3R`

Case 2: Operation on floating point data

```
IppStatus ippiPyramidLayerDownInitAlloc_<mod>(IppiPyramidDownState_<mod>**
    ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode);
```

Supported values for *mod* :

`32f_C1R`
`32f_C3R`

Parameters

ppState	Pointer to the pointer to the pyramid layer structure.
srcRoiSize	Maximal size of source image ROI in pixels.
rate	Ratio between neighbouring levels ($1 < rate \le 10$).
pKernel	Pointer to the symmetric separable convolution kernel.

kerSize	Size of the convolution kernel, should be odd and greater than 2.
mode	Specifies the type of interpolation for image downsampling; should be set to IPPI_INTER_LINEAR to perform bilinear interpolation.

Description

The function ippiPyramidLayerDownInitAlloc is declared in the ippcv.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory and initializes the structure *pState* to build a lower pyramid layer. This structure is used by the function ippiPyramidLayerDown and can be applied to process images with size not greater than *srcRoiSize*.
Generally the specified kernel *pKernel* should be symmetric. However if it is not symmetric, the function builds the symmetric kernel using its first half, and returns the warning.
For integer rates downsampling is performed by discarding rows and columns that are not multiples of the rate value. For non-integer rate bilinear interpolation is used (see appendix B "Linear Interpolation" for more information).
The symmetric separable kernel of odd size can be not Gaussian. If the sum of kernel elements is not equal to zero, then the kernel is normalized.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if srcRoiSize has a field with zero or negative value.
ippStsBadArgErr	Indicates an error condition if one of the parameters rate, kerSize, mode has wrong value.
ippStsMemAllocErr	Memory allocation error.
ippStsSymKernelExpected	Indicates a warning if the specified kernel is not symmetric.

PyramidLayerDownFree

Frees memory allocated for the lower pyramid layer structure.

Syntax

```
IppStatus ippiPyramidLayerDownFree_<mod>(IppiPyramidDownState_<mod>* pState);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

Parameters

pState	Pointer to the pyramid layer structure .

Description

The function `ippiPyramidLayerDownFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function `ippiPyramidLayerDownFree` for the lower pyramid layer structure *pState*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pState` is `NULL`.

PyramidLayerUpInitAlloc

Allocates memory and initializes a structure for creating an upper pyramid layer.

Syntax

Case 1: Operation on integer data

```
IppStatus ippiPyramidLayerUpInitAlloc_<mod>(IppiPyramidUpState_<mod>** ppState,
    IppiSize dstRoiSize,  Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode);
```

Supported values for *mod* :

```
8u_C1R          16u_C1R
8u_C3R          16u_C3R
```

Case 2: Operation on floating point data

```
IppStatus ippiPyramidLayerUpInitAlloc_<mod>(IppiPyramidUpState_<mod>** ppState,
    IppiSize dstRoiSize, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode);
```

Supported values for *mod* :

```
32f_C1R
32f_C3R
```

Parameters

pState	Pointer to the pointer to the pyramid layer structure.
dstRoiSize	Maximal size of the destination image ROI in pixels.
rate	Ratio between neighbouring levels ($1 < rate \le 10$).
pKernel	Pointer to the symmetric separable convolution kernel.
kerSize	Size of the convolution kernel, should be odd and greater than 2.
mode	Specifies the type of interpolation for image upsampling; should be set to `IPPI_INTER_LINEAR` to perform bilinear interpolation.

Description

The function `ippiPyramidLayerUpInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory and initializes the structure *pState* to build an upper pyramid layer. This structure is used by the function `ippiPyramidLayerUp` and can be applied to process images with size not greater than *dstRoiSize*.
Generally the specified kernel *pKernel* should be symmetric. However if it is not symmetric, the function builds the symmetric kernel using its first half, and returns the warning.
For integer rates upsampling is performed by inserting zero rows and columns that are not multiples of the rate value. For non-integer rate bilinear interpolation is used to calculate kernel coefficients for pixels with non-integer coordinates.
The symmetric separable kernel of odd size can be not Gaussian. If the sum of kernel elements is not equal to zero, then the kernel is normalized.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if dstRoiSize has a field with zero or negative value.
ippStsBadArgErr	Indicates an error condition if one of the parameters rate, kerSize, mode has wrong value, or the sum of the kernel elements is not positive.
ippStsMemAllocErr	Memory allocation error.
ippStsSymKernelExpected	Indicates a warning if the specified kernel is not symmetric.

PyramidLayerUpFree

Frees memory allocated for the upper pyramid layer structure.

Syntax

```
IppStatus ippiPyramidLayerUpFree_<mod>(IppiPyramidUpState_<mod>* pState);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

Parameters

pState	Pointer to the pyramid layer structure.

Description

The function ippiPyramidLayerUpFree is declared in the ippcv.h file. This function frees memory allocated by the function ippiPyramidLayerUpInitAlloc for the upper pyramid layer structure *pState*.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pState is NULL.

GetPyramidDownROI

Computes the size of the lower pyramid layer.

Syntax

```
IppStatus ippiGetPyramidDownROI(IppiSize srcRoiSize, IppiSize* pDstRoiSize,
    Ipp32f rate);
```

Parameters

srcRoiSize	Size of the source pyramid layer ROI in pixels.
pDstRoiSize	Pointer to the size of the destination (lower) pyramid layer ROI in pixels.
rate	Ratio between source and destination layers ($1 < rate \le 10$).

Description

The function `ippiGetPyramidDownROI` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the lower pyramid layer *pDstRoiSize* for a source layer of a given size *srcRoiSize* and specified size ratio *rate* between them in accordance with the following formulas:

$$pDstRoiSize.width = max(1, min(\lceil srcRoiSize.width / rate \rceil, srcRoiSize.width - 1))$$

$$pDstRoiSize.height = max(1, min(\lceil srcRoiSize.height / rate \rceil, srcRoiSize.height - 1))$$

NOTE. Since for the non-integer *rate* results depend on the computational precision, it is strongly recommended to use this function in computations.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pDstRoiSize* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcRoiSize* has a field with zero or negative value.

ippStsBadArgErr	Indicates an error condition if *rate* is out of the range.

GetPyramidUpROI

Computes the size of the upper pyramid layer.

Syntax

```
IppStatus ippiGetPyramidUpROI(IppiSize srcRoiSize, IppiSize* pDstRoiSizeMin,
    IppiSize* pDstRoiSizeMax, Ipp32f rate);
```

Parameters

srcRoiSize	Size of the source pyramid layer ROI in pixels.
pDstRoiSizeMin	Pointer to the minimal size of the destination (upper) pyramid layer ROI in pixels.
pDstRoiSizeMax	Pointer to the maximal size of the destination (upper) pyramid layer ROI in pixels.
rate	Ratio between source and destination layers ($1 < rate \le 10$).

Description

The function ippiGetPyramidUpROI is declared in the ippcv.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function computes possible sizes of the upper pyramid layer *pDstRoiSizeMin* and *pDstRoiSizeMax* for a source layer of a given size *srcRoiSize* and specified size ratio *rate* between them in accordance with the following formulas:

maximum size *pDstRoiSizeMax*

$$pDstRoiMax.width = max(srcRoiSize.width + 1, \lfloor srcRoiSize.width \cdot rate \rfloor)$$
$$pDstRoiMax.height = max(srcRoiSize.height + 1, \lfloor srcRoiSize.height \cdot rate \rfloor)$$

minimum size *pDstRoiSizeMin*

if the width and height of the source layer ROI is greater than 1:

$$pDstRoiMin.width = max(srcRoiSize.width + 1, \lfloor (srcRoiSize.width - 1) \cdot rate \rfloor)$$
$$pDstRoiMin.height = max(srcRoiSize.height + 1, \lfloor (srcRoiSize.height - 1) \cdot rate \rfloor)$$

if the width and height of the source layer ROI is equal to1:

pDstRoiMin.width = 1

pDstRoiMin.height = 1

NOTE. Since for the non-integer *rate* results depend on the computational precision, it is strongly recommended to use this function in computations.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if srcRoiSize has a field with zero or negative value.
ippStsBadArgErr	Indicates an error condition if *rate* is out of the range.

PyramidLayerDown

Creates a lower pyramid layer.

Syntax

```
IppStatus ippiPyramidLayerDown_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize
    dstRoiSize, IppiPyramidDownState_<mod>* pState);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

srcRoiSize	Size of the source image ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination image ROI in pixels.
pState	Pointer to the pyramid layer structure.

Description

The function `ippiPyramidLayerDown` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates a lower pyramid layer *pDst* from the source image *pSrc*, that is, it applies the convolution kernel to the source image using the mirror border and then performs downsampling. Before calling `ippiPyramidLayerDown`, the pyramid layer structure *pState* should be initialized by calling the function `ippiPyramidLayerDownInitAlloc` and the kernel and the downsampling ratio should be specified beforehand.
The function can process images with *srcRoiSize* not greater than *srcRoiSize* parameter specified in the function `ippiPyramidLayerDownInitAlloc`.

NOTE. This function uses the mirrored border.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* is less than *srcRoiSize.width * <pixelSize>* , or *dstStep* is less than *dstRoiSize.width * <pixelSize>*.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

ippStsBadArgErr	Indicates an error condition if *pState->rate* has wrong value.

PyramidLayerUp

Creates an upper pyramid layer.

Syntax

```
IppStatus ippiPyramidLayerUp_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidUpState_<mod>* pState);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of source image ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of destination image ROI in pixels.
pState	The pointer to the pyramid layer structure.

Description

The function ippiPyramidLayerUp is declared in the ippcv.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates an upper pyramid layer *pDst* from the source image *pSrc*, that is, it performs upsampling of the source image and then applies the convolution kernel using the mirror border. Before calling ippiPyramidupUp function, the pyramid layer structure *pState* should be initialized by calling the function ippiPyramidLayerUpInitAlloc and

the kernel and the upsampling ratio should be specified beforehand.
The function can process images with *srcRoiSize* not greater than *srcRoiSize* parameter specified in the function ippiPyramidLayerUpInitAlloc.

NOTE. This function uses the mirrored border.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* is less than *srcRoiSize.width* * *<pixelSize>* , or *dstStep* is less than *dstRoiSize.width* * *<pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error condition if *pState->rate* has wrong value.

Example of Using General Pyramid Functions

The following Example 14-6 shows how different general pyramids functions can be used to create the Gaussian and Laplacian pyramids:

Example 14-6 Building Gaussian and Laplacian Pyramids

```
void UsePyramids(Ipp32f *pSrc, IppiSize srcRoi, int srcStep, Ipp32f *pkernel,
                    int kerSize) {
    IppiPyramid *gPyr;   // pointer to Gaussian pyramid structure
    IppiPyramid *lPyr;   // pointer to Laplacian pyramid structure

//  allocate pyramid structures
    ippiPyramidInitAlloc (&gPyr, 1000, srcRoi, rate);
    ippiPyramidInitAlloc (&lPyr, 1000, srcRoi, rate);
{
       IppiPyramidDownState_32f_C1R **gState =
                      (IppiPyramidDownState_32f_C1R*)&(gPyr->pState);
       IppiPyramidUpState_32f_C1R   **lState =
                      (IppiPyramidUpState_32f_C1R*)  &(lPyr->pState);
       Ipp32f **gImage = (Ipp32f**)(gPyr->pImage);
       Ipp32f **lImage = (Ipp32f**)(lPyr->pImage);
       IppiSize *pRoi   = dPyr->pRoi;
       int *gStep = gPyr->pStep,
       int *lStep = lPyr->pStep;
       int level = gPyr->level;
       Ipp32f *ptr;
       int step;

       //  allocate structures to calculate pyramid layers
       ippiPyramidLayerDownInitAlloc_32f_C1R(gState, srcRoi, rate, pKernel,
                                                kerSize, mode);
       ippiPyramidLayerUpInitAlloc_32f_C1R  (lState, srcRoi, rate, pKernel,
                                                kerSize, mode);
//     build Gaussian pyramid with level+1 layers
       gImage[0] = pSrc;
       gStep[0]  = srcStep;
       for (i=1; i<=level; i++) {
          gImage[i] = ippiMalloc_32f_C1(pRoi[i].width,pRoi[i].height,gStep+i);
          ippiPyramidLayerDown_32f_C1R(gImage[i-1], gStep[i-1], pRoi[i-1],
                                    gImage[i],   gStep[i],   pRoi[i], *gState);
       }
```

Example 14-6 Building Gaussian and Laplacian Pyramids (continued)

```
 //     build Laplacian pyramid with level layers
       ptr = ippiMalloc_32f_C1(srcRoi.width,srcRoi.height,&step);
       for (i=level-1; i>=0; i--) {
          lImage[i] = ippiMalloc_32f_C1(pRoi[i].width,pRoi[i].height,gStep+i);
          ippiPyramidLayerUp_32f_C1R(gImage[i+1], gStep[i+1], pRoi[i+1],
                                     ptr, step, pRoi[i], *lState);
          ippiSub_32f_C1R(ptr, step, gImage[i], gStep[i],
                          lImage[i], lStep[i], pRoi[i]);
       }
       ippiFree(ptr);

       ippiPyramidLayerDownFree_32f_C1R(*gStep);
       ippiPyramidLayerUpFree_32f_C1R  (*lStep);

//     use Gaussian and Laplacian pyramids

//     free allocated images
       for (i=1; i<=level; i++) {
          ippiFree(gImage[i]);
          ippiFree(lImage[i-1]);
       }
    }
//  free pyramid structures
    ippiPyramidFree (gPyr);
    ippiPyramidFree (lPyr);
}
```

Image Inpainting

The functions described in this section allows to restore the unknown image portions. They could be used to repair damaged parts of images and to remove some objects from images. Fast direct methods of inpainting that allow for run-time correcting of video frames are supported.

InpaintInitAlloc

Allocates memory and initializes a structure for image inpainting.

Syntax

```
IppStatus ippiInpaintInitAlloc_8u_C1R(IppiInpaintState_8u_C1R** ppState,
    const Ipp32f* pDist, int distStep, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp32f radius, IppiInpaintFlag flags);
IppStatus ippiInpaintInitAlloc_8u_C3R(IppiInpaintState_8u_C3R** ppState,
    const Ipp32f* pDist, int distStep, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp32f radius, IppiInpaintFlag flags);
```

Parameters

ppState	Double pointer to the state structure for the image inpaiting.
pDist	Pointer to the ROI of the image of distances.
distStep	Distance in bytes between starts of consecutive lines in the image of distances.
pMsk	Pointer to the mask image ROI.
mskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the image ROI in pixels.
radius	Radius of the neghborhood used for inpainting.
flags	Specifies algorithm for image inpainting; following values are possible: `IPP_INPAINT_TELEA` Telea algorithm; `IPP_INPAINT_NS` Navier-Stokes equation.

Description

The function `ippiInpaintInitAlloc` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory and initializes the structure *ppState* for direct methods of image inprinting. This structure is used by the function `ippiInpaint` and can be applied to process images with ROI of the same size *roiSize*. Zero pixels of the mask image *pMsk* correspond to known image pixels, non-zero pixels - correspond to unknown image pixels that should be restored. The distance image *pDist* specifies the order of pixel inpainting. Values of unknown pixels are restored in ascending order in dependence on their distances. The parameter *radius* specifies the radius of the circular neighborhood that affects the restoration of the central pixel.
The parameter specifies the methods of direct inprinting. Two methods are supported: Telea algorithm [Telea04], and Navier-Stokes equation [Bert01]

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *distStep* or *mskStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBadArgErr`	Indicates an error condition if *radius* is less than 1, or *flags* has an illegal value.
`ippStsMemAllocErr`	Memory allocation error.

InpaintFree

Frees memory allocated for the structure for image inpainting.

Syntax

```
IppStatus ippiInpaintFree_8u_C1R(IppiInpaintState_8u_C1R* pState);
IppStatus ippiInpaintFree_8u_C3R(IppiInpaintState_8u_C3R* pState);
```

Parameters

pState	Pointer to the state structure for the image inpaiting.

Description

The function `ippiInpaintFree` is declared in the `ippcv.h` This function frees memory allocated by the function `ippiInpaintInitAlloc` for the inpainting structure *pState*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the *pState* pointer is `NULL`.

Inpaint

Restores unknown image pixels.

Syntax

```
IppStatus ippiInpaint_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiInpaintState_8u_C1R* pState);
IppStatus ippiInpaint_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, IppiInpaintState_8u_C3R* pState);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the image ROI in pixels.
pState	The pointer to the inpainting structure.

Description

The function `ippiInpaint` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function reconstructs damaged part of the image, or removes a selected object (see Figure 14-9). The image part to restore is defined by the mask that is created when the inpainting structure *pState* is initialized by the function `ippiInpaintInitAlloc`. Different distant transforms can be used, but the Fast Marching Method (`ippiFastMarching`) provides the best results. The order of pixel restoration is defined by the distance through the initialization the inpainting structure *pState* by the function `ippiInpaintInitAlloc`. Pixels are restored in according to the growing of their distance value. When a pixel is inpainted, it is treated as the known one.

Two algorithms of direct inpainting are supported (controlled by the parameter *flags* of the function `ippiInpaintInitAlloc`):

- image restoration of the unknown pixel by the weighted sum of approximations by known pixels in the neighborhood (*flags* = `IPP_INPAINT_TELEA`) [Telea04],
- image restoration based on the Navier-Stokes equations (*flags* = `IPP_INPAINT_NS`) [Bert01].

The inpainting structure *pState* must be initialized by the function `ippiInpaintInitAlloc`, and may be used to perform restoration of several different images of the same size *roiSize*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value, or if differs from the corresponding parameter that is specified when the inpainting structure is initialized by the `ippiInpaintInitAlloc`.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* is less than *roiSize.width* * <*pixelSize*>.

Figure 14-9 Image Inprinting

initial image restored image

Image Segmentation

THis section describes the functions that perform image segmentation using different techniques. These functions allow to extract parts of the image that can be associated with objects of the real world. Watershed and gradient segmentation are region-based methods to split image into the distinctive areas.

Background/foreground segmentation allows for distinguishing between moving objects and stable areas of the background.

LabelMarkersGetBufferSize

Computes the size of the working buffer for the marker labeling.

Syntax

```
IppStatus ippiLabelMarkersGetBufferSize_8u_C1R(IppiSize roiSize, int*
    pBufferSize);
IppStatus ippiLabelMarkersGetBufferSize_16u_C1R(IppiSize roiSize, int*
    pBufferSize);
```

Parameters

roiSize	Size of the source image ROI in pixels.
pBufferSize	Pointer to the computed size of the working buffer.

Description

The function `ippiLabelMarkersGetBufferSize` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the `ippiLabelMarkers` function. The buffer with the length *pBufferSize*[0] can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter *roiSize*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer *pBufferSize* is NULL.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

LabelMarkers

Labels markers in image with different values.

Syntax

```
IppStatus ippiLabelMarkers_8u_C1IR(Ipp8u* pMarker, int markerStep, IppiSize
    roiSize, int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u*
    pBuffer);
IppStatus ippiLabelMarkers_16u_C1IR(Ipp16u* pMarker, int markerStep,
    IppiSize roiSize, int minLabel, int maxLabel, IppiNorm norm, int* pNumber,
    Ipp8u* pBuffer);
```

Parameters

pMarker	Pointer to the source and destination image ROI.
markerStep	Distance in bytes between starts of consecutive lines in the source and destination image.
minLabel	Minimal value of the marker label (0 < *minLabel* ≤*maxLabel*).
maxLabel	Maximal value of the marker label (*minLabel* ≤*maxLabel* < 255 for 8-bit markers, and *minLabel* ≤*maxLabel* < 65535 for 16-bit markers).
roiSize	Size of the source and destination image ROI in pixels.
norm	Specifies type of the norm to form the mask for marker propagation: ippiNormInf — Infinity norm (8-connectivity); ippiNormL1 — L1 norm (4-connectivity).
pNumber	Pointer to the number of markers.
pBuffer	Pointer to the working buffer.

Description

The function ippiLabelMarkers is declared in the ippcv.h It operates with ROI (see Regions of Interest in Intel IPP).

This function labels markers in the image *pSrcDst* with different integer values. Each connected set of non-zero image pixels is treated as the separate marker. 4- or 8-connectivity can be used depending on the norm type. All pixels belonging to the same marker are set to the same value from the interval [*minLabel*, *maxLabel*]. Two markers can be labeled with the same value if the number of connected components exceeds *minLabel*-*maxLabel*+1. The image with labeled markers can be used as the seed image for segmentation by functions ippiSegmentWatershed or ippiSegmentGradient functions.

The function requires the working buffer *pBuffer* whose size should be computed by the function ippiLabelMarkersGetBufferSize beforehand.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *markerStep* is less than *roiSize.width* * <*pixelSize*>.
ippStsNotEvenStepErr	Indicates an error condition if *markerStep* for 16-bit images is not divisible by 2.
ippStsBadArgErr	Indicates an error condition if one of the *minLabel*, *maxLabel*, and *norm* has an illegal value.

SegmentWatershedGetBufferSize

Computes the size of the working buffer for the watershed segmentation.

Syntax

```
IppStatus ippiSegmentWatershedGetBufferSize_8u_C1R(IppiSize roiSize, int*
    pBufferSize);
IppStatus ippiSegmentWatershedGetBufferSize_8u16u_C1R(IppiSize roiSize,
    int* pBufferSize);
```

Parameters

roiSize	Size of the source image ROI in pixels.
pBufferSize	Pointer to the computed size of the working buffer.

Description

The function `ippiSegmentWatershedGetBufferSize` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the `ippiSegmentWatershed` function. The buffer with the length *pBufferSize*[0] can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter *roiSize*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer *pBufferSize* is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

SegmentWatershed

Performs watershed image segmentation using markers.

Syntax

```
IppStatus ippiSegmentWatershed_8u_C1IR(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pMarker, int markerStep, IppiSize roiSize, IppiNorm norm, int flags,
    Ipp8u* pBuffer);
IppStatus ippiSegmentWatershed_8u16u_C1IR(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pMarker, int markerStep, IppiSize roiSize, IppiNorm norm, int
    flags, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pMarker	Pointer to the ROI of the source and destination image of markers.
markerStep	Distance in bytes between starts of consecutive lines in the image of markers.
roiSize	Size of the source and destination image ROI in pixels.
norm	Specifies type of the norm to form the mask for marker propagation: `ippiNormInf` Infinity norm (8-connectivity, 3x3 rectangular mask); `ippiNormL1` L1 norm (4-connectivity, 3x3 cross mask); `ippiNormL2` approximation of L2 norm (8-connectivity, 3x3 mask) [Borge86]; `ippiNormFM` fast marching distance [Telea04];
flags	Specifies the algorithm of segmentation; it is the logical sum of two values: mandatory value, one of the following values: `IPP_SEGMENT_QUEUE` Priority queue is used to define the order of pixel processing; `IPP_SEGMENT_DISTANCE` distance transform is used for the segmentation; and optional value, one of the following values: `IPP_SEGMENT_BORDER_4` pixels of the 4-connectivity border between image segments are marked with value `IPP_MAX_8U` (255). `IPP_SEGMENT_BORDER_8` pixels of the 8-connectivity border between image segments are marked with value `IPP_MAX_8U` (255).
pBuffer	Pointer to the working buffer.

Description

The function `ippiSegmentWatershed` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function performs image watershed segmentation with markers. Non-zero pixels of *pMarker* image belong to water source markers. Marker values propagate through the whole image according to the watershed algorithm. Image segments are formed by groups of connected *pMarker* pixels with the same value. The parameter *norm* controls marker propagation connectivity. Watershed segmentation is preferable for images with local minimums, for example, gradient images. Image markers generally correspond to these local minimums and can be created, for example, manually or using morphological reconstruction.

The parameter *flags* specifies how watershed segmentation is performed. This parameter is a logical sum of two values.

Required value specifies algorithm of segmentation. Possible values:

IPP_SEGMENT_QUEUE specifies the classic watershed segmentation scheme with priority queue [Vicent91].

IPP_SEGMENT_DISTANCE specifies the watershed segmentation by calculating the topographic distance for each pixel [Lotufo00], [Meyer94].

Optional additional values of the *flags*: IPP_SEGMENT_BORDER_4 and IPP_SEGMENT_BORDER_8 specifies the border of the segments. All pixels adjacent to the differently marked pixels are considered as border pixels, and their values are set to IPP_MAX_8U (255) for 8-bit markers, or IPP_MAX_16U (65535) for 16-bit markers. Note in this case value IPP_MAX_8U (IPP_MAX_16U) should not be used to mark segments.
If these optional values are not specified, segments are formed without borders.

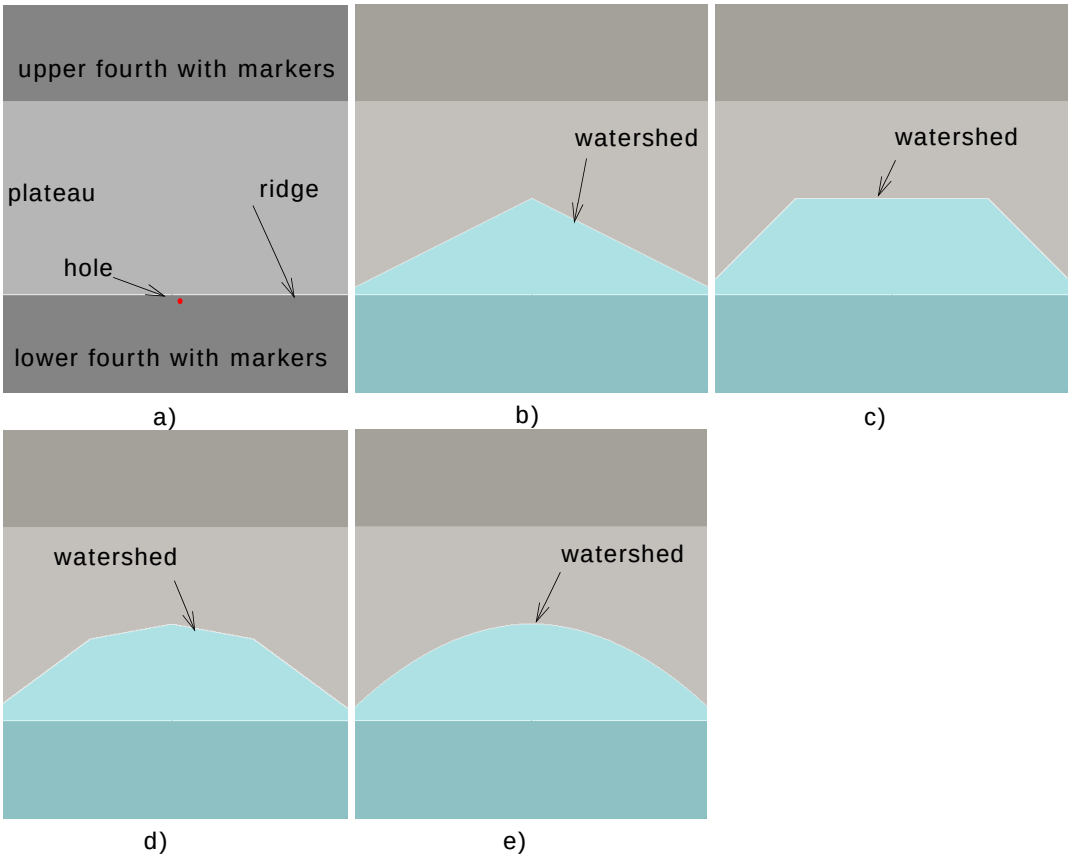
The function requires the working buffer *pBuffer* whose size should be computed by the function ippiSegmentWatershedGetBufferSize beforehand.

Figure 14-10 shows the plateau filling through the watershed segmentation with different values of the *norm* parameters. Initial image (a) has the labeled with markers upper and lower fourths with low pixel value, the central plateau between them, the ridge between the plateau and the lower forth with one pixel hole in the center of it. The following pictures are segmentation results: b) - for L1 norm (block distance), c)- Linf norm (chessboard distance), d) - approximate L2 (Euclidian) norm [Borge86], e) Fast Marching distance.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the *srcStep* or *markerStep* is less than *roiSize.width* * <*pixelSize*>.
ippNotEvenStsStepErr	Indicates an error condition if one of the *srcStep* or *markerStep* for 16-bit integer images is not divisible by 2.
ippStsBadArgErr	Indicates an error condition if *norm* has an illegal value.

Figure 14-10 Watershed Segmentation with Different Norms



SegmentGradientGetBufferSize

Computes the size of the working buffer for the gradient segmentation.

Syntax

```
IppStatus ippiSegmentGradientGetBufferSize_8u_C1R(IppiSize roiSize, int*
    pBufferSize);
IppStatus ippiSegmentGradientGetBufferSize_8u_C3R(IppiSize roiSize, int*
    pBufferSize);
```

Parameters

roiSize	Size of the source image ROI in pixels.
pBufferSize	Pointer to the computed size of the working buffer.

Description

The function `ippiSegmentGradientGetBufferSize` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the `ippiSegmentGradient` function. The buffer with the length *pBufferSize*[0] can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter *roiSize*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer *pBufferSize* is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

SegmentGradient

Performs image segmentation by region growing to the least gradient direction.

Syntax

```
IppStatus ippiSegmentGradient_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pMarker, int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u*
    pBuffer);
IppStatus ippiSegmentGradient_8u_C3IR(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pMarker, int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u*
    pBuffer);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMarker	Pointer to the ROI of the source and destination image of markers.
markerStep	Distance in bytes between starts of consecutive lines in the image of markers.
roiSize	Size of the source and destination image ROI in pixels.
norm	Specifies type of the norm to form the mask for marker propagation: `ippiNormInf` Infinity norm (8-connectivity, 3x3 rectangular mask); `ippiNormL1` L1 norm (4-connectivity, 3x3 cross mask);
flags	optional flag: `IPP_SEGMENT_BORDER_4` pixels of the 4-connectivity border between image segments are marked with value `(IPP_MAX_8U)-1` (254). `IPP_SEGMENT_BORDER_8` pixels of the 8-connectivity border between image segments are marked with value `(IPP_MAX_8U)-1` (254).
pBuffer	Pointer to the working buffer.

Description

The function `ippiSegmentGradient` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function performs image segmentation by region growing with markers. Non-zero pixels of *pMarker* image belong to initial image regions. Marker values propagate through the whole image in the direction of the least value of the absolute value of the image gradient. For 3-channel image the gradient is calculated as the maximum of channel gradients. Image segments are formed by groups of connected *pMarker* pixels with the same value. The parameter *norm* controls marker propagation connectivity. Gradient segmentation is generally done for an image without explicit calculation of the image gradient. [Meyer92]

If `IPP_SEGMENT_BORDER` flag is defined, then the pixels adjacent to differently marked pixels are assumed to be border pixels and are set to a special value (254). This value must not be used to mark segments in this case.

Another special value (255) is used inside the function and can not be used to mark segment inany case.

The function requires the working buffer *pBuffer* whose size should be computed by the function `ippiSegmentGradientGetBufferSize` beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of the *srcStep* or *markerStep* is less than *roiSize.width* * <*pixelSize*>.
`ippStsBadArgErr`	Indicates an error condition if *norm* has an illegal value.

BoundSegments

Marks pixels belonging to segment boundaries.

Syntax

```
IppStatus ippiBoundSegments_8u_C1IR(Ipp8u* pMarker, int markerStep,
    IppiSize roiSize, Ipp8u val, IppiNorm norm);
IppStatus ippiBoundSegments_16u_C1IR(Ipp16u* pMarker, int markerStep,
    IppiSize roiSize, Ipp16u val, IppiNorm norm);
```

Parameters

pMarker	Pointer to the ROI of the source and destination image of markers.
markerStep	Distance in bytes between starts of consecutive lines in the image of markers.
roiSize	Size of the source and destination image ROI in pixels.
val	Value of the boundary pixel.
norm	Specifies type of the norm gor pixel neighborhood: `ippiNormInf` Infinity norm (8-connectivity); `ippiNormL1` L1 norm (4-connectivity).

Description

The function `ippiBoundSegments` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function detects segment boundaries in the *pMarker* image and sets border pixels to the value *val*. A segment is the set of connected pixels of the *pMarker* image with the same value not equal to *val*. After boundaries are marked, the *pMarker* image does not contain any pair of adjacent in *norm* pixels with the same value that not equal to *val*.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if *markerStep* is less than *roiSize.width* * <*pixelSize*>.
ippNotEvenStsStepErr	Indicates an error condition if *markerStep* for 16-bit integer images is not divisible by 2.
ippStsBadArgErr	Indicates an error condition if *norm* has an illegal value.

ForegroundHistogramInitAlloc

Allocates memory and initializes a state structure for foreground/background segmentation using histograms.

Syntax

```
IppStatus ippiForegroundHistogramInitAlloc_8u_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize roiSize, IppFGHistogramModel* pModel,
    IppFGHistogramState_8u_C1R** ppState)
```

```
IppStatus ippiForegroundHistogramInitAlloc_8u_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize roiSize, IppFGHistogramModel* pModel,
    IppFGHistogramState_8u_C3R** ppState)
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source and destination image ROI in pixels.
pModel	Pointer to the structure of the histogram statistical model.
ppState	Pointer to the pointer to the segmentation state structure.

Description

The function `ippiForegroundHistogramInitAlloc` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory, initializes the state structure for the histogram modeling of foreground and background and returns the pointer *ppState* to it. It is used by the functions `ippiForegroundHistogram` and `ippiForegroundHistogramUpdate` that detect foreground pixels and update the statistical model for the segmentation.

The histogram statistical model implements detecting both stationary and moving background objects [Li03]. The model keeps histograms of quantized pixel values and quantized velocities of pixel value changes.

The histogram statistical model is defined by following fields of *pModel* structure:

valueQuant	Number of quantization levels for pixel values (degree of 2, *valueQuant* ≥ 2).
changeQuant	Number of quantization levels for the speed of pixel value change (degree of 2, *valueQuant* ≥ 2).
valueUse	Number of most frequent pixel value levels used for classification (*valueUse* ≥ *valueAll*).
valueAll	Number of all pixel value levels monitored by the model (*valueAll* > 0).
changeUse	Number of most frequent speed levels used for classification (*changeUse* ≥ *changeAll*).
changeAll	Number of all speed levels monitored by the model (*changeAll* > 0).
updBGProb	Speed of updating of the background probability (0.5 > *updBGProb* > 0).
updBGRef	Speed of updating of the reference background image (0.5 > *updBGRef* > 0).
numFrame	Number of background frames for adjustment of shadows searching. Only for 3-channels images.
detectionRate	Speed of shadows searching. Only for 3-channels images (0 > *detectionRate* > 1).
brightnessDistortion	Lower bound of the normalized brightness distortion for shadows. Only for 3-channels images (0 > *brightnessDistortion* > 1).
shadowBG	Flag of shadows searching. If *shadowBG* is 1, then pixels belonging to colored shadows are marked as background, otherwise they may be included in foreground areas. Only for 3-channels images (0,1).

valueAll (*changeAll*) are the most frequent levels; they are monitored by the model. First of them *valueUse* (*changeUse*) are used for foreground/background classification. The probabilities to classify pixels as background and pixel values of the reference background image are updated using one pole IIR filters with parameters *updBGProb* and *updBGRef* respectively. If *shadowBG* is set to 1, then the search of colored shadows is performed for three-channel images. In this case foreground pixels, that are detected as shadow, are marked as background pixels.

The structure can be used to process image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the *srcStep* is less than *roiSize.width* * <*pixelSize*>.
ippStsBadArgErr	Indicates an error condition if *pModel* has an illegal value.

ForegroundHistogramFree

Frees memory allocated for the foreground/background segmentation structure.

Syntax

```
IppStatus ippiForegroundHistogramFree_8u_C1R(IppFGHistogramState_8u_C1R*
    pState);
IppStatus ippiForegroundHistogramFree_8u_C3R(IppFGHistogramState_8u_C3R*
    pState);
```

Parameters

pState	Pointer to the segmentation state structure.

Description

The function ippiForegroundHistogramFree is declared in the ippcv.h This function frees memory allocated by the function ippiForegroundHistogramInitAlloc for the state structure *pState* for the histogram modeling of foreground and background.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if *pState* is NULL.

ForegroundHistogram

Calculates foreground mask using histograms.

Syntax

```
IppStatus ippiForegroundHistogram_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pMask, int maskStep, IppiSize roiSize, IppFGHistogramState_8u_C1R*
    pState);
IppStatus ippiForegroundHistogram_8u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pMask, int maskStep, IppiSize roiSize, IppFGHistogramState_8u_C3R*
    pState);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the ROI of the destination foreground mask image.
maskStep	Distance in bytes between starts of consecutive lines in the foreground mask image.
roiSize	Size of the source and destination image ROI in pixels.
pState	Pointer to the segmentation state structure.

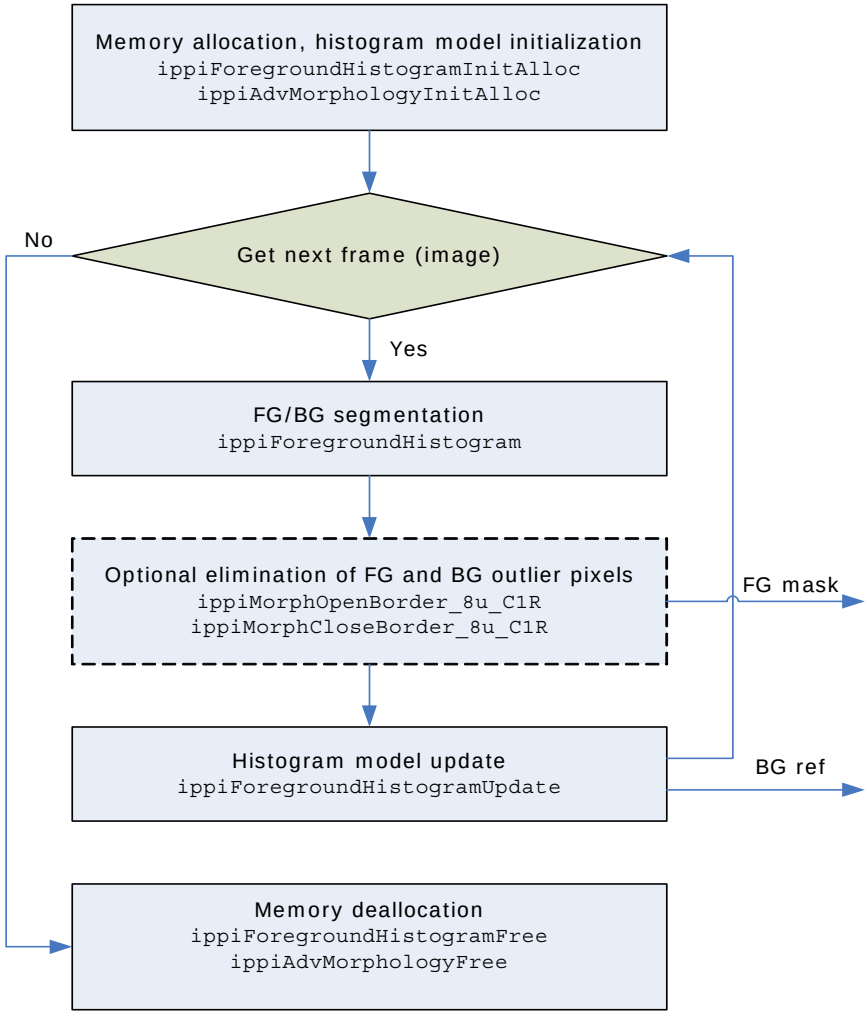
Description

The function `ippiForegroundHistogram` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This foreground/background segmentation algorithm processes the sequence of images and trains statistical models for foreground and background pixels for this sequence. The algorithm applies the model to each image, classifies its pixels as belonging to the foreground or background of the scene, and then updates the model with the image data. Figure 14-11 shows the scheme of the algorithm using the statistical histogram model for each pixel [Li03]. The function `ippiForegroundHistogram` applies the histogram statistical model to the image *pSrc,* and sets pixels of *pMask* image corresponding to the background pixels of the *pSrc* to zero, and foreground pixels - to non-zero value. Then the statistical model is updated by the function `ippiForegroundHistogramUpdate`.

The function `ippiForegroundHistogram` can process images with *roiSize* that is not greater than *roiSize* specified in the function `ippiForegroundHistogramInitAlloc`.

Figure 14-11 Foreground/Background Segmentation Using Histogram Model



Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the *srcStep* or *maskStep* is less than *roiSize.width* * <*pixelSize*>.

ForegroundHistogramUpdate

Updates histogram statistical model for foreground segmentation.

Syntax

```
IppStatus ippiForegroundHistogramUpdate_8u_C1R(const Ipp8u* pSrc, int
    srcStep, const Ipp8u* pMask, int maskStep, Ipp8u* pRef, int refStep,
    IppiSize roiSize, IppFGHistogramState_8u_C1R* pState);
IppStatus ippiForegroundHistogramUpdate_8u_C3R(const Ipp8u* pSrc, int
    srcStep, const Ipp8u* pMask, int maskStep, Ipp8u* pRef, int refStep,
    IppiSize roiSize, IppFGHistogramState_8u_C3R* pState);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the ROI of the foreground mask image.
maskStep	Distance in bytes between starts of consecutive lines in the foreground mask image.
pRef	Pointer to the ROI of the source and destination reference background mask image.
refStep	Distance in bytes between starts of consecutive lines in the reference background mask image.
roiSize	Size of the source and destination image ROI in pixels.
pState	Pointer to the segmentation state structure.

Description

The function `ippiForegroundHistogramUpdate` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function updates the histogram statistical model using *pSrc* image and foreground mask *pMask*. Between calls of the function `ippiForegroundHistogram` and `ippiForegroundHistogramUpdate` functions, the mask image *pMask* is generally processed to delete foreground and background outlier pixels. When the model is updated, it is ready to process the next image.

This function uses the reference background image *pRef* that is initialized during the first call of the function. This image is updated after each call of the function `ippiForegroundHistogramUpdate`.

The function `ippiForegroundHistogram` can process images with *roiSize* that is not greater than *roiSize* specified in the function `ippiForegroundHistogramInitAlloc`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of the *srcStep*, *maskStep*, or *refStep* is less than *roiSize.width* * <*pixelSize*>.

ForegroundGaussianInitAlloc

Allocates memory and initializes a state structure for foreground/background segmentation using Gaussian mixtures.

Syntax

```
IppStatus ippiForegroundGaussianInitAlloc_8u_C1R(const Ipp8u* pSrc, int
    srcStep, IppiSize roiSize, IppFGGaussianModel* pModel,
    IppFGGaussianState_8u_C1R** ppState);
IppStatus ippiForegroundGaussianInitAlloc_8u_C3R(const Ipp8u* pSrc, int
    srcStep, IppiSize roiSize, IppFGGaussianModel* pModel,
    IppFGGaussianState_8u_C3R** ppState);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source and destination image ROI in pixels.
pModel	Pointer to the structure of the Gaussian statistical model.
ppState	Pointer to the pointer to the segmentation state structure.

Description

The function `ippiForegroundGaussianInitAlloc` is declared in the `ippcv.h` It operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory, initializes the state structure for the foreground and background modeling by Gaussian mixtures and returns the pointer *ppState* to it. This structure is used by the function `ippiForegroundGaussian` that detects foreground pixels and updates the statistical model for the segmentation.

The Gaussian mixture statistical model is defined by following fields of *pModel* structure:

numGauss	Number of components in the Gaussian mixture for each frame (usually, 3-5).
priorBack	Minimum prior probability of the background.
updBGProb	Speed of updating of parameters of selected mixture component (0.5 > *updBGProb* > 0).
winSize	Number of last frames affecting model parameter update (100-200).
numFrame	Number of background frames for adjustment of shadows searching. Only for 3-channels images.
detectionRate	Speed of shadows searching. Only for 3-channels images (0 > *detectionRate* > 1).
brightnessDistortion	Lower bound of the normalized brightness distortion for shadows. Only for 3-channels images (0 > *brightnessDistortion* > 1).
shadowBG	Flag of shadows searching. If *shadowBG* is 1, then pixels belonging to colored shadows are marked as background, otherwise they may be included in foreground areas. Only for 3-channels images (0,1).

The Gaussian mixture statistical model models each background pixel by a Gaussian mixture with a small number of components *numGauss* [Kadew01]. Most important components with sum of weights greater than *priorBack* are used to model the background. Mixture components are updated while processing each new frame. The

learning speed of the component that matches a background pixel is defined by *updBGChange*. All mixture parameters are then updated using statistics from last *winSize* frames. If *shadowBG* is set to 1, then the search of colored shadows is performed for three-channel images. In this case foreground pixels, that are detected as shadow, are marked as background pixels.

The structure can be used to process image whose width and height are equal to or less than corresponding fields of *roiSize*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the *srcStep* is less than *roiSize.width* * <*pixelSize*>.

ForegroundGaussianFree

Frees memory allocated for the foreground/background segmentation structure.

Syntax

```
IppStatus ippiForegroundGaussianFree_8u_C1R(IppFGGaussianState_8u_C1R*
    pState);
IppStatus ippiForegroundGaussianFree_8u_C3R(IppFGGaussianState_8u_C3R*
    pState);
```

Parameters

pState	Pointer to the segmentation state structure.

Description

The function ippiForegroundGaussianFree is declared in the ippcv.h This function frees memory allocated by the function ippiForegroundGaussianInitAlloc for the state structure *pState* for the Gaussian modeling of foreground and background.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if *pState* is NULL.

ForegroundGaussian

Calculates foreground mask using Gaussians.

Syntax

```
IppStatus ippiForegroundGaussian_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pRef, int refStep, Ipp8u* pDst, int dstStep, IppiSize roiSize,
    IppFGGaussianState_8u_C1R* pState);
IppStatus ippiForegroundGaussian_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pRef, int refStep, Ipp8u* pDst, int dstStep, IppiSize roiSize,
    IppFGGaussianState_8u_C1R* pState);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pRef	Pointer to the ROI of the destination reference background mask image.
refStep	Distance in bytes between starts of consecutive lines in the reference background mask image.
pDst	Pointer to the ROI of the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI in pixels.
pState	Pointer to the segmentation state structure.

Description

The function ippiForegroundGaussian is declared in the ippcv.h It operates with ROI (see Regions of Interest in Intel IPP).

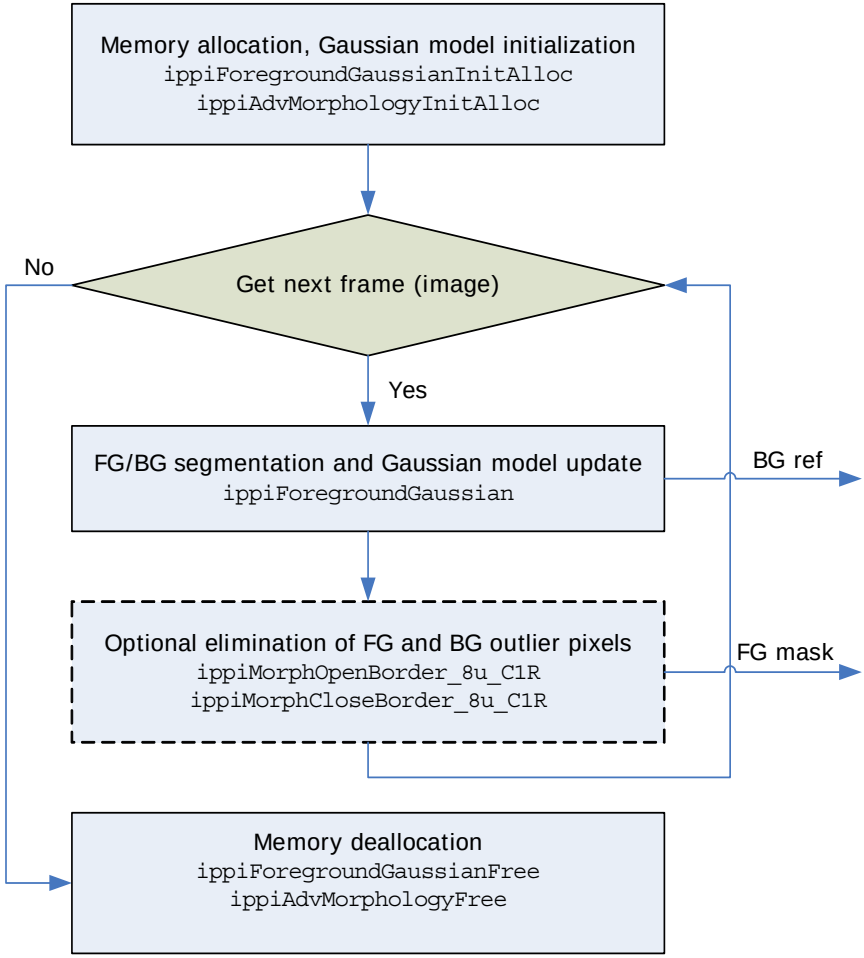
This foreground/background segmentation algorithm processes the sequence of images and trains statistical models for foreground and background pixels for this sequence. The algorithm applies the model to each image, classifies its pixels as belonging to the foreground or background of the scene, and then updates the model with the image data. Figure 14-12 shows the scheme of the algorithm using the Gaussian mixture statistical model for each pixel [Kadew01]. The function `ippiForegroundGaussian` applies the Gaussian mixture statistical model to the image *pSrc,* and sets pixels of *pMask* image corresponding to the background pixels of the *pSrc* to zero, and foreground pixels - to non-zero value. Then the statistical model is updated. The mask image *pMask* is usually processed to delete foreground and background outlier pixels. The function calculates also the reference background image *pRef*.

The function `ippiForegroundGaussian` can process images with *roiSize* that is not greater than *roiSize* specified in the function `ippiForegroundGaussianInitAlloc`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of the *srcStep*, *dstStep*, or *refStep* is less than *roiSize.width* * <*pixelSize*>.

Figure 14-12 Foreground/Background Segmentation Using the Gaussian Model

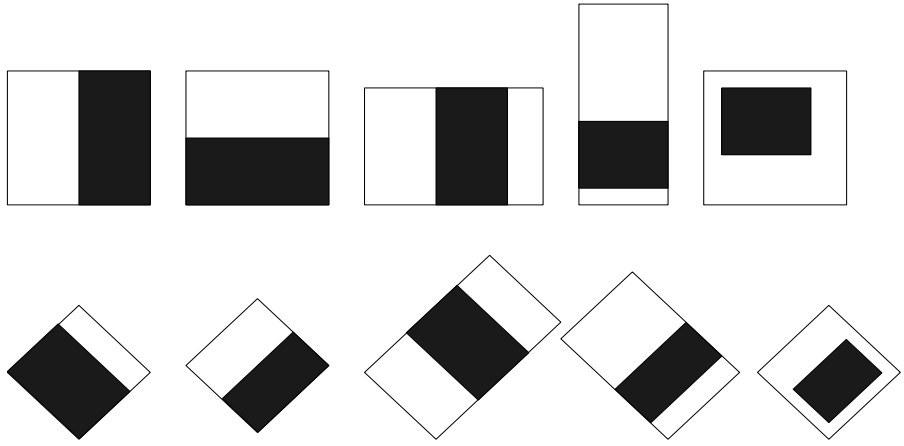


Pattern Recognition

Object Detection Using Haar-like Features

The object detector described in [Viola99] and [Lein02] is based on Haar classifiers. Each classifier uses k rectangular areas (Haar features) to make decision if the region of the image looks like the predefined image or not. Figure 14-13 shows different types of Haar features.

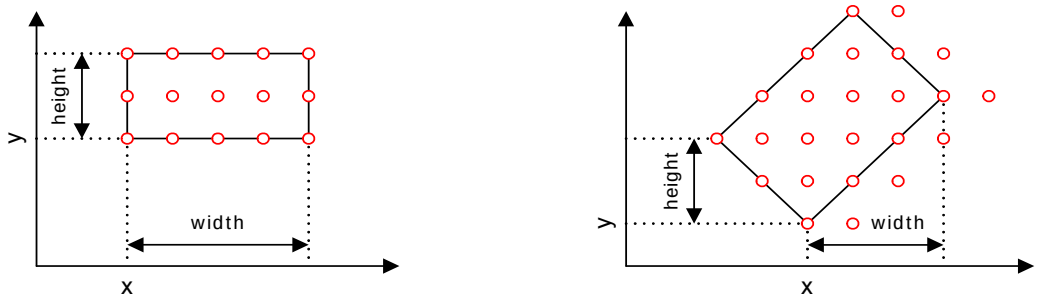
Figure 14-13 Types of Haar Features



In the Intel IPP Haar features are represented using `IppRect` structure. Figure 14-14 shows how it can be done for common and tilted features.

.

Figure 14-14 Representing Haar Features



Haar feature defined by IppiRect structure
x=2, y=3, width=5, height=3 (15 pixels)

Tilted Haar feature defined by IppiRect structure
x=4, y=1, width=4, height=3 (24 pixels)

When the classifier K_t is applied to the pixel (i, j) of the image A, it yields the value *val1(t)* if

$$\sum_{i=1}^{k}\left(w_1 \cdot \sum_{u=i+R_1y}^{R_1y+R_1height-1} \sum_{v=j+R_1yx}^{R_1y+R_1width-1} A_{uv}\right) < norm(i,\ j) \cdot\ threshold(t)$$

and *val2(t)* otherwise.

Here w_1 is a feature weight, *norm(i, j)* is the norm factor (generally the standard deviation on the rectangle containing all features), *threshold(t)*, *val1(t)* and *val2(t)* are parameters of the classifier. For fast computation the integral representation of an image is used. Haar classifiers are organized in sequences called *stages* (*classification stages*). The stage value is the sum of its classifier values. During feature detecting stages are consequently applied to the region of the image until the stage value becomes less than the threshold value or all stages are passed.

The use of the Intel IPP pattern recognition functions is demonstrated in the face detecting sample. See *Intel IPP JPEG Computer Vision Samples* downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

HaarClassifierInitAlloc

Allocates memory and initializes the structure for standard Haar classifiers.

Syntax

```
IppStatus ippiHaarClassifierInitAlloc_32f(IppiHaarClassifier_32f** pState,
    const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold,
    const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int length);
IppStatus ippiHaarClassifierInitAlloc_32s(IppiHaarClassifier_32s** pState,
    const IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold,
    const Ipp32s* pVal1, const Ipp32s* pVal2, const int* pNum, int length);
```

Parameters

ppState	Pointer to the pointer to the Haar classifier structure.
pFeature	Pointer to the array of features.
pWeight	Pointer to the array of feature weights.
pThreshold	Pointer to the array of classifier threshold values.
pVal1, pVal2	Pointers to the arrays of classifier result values.
pNum	Pointer to the array of classifier lengths.
length	Number of classifiers in the stage.

Description

The function `ippiHaarClassifierInitAlloc` is declared in the `ippcv.h` file. This function allocates memory and initializes the structure for the sequence of Haar classifiers - classification stage. *i*-th classifier in the stage has *pNum*[*i*] rectangular features. Each feature is defined by the certain rectangle with horizontal and vertical sides. The length of vectors *pThreshold*, *pVal1*, and *pVal2* is equal to *length*, the length of vectors *pFeature*, *pWeight* is equal to:

$$\sum_{i=0}^{length-1} pNum[i]$$

Result of applying classifiers to the image are computed in accordance with formula on page 14-94.

All features of the classifier initialized by the function `ippiHaarClassifierInitAlloc` have vertical and horizontal sides (left part of Figure 14-14). Some of these features later can be tilted by calling the function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`length`* or one of *`pNum[i]`* is less than or equal to 0.
`ippStsBadArgErr`	Indicates an error condition if one of features is defined incorrectly.
`ippStsMemAllocErr`	Memory allocation error

TiltedHaarClassifierInitAlloc

Allocates memory and initializes the structure for tilted Haar classifiers.

Syntax

```
IppStatus ippiTiltedHaarClassifierInitAlloc_32f(IppiHaarClassifier_32f**
    pState, const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f*
    pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int
    length);
IppStatus ippiTiltedHaarClassifierInitAlloc_32s(IppiHaarClassifier_32s**
    pState, const IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s*
    pThreshold, const Ipp32s* pVal1, const Ipp32s* pVal2, const int* pNum, int
    length);
```

Parameters

`ppState`	Pointer to the pointer to the Haar classifier structure.
`pFeature`	Pointer to the array of features.
`pWeight`	Pointer to the array of feature weights.
`pThreshold`	Pointer to the array of classifier threshold values.
`pVal1, pVal2`	Pointers to the arrays of classifier result values.
`pNum`	Pointer to the array of classifier lengths.
`length`	Number of classifiers in the stage.

Description

The function `ippiHaarClassifierInitAlloc` is declared in the `ippcv.h` file. This function allocates memory and initializes the structure for the sequence of Haar classifiers - classification stage. *t*-th classifier in the stage has *pNum*[*i*] rectangular features. Each feature is defined by the certain rectangle with sides tilted by 45 degrees. The points with minimum and maximum row numbers should be specified. The length of vectors *pFeature*, *pWeight*, *pThreshold*, *pVal1*, and *pVal2* is equal to:

$$\sum_{i=0}^{length-1} pNum[i]$$

Result of applying classifiers to the image are computed in accordance with formula on page 14-94.

All features of the classifier initialized by the function `ippiTiltedHaarClassifierInitAlloc` have tilted sides (right part of Figure 14-14).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *length* or one of *pNum*[*i*] is less than or equal to 0; or if the sum of all elements of *pNum* is not equal to *length*.
`ippStsBadArgErr`	Indicates an error condition if one of features is defined incorrectly.
`ippStsMemAllocErr`	Memory allocation error

HaarClassifierFree

Frees memory allocated for the Haar classifier structure.

Syntax

```
IppStatus ippiHaarClassifierFree_32f(IppiHaarClassifier_32f* pState);
IppStatus ippiHaarClassifierFree_32s(IppiHaarClassifier_32s* pState);
```

Parameters

pState	Pointer to the Haar classifier structure.

Description

The function `ippiHaarClassifierFree` is declared in the `ippcv.h` file. This function frees memory allocated for the Haar classifier structure *pState* by the function `ippiHaarClassifierInitAlloc` or `ippiTiltedHaarClassifierInitAlloc` .

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if the *pState* pointer is `NULL`.

GetHaarClassifierSize

Returns the size of the Haar classifier.

Syntax

```
IppStatus ippiGetHaarClassifierSize_32f(IppiHaarClassifier_32f* pState,
    IppiSize* pSize);
IppStatus ippiGetHaarClassifierSize_32s(IppiHaarClassifier_32s* pState,
    IppiSize* pSize);
```

Parameters

pState	Pointer to the Haar classifier structure.
pSize	Pointer to the size of Haar classifier structure.

Description

The function `ippiGetHaarClassifierSize` is declared in the `ippcv.h` file. This function computes the minimum size of the window containing all features of the Haar classifier described by the *pState*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if the *pState* pointer is `NULL`.

TiltHaarFeatures

Modifies a Haar classifier by tilting specified features.

Syntax

```
IppStatus ippiTiltHaarFeatures_32f(const Ipp8u* pMask, int flag,
    IppiHaarClassifier_32f* pState);
IppStatus ippiTiltHaarFeatures_32s(const Ipp8u* pMask, int flag,
    IppiHaarClassifier_32s* pState);
```

Parameters

pMask	Pointer to the mask vector.
flag	Flag to choose the direction of feature tilting.
pState	Pointer to the Haar classifier structure.

Description

The function `ippiTiltHaarFeatures` is declared in the `ippcv.h` file. This function tilts specified features of the Haar classifier created by the function `ippiHaarClassifierInitAlloc`. Non-zero elements of previously prepared vector *pMask* indicates the features that will be tilted. The *flag* parameter specifies how the features are tilted: if its value is equal to 0, the feature is tilted around the left top corner clockwise, if it is equal to 1, the feature is tilted around the bottom left corner counter-clockwise.

This mixed classifier containing both common and tilted features can be used by the function `ippiApplyMixedHaarClassifier`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsBadArgErr`	Indicates an error condition if the classifier is tilted already.

ApplyHaarClassifier

Applies a Haar classifier to an image.

Syntax

```
IppStatus ippiApplyHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize
    roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);
```

```
IppStatus ippiApplyHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep,
    const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize
    roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);
```

```
IppStatus ippiApplyHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep,
    const Ipp32s* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize
    roiSize, int* pPositive, Ipp32s threshold, IppiHaarClassifier_32s* pState,
    int scaleFactor);
```

Parameters

pSrc	Pointer to the ROI in the source image of integrals.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pNorm	Pointer to the ROI in the source image of norm factors.
normStep	Distance in bytes between starts of consecutive lines in the image of the norm factors.
pMask	Pointer to the source and destination image of classification decisions.
maskStep	Distance in bytes between starts of consecutive lines in the image of classification decisions.
pPositive	Pointer to the number of positive decisions.
roiSize	Size of the source and destination images ROI in pixels.
threshold	Stage threshold value.
pState	Pointer to the Haar classifier structure.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function `ippiApplyHaarClassifier` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function applies the Haar classifier *pState* previously initialized by the function ippiHaarClassifierInitAlloc or ippiTiltedHaarClassifierInitAlloc to pixels of the source image ROI *pSrc*. The source image should be in the integral representation, it can be obtained by calling one of the integral functions beforehand. The sum of pixels on feature rectangles is computed as:

$$\sum_{l=1}^{k}(pSrc[i+y_l, j+x_l] - pSrc[i+Y_l, j+x_l] - pSrc[i+y_l, j+X_l] + pSrc[i+Y_l, j+X_l])\cdot w_l$$

Here (y_l, x_l) and (Y_l, X_l) are coordinates of top left and right bottom pixels of l-th rectangle of the feature , and w_l is the feature weight. For i = 0...*roiSize.height* - 1, j = 0...*roiSize.width* - 1 all pixels referred in the above formula should be allocated in memory.

The input value of *pPositive*[0] is used as a hint to choose the calculation algorithm. If it is greater than or equal to *roiSize.width***roiSize.height* the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of *pMask* image. If the sum is less than *threshold* than the negative decision is made and the value of the corresponding pixel of the *pMask* image is set to zero. The number of positive decisions is assigned to the pPositive[0].

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the image step values is less than *roiSize.width**<*pixelSize*>.
ippStsNorEvenStepErr	Indicates an error condition if one of the image step values is not divisible by 4 for 32-bit images.

ApplyMixedHaarClassifier

Applies a mixed Haar classifier to an image.

Syntax

```
IppStatus ippiApplyMixedHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u*
    pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold,
    IppiHaarClassifier_32f* pState);
IppStatus ippiApplyMixedHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int
    srcStep, const Ipp32s* pTilt, int tiltStep, const Ipp32f* pNorm, int
    normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive,
    Ipp32f threshold, IppiHaarClassifier_32f* pState);
IppStatus ippiApplyMixedHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int
    srcStep, const Ipp32s* pTilt, int tiltStep, const Ipp32s* pNorm, int
    normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive,
    Ipp32s threshold, IppiHaarClassifier_32s* pState, int scaleFactor);
```

Parameters

pSrc	Pointer to the ROI in the source image of integrals.
srcStep	Distance in bytes between starts of consecutive lines in the source image of integrals.
pTilt	Pointer to the ROI in the source image of tilted integrals.
tiltStep	Distance in bytes between starts of consecutive lines in the source image of tilted integrals.
pNorm	Pointer to the ROI in the source image of norm factors.
normStep	Distance in bytes between starts of consecutive lines in the image of the norm factors.
pMask	Pointer to the source and destination image of classification decisions.
maskStep	Distance in bytes between starts of consecutive lines in the image of classification decisions.
pPositive	Pointer to the number of positive decisions.
roiSize	Size of the source and destination images ROI in pixels.
threshold	Stage threshold value.
pState	Pointer to the mixed Haar classifier structure.

scaleFactor	Scale factor (see Integer Result Scaling).

Description

The function ippiApplyMixedHaarClassifier is declared in the ippcv.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function applies the mixed Haar classifier *pState* to the ROI of the source images *pSrc* and *pTilt*. The mixed Haar classifier is a classifier initialized by the function ippiHaarClassifierInitAlloc and then modified by the function ippiTiltHaarFeatures.The source images must be in the integral representation, they can be obtained by calling one of the integral functions beforehand. Common features are applied to the *pSrc* image, and tilted features are applied to the *pTilt* image.The sum of pixels on feature rectangles is computed as:

$$\sum_{l=1}^{k}(pSrc[i+y_l, j+x_l] - pSrc[i+Y_l, j+x_l] - pSrc[i+y_l, j+X_l] + pSrc[i+Y_l, j+X_l])\cdot w_l$$

or

$$\sum_{l=1}^{k}(pTilt[i+y_l, j+x_l] - pTilt[i+Y_l, j+x_l] - pTilt[i+y_l, j+X_l] + pTilt[i+Y_l, j+X_l])\cdot w_l$$

Here (y_l, x_l) and (Y_l, X_l) are coordinates of top left and right bottom pixels of l-th rectangle of the feature , and w_l is the feature weight. For i = 0...*roiSize.height* - 1, j = 0...*roiSize.width* - 1 all pixels referred in the above formula should be allocated in memory.

The input value of *pPositive*[0] is used as a hint to choose the calculation algorithm. If it is greater than or equal to *roiSize.width***roiSize.height* the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of *pMask* image. If the sum is less than *threshold* than the negative decision is made and the value of the corresponding pixel of the *pMask* image is set to zero. The number of positive decisions is assigned to the pPositive[0].

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the image step values is less than *roiSize.width*<pixelSize>*.
ippStsNorEvenStepErr	Indicates an error condition if one of the image step values is not divisible by 4 for 32-bit images.

Camera Calibration and 3D Reconstruction

Correction of Camera Lens Distortion

Digital camera usually introduces significant distortion caused by the camera and lens. These distortions cause errors in any analysis of the image. The functions described in this section correct these distortion using intrinsic camera parameters and distortion coefficients. These intrinsic camera parameters are focal lengths fx, fy, and principal point coordinates cx, cy. The distortion is characterized by two coefficients of radial distortions k_1, k_2 and two coefficients of tangential distortions p_1, p_2.

The undistorted coordinates x_u and y_u of point with coordinates (x_d, y_d) are computed in accordance with the following formulas:

$$x_u = x_d \cdot \left(1 + k_1 r^2 + k_2 r^4\right) + 2p_1 x_d y_d + p_2 \cdot \left(r^2 + 2x_d^2\right)$$

$$y_u = y_d \cdot \left(1 + k_1 r^2 + k_2 r^4\right) + 2p_2 x_d y_d + p_1 \cdot \left(r^2 + 2y_d^2\right)$$

Here $r^2 = x_d^2 + y_d^2$, $x_d = (j\text{-}cx)/fx$, $y_d = (i\text{-}cy)/fy$; i and j are row and columns numbers of the pixel. The pixel value is computed using bilinear interpolation of four nearest pixel of the source image. If undistorted coordinates are outside the image, then the destination pixel is not changed.

UndistortGetSize

Computes the size of the external buffer.

Syntax

```
IppStatus ippiUndistortGetSize(IppiSize roiSize, int* pBufsize);
```

Parameters

roiSize	Size of source and destination images ROI in pixels.
pBufSize	Pointer to the computed value of the buffer size.

Description

The function `ippiUndistortGetSize` is declared in the `ippcv.h` file. This function computes the size of the temporary external buffer that is used by the functions `ippiUndistortRadial`. The buffer of the computed size can be used to process smaller images as well.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pBufSize* is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

UndistortRadial

Corrects radial distortions of the single image.

Syntax

```
IppStatus ippiUndistortRadial_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f
    fy, Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
```

Supported values for *mod* :

`8u_C1R`	`16u_C1R`	`32f_C1R`
`8u_C3R`	`16u_C3R`	`32f_C3R`

Parameters

pSrc	Pointer to the ROI in the source distorted image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination corrected image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of source and destination images ROI in pixels.
fx	Focal lengths along the x axis.
fy	Focal lengths along the y axis.

cx	x-coordinate of the principal point.
cy	y-coordinate of the principal point.
k1	First coefficient of radial distortion.
k2	Second coefficient of radial distortion.
pBuffer	Pointer to the external buffer.

Description

The function `ippiUndistortRadial` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function corrects radial distortions of the single source image *pSrc* and stores corrected image in the *pDst*. Correction is performed accounting camera parameters *fx*, *fy*, *cx*, *cy* and radial distortion parameters *k1*, *k2*. The function can also pass the pointer to the external buffer *pBuffer* whose size should be computed previously using the function `ippiUndistortGetSize` .
If a null pointer is passed, slower computations without an external buffer will be performed.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` is less than `roiSize.width * <pixelSize>` , or `dstStep` is less than `roiSize.width * <pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if `fx` or `fy` is equal to 0.

CreateMapCameraUndistort

Creates look-up tables of coordinates of corrected image.

Syntax

```
IppStatus ippiCreateMapCameraUndistort_32f_C1R(Ipp32f* pxMap, int xStep,
    Ipp32f* pyMap, int yStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy, Ipp32f
    cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp32f p1, Ipp32f p2, Ipp8u* pBuffer);
```

Parameters

pxMap	Pointer to the destination x coordinate look-up buffer.
xStep	Distance in bytes between starts of consecutive lines in the *pxMap* image.
pyMap	Pointer to the destination y coordinate look-up buffer.
yStep	Distance in bytes between starts of consecutive lines in the *pyMap* image.
roiSize	Size of source and destination images ROI in pixels.
fx	Focal lengths along the *x* axis.
fy	Focal lengths along the *y* axis.
cx	*x*- coordinate of the principal point.
cy	*y*- coordinate of the principal point.
k1	First coefficient of radial distortion.
k2	Second coefficient of radial distortion.
p1	First coefficient of tangential distortion.
p2	Second coefficient of tangential distortion.
pBuffer	Pointer to the external buffer.

Description

The function `ippiCreateMapCameraUndistort` is declared in the `ippcv.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates the look-up tables of *x*- and *y*-coordinates *pxMap* and *pyMap* respectively. These coordinates are computed in accordance with camera parameters *fx*, *fy*, *cx*, *cy*, and distortion parameters *k1*, *k2*, *p1*, *p2*. The created tables can be used by the Intel IPP function ippiRemap to remap the distorted source image and get the corrected image.

To accelerate the computations the function can pass the pointer to the external buffer *pBuffer* whose size should be computed previously using the function ippiUndistortGetSize.
If a null pointer is passed, slower computations without an external buffer will be performed.

The following Example 14-7 demonstrates how to correct camera lens distortion for set of images using Intel IPP functions.

Example 14-7 Correction of Set of Distorted Images

```
Ipp32f *pxMap, *pyMap, *pBuffer;
Int xStep, yStep, buflen;
IppiSize roiSize;
IppiRect rect;


ippiUndistortGetSize (roiSize, &buflen);
buffer = ippiMalloc_8u(buflen);
xMap = ippiMalloc_32f(roiSize.x, roiSize,y, *xStep);
yMap = ippiMalloc_32f(roiSize.x, roiSize,y, *yStep);

ippiCreateMapCameraUndistort_32f_C1R (pxMap, pxStep, yMap, yStep, roiSize,
                                      fx, fy, cx, cy, k1, k2, 0, 0, pBuffer);
rect.x = 0;             rect.y = 0;
rect.width = roiSize.width; rect.height = roiSize.height;

…

ippiRemap_32f(pSrc, roiSize, srcStep, rect, pxMap, xStep, pyMap, yStep,
              pDst, dstStep, roiSize, IPPI_INTER_LINEAR);

…

ippiFree(yMap);
ippiFree(xMap);
ippsFree(buffer);
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if *pxMap* or *pyMap* is NULL.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if : *xStep* is less than *roiSize.width* * *<pixelSize>* , or *yStep* is less than *roiSize.width* * *<pixelSize>*.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBadArgErr	Indicates an error when *fx* or *fy* is equal to 0.

Image Enchancement Functions

This section describes the functions that are used in the high-quality upsampling, or increasing the spatial resolution of the video stream (Haussecker-Nestares super-resolution algorithm).

Algorithm Overview

For every frame in the input stream the algorithm computes the corresponding high-resolution (upsampled) frame by combining information from the original low-resolution frame (the central frame) and several low-resolution frames preceding and following the central frame in the video stream. The algorithm consists of 2 parts:

- Registration, or finding position in the central frame of each pixel of each low-resolution frame. That is, the dense optical flow from each surrounding frames to the central frame is calculated.
- Robust estimation of value of each pixel in the high-resolution frame using the Bayesian framework.

The Intel IPP functions described below implement only second part.

Once the optical flow is computed, the Bayesian formulation is reduced to the following optimization problem:

$$g = \arg\min_{g} C(g);$$

$$C(g) = \sum_{i,j} w_{i,j} \varphi_L\left(\frac{f_{i,j} - A_{i,j} g}{\sigma_L}\right) + \lambda \sum_{(i,j):i<j \ \& \ |x(i)-x(j)|+|y(i)-y(j)|=1} \varphi_P\left(\frac{g_i - g_j}{\sigma_P}\right)$$

where:

g - unknown high-resolution frame represented as a 1D vector which elements are pixel intensities in the raster scan order.

$w_{i,j}$ - confidence weights that make certain links between low- and high-resolution frames more or less important depending on the accuracy of the estimated motion field at corresponding areas and so on.

f_i - i-th low-resolution frame, represented as a 1D vector.

$f_{i,j}$ - intensity of the j-th pixel of f_i.

A_i - mapping from g to f_i, also known as the point spread function PSF (see "Point Spread Function" for more details). It combines optical flow, blurring and downsampling. As A_i is a linear mapping (each low-resolution pixel is a linear combination of several pixels in high-resolution image, $A_{i,j}g$ is a dot product of the j-th row of matrix A_i and vector g.

σ_L, σ_P - standard deviations in the likelihood and prior parts of the target function, respectively.

φ_L, φ_P - robust error functions (see "Error Functions") for the likelihood and prior parts of the target function, respectively.

λ - parameter that regulates smoothness, or the relative weight of the prior part.

$(i,j): i<j \& |x(i)-x(j)|+|y(i)-y(j)|=1$ means that the summation is done over all ordered pairs (g_i, g_j) where g_i and g_j are intensities of the horizontally or vertically adjacent pixels of g.

This problem is solved using the conjugate gradient algorithm, which main steps are:

- Computing gradient of $C(g)$:

$$\nabla C(g) = \sum_{i,j} \frac{w_{i,j}}{\sigma_L} A_{i,j}^T \psi_L\left(\frac{f_{i,j} - A_{i,j}g}{\sigma_L}\right) + \lambda \sum_{(i,j):|x(i)-x(j)|+|y(i)-y(j)|=1} \frac{I_i}{\sigma_P} \psi_P\left(\frac{g_i - g_j}{\sigma_P}\right)$$

  where ψ_L and ψ_P are derivatives of the φ_L and φ_P, respectively, and I_i is the i-th column of the identity matrix.

- Solving 1D optimization problem:

$$\alpha = \arg\min_{\alpha} C(g + \alpha d)$$

  where d is the current "conjugate" direction. The search is performed using the Newton-Raphson method, if it fails then the golden-section search (GSS) is used.

- Computing cost $C(g)$ to determine when to stop the algorithm (as soon as the decrease of the cost function becomes small enough or even negative).

Point Spread Function

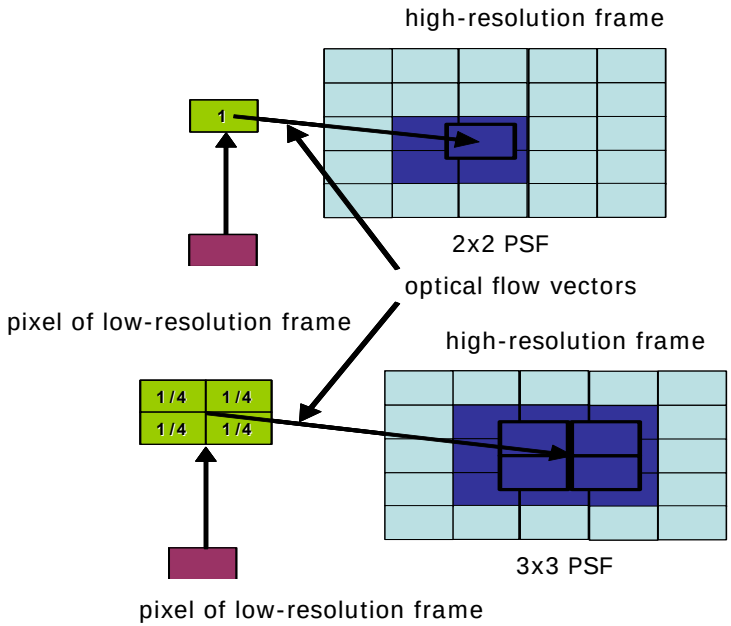
The point spread function (PSF) maps $f_{i,j}$ to g or vice versa. To do this, blurring, decimation and motion need to be modeled at high resolution.

In case of superresolution with scale factor 2 a simple 2x2 box filter is used for blurring and downsampling. For motion field more complex interpolation is used. Nearest neighbor interpolation results in the 3x3 PSF, in other word each low-resolution image pixel is mapped to 3x3 area in high-resolution image.

Another useful mode of this algorithm is x1 upsampling, it may be used for quick de-noising and de-blocking of chrominance planes. In this case motion field is not interpolated, and blurring is not performed, thus it can be done with the 2x2 PSF.

Figure 14-15 shows the above described types of PSF.

Figure 14-15 Mapping Low-resolution frame with the PSF



Error Functions

Error function must be a symmetrical non-negative function that has second derivative. Moreover it must be convex near the zero. A robust error function is an error function that grows slower than the quadratic function, thus outliers do not influence the target cost function much. There are many functions that satisfy the above requirements. Intel IPP provides the Cauchy function (see “Intel IPP Reference Manual” vol.1):

$$\varphi(x) = \frac{1}{2}\log\left(1 + \left(\frac{x}{C}\right)^2\right)$$

where C is a user-supplied parameter.

SRHNInitAlloc_PSF3x3, SRHNInitAlloc_PSF2x2

Allocates memory and initializes the internal representation of the PSF table.

Syntax

```
IppStatus ippiSRHNInitAlloc_PSF3x3(IppiSRHNSpec_PSF3x3** ppPSF, const
    Ipp32f coeffTab[][9], int tabSize);
IppStatus ippiSRHNInitAlloc_PSF2x2(IppiSRHNSpec_PSF2x2** ppPSF, const
    Ipp32f coeffTab[][4], int tabSize);
```

Parameters

ppPSF	Double pointer to the PSF table.
coeffTab	Array containing the input PSF table with the coefficients.
tabSize	The number of elements in the *coeffTab*.

Description

The functions `ippiSRHNInitAlloc_PSF3x3` and `ippiSRHNInitAlloc_PSF2x2` are declared in the `ippcv.h` file. These functions allocate memory and initialize the internal table representation of the 3x3 or 2x2 point spread functions (PSF) respectively.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error if the *ppPSF* pointer is NULL.
ippStsSizeErr	Indicates an error if the *tabSize* is less than or equal to 0.

SRHNFree_PSF3x3, SRHNFree_PSF3x3

Deallocates the internal representation of the PSF table.

Syntax

```
IppStatus ippiSRHNFree_PSF3x3(IppiSRHNSpec_PSF3x3* pPSF);
IppStatus ippiSRHNFree_PSF2x2(IppiSRHNSpec_PSF2x2* pPSF);
```

Parameters

pPSF	Ppointer to the PSF table.

Description

The functions ippiSRHNFree_PSF3x3 and ippiSRHNFree_PSF2x2 are declared in the ippcv.h file. These functions deallocates the internal table representation of the 3x3 or 2x2 point spread functions (PSF) respectively.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the *pPSF* pointer is NULL.

SRHNCalcResidual_PSF3x3, SRHNCalcResidual_PSF2x2

Computes residuals for the likelihood part of the target function or its gradient.

Syntax

```
IppStatus ippiSRHNCalcResidual_PSF3x3_8u32f_C1R(const Ipp32f* pHighRes, int
    highResStep, const Ipp8u* pLowRes, const Ipp32s* pOffset, const Ipp16u*
    pCoeff, Ipp32f* pRes, int len, const IppiSRHNSpec_PSF3x3* pPSF);
IppStatus ippiSRHNCalcResidual_PSF3x3_16u32f_C1R(const Ipp32f* pHighRes,
    int highResStep, const Ipp16u* pLowRes, const Ipp32s* pOffset, const
    Ipp16u* pCoeff, Ipp32f* pRes, int len, const IppiSRHNSpec_PSF3x3* pPSF);

IppStatus ippiSRHNCalcResidual_PSF2x2_16u32f_C1R(const Ipp32f* pHighRes,
    int highResStep, const Ipp16u* pLowRes, const Ipp32s* pOffset, const
    Ipp16u* pCoeff, Ipp32f* pRes, int len, const IppiSRHNSpec_PSF2x2* pPSF);
IppStatus ippiSRHNCalcResidual_PSF2x2_8u32f_C1R(const Ipp32f* pHighRes, int
    highResStep, const Ipp8u* pLowRes, const Ipp32s* pOffset, const Ipp16u*
    pCoeff, Ipp32f* pRes, int len, const IppiSRHNSpec_PSF2x2* pPSF);
```

Parameters

pHighRes	Pointer to high-resolution image.
highResStep	Distance in bytes between starts of consecutive lines in the *pHighRes* image.
pLowRes	Pointer to the array of pixel values from low-resolution images.
pOffset	Pointer to the array of offsets of 3x3 or 2x2 areas in the high-resolution image.
pCoeff	Pointer to the array of indexes of the coefficients in the input PSF table.
pRes	Pointer to the output array containing the calculated residuals.
len	Number of elements in the arrays *pLowRes*, *pOffset*, *pCoeff* and *pRes*.
pPSF	Pointer to the initialized table for 3x3 or 2x2 PSF.

Description

The functions SRHNCalcResidual_PSF3x3 and SRHNCalcResidual_PSF2x2 are declared in the ippcv.h file.

These functions computes residuals *pRes* for the likelihood part of the target function $C(g)$ or its gradient $\nabla C(g)$. The calculated values are used as the arguments of the robust error functions or their derivatives. The following pseudocode shows how the calculation is performed.

```
for( int i = 0; i < len; i++ ) {
      const Ipp32f* ptr = hiRes + ofs[i];
      const Ipp32f* c = psf->cTab[coeff[i]];
      Ipp32f sum = 0;
      for( int dy = 0; dy < K; dy++ )
         for( int dx = 0; dx < K; dx++ )
            sum += c[dy*K+dx]*ptr[dy*hiResStep + dx];
      residual[i] = sum - lowRes[i];
}
```

Here K=3 for the 3x3 PSF, and K=2 for the 2x2 PSF.

The results stored in the *pRes* then can be passed to the robust error functions: ippsCauchy, ippsCauchyD, and ippsCauchyDD2 (see section"Arithmetic Functions" in the "Intel IPP Reference Manual" vol.1).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error if *len* is less than 0.
ippStsStepErr	Indicates an error if *highResStep* is less than 2*sizeof(Ipp32f) for 3x3 PSF, and less than 3*sizeof(Ipp32f) for the 2x2 PSF.

SRHNUpdateGradient_PSF3x3, SRHNUpdateGradient_PSF2x2

Updates the likelihood part of the gradient.

Syntax

```
IppStatus ippiSRHNUpdateGradient_PSF3x3_32f_C1R(Ipp32f* pGrad, int
    gradStep, const Ipp32s* pOffset, const Ipp16u* pCoeff, const Ipp32f*
    pDerivRes, const Ipp8u* pWeight, int len, const Ipp32f* pWeightTab, const
    IppiSRHNSpec_PSF3x3* pPSF);
```

```
IppStatus ippiSRHNUpdateGradient_PSF2x2_32f_C1R(Ipp32f* pGrad, int
    gradStep, const Ipp32s* pOffset, const Ipp16u* pCoeff, const Ipp32f*
    pDerivRes, const Ipp8u* pWeight, int len, const Ipp32f* pWeightTab, const
    IppiSRHNSpec_PSF2x2* pPSF);
```

Parameters

pGrad	Pointer to the gradient.
gradStep	Distance in bytes between starts of consecutive lines in the *pGrad*.
pOffset	Pointer to the array of offsets of 3x3 or 2x2 areas in the high-resolution image.
pCoeff	Pointer to the array of indexes of the coefficients in the input PSF table.
pDerivRes	Pointer to the array containing the calcuated derivatives of the robust error function.
pWeight	Pointer to the array of indexes of the values in the *pWeightTab*.
len	Number of elements in the arrays *pOffset*, *pCoeff*, *pDerivRes*, and *pWeight*.
pWeightTab	Pointer to the table of the confidence weights.
pPSF	Pointer to the initialized table for 3x3 or 2x2 PSF.

Description

The functions `SRHNUpdateGradient_PSF3x3` and `SRHNUpdateGradient_PSF2x2` are declared in the `ippcv.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

Theses functions update gradient of the target function $\nabla C(g)$. using the calculated derivatives of the robust error function. The following pseudocode shows how the calculation is performed.

```
for( int i = 0; i < len; i++ ) {
      const Ipp32f* ptr = grad + ofs[i];
      const Ipp32f* c = psf->cTab[coeff[i]];
      Ipp32f w = wTab[weight[i]]*dRes[i];
      for( int dy = 0; dy < K; dy++ )
         for( int dx = 0; dx < K; dx++ )
            ptr[dy*gradStep+dx] += c[dy*K+dx]*w;
}
```

Here `K=3` for the 3x3 PSF, and `K=2` for the 2x2 PSF.

CAUTION. When the gradient is calculated, the functions `ippiSRHNCalcResidual` and `ippiSRHNUpdateGradient` are called sequentially (and the function `ippsCauchyD` between them). The same array *pOffset* can be passed to both these functions (with the same PSF type) but only when the parameters *highResStep* an *gradStep* are equal.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error if *len* is less than 0.
`ippStsStepErr`	Indicates an error if *highResStep* is less than `2*sizeof(Ipp32f)` for 3x3 PSF, and less than `3*sizeof(Ipp32f)` for the 2x2 PSF.

Image Compression Functions

15

This chapter describes the Intel® IPP image processing functions that prepare data and perform still image compression and coding in accordance with JPEG and JPEG2000 standards (see [ISO10918] and [ISO15444]).
These functions are grouped into the following sections:

Support Functions

JPEG Coding:

Color Conversion Functions

Combined Color Conversion Functions

Quantization Functions

Combined DCT Functions

Level Shift Functions

Sampling Functions

Planar-to-Pixel and Pixel-to-Planar Conversion Functions

Huffman Codec Functions

Lossless Mode of Operation:

Functions for Lossless JPEG Coding

JPEG2000 Coding:

Wavelet Transform Functions

JPEG2000 Entropy Coding and Decoding Functions

Component Transform Functions

RLE Coding:

Functions for RLE Coding

Texture Image Coding:

Texture Compression Functions

Each section starts with a table that lists functions described in more detail later in this section, together with brief description of operations these functions perform.

NOTE. For simplicity, `ippi` prefix is omitted in function group names given in the summary tables. Function prototypes and references in the text contain full names.

The use of the Intel IPP JPEG functions is demonstrated in Intel IPP Samples. See *Intel IPP JPEG Processing Samples* downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

Support Functions

This section describes the support function that returns information about the current version of the used Intel IPP JPEG codec software.

ippjGetLibVersion

Returns information of the library version.

Syntax

```
const IppLibraryVersion*  ippjGetLibVersion(void);
```

Description

The function `ippjGetLibVersion` is declared in the `ippj.h` file. This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of the Intel IPP JPEG codec library. There is no need to free memory referenced by this pointer, as it points to a static variable. The `IppLibraryVersion` structure contains the following fields:

major	The major number of the current library version.
minor	The minor number of the current library version.

Name	The name of the library.
Version	The library version ASCII string.
BuildDate	The library version build date.

Return Values

The return value of the function is a pointer to a structure `IppLibraryVersion`.

Color Conversion Functions

This section describes Intel IPP color conversion functions that are specific for JPEG codec.

These functions are listed in the table Table 15-1.

Table 15-1 Color Conversion Functions

Function Base Name	Description
RGBToY_JPEG	Converts RGB images to gray scale.
BGRToY_JPEG	Converts BGR images to gray scale.
RGBToYCbCr_JPEG	Converts RGB images to the YCbCr color model.
YCbCrToRGB_JPEG	Converts YCbCr images to the RGB color model.
RGB565ToYCbCr_JPEG, RGB555ToYCbCr_JPEG	Convert 16-bit RGB images to the YCbCr color model.
YCbCrToRGB565_JPEG, YCbCrToRGB555_JPEG	Convert YCbCr images to the 16-bit RGB images.
BGRToYCbCr_JPEG	Converts BGR images to the YCbCr color model.
YCbCrToBGR_JPEG	Converts YCbCr images to the BGR color model.
BGR565ToYCbCr_JPEG, BGR555ToYCbCr_JPEG	Convert 16-bit BGR images to the YCbCr color model.
YCbCrToBGR565_JPEG, YCbCrToBGR555_JPEG	Convert YCbCr images to the 16-bit BGR images.
YCbCr422ToRGB_JPEG	Converts 16-bit per pixel YCbCr image to 24-bit per pixel RGB image.
CMYKToYCCK_JPEG	Converts CMYK images to the YCCK color model.
YCCKToCMYK_JPEG	Converts YCCK images to the CMYK color model.

RGBToY_JPEG

Converts an RGB image to gray scale.

Syntax

Case 1: Operation on planar data

```
IppStatus ippiRGBToY_JPEG_8u_P3C1R(const Ipp8u* pSrcRGB[3],
    int srcStep, Ipp8u*  pDstY, int dstStep, IppiSize roiSize);
```

Case 2: Operation on pixel-order data

```
IppStatus ippiRGBToY_JPEG_8u_C3C1R(const Ipp8u* pSrcRGB, int srcStep,
    Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

Parameters

pSrcRGB	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstY	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRGBToY_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts the RGB image to gray scale using the following formula:

```
Y  =  0.299*R + 0.587*G + 0.114*B
```

Example 15-1 shows how to use the function `ippiRGBToY_JPEG_8u_C3C1R`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

Example 15-1 Using the function `ippiRGBToY_JPEG`

```
Ipp8u srcRGB[9*3] = { 255, 0, 0, 255, 0, 0, 255, 0, 0,
                      0, 255, 0, 0, 255, 0, 0, 255, 0,
                      0, 0, 255, 0, 0, 255, 0, 0, 255};
Ipp8u dstY[3*3];
IppiSize roiSize = { 9, 3 };
```

Example 15-1 Using the function ippiRGBToY_JPEG

```
ippiRGBToY_JPEG_8u_C3C1R ( srcRGB, 9, dstY, 9, roiSize );

result:
255 0   0   255 0   0   255 0   0
0   255 0   0   255 0   0   255 0        src
0   0   255 0   0   255 0   0   255

 76  76  76
150 150 150        dst
 29  29  29
```

BGRToY_JPEG

Converts a BGR image to gray scale.

Syntax

```
IppStatus ippiBGRToY_JPEG_8u_C3C1R(const Ipp8u* pSrcBGR, int srcStep,
    Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

Parameters

pSrcBGR	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstY	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiBGRToY_JPEG is declared in the ippj.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a BGR image to gray scale using the following formula:

```
Y  =  0.299*R + 0.587*G + 0.114*B
```

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

RGBToYCbCr_JPEG

Converts an RGB image to YCbCr color model.

Syntax

Case 1: Operation on planar data

```
IppStatus ippiRGBToYCbCr_JPEG_8u_P3R(const Ipp8u* pSrcRGB[3], int
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

Case 2: Operation on pixel-order data

```
IppStatus ippiRGBToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrcRGB, int srcStep,
    Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

Parameters

pSrcRGB	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstYCbCr	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function ippiRGBToYCbCr_JPEG is declared in the ippj.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an RGB image to the YCbCr color model using the following formulas:

```
Y =   0.299*R   + 0.587*G   + 0.114*B
Cb = -0.16874*R - 0.33126*G + 0.5*B     + 128
Cr =  0.5*R     - 0.41869*G - 0.08131*B + 128
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

YCbCrToRGB_JPEG

Converts an YCbCr image to the RGB color model.

Syntax

Case 1: Operation on planar data

```
IppStatus ippiYCbCrToRGB_JPEG_8u_P3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp8u* pDstRGB[3], int dstStep, IppiSize roiSize);
```

Case 2: Operation on pixel-order data

```
IppStatus ippiYCbCrToRGB_JPEG_8u_P3C3R(const Ipp8u* pSrcYCbCr[3], int
    srcStep, Ipp8u* pDstRGB, int dstStep, IppiSize roiSize);
```

Parameters

pSrcYCbCr	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstRGB	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYCbCrToRGB_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an YCbCr image to the RGB image according to the following formulas:

```
R = Y  + 1.402*Cr - 179.456
G = Y  - 0.34414*Cb - 0.71414*Cr + 135.45984
B = Y  + 1.772*Cb - 226.816
```

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

RGB565ToYCbCr_JPEG, RGB555ToYCbCr_JPEG

Convert an RGB image to YCbCr color model.

Syntax

```
IppStatus ippiRGB565ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcRGB,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
IppStatus ippiRGB555ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcRGB,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

Parameters

pSrcRGB	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstYCbCr	An array of pointers to the ROIs in the separate destination color planes.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiRGB555ToYCbCr_JPEG` and `ippiRGB555ToYCbCr_JPEG` are declared in the `ippj.h` file. These functions convert an RGB image to the YCbCr image using the same formulas as in `ippiRGBToYCbCr_JPEG` function for computing `Y`, `Cb`, and `Cr` component values. The source image *pSrcRGB* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), or `RGB555` (5 bits for red, green, blue).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

YCbCrToRGB565_JPEG, YCbCrToRGB555_JPEG

Convert an YCbCr image to the RGB color model.

Syntax

```
IppStatus ippiYCbCrToRGB565_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp16u* pDstRGB, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp16u* pDstRGB, int dstStep, IppiSize roiSize);
```

Parameters

pSrcYCbCr	An array of pointers to the ROIs in the separate source color planes.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDstRGB	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCrToRGB565_JPEG` and `ippiYCbCrToRGB565_JPEG` are declared in the `ippj.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert an YCbCr image to the RGB format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `R`, `G`, and `B` component values. The destination image *pDstRGB* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), or `RGB555` (5 bits for red, green, blue).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

BGRToYCbCr_JPEG

Converts a BGR image to YCbCr color model.

Syntax

```
IppStatus ippiBGRToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrcBGR,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

Parameters

pSrcBGR	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDstYCbCr	An array of pointers to the ROIs in the separate destination color planes.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiBGRToYCbCr_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a BGR image to the YCbCr color model using the same formulas as in `ippiRGBToYCbCr_JPEG` function for computing `Y`, `Cb`, and `Cr` component values.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

YCbCrToBGR_JPEG

Converts an YCbCr image to the BGR color model.

Syntax

```
IppStatus ippiYCbCrToBGR_JPEG_8u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp8u* pDstBGR, int dstStep, IppiSize roiSize);
```

Parameters

pSrcYCbCr	An array of pointers to the ROIs in the separate source color planes.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstBGR	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYCbCrToBGR_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an YCbCr image to the BGR image using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `R`, `G`, and `B` component values.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

BGR565ToYCbCr_JPEG, BGR555ToYCbCr_JPEG

Convert a BGR image to YCbCr color model.

Syntax

```
IppStatus ippiBGR565ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcBGR, int
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
IppStatus ippiBGR555ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

Parameters

pSrcBGR	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstYCbCr	An array of pointers to the ROIs in the separate destination color planes.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions ippiBGR555ToYCbCr_JPEG and ippiBGR555ToYCbCr_JPEG are declared in the ippj.h file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert an BGR image to the YCbCr image using the same formulas as in ippiRGBToYCbCr_JPEG function for computing Y, Cb, and Cr component values. The source image *pSrcBGR* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

YCbCrToBGR565_JPEG, YCbCrToBGR555_JPEG

Convert an YCbCr image to the BGR color model.

Syntax

```
IppStatus ippiYCbCrToBGR565_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp16u* pDstBGR, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp16u* pDstBGR, int dstStep, IppiSize roiSize);
```

Parameters

pSrcYCbCr	An array of pointers to the ROIs in the separate source color planes.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstBGR	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The functions `ippiYCbCrToBGR565_JPEG` and `ippiYCbCrToBGR565_JPEG` are declared in the `ippj.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert an YCbCr image to the BGR format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `R`, `G`, and `B` component values. The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), or `BGR555` (5 bits for blue, green, red).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

YCbCr422ToRGB_JPEG

Converts 16-bit per pixel YCbCr image to 24-bit per pixel RGB image.

Syntax

```
IppStatus ippiYCbCr422ToRGB_JPEG_8u_C2C3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYCbCr422ToRGB_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an YCbCr image image *pSrc*, packed in 4:2:2 sampling format (see Table 6-2 and Table 6-3 for more details) to the `RGB` image *pDst* according to the same formulas as the function `ippiYCbCrToRGB_JPEG` does.
The output `RGB` values are saturated to the range `[0..255]`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

CMYKToYCCK_JPEG

Converts a CMYK image to the YCCK color model.

Syntax

Case 1: Operation on planar data

```
IppStatus ippiCMYKToYCCK_JPEG_8u_P4R(const Ipp8u* pSrcCMYK[4],
    int srcStep, Ipp8u* pDstYCCK[4], int dstStep, IppiSize roiSize);
```

Case 2: Operation on pixel-order data

```
IppStatus ippiCMYKToYCCK_JPEG_8u_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp8u* pDstYCCK[4], int dstStep, IppiSize roiSize);
```

Parameters

pSrcCMYK	An array of pointers to the ROIs in the separate source color planes.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstYCCK	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiCMYKToYCCK_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a CMYK image to the YCCK image in two steps. First, conversion is done into RGB format:

```
R = 255 – C
G = 255 – M
B = 255 – Y
```

After that, conversion to YCCK image is performed as:

```
Y =   0.299*R   + 0.587*G   + 0.114*B
Cb = -0.16874*R - 0.33126*G + 0.5*B     + 128
Cr =  0.5*R     - 0.41869*G - 0.08131*B + 128
```

The values of *K* channel are written without modification.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

YCCKToCMYK_JPEG

Converts an YCCK image to the CMYK color model.

Syntax

```
IppStatus ippiYCCKToCMYK_JPEG_8u_P4R(const Ipp8u* pSrcYCCK[4],
    int srcStep, Ipp8u* pDstCMYK[4], int dstStep, IppiSize roiSize);
IppStatus ippiYCCKToCMYK_JPEG_8u_P4C4R(const Ipp8u* pSrcYCCK[4],
    int srcStep, Ipp8u* pDstCMYK, int dstStep, IppiSize roiSize);
```

Parameters

pSrcYCCK	An array of pointers to the ROIs in the separate source color planes.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstCMYK	Pointer to the destination image ROI; an array of pointers to the ROIs in the separate destination color planes for planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiYCCKToCMYK_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an YCCK image to the CMYK image in two steps. First, conversion is done into RGB format as:

```
R = Y  + 1.402*Cr - 179.456
G = Y  - 0.34414*Cb - 0.71414*Cr + 135.45984
B = Y  + 1.772*Cb - 226.816
```

After that, conversion to CMYK image is performed as follows:

```
C = 255 – R
M = 255 – G
Y = 255 – B
```

The values of `K` channel are written without modification.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any od the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`roiSize`* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *`srcStep`* or *`dstStep`* has a zero or negative value.

Combined Color Conversion Functions

This section describes Intel IPP functions that perform sampling, color conversion and level shift (transition from unsigned data range to the signed one and vice versa) and are specific for JPEG codec. These functions are listed in the table Table 15-2.

Table 15-2 Combined Color Conversion Functions

Function Base Name	Description
RGBToYCbCr444LS_MCU	Converts RGB images to YCbCr color model and creates 444 MCU.
RGBToYCbCr422LS_MCU	Converts RGB images to YCbCr color model and creates 422 MCU.
RGBToYCbCr411LS_MCU	Converts RGB images to YCbCr color model and creates 411 MCU.
BGRToYCbCr444LS_MCU	Converts BGR images to YCbCr color model and creates 444 MCU.
BGR565ToYCbCr444LS_MCU, BGR555ToYCbCr444LS_MCU	Convert 16-bit BGR images to YCbCr color model and create 444 MCU.
BGRToYCbCr422LS_MCU	Converts BGR images to YCbCr color model and creates 422 MCU.
BGR565ToYCbCr422LS_MCU, BGR555ToYCbCr422LS_MCU	Convert 16-bit BGR images to YCbCr color model and create 422 MCU
BGRToYCbCr411LS_MCU	Converts BGR images to YCbCr color model and creates 411 MCU.
BGR565ToYCbCr411LS_MCU, BGR555ToYCbCr411LS_MCU	Convert 16-bit BGR images to YCbCr color model and create 411 MCU
CMYKToYCCK444LS_MCU	Converts CMYK images to YCCK color model and creates 4444 MCU.
CMYKToYCCK422LS_MCU	Converts CMYK images to YCCK color model and creates 4224 MCU.
CMYKToYCCK411LS_MCU	Converts CMYK images to YCCK color model and creates 4114 MCU.
YCbCr444ToRGBLS_MCU	Creates YCbCr image from 444 MCU and converts it to RGB color model.
YCbCr422ToRGBLS_MCU	Creates YCbCr image from 422 MCU and converts it to RGB color model.

Table 15-2 Combined Color Conversion Functions (continued)

Function Base Name	Description
YCbCr411ToRGBLS_MCU	Creates YCbCr image from 411 MCU and converts it to RGB color model.
YCbCr444ToBGRLS_MCU	Creates YCbCr image from 444 MCU and converts it to BGR color model.
YCbCr444ToBGR565LS_MCU, YCbCr444ToBGR555LS_MCU	Create YCbCr image from 444 MCU and convert it to 16-bit BGR image.
YCbCr422ToBGRLS_MCU	Creates YCbCr image from 422 MCU and converts it to BGR color model.
YCbCr422ToBGR565LS_MCU, YCbCr422ToBGR555LS_MCU	Create YCbCr image from 422 MCU and convert it to 16-bit BGR image.
YCbCr411ToBGRLS_MCU	Creates YCbCr image from 411 MCU and converts it to BGR color model.
YCbCr411ToBGR565LS_MCU, YCbCr411ToBGR555LS_MCU	Create YCbCr image from 411 MCU and convert it to 16-bit BGR image.
YCCK444ToCMYKLS_MCU	Creates YCCK image from 4444 MCU and converts it to CMYK color model.
YCCK422ToCMYKLS_MCU	Creates YCCK image from 4224 MCU and converts it to CMYK color model.
YCCK411ToCMYKLS_MCU	Creates YCCK image from 4114 MCU and converts it to CMYK color model.

Here MCU stands for JPEG Minimum Coded Unit, that is, an interleaved set of blocks of size determined by the sampling factors (see Figure 6-13), or a single block in a non interleaved scan.

All functions implement the level shift operation in accordance with its definition by [ISO10918], *Annex A.3.1*, *Level Shift*.

RGBToYCbCr444LS_MCU

Converts an RGB image to the YCbCr color model and creates 444 MCU.

Syntax

```
IppStatus ippiRGBToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrcRGB	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The function `ippiRGBToYCbCr444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an RGB image to the YCbCr image using the same formulas as in `ippiRGBToYCbCr_JPEG` function for computing `Y`, `Cb`, and `Cr` component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 444 MCU (see Figure 6-13).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

RGBToYCbCr422LS_MCU

Converts an RGB image to the YCbCr color model and creates 422 MCU.

Syntax

```
IppStatus ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrcRGB	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The function `ippiRGBToYCbCr422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an RGB image to the YCbCr image using the same formulas as in `ippiRGBToYCbCr_JPEG` function for computing `Y`, `Cb`, and `Cr` component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 422 MCU (see Figure 6-13).

Downsampling is performed by plain averaging.

Example 15-2 shows how to use the function `ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

Example 15-2 Using the function ippiRGBToYCbCr422LS_MCU

```
int k=0;
int buf;
Ipp8u SrcRGB[48*8];
for(int i = 0; i < 48*8; i++){
    if(k==48) k=0;
    SrcRGB[i] = k++;
}
Ipp16s* DstMCU[3];
DstMCU[0] = ippiMalloc_16s_C1(16,8,&buf);  // no control on return value
DstMCU[1] = ippiMalloc_16s_C1(8,8,&buf);
DstMCU[2] = ippiMalloc_16s_C1(8,8,&buf);

ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R(SrcRGB, 48, DstMCU);

result:

    DstMCU[0]
-128 -125 -122 -119 -116 -113 -110 -107 -128 -125 -122 -119 -116 -113 -110 -107
-128 -125 -122 -119 -116 -113 -110 -107 -128 -125 -122 -119 -116 -113 -110 -107
-128 -125 -122 -119 -116 -113 -110 -107 -128 -125 -122 -119 -116 -113 -110 -107
-128 -125 -122 -119 -116 -113 -110 -107 -128 -125 -122 -119 -116 -113 -110 -107
-104 -101 -98 -95 -92 -89 -86 -83 -104 -101 -98 -95 -92 -89 -86 -83
-104 -101 -98 -95 -92 -89 -86 -83 -104 -101 -98 -95 -92 -89 -86 -83
-104 -101 -98 -95 -92 -89 -86 -83 -104 -101 -98 -95 -92 -89 -86 -83
-104 -101 -98 -95 -92 -89 -86 -83 -104 -101 -98 -95 -92 -89 -86 -83

    DstMCU[1]
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

    DstMCU[2]
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
```

RGBToYCbCr411LS_MCU

Converts an RGB image to the YCbCr color model and creates 411 MCU.

Syntax

```
IppStatus ippiRGBToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrcRGB	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The function `ippiRGBToYCbCr411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function converts an RGB image to the YCbCr image using the same formulas as in `ippiRGBToYCbCr_JPEG` function for computing `Y`, `Cb`, and `Cr` component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 411 MCU (see Figure 6-13).

Downsampling is performed by plain averaging.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

BGRToYCbCr444LS_MCU

Converts a BGR image to the YCbCr color model and creates 444 MCU.

Syntax

```
IppStatus ippiBGRToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrcBGR	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The function `ippiBGRToYCbCr444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function converts a BGR image to the YCbCr image using the same formulas as in `ippiRGBToYCbCr_JPEG` function for computing `Y`, `Cb`, and `Cr` component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 444 MCU (see Figure 6-13).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

BGR565ToYCbCr444LS_MCU, BGR555ToYCbCr444LS_MCU

Convert a BGR image to the YCbCr color model and create 444 MCU.

Syntax

```
IppStatus ippiBGR565ToYCbCr444LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
IppStatus ippiBGR555ToYCbCr444LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrcBGR	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The functions `ippiBGR565ToYCbCr444LS_MCU` and `ippiBGR555ToYCbCr444LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert a source BGR image to the YCbCr 444 MCU (see Figure 6-13). The source image *pSrcBGR* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), or `BGR555` (5 bits for blue, green, red).

Source data are converted to the signed `Ipp16s` range `[-128..127]` by transforming them at first to the unsigned `Ipp8u` range with following level shift operation (by subtracting 128). `Y`, `Cb`, and `Cr` component values are computed using the same formulas as in `ippiRGBToYCbCr_JPEG` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

BGRToYCbCr422LS_MCU

Converts an BGR image to the YCbCr color model and creates 422 MCU.

Syntax

```
IppStatus ippiBGRToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrcBGR	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The function `ippiBGRToYCbCr422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an BGR image to the YCbCr image using the same formulas as in `ippiRGBToYCbCr_JPEG` function for computing `Y`, `Cb`, and `Cr` component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 422 MCU (see Figure 6-13).

Downsampling is performed by plain averaging.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

BGR565ToYCbCr422LS_MCU, BGR555ToYCbCr422LS_MCU

Convert an BGR image to the YCbCr color model and create 422 MCU.

Syntax

```
IppStatus ippiBGR565ToYCbCr422LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
IppStatus ippiBGR555ToYCbCr422LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrcBGR	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The functions `ippiBGR565ToYCbCr422LS_MCU` and `ippiBGR555ToYCbCr422LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert a source BGR image to the YCbCr 422 MCU (see Figure 6-13). The source image *pSrcBGR* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), or `BGR555` (5 bits for blue, green, red).

Source data are converted to the signed `Ipp16s` range `[-128..127]` by transforming them at first to the unsigned `Ipp8u` range with following level shift operation (by subtracting 128). `Y`, `Cb`, and `Cr` component values are computed using the same formulas as in `ippiRGBToYCbCr_JPEG` function.

Downsampling is performed by plain averaging.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.

ippStsStepErr	Indicates an error condition if *srcStep* has a zero or negative value.

BGRToYCbCr411LS_MCU

Converts an BGR image to the YCbCr color model and creates 411 MCU.

Syntax

```
IppStatus ippiBGRToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrcBGR	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The function `ippiBGRToYCbCr411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts an BGR image to the YCbCr image using the same formulas as in `ippiRGBToYCbCr_JPEG` function for computing `Y`, `Cb`, and `Cr` component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 411 MCU (see Figure 6-13).

Downsampling is performed by plain averaging.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is `NULL`.
ippStsStepErr	Indicates an error condition if *srcStep* has a zero or negative value.

BGR565ToYCbCr411LS_MCU, BGR555ToYCbCr411LS_MCU

Convert an BGR image to the YCbCr color model and create 411 MCU.

Syntax

```
IppStatus ippiBGR565ToYCbCr411LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
IppStatus ippiBGR555ToYCbCr411LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrcBGR	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The functions `ippiBGR565ToYCbCr411LS_MCU` and `ippiBGR555ToYCbCr411LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions convert a source BGR image to the YCbCr 411 MCU (see Figure 6-13). The source image *pSrcBGR* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), or `BGR555` (5 bits for blue, green, red).

Source data are converted to the signed `Ipp16s` range `[-128..127]` by transforming them at first to the unsigned `Ipp8u` range with following level shift operation (by subtracting 128). `Y`, `Cb`, and `Cr` component values are computed using the same formulas as in `ippiRGBToYCbCr_JPEG` function.

Downsampling is performed by plain averaging.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.

ippStsStepErr	Indicates an error condition if *srcStep* has a zero or negative value.

CMYKToYCCK444LS_MCU

Converts a CMYK image to the YCCK color model and creates 4444 MCU.

Syntax

```
IppStatus ippiCMYKToYCCK444LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp16s* pDstMCU[4]);
```

Parameters

pSrcCMYK	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 4 pointers to the destination image blocks.

Description

The function `ippiCMYKToYCCK444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a CMYK image to the YCCK image in the same way as the function `ippiCMYKToYCCK_JPEG` does.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 4444 MCU.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is `NULL`.
ippStsStepErr	Indicates an error condition if *srcStep* has a zero or negative value.

CMYKToYCCK422LS_MCU

Converts a CMYK image to the YCCK color model and creates 4224 MCU.

Syntax

```
IppStatus ippiCMYKToYCCK422LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp16s* pDstMCU[4]);
```

Parameters

pSrcCMYK	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 4 pointers to the destination image blocks.

Description

The function `ippiCMYKToYCCK422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts a CMYK image to the YCCK image in the same way as the function `ippiCMYKToYCCK_JPEG` does.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 4224 MCU.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

CMYKToYCCK411LS_MCU

Converts a CMYK image to the YCCK color model and creates 4114 MCU.

Syntax

```
IppStatus ippiCMYKToYCCK411LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp16s* pDstMCU[4]);
```

Parameters

pSrcCMYK	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 4 pointers to the destination image blocks.

Description

The function `ippiCMYKToYCCK411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function converts a CMYK image to the YCCK image in the same way as the function `ippiCMYKToYCCK_JPEG` does.

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 4114 MCU.

Downsampling is performed by plain averaging.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

YCbCr444ToRGBLS_MCU

Creates an YCbCr image from 444 MCU and converts it to the RGB color model.

Syntax

```
IppStatus ippiYCbCr444ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDstRGB, int dstStep);
```

Parameters

pSrcMCU	Array of 3 pointers to the source image blocks.
pDstRGB	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiYCbCr444ToRGBLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function creates YCbCr 8x8 image from 444 MCU and converts it to the RGB format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `R`, `G`, and `B` component values.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCbCr422ToRGBLS_MCU

Creates an YCbCr image from 422 MCU and converts it to the RGB color model.

Syntax

```
IppStatus ippiYCbCr422ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDstRGB, int dstStep);
```

Parameters

pSrcMCU	Array of 3 pointers to the source image blocks.
pDstRGB	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiYCbCr422ToRGBLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function creates YCbCr 16x8 image from 422 MCU and converts it to the RGB format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `R`, `G`, and `B` component values.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Upsampling is performed by replication of the neighbor value.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCbCr411ToRGBLS_MCU

Creates an YCbCr image from 411 MCU and converts it to the RGB color model.

Syntax

```
IppStatus ippiYCbCr411ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDstRGB, int dstStep);
```

Parameters

pSrcMCU	Array of 3 pointers to the source image blocks.
pDstRGB	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiYCbCr411ToRGBLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function creates YCbCr 16x16 image from 411 MCU and converts it to the RGB format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `R`, `G`, and `B` component values.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Upsampling is performed by replication of the neighbor value.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCbCr444ToBGRLS_MCU

Creates an YCbCr image from 444 MCU and converts it to the BGR color model.

Syntax

```
IppStatus ippiYCbCr444ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDstBGR, int dstStep);
```

Parameters

pSrcMCU	Array of 3 pointers to the source image blocks.
pDstBGR	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiYCbCr444ToBGRLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function creates YCbCr 8x8 image from 444 MCU and converts it to the BGR format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `B`, `G`, and `R` component values.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCbCr444ToBGR565LS_MCU, YCbCr444ToBGR555LS_MCU

Create an YCbCr image from 444 MCU and convert it to the BGR color model.

Syntax

```
IppStatusippiYCbCr444ToBGR565LS_MCU_16s16u_P3C3R(constIpp16s*pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
IppStatusippiYCbCr444ToBGR555LS_MCU_16s16u_P3C3R(constIpp16s*pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
```

Parameters

pSrcMCU	Array of 3 pointers to the source image blocks.
pDstBGR	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The functions `ippiYCbCr444ToBGR565LS_MCU` and `ippiYCbCr444ToBGR555LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see Regions of Interest in Intel IPP). These functions create YCbCr 8x8 image from 444 MCU (see Figure 6-13) and convert it to the BGR format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `B`, `G`, and `R` component values.

Additionally, the functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).
The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), or `BGR555` (5 bits for blue, green, red).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCbCr422ToBGRLS_MCU

Creates a YCbCr image from 422 MCU and converts it to the BGR color model.

Syntax

```
IppStatus ippiYCbCr422ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDstBGR, int dstStep);
```

Parameters

pSrcMCU	Array of 3 pointers to the source image blocks.
pDstBGR	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiYCbCr422ToBGRLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function creates YCbCr 16x8 image from 422 MCU (see Figure 6-13) and converts it to the BGR format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `B`, `G`, and `R` component values.Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCbCr422ToBGR565LS_MCU, YCbCr422ToBGR555LS_MCU

Create an YCbCr image from 422 MCU and convert it to the BGR color model.

Syntax

```
IppStatusippiYCbCr422ToBGR565LS_MCU_16s16u_P3C3R(constIpp16s*pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
IppStatusippiYCbCr422ToBGR555LS_MCU_16s16u_P3C3R(constIpp16s*pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
```

Parameters

pSrcMCU	Array of 3 pointers to the source image blocks.
pDstBGR	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The functions `ippiYCbCr444ToBGR565LS_MCU` and `ippiYCbCr444ToBGR555LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see Regions of Interest in Intel IPP). These functions create YCbCr 8x8 image from 422 MCU (see Figure 6-13) and convert it to the BGR format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `B`, `G`, and `R` component values.
Upsampling is performed by replication of the neighbor value.

Additionally, the functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).
The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), or `BGR555` (5 bits for blue, green, red).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCbCr411ToBGRLS_MCU

Creates an YCbCr image from 411 MCU and converts it to the BGR color model.

Syntax

```
IppStatus ippiYCbCr411ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDstBGR, int dstStep);
```

Parameters

pSrcMCU	Array of 3 pointers to the source image blocks.
pDstBGR	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiYCbCr411ToBGRLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function creates YCbCr 16x16 image from 411 MCU and converts it to the BGR format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `B`, `G`, and `R` component values.Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCbCr411ToBGR565LS_MCU, YCbCr411ToBGR555LS_MCU

Create an YCbCr image from 411 MCU and convert it to the BGR color model.

Syntax

```
IppStatusippiYCbCr411ToBGR565LS_MCU_16s16u_P3C3R(constIpp16s*pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
IppStatusippiYCbCr411ToBGR555LS_MCU_16s16u_P3C3R(constIpp16s*pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
```

Parameters

pSrcMCU	Array of 3 pointers to the source image blocks.
pDstBGR	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The functions `ippiYCbCr411ToBGR565LS_MCU` and `ippiYCbCr411ToBGR555LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see Regions of Interest in Intel IPP). These functions create YCbCr 8x8 image from 411 MCU (see Figure 6-13) and convert it to the BGR format using the same formulas as in `ippiYCbCrToRGB_JPEG` function for computing `B`, `G`, and `R` component values.
Upsampling is performed by replication of the neighbor value.

Additionally, the functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).
The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see Figure 6-14), and it can be in one of two possible formats : `BGR565` (5 bits for blue, 6 bits for green, 5 bits for red), or `BGR555` (5 bits for blue, green, red).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCCK444ToCMYKLS_MCU

Creates YCCK image from 4444 MCU and converts it to the CMYK color model.

Syntax

```
IppStatus ippiYCCK444ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4],
    Ipp8u* pDstCMYK, int dstStep);
```

Parameters

pSrcMCU	Array of 4 pointers to the source image blocks.
pDstCVBYK	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiYCCK444ToCMYKLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function creates YCCK 8x8 image from 4444 MCU and converts it to the CMYK color model in the same way as the function `ippiYCCKToCMYK_JPEG` does.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCCK422ToCMYKLS_MCU

Creates an YCCK image from 4224 MCU and converts it to the CMYK color model.

Syntax

```
IppStatus ippiYCCK422ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4],
    Ipp8u* pDstCMYK, int dstStep);
```

Parameters

pSrcMCU	Array of 4 pointers to the source image blocks.
pDstCVBYK	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiYCCK422ToCMYKLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function creates YCCK 16x8 image from 4224 MCU and converts it to the CMYK color model in the same way as the function `ippiYCCKToCMYK_JPEG` does. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

YCCK411ToCMYKLS_MCU

Creates an YCCK image from 4114 MCU and converts it to the CMYK color model.

Syntax

```
IppStatus ippiYCCK411ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4],
    Ipp8u* pDstCMYK, int dstStep);
```

Parameters

pSrcMCU	Array of 4 pointers to the source image blocks.
pDstCVBYK	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiYCCK411ToCMYKLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function creates YCCK 16x16 image from 4114 MCU and converts it to the CMYK color model in the same way as the function `ippiYCCKToCMYK_JPEG` does. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

Quantization Functions

This section describes Intel IPP functions that perform quantization, dequantization and preparation of quantization tables and are specific for JPEG codec. These functions are listed in the Table 15-3.

Table 15-3 Quantization Functions

Function Base Name	Description
QuantFwdRawTableInit_JPEG	Modifies raw quantization tables in accordance with quality factor.
QuantFwdTableInit_JPEG	Prepares quantization tables in a format suitable for an encoder.
QuantFwd8x8_JPEG	Performs quantization of an 8x8 block of DCT coefficients.
QuantInvTableInit_JPEG	Prepares quantization tables in a format suitable for a decoder.
QuantInv8x8_JPEG	Performs dequantization of DCT coefficients in 8x8 block

QuantFwdRawTableInit_JPEG

Modifies raw quantization tables in accordance with the quality factor.

Syntax

```
IppStatus ippiQuantFwdRawTableInit_JPEG_8u(Ipp8u* pQuantRawTable,
    int qualityFactor);
```

Parameters

pQuantRawTable	Pointer to the raw quantization table.
qualityFactor	JPEG quality factor, must be in the range `[1..100]`.

Description

The function ippiQuantFwdRawTableInit_JPEG is declared in the ippj.h file. This function modifies the initial raw quantization table in accordance with the quality factor using the algorithm which is specified in IJG JPEG Library, version 6b. This function is optional for JPEG codec implementation.

Example 15-3 and Example 15-4 show how to use the function ippiQuantFwdRawTableInit_JPEG_8u.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers are NULL.

QuantFwdTableInit_JPEG

Prepares quantization tables suitable for fast encoding.

Syntax

```
IppStatus ippiQuantFwdTableInit_JPEG_8u16u(
    const Ipp8u* pQuantRawTable, Ipp16u* pQuantFwdTable);
```

Parameters

pQuantRawTable	Pointer to the raw quantization table.
pQuantFwdTable	Pointer to the quantization table for the encoder.

Description

The function ippiQuantFwdTableInit_JPEG is declared in the ippj.h file. This function prepares quantization table in a format that is suitable for fast encoding. Initial raw quantization tables have zigzag order by definition. This function performs reordering transformation that converts the zigzag sequence of table elements to conventional order (left-to-right, top-to-bottom).
To avoid division during quantization, the function ippiQuantFwdTableInit_JPEG scales the array by 15 bits.

The pointer *pQuantRawTable* points to the array of 64 elements as is required by [ISO10918], *Annex B.2.4.1*, *Quantization Table Specification Syntax*.
The pointer *pQuantFwdTable* points to an array containing 64 values of Ipp16u type.

Example 15-3 and Example 15-4 show how to use the function ippiQuantFwdTableInit_JPEG_8u16u.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or both of the specified pointers are NULL.

QuantFwd8x8_JPEG

Performs quantization of an 8x8 block of DCT coefficients.

Syntax

```
IppStatus ippiQuantFwd8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*
    pQuantFwdTable);
```

Parameters

pSrcDst	Pointer to the 8x8 block of DCT coefficients.
pQuantFwdTable	Pointer to the quantization table for the encoder.

Description

The function ippiQuantFwd8x8_JPEG is declared in the ippj.h file. This function performs quantization of computed DCT coefficients for an 8x8 block. Quantization is defined in [ISO10918], *Annex A.3.4*, *DCT Coefficient Quantization and Dequantization*, as

$$Sq(u,v) = round(S(u,v)/Q(u,v)),$$

where Sq(u,v) is the quantized DCT coefficient,
S(u,v) is a computed DCT coefficient, and
Q(u,v) is a quantization table element.

Rounding is to the nearest integer. To avoid division, quantization table elements are scaled up by 2^{15}. After that, division operation is replaced by multiplication of DCT coefficients by scaled quantization table elements. Then the products are rescaled back by 15 bits.

Example 15-3 shows how to use the function `ippiQuantFwd8x8_JPEG_16s_C1I`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or both specified pointers are `NULL`.

Example 15-3 Using Intel IPP for Quantization of 8x8 Block of DCT Coefficients

```
Ipp8u pQuantRawTable[64] = {16,  11,  12,  14,  12,  10,  16,  14,
                            13,  14,  18,  17,  16,  19,  24,  40,
                            26,  24,  22,  22,  24,  49,  35,  37,
                            29,  40,  58,  51,  61,  60,  57,  51,
                            56,  55,  64,  72,  92,  78,  64,  68,
                            87,  69,  55,  56,  80, 109,  81,  87,
                            95,  98, 103, 104, 103,  62,  77, 113,
                           121, 112, 100, 120, 92,  101, 103,  99 };

Ipp16s dstFwd[8*8] = { 4,  4, 4,  4,  4,  4,  4, 4,
                       4,  3, 3,  3, -3,  3, -3, 4,
                       4, -3, 2,  0,  0,  2,  3, 4,
                       4,  3, 0,  1, -1,  0, -3, 4,
                       4, -3, 0, -1,  1,  0,  3, 4,
                       4,  3, 2,  0,  0,  2, -3, 4,
                       4, -3, 3, -3,  3, -3,  3, 4,
                       4,  4, 4,  4,  4,  4,  4, 4};

Ipp16u pQuantFwdTable[64];
int quality = 75;

ippiQuantFwdRawTableInit_JPEG_8u ( pQuantRawTable, quality );

ippiQuantFwdTableInit_JPEG_8u16u ( pQuantRawTable, pQuantFwdTable );

ippiQuantFwd8x8_JPEG_16s_C1I ( dstFwd, pQuantFwdTable );

result:
pQuantRawTable (with quality)
8 6 6 7 6 5 8 7
7 7 9 9 8 10 12 20
13 12 11 11 12 25 18 19
15 20 29 26 31 30 29 26
28 28 32 36 46 39 32 34
44 35 28 28 40 55 41 44
48 49 52 52 52 31 39 57
61 56 50 60 46 51 52 50

pQuantFwdTable
4096 5461 6554 4096 2731 1638 1260 1057
5461 5461 4681 3277 2521 1130 1092 1170
4681 4681 4096 2731 1638 1130  936 1170
```

Example 15-3 Using Intel IPP for Quantization of 8x8 Block of DCT Coefficients

```
4681 3641 2979 2185 1260  745  819 1057
3641 2979 1725 1170  964  596  630  840
2731 1820 1170 1024  799  630  575  712
1311 1024  840  745  630  537  546  643
 910  712  683  669  585  655  630  655


dstFwd (after pQuantFwdTable)
0 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

QuantInvTableInit_JPEG

Prepares quantization tables suitable for fast decoding.

Syntax

```
IppStatus ippiQuantInvTableInit_JPEG_8u16u(const Ipp8u* pQuantRawTable,
    Ipp16u* pQuantInvTable);
```

Parameters

pQuantRawTable	Pointer to the raw quantization table.
pQuantInvTable	Pointer to the quantization table for decoding.

Description

The function `ippiQuantInvTableInit_JPEG` is declared in the `ippj.h` file. This function prepares a quantization table for fast decoding. It also performs reordering transformation that converts the zigzag sequence of table elements to conventional order (left-to-right, top-to-bottom) that is more suitable for a decoder. See [ISO10918], *Annex A.3.6*, *Zig-zag Sequence* for more information on zigzag order.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or both of the specified pointers are NULL.

QuantInv8x8_JPEG

Performs dequantization of an 8x8 block of DCT coefficients

Syntax

```
IppStatus ippiQuantInv8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*
    pQuantInvTable);
```

Parameters

pSrcDst	Pointer to the DCT coefficients of the 8x8 block.
pQuantInvTable	Pointer to the quantization table for decoding.

Description

The function ippiQuantInv8x8_JPEG is declared in the ippj.h file. This function performs dequantization of the 8x8 block of DCT coefficients. Dequantization operation is defined in [ISO10918], *Annex A.3.4*, *DCT Coefficient Quantization and Dequantization*.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or both of the specified pointers are NULL.

Combined DCT Functions

This section describes functions that perform Discrete Cosine Transform (DCT) plus some additional operations - level shift, quantization (or dequantization) - and are specific for JPEG codec. These functions are listed in Table 15-4.

Table 15-4 Functions that combine Quantization, DCT, and Level Shift.

Function Base Name	Description
DCTQuantFwd8x8_JPEG	Performs forward DCT and quantization.
DCTQuantFwd8x8LS_JPEG	Performs forward DCT, quantization and level shift.
DCTQuantInv8x8_JPEG	Performs inverse DCT and dequantization.
DCTQuantInv8x8LS_JPEG	Performs inverse DCT, dequantization and level shift.
DCTQuantInv8x8LS_1x1_JPEG DCTQuantInv8x8LS_2x2_JPEG DCTQuantInv8x8LS_4x4_JPEG	Performs fast inverse DCT, dequantization and level shift for 8x8 block with zero high frequency components.
DCTQuantInv8x8To4x4LS_JPEG DCTQuantInv8x8To2x2LS_JPEG	Performs fast inverse DCT, dequantization, level shift and downsampling.

These functions implement component operations in accordance with their definition by [ISO10918], specifically:

- for DCT, as in [ISO10918], *Annex A.3.3*, *FDCT and IDCT*;
- for quantization/dequantization, as in [ISO10918], *Annex A.3.4*, *DCT Coefficient Quantization and Dequantization*;
- for level shift, as in [ISO10918], *Annex A.3.1*, *Level Shift*.

NOTE. The IPP functions described in this section use quantization tables in special format that is suitable for fast encoding/decoding. Initial raw quantization tables must be previously transformed using functions ippiQuantFwdTableInit_JPEG and ippiQuantInvTableInit_JPEG.

DCTQuantFwd8x8_JPEG

Performs forward DCT and quantization of an 8x8 block.

Syntax

```
IppStatus ippiDCTQuantFwd8x8_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s*
    pDst, const Ipp16u* pQuantFwdTable);
IppStatus ippiDCTQuantFwd8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*
    pQuantFwdTable);
```

Parameters

pSrc	Pointer to the 8x8 block in the source image.
pDst	Pointer to the destination 8x8 block of the quantized DCT coefficients.
pSrcDst	Pointer to the source and destination 8x8 block for in-place operation.
pQuantFwdTable	Pointer to the quantization table for the encoder.

Description

The function `ippiDCTQuantFwd8x8_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a forward DCT and quantization of DCT coefficients.

The function uses a quantization table suitable for fast encoding. This table must be prepared using the function `ippiQuantFwdTableInit_JPEG`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.

DCTQuantFwd8x8LS_JPEG

Performs level shift, forward DCT and quantization of an 8x8 block.

Syntax

```
IppStatus ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDst, const Ipp16u* pQuantFwdTable);
IppStatus ippiDCTQuantFwd8x8LS_JPEG_16u16s_C1R(const Ipp16u*  pSrc,
    int srcStep, Ipp16s* pDst, const Ipp32f* pQuantFwdTable);
```

Parameters

pSrc	Pointer to the 8x8 block in the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination 8x8 block of the quantized DCT coefficients.
pQuantFwdTable	Pointer to the quantization table for the encoder.

Description

The function `ippiDCTQuantFwd8x8LS_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a forward DCT, quantization of DCT coefficients, and data conversion. Data from the unsigned `Ipp8u` range `[0..255]` are converted to the signed `Ipp16s` range `[-128..127]`.

The function uses a quantization table suitable for fast encoding. This table must be prepared using the function `ippiQuantFwdTableInit_JPEG`.

Example 15-4 shows how to use the function `ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.

Example 15-4 Using Combined DCT Function

```
Ipp8u pQuantRawTable[64] = { 16,  11,  12,  14,  12,  10,  16,  14,
                             13,  14,  18,  17,  16,  19,  24,  40,
                             26,  24,  22,  22,  24,  49,  35,  37,
                             29,  40,  58,  51,  61,  60,  57,  51,
                             56,  55,  64,  72,  92,  78,  64,  68,
                             87,  69,  55,  56,  80, 109,  81,  87,
                             95,  98, 103, 104, 103,  62,  77, 113,
                            121, 112, 100, 120,  92, 101, 103,  99};
Ipp8u src[8*8] = {  4, 4, 4, 4, 4, 4, 4, 4,
                    4, 3, 3, 3, 3, 3, 3, 4,
                    4, 3, 2, 2, 2, 2, 3, 4,
                    4, 3, 2, 1, 1, 2, 3, 4,
                    4, 3, 2, 1, 1, 2, 3, 4,
                    4, 3, 2, 2, 2, 2, 3, 4,
                    4, 3, 3, 3, 3, 3, 3, 4,
                    4, 4, 4, 4, 4, 4, 4, 4};
Ipp16s dst[64];
Ipp16u pQuantFwdTable[64];
int quality = 75;

ippiQuantFwdRawTableInit_JPEG_8u ( pQuantRawTable, quality );

ippiQuantFwdTableInit_JPEG_8u16u ( pQuantRawTable, pQuantFwdTable );

ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R ( src, 8, dst, pQuantFwdTable );

result:

pQuantRawTable (with quality)
 8  6  6  7  6  5  8  7
 7  7  9  9  8  0 12 20
13 12 11 11 12 25 18 19
15 20 29 26 31 30 29 26
28 28 32 36 46 39 32 34
44 35 28 28 40 55 41 44
48 49 52 52 52 31 39 57
61 56 50 60 46 51 52 50

        pQuantFwdTable
4096 5461 6554 4096 2731 1638 1260 1057
5461 5461 4681 3277 2521 1130 1092 1170
4681 4681 4096 2731 1638 1130  936 1170
4681 3641 2979 2185 1260  745  819 1057
3641 2979 1725 1170  964  596  630  840
2731 1820 1170 1024  799  630  575  712
1311 1024  840  745  630  537  546  643
 910  712  683  669  585  655  630  655

    dst
-125 0 1 0 0 0 0 0
```

Example 15-4 Using Combined DCT Function

```
0  0  0  0  0  0  0  0
1  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
```

DCTQuantInv8x8_JPEG

Performs dequantization and inverse DCT of an 8x8 block.

Syntax

```
IppStatus ippiDCTQuantInv8x8_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s*
    pDst, const Ipp16u* pQuantInvTable);
IppStatus ippiDCTQuantInv8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*
    pQuantInvTable);
```

Parameters

pSrc	Pointer to the 8x8 block of quantized DCT coefficients.
pDst	Pointer to the 8x8 block in the destination image.
pSrcDst	Pointer to the source and destination 8x8 block for in-place operation.
pQuantInvTable	Pointer to the quantization table for the decoder.

Description

The function `ippiDCTQuantInv8x8_JPEG` is declared in the `ippj.h` file. This function performs an inverse DCT and dequantization of DCT coefficients.

The function uses a quantization table suitable for fast decoding. This table must be prepared using the function `ippiQuantInvTableInit_JPEG`.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error condition when any of the specified pointers is NULL.

DCTQuantInv8x8LS_JPEG

Performs dequantization, inverse DCT and level shift of an 8x8 block.

Syntax

```
IppStatus ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R(const Ipp16s* pSrc, Ipp8u*
    pDst, int dstStep, const Ipp16u* pQuantInvTable);
IppStatus ippiDCTQuantInv8x8LS_JPEG_16s16u_C1R(const Ipp16s* pSrc,
    Ipp16u*  pDst, int dstStep, const Ipp32f* pQuantInvTable);
```

Parameters

pSrc	Pointer to the 8x8 block of quantized DCT coefficients.
pDst	Pointer to the 8x8 block in the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pQuantInvTable	Pointer to the quantization table for the decoder.

Description

The function ippiDCTQuantInv8x8LS_JPEG is declared in the ippj.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs an inverse DCT, dequantization of DCT coefficients and data conversion. The function ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R converts data from the signed Ipp16s range [-128..127] to the unsigned Ipp8u range [0..255].

The function uses a quantization table suitable for fast decoding. This table must be prepared using the function ippiQuantInvTableInit_JPEG.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.

DCTQuantInv8x8LS_1x1_JPEG
DCTQuantInv8x8LS_2x2_JPEG
DCTQuantInv8x8LS_4x4_JPEG

Performs dequantization, inverse DCT and level shift of an 8x8 block with zero high frequency components.

Syntax

```
IppStatus ippiDCTQuantInv8x8LS_4x4_JPEG_16s8u_C1R(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstStep, const Ipp16u* pQuantInvTable);
IppStatus ippiDCTQuantInv8x8LS_2x2_JPEG_16s8u_C1R(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstStep, const Ipp16u* pQuantInvTable);
IppStatus ippiDCTQuantInv8x8LS_1x1_JPEG_16s8u_C1R(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstStep, const Ipp16u* pQuantInvTable);
```

Parameters

pSrc	Pointer to the 8x8 block of quantized DCT coefficients.
pDst	Pointer to the 8x8 block in the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pQuantInvTable	Pointer to the quantization table for the decoder.

Description

These functions are declared in the `ippj.h` file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions perform fast inverse DCT of an 8x8 block of coefficients containing non-zero components exclusively in the specified top-left part of the block. All coefficients outside the specified parts are equal to 0. The function `ippiDCTQuantInv8x8LS_4x4` operates with coefficients in the 4x4 top-left quadrant of the block, `ippiDCTQuantInv8x8LS_2x2` operates with coefficients in the 2x2 top-left part of the block, `ippiDCTQuantInv8x8LS_1x1` operates with only DC non-zero coefficient. All functions perform dequantization of DCT coefficients and data conversion. The functions converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]`.

The function uses a quantization table suitable for fast decoding. This table must be prepared using the function ippiQuantInvTableInit_JPEG.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.

DCTQuantInv8x8To4x4LS_JPEG
DCTQuantInv8x8To2x2LS_JPEG

Performs dequantization, inverse DCT, level shift, and downsampling of an 8x8 block.

Syntax

```
IppStatus ippiDCTQuantInv8x8To4x4LS_JPEG_16s8u_C1R(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstStep, const Ipp16u* pQuantInvTable);
IppStatus ippiDCTQuantInv8x8To2x2LS_JPEG_16s8u_C1R(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstStep, const Ipp16u* pQuantInvTable);
```

Parameters

pSrc	Pointer to the 8x8 block of quantized DCT coefficients.
pDst	Pointer to the block in the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pQuantInvTable	Pointer to the quantization table for the decoder.

Description

These functions are declared in the ippj.h file. They operate with ROI (see Regions of Interest in Intel IPP).

These functions perform an inverse DCT, dequantization of DCT coefficients, downsamling and data conversion. The function ippiDCTQuantInv8x8To4x4LS_JPEG perfroms downsampling with factor 1:2, that is the output block contains 4x4 elements. The function ippiDCTQuantInv8x8To2x2LS_JPEG perfroms downsampling with factor 1:4, that is the output block contains 2x2 elements.

The functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]`.

The function uses a quantization table suitable for fast decoding. This table must be prepared using the function `ippiQuantInvTableInit_JPEG`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.

Level Shift Functions

This section describes the Intel IPP functions that perform level shift operation and are specific for JPEG codec. These functions are listed in Table 15-5.

Table 15-5 Level Shift Functions

Function Base Name	Description
Sub128_JPEG	Performs level shift to the signed representation.
Add128_JPEG	Performs level shift to the unsigned representation.

The level shift operation is implemented in accordance with its definition in [ISO10918], *Annex A.3.1*, *Level Shift*.

Sub128_JPEG

Converts data from the unsigned `8u` *range to the signed* `16s` *range.*

Syntax

```
IppStatus ippiSub128_JPEG_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiSub128_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts every sample of the source image ROI from the unsigned Ipp8u range `[0..255]` to the signed Ipp16s range `[-128..127]`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

Add128_JPEG

Restores samples to the unsigned representation.

Syntax

```
IppStatus ippiAdd128_JPEG_16s8u_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiAdd128_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts every sample of source image from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or both of the specified pointers are `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`roiSize`* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *`srcStep`* or *`dstStep`* has a zero or negative value.

Sampling Functions

This section describes Intel IPP functions that perform sampling and are specific for JPEG codec. These functions are listed in Table 15-6.

Table 15-6 Sampling Functions

Function Base Name	Description
`SampleDownH2V1_JPEG`	Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image.
`SampleDownH2V2_JPEG`	Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image.
`SampleDownRowH2V1_Box_JPEG`	Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row.
`SampleDownRowH2V2_Box_JPEG`	Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row.
`SampleUpH2V1_JPEG`	Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image.
`SampleUpH2V2_JPEG`	Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image.
`SampleUpRowH2V1_Triangle_JPEG`	Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row.
`SampleUpRowH2V2_Triangle_JPEG`	Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row.
`SampleDown444LS_MCU`	Creates 444 MCU with level shift

Table 15-6 Sampling Functions (continued)

Function Base Name	Description
SampleDown422LS_MCU	Creates 422 MCU with level shift
SampleDown411LS_MCU	Creates 411 MCU with level shift
SampleUp444LS_MCU	Creates pixel-order image from 444 MCU with level shift
SampleUp422LS_MCU	Creates pixel-order image from 422 MCU with level shift
SampleUp411LS_MCU	Creates pixel-order image from 411 MCU with level shift

SampleDownH2V1_JPEG

Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image.

Syntax

```
IppStatus ippiSampleDownH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcSize	Size of the source ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstSize	Size of the destination ROI in pixels.

Description

The function `ippiSampleDownH2V1_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs 2:1 horizontal sampling and 1:1 vertical sampling of an image. Downsampling is performed as a simple “box” filter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or both of the specified pointers are NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value

SampleDownH2V2_JPEG

Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image.

Syntax

```
IppStatus ippiSampleDownH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcSize	Size of the source ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstSize	Size of the destination ROI in pixels.

Description

The function ippiSampleDownH2V2_JPEG is declared in the ippj.h file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs 2:1 horizontal sampling and 2:1 vertical sampling of an image. Downsampling is performed as a simple “box” filter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or both of the specified pointers are NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

SampleDownRowH2V1_Box_JPEG

Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row.

Syntax

```
IppStatus ippiSampleDownRowH2V1_Box_JPEG_8u_C1(const Ipp8u* pSrc,
    int srcWidth, Ipp8u* pDst);
```

Parameters

pSrc	Pointer to the source image row.
pDst	Pointer to the destination image row.
srcWidth	Width of the source row in pixels.

Description

The function ippiSampleDownRowH2V1_Box_JPEG is declared in the ippj.h file. This function performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row. Downsampling is performed as a simple “box” filter.

Example 15-5 shows how to use the function ippiSampleDownRowH2V1_Box_JPEG_8u_C1.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or both of the specified pointers are NULL.

ippStsSizeErr	Indicates an error condition if *srcWidth* has a zero or negative value.

Example 15-5 Using the function ippiSampleDownRowH2V1_Box_JPEG

```
void func_sampling()
{
    Ipp8u pSrc[8*8];
    IppiSize srcSize = {8, 8};
    int srcStep = 8*sizeof(Ipp8u);
    int srcWidth = 8;

    Ipp8u pDst[8*8];

    ippiImageJaehne_8u_C1R( pSrc, srcStep, srcSize);

    for(int r = 0; r < 8; r++)
    {
        ippiSampleDownRowH2V1_Box_JPEG_8u_C1( &pSrc[8*r], srcWidth, &pDst[4*r]);
    }
}
Result:
            pSrc (8x8)                                  pDst (8x4)
  0   67  165  209  209  165   67    0              34  187  187   33
 67  209  255  250  250  255  209   67             138  252  253  138
165  255  226  188  188  226  255  165             210  207  207  210
209  250  188  140  140  188  250  209             230  164  164  229
209  250  188  140  140  188  250  209             230  164  164  229
165  255  226  188  188  226  255  165             210  207  207  210
 67  209  255  250  250  255  209   67             138  252  253  138
  0   67  165  209  209  165   67    0              34  187  187   33
```

SampleDownRowH2V2_Box_JPEG

Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row.

Syntax

```
IppStatus ippiSampleDownRowH2V2_Box_JPEG_8u_C1(const Ipp8u* pSrc1,
    const Ipp8u* pSrc2, int srcWidth, Ipp8u* pDst);
```

Parameters

pSrc1	Pointer to the source image row.
pSrc2	Pointer to the subsequent source image row.

pDst	Pointer to the destination image row.
srcWidth	Width of the source row in pixels.

Description

The function `ippiSampleDownRowH2V2_Box_JPEG` is declared in the `ippj.h` file. This function performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row. Downsampling is performed as a simple “box” filter.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcWidth* has a zero or negative value.

SampleUpH2V1_JPEG

Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image.

Syntax

```
IppStatus ippiSampleUpH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

Parameters

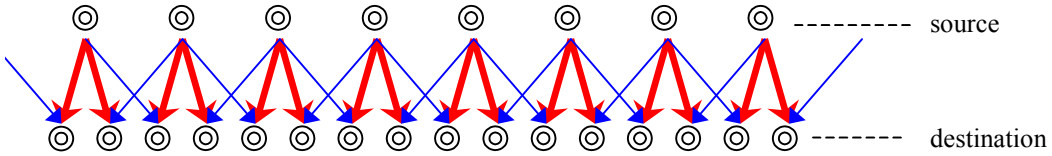
pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcSize	Size of the source ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstSize	Size of the destination ROI in pixels.

Description

The function `ippiSampleUpH2V1_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs 1:2 horizontal sampling and 1:1 vertical sampling of an image. Sampling is performed following the scheme of a "triangle" filter as shown in Figure 15-1 below:

Figure 15-1 Triangle Sampling Scheme



Here thick red lines denote source sample weight ¾ and thin blue lines denote source sample weight ¼ in their contribution to the destination sample value.

NOTE. This function requires that some samples outside the source image ROI are defined. Specifically, the left-most column must be present and the right-most column may be required.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or both of the specified pointers are `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`roiSize`* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if *`srcStep`* or *`dstStep`* has a zero or negative value

SampleUpH2V2_JPEG

Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image.

Syntax

```
IppStatus ippiSampleUpH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcSize	Size of the source ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstSize	Size of the destination ROI in pixels.

Description

The function `ippiSampleUpH2V2_JPEG` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs 1:2 horizontal sampling and 1:2 vertical sampling of an image. Sampling is performed following the scheme of a “triangle” filter as shown in Figure 15-1.

NOTE. This function requires that some samples outside the source image ROI are defined. Specifically, the left-most column and the top-most row must be present, whereas the right-most column and the bottom-most row may be required.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or both of the specified pointers are `NULL`.

ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.

SampleUpRowH2V1_Triangle_JPEG

Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row.

Syntax

```
IppStatus ippiSampleUpRowH2V1_Triangle_JPEG_8u_C1(const Ipp8u* pSrc,
    int srcWidth, Ipp8u* pDst);
```

Parameters

pSrc	Pointer to the source image row.
pDst	Pointer to the destination image row.
srcWidth	Width of the source row in pixels.

Description

The function `ippiSampleUpRowH2V1_Triangle_JPEG` is declared in the `ippj.h` file. This function performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row. Sampling is performed following the scheme of a “triangle” filter as shown in Figure 15-1. This function does not require samples outside the source image row.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or both of the specified pointers are NULL.
ippStsSizeErr	Indicates an error condition if *srcWidth* has a zero or negative value.

SampleUpRowH2V2_Triangle_JPEG

Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row.

Syntax

```
IppStatus ippiSampleUpRowH2V2_Triangle_JPEG_8u_C1(const Ipp8u* pSrc1,
    const Ipp8u* pSrc2, int srcWidth, Ipp8u* pDst);
```

Parameters

pSrc1	Pointer to the source image row.
pSrc2	Pointer to the subsequent source image row.
pDst	Pointer to the destination image row.
srcWidth	Width of the source row in pixels.

Description

The function `ippiSampleUpRowH2V2_Triangle_JPEG` is declared in the `ippj.h` file. This function performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row. Sampling is performed following the scheme of a “triangle” filter as shown in Figure 15-1. This function does not require samples outside the source image rows.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all specified pointers are `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcWidth* has a zero or negative value.

SampleDown444LS_MCU

Creates 444 MCU with level shift from pixel-order data.

Syntax

```
IppStatus ippiSampleDown444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Array of 3 pointers to the destination image blocks.

Description

The function `ippiSampleDown444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts interleaved data to the 444 MCU that consists of three 8x8 blocks - one block for each channel (see Figure 6-13).

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting `128`).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers are `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

SampleDown422LS_MCU

Creates 422 MCU with level shift from pixel-order data.

Syntax

```
IppStatus ippiSampleDown422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Pointer to the array of 3 pointers to the destination blocks.

Description

The function `ippiSampleDown422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts interleaved data to the full MCU with 4:2:2 downsampling. Thus, final MCU consists of four blocks - two 8x8 blocks for channel 1, one 8x8 block for channel 2, and one 8x8 block for channel 3 (see Figure 6-13).

Downsampling is performed by averaging the corresponding pixel values.

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting `128`).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers are `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

SampleDown411LS_MCU

Creates 411 MCU with level shift from pixel-order data.

Syntax

```
IppStatus ippiSampleDown411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDstMCU[3]);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Pointer to the array of 3 pointers to the destination blocks.

Description

The function `ippiSampleDown411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function converts interleaved data to the full MCU with 4:1:1 downsampling. Thus, final MCU consists of six blocks - four 8x8 blocks for channel 1, one 8x8 block for channel 2, and one 8x8 block for channel 3 (see Figure 6-13).

Downsampling is performed by averaging the corresponding pixel values.

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers are `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

SampleUp444LS_MCU

Creates pixel-order image from 444 MCU and performs level shift.

Syntax

```
IppStatus ippiSampleUp444LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDst, int dstStep);
```

Parameters

pSrcMCU	Pointer to the array of 3 pointers to the source blocks.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiSampleUp444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates pixel-order image from 444 MCU (see Figure 6-13).

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

SampleUp422LS_MCU

Creates pixel-order image from 422 MCU and performs level shift.

Syntax

```
IppStatus ippiSampleUp422LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDst, int dstStep);
```

Parameters

pSrcMCU	Pointer to the array of 3 pointers to the source blocks.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiSampleUp422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates pixel-order image from 422 MCU (see Figure 6-13). Upsampling is performed as a simple box filter.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

SampleUp411LS_MCU

Creates pixel-order image from 411 MCU and performs level shift.

Syntax

```
IppStatus ippiSampleUp411LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp8u* pDst, int dstStep);
```

Parameters

pSrcMCU	Pointer to the array of 3 pointers to the source blocks.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiSampleUp411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates pixel-order image from 411 MCU (see Figure 6-13). Upsampling is performed as a simple box filter.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *dstStep* has a zero or negative value.

Planar-to-Pixel and Pixel-to-Planar Conversion Functions

This section describes the functions that convert planar (not-interleaved) data to pixel-order (interleaved) data and vice versa. Table 15-7 lists these functions described in more detail later in this section.

Table 15-7 Pixel-to-Planar and Planar-to-Pixel Conversion Functions

Function Base Name	Description
Split422LS_MCU	Creates 422 MCU from 422 interleaved data with level shift.
Join422LS_MCU	Creates 422 interleaved data from 422 MCU with level shift.

Split422LS_MCU

Creates 422 MCU from 422 interleaved data with level shift.

Syntax

```
IppStatus ippiSplit422LS_MCU_8u16s_C2P3R(const Ipp8u* pSrc, int SrcStep,
    Ipp16s* pDstMCU[3]);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstMCU	Pointer to the array of 3 pointers to the destination blocks.

Description

The function `ippiSplit422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates 422 MCU from 422 pixel-order data (see Figure 6-15), that is, converts pixel-order data to planar form without re-sampling. Final MCU consists of four blocks - two 8x8 blocks for channel 1, one 8x8 block for channel 2, and one 8x8 block for channel 3 (see Figure 6-13).

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers are `NULL`.
`ippStsStepErr`	Indicates an error condition if *srcStep* has a zero or negative value.

Join422LS_MCU

Creates 422 interleaved data from 422 MCU with level shift.

Syntax

```
IppStatus ippiJoin422LS_MCU_16s8u_P3C2R(const Ipp16s* pSrcMCU[3], Ipp8u*
    pDst, int dstStep);
```

Parameters

pSrcMCU	Pointer to the array of 3 pointers to the source blocks.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

Description

The function `ippiJoin422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates 422 interleaved data (see Table 6-2) from 422 MCU (see Figure 6-13), that is, converts planar data to pixel-order form without re-sampling. Output data is a repetitive sequence of pixels Ch1Ch2Ch1Ch3...

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers are `NULL`.
`ippStsStepErr`	Indicates an error condition if *`dstStep`* has a zero or negative value.

Huffman Codec Functions

This section describes Huffman encoding and decoding functions that are specific for JPEG codec. These functions are listed in Table 15-8.

Table 15-8 Huffman Codec Functions

Function Base Name	Description
Encoder Functions	
`EncodeHuffmanRawTableInit_JPEG`	Creates raw Huffman tables using Huffman symbols statistics.
`EncodeHuffmanSpecGetBufSize_JPEG`	Returns the length of the `IppiEncodeHuffmanSpec` structure.
`EncodeHuffmanSpecInit_JPEG`	Creates Huffman table in format that is suitable for an encoder.
`EncodeHuffmanSpecInitAlloc_JPEG`	Allocates memory and creates Huffman table in format that is suitable for an encoder.
`EncodeHuffmanSpecFree_JPEG`	Frees memory allocated by the `ippiEncodeHuffmanSpecInitAlloc_JPEG_8u` function.
`EncodeHuffmanStateGetBufSize_JPEG`	Returns the length of the `IppiEncodeHuffmanState` structure.
`EncodeHuffmanStateInit_JPEG`	Initializes the IppiEncodeHuffmanState structure.
`EncodeHuffmanStateInitAlloc_JPEG`	Allocates memory and initializes the `IppiEncodeHuffmanState` structure.
`EncodeHuffmanStateFree_JPEG`	Frees memory allocated by the `ippiEncodeHuffmanStateInitAlloc_JPEG` function.

Table 15-8 Huffman Codec Functions (continued)

EncodeHuffman8x8_JPEG	Performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.
EncodeHuffman8x8_Direct_JPEG	Directly performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.
GetHuffmanStatistics8x8_JPEG	Computes Huffman symbols statistics for the baseline encoding.
GetHuffmanStatistics8x8_DCFirst_JPEG	Computes Huffman symbols statistics for the progressive encoding (DC coefficient).
GetHuffmanStatistics8x8_ACFirst_JPEG	Computes Huffman symbols statistics for the progressive encoding (AC coefficients, the first scan).
GetHuffmanStatistics8x8_ACRefine_JPEG	Computes Huffman symbols statistics for the progressive encoding (AC coefficients, subsequent scans).
EncodeHuffman8x8_DCFirst_JPEG	Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).
EncodeHuffman8x8_DCRefine_JPEG	Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).
EncodeHuffman8x8_ACFirst_JPEG	Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).
EncodeHuffman8x8_ACRefine_JPEG	Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).
Decoder Functions	
DecodeHuffmanSpecGetBufSize_JPEG	Returns the length of the IppiDecodeHuffmanSpec structure.
DecodeHuffmanSpecInit_JPEG	Creates Huffman table in format that is suitable for a decoder.
DecodeHuffmanSpecInitAlloc_JPEG	Allocates memory and creates Huffman table in format that is suitable for a decoder.
DecodeHuffmanSpecFree_JPEG	Frees memory allocated by the ippiDecodeHuffmanSpecInitAlloc_JPEG function.
DecodeHuffmanStateGetBufSize_JPEG	Returns the length of the IppiDecodeHuffmanState structure
DecodeHuffmanStateInit_JPEG	Initializes the IppiDecodeHuffmanState structure.

Table 15-8 Huffman Codec Functions (continued)

DecodeHuffmanStateInitAlloc_JPEG	Allocates memory and initializes the IppiDecodeHuffmanState structure.
DecodeHuffmanStateFree_JPEG	Frees memory allocated by the ippiDecodeHuffmanStateInitAlloc_JPEG function.
DecodeHuffman8x8_JPEG	Performs Huffman baseline decoding of 8x8 block of the quantized DCT coefficients.
DecodeHuffman8x8_Direct_JPEG	Directly performs Huffman baseline decoding of 8x8 block of the quantized DCT coefficients.
DecodeHuffman8x8_DCFirst_JPEG	Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).
DecodeHuffman8x8_DCRefine_JPEG	Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).
DecodeHuffman8x8_ACFirst_JPEG	Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).
DecodeHuffman8x8_ACRefine_JPEG	Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).

EncodeHuffmanRawTableInit_JPEG

Creates raw Huffman tables using Huffman symbols statistics.

Syntax

```
IppStatusippiEncodeHuffmanRawTableInit_JPEG_8u(constintpStatistics[256],
    Ipp8u* pListBits, Ipp8u* pListVals);
```

Parameters

pStatistics	Pointer to the buffer that contains Huffman symbol statistics.
pListBits	Pointer to the Bits list.
pListVals	Pointer to the Vals list.

Description

The function `ippiEncodeHuffmanRawTableInit_JPEG` is declared in the `ippj.h` file. This function builds raw Huffman tables using the previously computed Huffman symbols statistics. An array for statistics is indexed with Huffman symbol value. The values in the array are frequencies of a given symbol.

The `Bits` and `Vals` lists are specified in [ISO10918], *Annex B, Figure B.7.*

To generate `Bits` and `Vals` lists, the function uses the procedure that is described in [ISO10918], *Annex K.2, A Procedure for Generating the Lists, Which Specify a Huffman Code Table.*

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all specified pointers are `NULL`.

EncodeHuffmanSpecGetBufSize_JPEG

Returns the length of the `IppiEncodeHuffmanSpec` *structure.*

Syntax

```
IppStatus ippiEncodeHuffmanSpecGetBufSize_JPEG_8u(int* pSize);
```

Parameters

pSize	Pointer to a variable that will receive the length of the `IppiEncodeHuffmanSpec` structure.

Description

The function `ippiEncodeHuffmanSpecGetBufSize_JPEG` is declared in the `ippj.h` file. This function computes the length of the `IppiEncodeHuffmanSpec` structure.

NOTE. The fields of the structure `IppiEncodeHuffmanSpec` are hidden from the user.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if the pointer is `NULL`.

EncodeHuffmanSpecInit_JPEG

Creates Huffman table in a format that is suitable for an encoder.

Syntax

```
IppStatus ippiEncodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits,
    const Ipp8u* pListVals, IppiEncodeHuffmanSpec* pEncHuffSpec);
```

Parameters

pListBits	Pointer to `Bits` list.
pListVals	Pointer to `Vals` list.
pEncHuffSpec	Pointer to the Huffman table for the encoder.

Description

The function `ippiEncodeHuffmanSpecInit_JPEG` is declared in the `ippj.h` file. This function creates Huffman table in a format that is suitable for the encoder.
The `Bits` and `Vals` lists are specified in [ISO10918], *Annex C, Huffman table specification.* To generate the table, the function uses the procedure that is described in [ISO10918], *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths.*

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or all specified pointers are NULL.
ippStsJPEGHuffTableErr	Indicates an error condition during the initialization of the table. It may happen when raw tables contain inadmissible values.

EncodeHuffmanSpecInitAlloc_JPEG

Allocates memory and creates Huffman table in a format that is suitable for an encoder.

Syntax

```
IppStatus ippiEncodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u* pListBits,
    const Ipp8u* pListVals, IppiEncodeHuffmanSpec** ppEncHuffSpec);
```

Parameters

pListBits	Pointer to the Bits list.
pListVals	Pointer to the Vals list.
ppEncHuffSpec	Pointer to the returned pointer to the Huffman table for the coder.

Description

The function ippiEncodeHuffmanSpecInitAlloc_JPEG is declared in the ippj.h file. This function allocates memory and creates Huffman table in a format that is suitable for the encoder.
The Bits and Vals lists are specified in [ISO10918], *Annex C, Huffman table specification.*
To generate the table, the function uses the procedure that is described in [ISO10918], *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths.*

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or all specified pointers are NULL.

`ippStsJPEGHuffTableErr`	Indicates an error condition when the table is initialized. It may happen when raw tables contain inadmissible values.

EncodeHuffmanSpecFree_JPEG

Frees memory allocated by the `ippiEncodeHuffmanSpecInitAlloc_JPEG_8u` *function.*

Syntax

```
IppStatus ippiEncodeHuffmanSpecFree_JPEG_8u(
    IppiEncodeHuffmanSpec*pEncHuffSpec);
```

Parameters

pEncHuffSpec	Pointer to the Huffman table for the encoder.

Description

The function `ippiEncodeHuffmanSpecFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated by the `ippiEncodeHuffmanSpecInitAlloc_JPEG` function for the Huffman table.

Return Values

`ippStsNoErr`	Indicates no error.

EncodeHuffmanStateGetBufSize_JPEG

Returns the length of the `IppiEncodeHuffmanState` *structure.*

```
IppStatus ippiEncodeHuffmanStateGetBufSize_JPEG_8u(int* pSize);
```

Parameters

pSize	Pointer to a variable that will receive the length of the `IppiEncodeHuffmanState` structure.

Description

The function `ippiEncodeHuffmanStateGetBufSize_JPEG` is declared in the `ippj.h` file. This function computes the length (in bytes) of the `IppiEncodeHuffmanState` structure.

NOTE. The fields of the structure `IppiEncodeHuffmanState` are hidden from the user.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if specified pointer is `NULL`.

EncodeHuffmanStateInit_JPEG

Initializes the `IppiEncodeHuffmanState` *structure.*

Syntax

```
IppStatus ippiEncodeHuffmanStateInit_JPEG_8u( IppiEncodeHuffmanState*
    pEncHuffState);
```

Parameters

pEncHuffState	Pointer to the `IppiEncodeHuffmanState` structure.

Description

The function `ippiEncodeHuffmanStateInit_JPEG` is declared in the `ippj.h` file. This function initializes the `IppiEncodeHuffmanState` to the initial state. This function must be called prior to the JPEG data stream encoding.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if one or all of the specified pointers are NULL.

EncodeHuffmanStateInitAlloc_JPEG

Allocates memory and initializes the `IppiEncodeHuffmanState` *structure.*

Syntax

```
IppStatus ippiEncodeHuffmanStateInitAlloc_JPEG_8u(IppiEncodeHuffmanState**
    ppEncHuffState);
```

Parameters

ppEncHuffState	Pointer to the IppiEncodeHuffmanState structure.

Description

The function ippiEncodeHuffmanStateInitAlloc_JPEG is declared in the ippj.h file. This function allocates memory and initializes the IppiEncodeHuffmanState structure to the initial state. This function must be called prior to the JPEG data stream encoding.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or all of the specified pointers are NULL.

EncodeHuffmanStateFree_JPEG

Frees memory allocated by the `ippiEncodeHuffmanStateInitAlloc_JPEG` *function.*

Syntax

```
IppStatus ippiEncodeHuffmanStateFree_JPEG_8u(IppiEncodeHuffmanState*
    pEncHuffState);
```

pEncHuffState	Pointer to the `IppiEncodeHuffmanState` structure.

Description

The function `ippiEncodeHuffmanStateFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated for the `IppiEncodeHuffmanState` structure.

Return Values

`ippStsNoErr`	Indicates no error.

EncodeHuffman8x8_JPEG

Performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.

Syntax

```
IppStatus ippiEncodeHuffman8x8_JPEG_16s1u_C1(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, Ipp16s* pLastDC,
    const IppiEncodeHuffmanSpec* pDcTable, const IppiEncodeHuffmanSpec*
    pAcTable, IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

Parameters

pSrc	Pointer to the 8x8 block of the quantized DCT coefficients.
pDst	Pointer to the buffer for the output bit stream.
dstLenBytes	Length in bytes of the buffer for the bit stream.
pDstCurrPos	Shift in bytes of the current byte in the output buffer.
pLastDC	Pointer to the DC coefficient of the previous 8x8 block.
pDcTable	Pointer to the Huffman table for the DC coefficients.
pAcTable	Pointer to the Huffman table for the AC coefficients.
pEncHuffState	Pointer to the `IppiEncodeHuffmanState` structure.
bFlushState	Set to 1 for the last 8x8 block in the scan.

Description

The function `ippiEncodeHuffman8x8_JPEG` is declared in the `ippj.h` file. This function performs encoding of an 8x8 block of the quantized DCT coefficients using *pDcTable* and *pAcTable* tables. Encoding procedure is specified in [ISO10918], *Annex F.1.2, Baseline*

Huffman Encoding Procedures.
Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter *bFlushState* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *dstLenBytes* has a zero or negative value, or *pDstCurrPos* is outside the limit of *dstLenBytes*.
`ippStsJPEGDCTRangeErr`	Indicates an error condition if a DCT coefficient is out of the allowed range `(-1023..1023)`.

EncodeHuffman8x8_Direct_JPEG

Directly performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.

Syntax

```
IppStatus ippiEncodeHuffman8x8_Direct_JPEG_16s1u_C1(const Ipp16s* pSrc,
    Ipp8u* pDst, int* pDstBitsLen, Ipp16s* pLastDC,
    const IppiEncodeHuffmanSpec* pDcTable,
    const IppiEncodeHuffmanSpec*  pAcTable);
```

Parameters

pSrc	Pointer to the 8x8 block of the quantized DCT coefficients.
pDst	Pointer to the buffer for the output bit stream.
pDstBitsLen	Length in bits of the buffer for the bit stream.
pLastDC	Pointer to the DC coefficient of the previous 8x8 block of the same color component.
pDcTable	Pointer to the Huffman table for the DC coefficients.
pAcTable	Pointer to the Huffman table for the AC coefficients.

Description

The function `ippiEncodeHuffman8x8_Direct_JPEG` is declared in the `ippj.h` file. This function directly performs encoding of an 8x8 block of the quantized DCT coefficients using *pDcTable* and *pAcTable* tables. The function does not require any additional structure for operation.
Encoding procedure is specified in [ISO10918], *Annex F.1.2, Baseline Huffman Encoding Procedures*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.

GetHuffmanStatistics8x8_JPEG

Computes Huffman symbols statistics for the baseline encoding.

Syntax

```
IppStatus ippiGetHuffmanStatistics8x8_JPEG_16s_C1(const Ipp16s* pSrc,
    int pDcStatistics[256], int pAcStatistics[256],  Ipp16s* pLastDC);
```

Parameters

pSrc	Pointer to the 8x8 block of the quantized DCT coefficients.
pDcStatistics	Pointer to the Huffman symbols statistics buffer for the DC coefficients.
pAcStatistics	Pointer to the Huffman symbols statistics buffer for the AC coefficients.
pLastDC	Pointer to the DC coefficient of the previous 8x8 block.

Description

The function `ippiGetHuffmanStatistics8x8_JPEG` is declared in the `ippj.h` file. This function computes the statistics of Huffman symbols for an 8x8 block of the quantized DCT coefficients for the baseline encoding.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or both of the specified pointers are NULL.
ippStsJPEGDCTRangeErr	Indicates an error condition if a DCT coefficient is out of the allowed range (-1023..1023).

GetHuffmanStatistics8x8_DCFirst_JPEG

Computes Huffman symbols statistics for the progressive encoding (DC coefficients).

Syntax

```
IppStatus ippiGetHuffmanStatistics8x8_DCFirst_JPEG_16s_C1(const Ipp16s* pSrc,
    int pDcStatistics[256], Ipp16s* pLastDC, int Al);
```

Parameters

pSrc	Pointer to the 8x8 block of the quantized DCT coefficients.
pDcStatistics	Pointer to the Huffman symbols statistics buffer for the DC coefficients.
pLastDC	Pointer to the DC coefficient of the previous 8x8 block.
Al	Successive approximation bit position low, it specifies the actual point transform.

Description

The function ippiGetHuffmanStatistics8x8_DCFirst_JPEG is declared in the ippj.h file. This function computes the Huffman symbols statistics for an 8x8 block of the quantized DCT coefficients for progressive encoding, the first scan, DC coefficients.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or both of the specified pointers are NULL.
ippStsJPEGDCTRangeErr	Indicates an error condition if a DCT coefficient is out of the allowed range (-1023..1023).

GetHuffmanStatistics8x8_ACFirst_JPEG

Computes Huffman symbols statistics for the progressive encoding (AC coefficients, the first scan).

Syntax

```
IppStatus ippiGetHuffmanStatistics8x8_ACFirst_JPEG_16s_C1(const Ipp16s* pSrc,
    int pAcStatistics[256], int Ss,  int Se, int Al, IppiEncodeHuffmanState*
    pEncHuffState, int bFlushState);
```

Parameters

pSrc	Pointer to the 8x8 block of the quantized DCT coefficients.
pAcStatistics	Pointer to the Huffman symbols statistics buffer for the AC coefficients.
Ss	Spectral selection start index.
Se	Spectral selection end index.
Al	Successive approximation bit positions low, it specifies the actual point transform.
pEncHuffState	Pointer to the `IppiEncodeHuffmanState` structure.
bFlushState	Set to 1 for the last 8x8 block in the scan.

Description

The function `ippiGetHuffmanStatistics8x8_ACFirst_JPEG` is declared in the `ippj.h` file. This function computes the Huffman symbols statistics for an 8x8 block of the quantized DCT coefficients for progressive encoding, the first scan, AC coefficients.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.
`ippStsJPEGDCTRangeErr`	Indicates an error condition if a DCT coefficient is out of the allowed range `(-1023..1023)`.

GetHuffmanStatistics8x8_ACRefine_JPEG

Computes Huffman symbols statistics for the progressive encoding (AC coefficients, subsequent scans).

Syntax

```
IppStatus ippiGetHuffmanStatistics8x8_ACRefine_JPEG_16s_C1(const Ipp16s* pSrc,
    int pAcStatistics[256], int Ss, int Se, int Al, IppiEncodeHuffmanState*
    pEncHuffState, int bFlushState);
```

Parameters

pSrc	Pointer to the 8x8 block of the quantized DCT coefficients.
pAcStatistics	Pointer to the Huffman symbols statistics buffer for the AC coefficients.
Ss	Spectral selection start index.
Se	Spectral selection end index.
Al	Successive approximation bit positions low, it specifies the actual point transform.
pEncHuffState	Pointer to the `IppiEncodeHuffmanState` structure.
bFlushState	Set to 1 for the last 8x8 block in the scan.

Description

The function `ippiGetHuffmanStatistics8x8_ACRefine_JPEG` is declared in the `ippj.h` file. This function computes the Huffman symbols statistics for an 8x8 block of the quantized DCT coefficients for progressive encoding, the subsequent scans, AC coefficients.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.
`ippStsJPEGOutOfBufErr`	Indicates an error condition if the buffer limits are exceeded.
`ippStsJPEGDCTRangeErr`	Indicates an error condition if a DCT coefficient is out of the allowed range `(-1023..1023)`.

EncodeHuffman8x8_DCFirst_JPEG

Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).

Syntax

```
IppStatus ippiEncodeHuffman8x8_DCFirst_JPEG_16s1u_C1(const Ipp16s* pSrc, Ipp8u*
    pDst, int dstLenBytes, int* pDstCurrPos, Ipp16s* pLastDC, int Al,
    IppiEncodeHuffmanSpec* pDcTable, IppiEncodeHuffmanState* pEncHuffState, int
    bFlushState);
```

Parameters

pSrc	Pointer to the 8x8 block of the quantized DCT coefficients.
pDst	Pointer to the buffer for the output bit stream.
dstLenBytes	Length in bytes of the buffer for the bit stream.
pDstCurrPos	Shift in bytes of the current byte in the output buffer.
pLastDC	Pointer to the DC coefficient of the previous 8x8 block.
Al	Successive approximation bit positions low, it specifies the actual point transform.
pDcTable	Pointer to the Huffman table for DC coefficients.
pEncHuffState	Pointer to the `IppiEncodeHuffmanState` structure.
bFlushState	Set it to 1 for the last 8x8 block in the scan

Description

The function `ippiEncodeHuffman8x8_DCFirst_JPEG` is declared in the `ippj.h` file. This function performs encoding of the DC coefficient from an 8x8 block of the quantized coefficients (progressive encoding, the first scan), using *pDcTable* table. The encoding procedure conforms to [ISO10918], *Annex G.1.2, Progressive Encoding Procedures with Huffman.*

Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter *bFlushState* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or all of the specified pointers are NULL.
ippStsJPEGOutOfBufErr	Indicates an error condition if the buffer limits are exceeded.
ippStsJPEGDCTRangeErr	Indicates an error condition if the DCT coefficient is out of the allowed range (-1023..1023).

EncodeHuffman8x8_DCRefine_JPEG

Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).

Syntax

```
IppStatus ippiEncodeHuffman8x8_DCRefine_JPEG_16s1u_C1(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Al,
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

Parameters

pSrc	Pointer to the 8x8 block of the quantized DCT coefficients.
pDst	Pointer to the buffer for the output bit stream.
dstLenBytes	Length in bytes of the buffer for the bit stream.
pDstCurrPos	Shift in bytes of the current byte in the output buffer.
Al	Successive approximation bit positions low, it specifies the actual point transform.
pEncHuffState	Pointer to the IppiEncodeHuffmanState structure.
bFlushState	Set it to 1 for the last 8x8 block in the scan

Description

The ippiEncodeHuffman8x8_DCRefine_JPEG is declared in the ippj.h file. This function performs encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (progressive encoding, the subsequent scans), using *pDcTable* table.
The encoding procedure conforms to [ISO10918], *Annex G.1.2, Progressive Encoding*

Procedures with Huffman.
Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter *`bFlushState`* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.
`ippStsJPEGOutOfBufErr`	Indicates an error condition if the buffer limits are exceeded.
`ippStsJPEGDCTRangeErr`	Indicates an error condition if the DCT coefficient is out of the allowed range `(-1023..1023)`.

EncodeHuffman8x8_ACFirst_JPEG

Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).

Syntax

```
IppStatus ippiEncodeHuffman8x8_ACFirst_JPEG_16s1u_C1(const Ipp16s* pSrc, Ipp8u*
    pDst, int dstLenBytes, int* pDstCurrPos, int Ss, int Se, int Al,
    IppiEncodeHuffmanSpec* pAcTable, IppiEncodeHuffmanState* pEncHuffState, int
    bFlushState);
```

Parameters

`pSrc`	Pointer to the 8x8 block of the quantized DCT coefficients
`pDst`	Pointer to the buffer for the output bit stream.
`dstLenBytes`	Length in bytes of the buffer for the bit stream.
`pDstCurrPos`	Shift in bytes of the current byte in the output buffer.
`Ss`	Spectral selection start index.
`Se`	Spectral selection end index.

Al	Successive approximation bit positions low, it specifies the actual point transform.
pAcTable	Pointer to the Huffman table for AC coefficients.
pEncHuffState	Pointer to the `IppiEncodeHuffmanState` structure.
bFlushState	Set it to 1 for the last 8x8 block in the scan.

Description

The function `ippiEncodeHuffman8x8_ACFirst_JPEG` is declared in the `ippj.h` file. This function performs encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (progressive encoding, the first scan), using *pAcTable* table.
The encoding procedure conforms to [ISO10918], *Annex G.1.2, Progressive Encoding Procedures with Huffman.*
Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter *bFlushState* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.
`ippStsJPEGOutOfBufErr`	Indicates an error condition if the buffer limits are exceeded.
`ippStsJPEGDCTRangeErr`	Indicates an error condition if the DCT coefficient is out of the allowed range `(-1023..1023)`.

EncodeHuffman8x8_ACRefine_JPEG

Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients.

Syntax

```
IppStatus ippiEncodeHuffman8x8_ACRefine_JPEG_16s1u_C1(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Ss, int Se, int Al,
    IppiEncodeHuffmanSpec*  pAcTable, IppiEncodeHuffmanState* pEncHuffState,
    int bFlushState);
```

Parameters

pSrc	Pointer to the 8x8 block of the quantized DCT coefficients.
pDst	Pointer to the buffer for the output bit stream.
dstLenBytes	Length in bytes of the buffer for the bit stream.
pDstCurrPos	Shift in bytes of the current byte in the output buffer.
Ss	Spectral selection start index.
Se	Spectral selection end index.
Al	Successive approximation bit positions low, it specifies the actual point transform.
pAcTable	Pointer to the Huffman table for AC coefficients.
pEncHuffState	Pointer to the `IppiEncodeHuffmanState` structure.
bFlushState	Set it to 1 for the last 8x8 block in the scan.

Description

The function `ippiEncodeHuffman8x8_ACRefine_JPEG` is declared in the `ippj.h` file. This function performs encoding of the AC coefficient from an 8x8 block of the quantized DCT coefficients (progressive encoding, the subsequent scans), using *pAcTable* table.
The encoding procedure conforms to [ISO10918], *Annex G.1.2, Progressive Encoding Procedures with Huffman.*
Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter *bFlushState* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or all of the specified pointers are NULL.
ippStsJPEGOutOfBufErr	Indicates an error condition if the buffer limits are exceeded.
ippStsJPEGDCTRangeErr	Indicates an error condition if the DCT coefficient is out of the allowed range (-1023..1023).

DecodeHuffmanSpecGetBufSize_JPEG

Returns the length of the IppiDecodeHuffmanSpec *structure.*

Syntax

```
IppStatus ippiDecodeHuffmanSpecGetBufSize_JPEG_8u(int* pSize);
```

Parameters

pSize	Pointer to a variable that will receive the length of the IppiDecodeHuffmanSpec structure.

Description

The function ippiDecodeHuffmanSpecGetBufSize_JPEG is declared in the ippj.h file. This function returns the length of the IppiDecodeHuffmanSpec structure.

NOTE. The fields of the structure IppiDecodeHuffmanSpec are hidden from the user.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if the pointer is NULL.

DecodeHuffmanSpecInit_JPEG

Creates Huffman table in a format that is suitable for a decoder

Syntax

```
IppStatus ippiDecodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits,
    const Ipp8u* pListVals, IppiDecodeHuffmanSpec* pDecHuffSpec);
```

Parameters

pListBits	Pointer to `Bits` list.
pListVals	Pointer to `Vals` list.
pDecHuffSpec	Pointer to the Huffman table for the decoder.

Description

The function `ippiDecodeHuffmanSpecInit_JPEG` is declared in the `ippj.h` file. This function creates Huffman table in a format that is suitable for a decoder.
The `Bits` and `Vals` lists are specified in [ISO10918], *Annex C, Huffman Table Specification.*
To generate the table, the function uses the procedure that is described in [ISO10918], *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths.*

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all specified pointers are `NULL`.
`ippStsJPEGHuffTableErr`	Indicates an error condition during the initialization of the table. It may happen when raw tables contain inadmissible values.

DecodeHuffmanSpecInitAlloc_JPEG

Allocates memory and creates Huffman table in a format that is suitable for a decoder.

Syntax

```
IppStatus ippiDecodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u* pListBits,
    const Ipp8u* pListVals,  IppiDecodeHuffmanSpec** ppDecHuffSpec);
```

Parameters

pListBits	Pointer to `Bits` list.
pListVals	Pointer to `Vals` list.
ppDecHuffSpec	Pointer to the returned pointer to the Huffman table for the decoder.

Description

The function `ippiDecodeHuffmanSpecInitAlloc_JPEG` is declared in the `ippj.h` file. This function allocates memory and creates the Huffman table in a format that is suitable for a decoder.

The function expects that `Bits` and `Vals` lists are supplied in the format specified in [ISO10918], *Annex C, Huffman Table Specification.*

To generate the table, the function uses the procedure that is described in [ISO10918], *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths.*

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.
`ippStsJPEGHuffTableErr`	Indicates an error condition during the initialization of the table. It may happen when raw tables contain inadmissible values.

DecodeHuffmanSpecFree_JPEG

Frees memory allocated by `ippiDecodeHuffmanSpecInitAlloc_JPEG` *function.*

Syntax

```
IppStatus ippiDecodeHuffmanSpecFree_JPEG_8u(IppiDecodeHuffmanSpec*
    pDecHuffSpec);
```

Parameters

pDecHuffSpec	Pointer to the Huffman table for the decoder.

Description

The function `ippiDecodeHuffmanSpecFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated by the `ippiDecodeHuffmanSpecInitAlloc_JPEG` function for the Huffman table.

Return Values

`ippStsNoErr`	Indicates no error.

DecodeHuffmanStateGetBufSize_JPEG

Returns the length of `IppiDecodeHuffmanState` *structure.*

Syntax

```
IppStatus ippiDecodeHuffmanStateGetBufSize_JPEG_8u(int* pSize);
```

Parameters

pSize	Pointer to a variable that will receive the length of the `IppiDecodeHuffmanState` structure.

Description

The function `ippiDecodeHuffmanStateGetBufSize_JPEG` is declared in the `ippj.h` file. This function returns the length in bytes of the `IppiDecodeHuffmanState` structure.

NOTE. The fields of the structure `IppiDecodeHuffmanState` are hidden from the user.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if the specified pointer is `NULL`.

DecodeHuffmanStateInit_JPEG

Initializes `IppiDecodeHuffmanState` *structure.*

Syntax

```
IppStatus ippiDecodeHuffmanStateInit_JPEG_8u(IppiDecodeHuffmanState*
    pDecHuffState);
```

Parameters

pDecHuffState	Pointer to the `IppiDecodeHuffmanState` structure.

Description

The function `ippiDecodeHuffmanStateInit_JPEG` is declared in the `ippj.h` file. This function initializes the `IppiDecodeHuffmanState` structure to the initial state and must be called prior to the JPEG bit stream decoding.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.

DecodeHuffmanStateInitAlloc_JPEG

Allocates memory and initializes `IppiDecodeHuffmanState` *structure.*

Syntax

```
IppStatus ippiDecodeHuffmanStateInitAlloc_JPEG_8u(IppiDecodeHuffmanState**
    ppDecHuffState);
```

Parameters

ppDecHuffState	Pointer to the `IppiDecodeHuffmanState` structure.

Description

The function `ippiDecodeHuffmanStateInit_JPEG` is declared in the `ippj.h` file. This function allocates memory and initializes the `IppiDecodeHuffmanState` structure to the initial state. This function must be called prior to the JPEG bit stream decoding.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.

DecodeHuffmanStateFree_JPEG

Frees memory allocated by `ippiDecodeHuffmanStateInitAlloc_JPEG` *function.*

Syntax

```
IppStatus ippiDecodeHuffmanStateFree_JPEG_8u(IppiDecodeHuffmanState*
    pDecHuffState);
```

Parameters

pDecHuffState	Pointer to the `IppiDecodeHuffmanState` structure.

Description

The function `ippiDecodeHuffmanStateFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated by the `ippiDecodeHuffmanStateInitAlloc_JPEG_8u` function for the `IppiDecodeHuffmanState` structure.

Return Values

`ippStsNoErr`	Indicates no error.

DecodeHuffman8x8_JPEG

Performs Huffman baseline decoding of 8x8 block of the quantized DCT Coefficients.

Syntax

```
IppStatus ippiDecodeHuffman8x8_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, Ipp16s*  pLastDC,
    int* pMarker, const IppiDecodeHuffmanSpec*  pDcTable,
    const IppiDecodeHuffmanSpec* pAcTable,  IppiDecodeHuffmanState*
    pDecHuffState);
```

Parameters

pSrc	Pointer to the source buffer with the JPEG bit stream.
srcLenBytes	Length in bytes of the buffer for the bit stream.
pSrcCurrPos	Shift in bytes of the current byte in the buffer.
pDst	Pointer to the 8x8 block of the quantized DCT coefficients.
pLastDC	Pointer to the DC coefficient of the previous 8x8 block.
pMarker	Pointer to a variable that will receive JPEG marker detected during decoding.
pDcTable	Pointer to the Huffman table for the DC coefficients.
pAcTable	Pointer to the Huffman table for the AC coefficients.
pDecHuffState	Pointer to the `IppiDecodeHuffmanState` structure.

Description

The function `ippiDecodeHuffman8x8_JPEG` is declared in the `ippj.h` file. This function decodes an 8x8 block of the quantized DCT coefficients using *pDcTable* and *pAcTable* tables.
The decoding procedure conforms to [ISO10918], *Annex F.2.2, Baseline Huffman Decoding Procedures.*
If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicate an error condition if *srcLenBytes* has a zero or negative value, or *pSrcCurrPos* is out of *srcLenBytes* limit.
`ippStsJPEGDCTRangeErr`	Indicates an error condition if the DCT coefficient is out of the allowed range `(-1023..1023)`.
`ippStsJPEGOutOfBufErr`	Indicates an error condition if the buffer limits are exceeded.
`ippStsJPEGMarkerWarn`	Indicates a warning if a JPEG marker is detected.

DecodeHuffman8x8_Direct_JPEG

Directly performs Huffman baseline decoding of an 8x8 block of the quantized DCT coefficients.

Syntax

```
IppStatus ippiDecodeHuffman8x8_Direct_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int* pSrcBitsLen, Ipp16s* pDst, Ipp16s* pLastDC, int* pMarker,
    Ipp32u* pPrefetchedBits, int* pNumValidPrefetchedBits, const
    IppiDecodeHuffmanSpec* pDcTable, const IppiDecodeHuffmanSpec*
    pAcTable);
```

Parameters

pSrc	Pointer to the source buffer with the JPEG bit stream.

pSrcBitsLen	Length in bits of the buffer for the bit stream.
pDst	Pointer to the 8x8 block of the quantized DCT coefficients.
pLastDC	Pointer to the DC coefficient of the previous 8x8 block of the same color component.
pMarker	Pointer to a variable that will receive JPEG marker detected during decoding.
pPrefetchedBits	Pointer to the prefetch buffer which contains the previously decoded bits.
pNumValidPrefetchedBits	Number of valid bits in the prefetch buffer.
pDcTable	Pointer to the Huffman table for the DC coefficients.
pAcTable	Pointer to the Huffman table for the AC coefficients.

Description

The function `ippiDecodeHuffman8x8_Direct_JPEG` is declared in the `ippj.h` file. This function directly decodes an 8x8 block of the quantized DCT coefficients using *pDcTable* and *pAcTable* tables. The function does not require any additional structure for operation. The decoding procedure conforms to [ISO10918], *Annex F.2.2, Baseline Huffman Decoding Procedures.*
If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

pLastDC should be set to 0 through the function initialization or after each restart interval. *pMarker* should be set to 0 through the function initialization or after the found marker has been processed. *pNumValidPrefetchedBits* should be set to 0 in the following cases: 1) through the function initialization; 2) after each restart interval; 3) after each found marker has been processed.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.
`ippStsSizeErr`	Indicates an error condition if *pSrcBitsLen* or *pNumValidPrefetchedBits* has negative value.
`ippStsJPEGDCTRangeErr`	Indicates an error condition if the DCT coefficient is out of the allowed range `(-1023..1023)`.
`ippStsJPEGOutOfBufErr`	Indicates an error condition if the buffer limits are exceeded.

DecodeHuffman8x8_DCFirst_JPEG

Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).

Syntax

```
IppStatus ippiDecodeHuffman8x8_DCFirst_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, Ipp16s* pLastDC,
    int* pMarker, int Al, IppiDecodeHuffmanSpec* pDcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

Parameters

pSrc	Pointer to the buffer with the JPEG bit stream.
srcLenBytes	Length in bytes of the buffer for the bit stream.
pSrcCurrPos	Shift in bytes of the current byte in the buffer.
pDst	Pointer to the 8x8 block of the quantized DCT coefficients.
pLastDC	Pointer to the DC coefficient of the previous 8x8 block.
pMarker	Pointer to a variable that will receive JPEG marker detected during decoding.
Al	Successive approximation bit positions low, it specifies the actual point transform.
pDcTable	Pointer to the Huffman table for the DC coefficients.
pDecHuffState	Pointer to the `IppiDecodeHuffmanState` structure.

Description

The function `ippiDecodeHuffman8x8_DCFirst_JPEG` is declared in the `ippj.h` file. This function decodes the DC coefficient from an 8x8 block of the quantized DCT coefficients, progressive mode, the first scan, using *pDcTable* table.
The decoding procedure conforms to [ISO10918], *Annex G.2, Progressive Decoding of the DCT.*
If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if one or all of the specified pointers are NULL.
ippStsJPEGDCTRangeErr	Indicates an error condition if the DCT coefficient is out of the allowed range (-1023..1023).
ippStsJPEGOutOfBufErr	Indicates an error condition if the buffer limits are exceeded.
ippStsJPEGMarkerWarn	Indicates a warning if a JPEG marker is detected.

DecodeHuffman8x8_DCRefine_JPEG

Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients.

Syntax

```
IppStatus ippiDecodeHuffman8x8_DCRefine_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, int Al,
    IppiDecodeHuffmanState* pDecHuffState);
```

Parameters

pSrc	Pointer to the buffer with the JPEG bit stream.
srcLenBytes	Length in bytes of the buffer for the bit stream.
pSrcCurrPos	Shift in bytes of the current byte in the buffer.
pDst	Pointer to the 8x8 block of the quantized DCT coefficients.
pMarker	Pointer to a variable that will receive a JPEG marker detected during decoding.
Al	Successive approximation bit positions low, it specifies the actual point transform.
pDecHuffState	Pointer to the IppiDecodeHuffmanState structure.

Description

The function ippiDecodeHuffman8x8_DCRefine_JPEG is declared in the ippj.h file. This function decodes the DC coefficient from an 8x8 block of the quantized DCT coefficients, progressive mode, the subsequent scans, using *pDcTable* table.
The decoding procedure conforms to [ISO10918], *Annex G.2, Progressive Decoding of the*

DCT.

If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one or all of the specified pointers are NULL.
ippStsJPEGDCTRangeErr	Indicates an error condition if the DCT coefficient is out of the allowed range (-1023..1023).
ippStsJPEGOutOfBufErr	Indicates an error condition if the buffer limits are exceeded.
ippStsJPEGMarkerWarn	Indicates a warning if a JPEG marker is detected.

DecodeHuffman8x8_ACFirst_JPEG

Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).

Syntax

```
IppStatus ippiDecodeHuffman8x8_ACFirst_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst,  int* pMarker,
    int Ss, int Se, int Al,  IppiDecodeHuffmanSpec* pAcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

Parameters

pSrc	Pointer to the buffer with the JPEG bit stream.
srcLenBytes	Length in bytes of the buffer for the bit stream.
pSrcCurrPos	Shift in bytes of the current byte in the buffer.
pDst	Pointer to the 8x8 block of the quantized DCT coefficients.
pMarker	Pointer to a variable that will receive JPEG marker detected during decoding.
Ss	Spectral selection start index.
Se	Spectral selection end index.

Al	Successive approximation bit positions low, it specifies the actual point transform.
pAcTable	Pointer to the Huffman table for the AC coefficients.
pDecHuffState	Pointer to the `IppiDecodeHuffmanState` structure.

Description

The function `ippiDecodeHuffman8x8_ACFirst_JPEG` is declared in the `ippj.h` file. This function decodes AC coefficients from an 8x8 block of the quantized DCT coefficients, progressive mode, the first scan, using *pAcTable* table.
The decoding procedure conforms to [ISO10918], *Annex G.2, Progressive Decoding of the DCT.*
If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.
`ippStsJPEGDCTRangeErr`	Indicates an error condition if the DCT coefficient is out of the allowed range `(-1023..1023)`.
`ippStsJPEGOutOfBufErr`	Indicates an error condition if the buffer limits are exceeded.
`ippStsJPEGMarkerWarn`	Indicates a warning if a JPEG marker is detected.

DecodeHuffman8x8_ACRefine_JPEG

Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).

Syntax

```
IppStatus ippiDecodeHuffman8x8_ACRefine_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker,
    int Ss, int Se, int Al, IppiDecodeHuffmanSpec* pAcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

Parameters

pSrc	Pointer to the buffer with the JPEG bit stream.
srcLenBytes	Length in bytes of the buffer for the bit stream.
pSrcCurrPos	Shift in bytes of the current byte in the buffer.
pDst	Pointer to the 8x8 block of the quantized DCT coefficients.
pMarker	Pointer to a variable that will receive JPEG marker detected during of decoding.
Ss	Spectral selection start index.
Se	Spectral selection end index.
Al	Successive approximation bit positions low, it specifies the actual point transform.
pAcTable	Pointer to the Huffman table for the AC coefficients.
pDecHuffState	Pointer to the `IppiDecodeHuffmanState` structure.

Description

The function `ippiDecodeHuffman8x8_ACRefine_JPEG` is declared in the `ippj.h` file. This function decodes AC coefficients from an 8x8 block of the quantized DCT coefficients, progressive mode, subsequent scans, using *pAcTable* table. The decoding procedure conforms to [ISO10918], *Annex G.2, Progressive Decoding of the DCT.*
If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one or all of the specified pointers are `NULL`.
`ippStsJPEGDCTRangeErr`	Indicates an error condition if the DCT coefficient is out of the allowed range `(-1023..1023)`.
`ippStsJPEGOutOfBufErr`	Indicates an error condition if the buffer limits are exceeded.
`ippStsJPEGMarkerWarn`	Indicates a warning if a JPEG marker is detected.

Functions for Lossless JPEG Coding

This section describes the functions that are specific for the lossless JPEG process with Huffman coding. These functions are listed in Table 15-9.

Table 15-9 Functions for Lossless JPEG Coding

Function Base Name	Description
DiffPredFirstRow_JPEG	Computes the differences between input sample and predictor for the first line
DiffPredRow_JPEG	Computes the differences between input sample and predictor for all lines but the first.
ReconstructPredFirstRow_JPEG	Reconstructs samples from the decoded differences between input samples and predictor for the first line.
ReconstructPredRow_JPEG	Reconstructs samples from the decoded differences between input samples and predictor for all lines but the first.
GetHuffmanStatisticsOne_JPEG	Computes Huffman symbol statistics
EncodeHuffmanOne_JPEG	Performs Huffman encoding of one difference.
DecodeHuffmanOne_JPEG	Decodes one Huffman coded difference.
EncodeHuffmanRow_JPEG	Performs Huffman encoding of one row of differences for each color component in the JPEG scan.
DecodeHuffmanRow_JPEG	Decodes one row for each color component of Huffman coded differences.

The Intel IPP functions use the coding model in accordance with [ISO10918], *Annex H, Lossless Mode of Operation.*

DiffPredFirstRow_JPEG

Computes the differences between input sample and predictor for the first line.

Syntax

```
IppStatus ippiDiffPredFirstRow_JPEG_16s_C1(const Ipp16s* pSrc,
    Ipp16s* pDst, int width, int P, int Pt);
```

Parameters

pSrc	Pointer to the row of samples.
pDst	Pointer to the row of calculated differences.
width	Row width in elements; has always the same value for all rows.
P	Sample precision derived from JPEG frame header, varies from 2 to 16.
Pt	Point transformation parameter derived from JPEG scan header; its value should be 0 or a positive integer.

Description

The function `ippiDiffPredFirstRow_JPEG` is declared in the `ippj.h` file. This function operates with the first row of samples *pSrc* at the start of the scan and at the beginning of each restart interval only. It computes modulo 2^{16} differences between input samples and predictor with the specified sample precision *P* and performs the point transformation with the specified parameter *Pt*. The function uses special predictors for the first line.

Example 15-6 shows how to use the function `ippiDiffPredFirstRow_JPEG_16s_C1`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *width* parameter has a negative value.
`ippStsBadArgrErr`	Indicates an error condition if *P* or *Pt* has an illegal value.

Example 15-6 Using the function `ippiDiffPredFirstRow_JPEG`

```
void func_DiffPredFirstRow()
{
    Ipp16s pSrc[4*4] = { 0,  1,  2,  4,
                         8, 16, 32, 64,
                        32, 16,  8,  4,
                         2,  1,  0,  1};
    Ipp16s pDst[4*4];
    int width = 4;
    int P = 8;
    int Pt = 2;
    int predictor = 2;

    ippiDiffPredFirstRow_JPEG_16s_C1(pSrc, pDst, width, P, Pt);

    for(int j = 1; j < 4; j++)
    {
```

Example 15-6 Using the function ippiDiffPredFirstRow_JPEG

```
ippiDiffPredRow_JPEG_16s_C1(pSrc+j*width, pSrc + (j-1)*width,
pDst+j*width, width, predictor);
    }
}
```

```
Result:
    pDst
-32    1    1    2   <- ïåðâàÿ ñòðîêà ïîëó÷åíà ñ ïîìîùüþ ippiDiffPredFirstRow
  8   15   30   60    îñòàëüíûå(2-4) ñ ïîìîùüþ ippiDiffPredRow
 24    0  -24  -60
-30  -15   -8   -3
```

DiffPredRow_JPEG

Computes the differences between input sample and predictor for all lines but the first.

Syntax

```
IppStatus ippiDiffPredRow_JPEG_16s_C1(const Ipp16s* pSrc,
    const Ipp16s*  pPrevRow, Ipp16s* pDst, int width, int predictor);
```

Parameters

pSrc	Pointer to the row of samples.
pPrevRow	Pointer to the adjacent row of samples just above the current row.
pDst	Pointer to the row of calculated differences.
width	Row width in elements; has always the same value for all rows.
predictor	Selected predictor, should be in the range [1, 7].

Description

The function `ippiDiffPredRow_JPEG` is declared in the `ippj.h` file. This function operates on all rows of samples except the first row at the start of the scan and at the beginning of the restart interval. It computes modulo 2^{16} differences between input samples *pSrc* and the specified predictor *predictor*. The first sample of the row use the sample from the row *pPrevRow* above as a predictor.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *width* parameter has a negative value.
ippStsBadArgrErr	Indicates an error condition if *predictor* has an illegal value.

ReconstructPredFirstRow_JPEG

Reconstructs samples from the decoded differences between input samples and predictor for the first line.

Syntax

```
IppStatus ippiReconstructPredFirstRow_JPEG_16s_C1(const Ipp16s* pSrc,
    Ipp16s* pDst, int width, int P, int Pt);
```

Parameters

pSrc	Pointer to the row of decoded differences.
pDst	Pointer to the row of reconstructed samples.
width	Row width in elements; has always the same value for all rows.
P	Sample precision derived from JPEG frame header, varies from 2 to 16.
Pt	Point transformation parameter derived from JPEG scan header; its value should be 0 or a positive integer.

Description

The function ippiReconstructPredFirstRow_JPEG is declared in the ippj.h file. This function operates on the first row of decoded differences *pSrc* at the start of the scan and restart interval only. It reconstructs output samples *pDst* with the specified sample precision *P* adding decoded differences modulo 2^{16} to the predictions. It also performs the point transformation with the specified parameter *Pt*.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *width* parameter has a negative value.
ippStsBadArgrErr	Indicates an error condition if *P* or *Pt* has an illegal value.

ReconstructPredRow_JPEG

Reconstructs samples from the decoded differences between input samples and predictor for all lines but the first.

Syntax

```
IppStatus ippiReconstructPredRow_JPEG_16s_C1(const Ipp16s* pSrc, const
    Ipp16s* pPrevRow, Ipp16s* pDst, int width, int predictor);
```

Parameters

pSrc	Pointer to the row of decoded differences.
pPrevRow	Pointer to the adjacent row of reconstructed samples just above the current row.
pDst	Pointer to the row of reconstructed samples.
width	Row width in elements (has always the same value for all rows).
predictor	Selected predictor, should be in the range [1, 7].

Description

The function ippiReconstructPredRow_JPEG is declared in the ippj.h file. This function operates on all rows of decoded differences *pSrc* except first rows at the start of the scan and restart interval. It reconstructs output samples *pDst* adding decoded differences modulo 2^{16} to the predictions specified by the *predictor* parameters. The first sample of the row use the reconstructed sample from the row *pPrevRow* above as a predictor.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *width* parameter has a negative value.

ippStsBadArgrErr	Indicates an error condition if *predictor* has an illegal value.

GetHuffmanStatisticsOne_JPEG

Computes Huffman symbol statistics.

Syntax

```
IppStatus ippiGetHuffmanStatisticsOne_JPEG_16s_C1(const Ipp16s* pSrc,
    int pHuffStatistics[256]);
```

Parameters

pSrc	Pointer to the row of decoded differences.
pHuffStatistics	Pointer to the calculated statistics buffer.

Description

The function ippiGetHuffmanStatisticsOne_JPEG is declared in the ippj.h file. This function computes frequency of Huffman symbols to be encoded. Based on these statistics, an optimal Huffman table can be built by the ippiEncodeHuffmanRwaTableInit_JPEG function. This table saves space by removing unused symbols and assigning shorter codes for more frequent symbols.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

EncodeHuffmanOne_JPEG

Performs Huffman encoding of one difference.

Syntax

```
IppStatus ippiEncodeHuffmanOne_JPEG_16s1u_C1(const Ipp16s* pSrc,
    Ipp8u* pDst, int nDstLenBytes, int* pDstCurrPos,
    const IppiEncodeHuffmanSpec* pEncHuffTable,
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

Parameters

pSrc	Pointer to the difference to be encoded. This pointer can be NULL if *bFlusfState* = 1.
pDst	Pointer to the output bitstream buffer.
nDstLenBytes	Number of available bytes in the output buffer.
pDstCurrPos	Pointer to the current byte in the output buffer. This pointer is updated in the function.
pEncHuffTable	Pointer to the IppiEncodeHuffmanSpec structure that contains the Huffman code table. This pointer can be NULL if *bFlusfState* = 1.
pEncHuffState	Pointer to the IppiEncodeHuffmanState structure that contains the Huffman encoder state.
bFlushState	Setting this parameter to 1 forces the function to flush collected bits from the state structure to the bitstream, setting it to 0 forces to perform encoding.

Description

The function ippiEncodeHuffmanOne_JPEG is declared in the ippj.h file. This function encodes one difference *pSrc* using Huffman code table *pEncHuffTable*. Only full bytes are written to the output buffer. The IppiEncodeHuffmanState structure collects the bits that do not make up a complete byte. To force adding the bits accumulated in the IppiEncodeHuffmanState to the output buffer, the parameter *bFlushState* should be set to 1 when the last sample in scan or restart interval is encoded. In all other cases this parameter must be set to zero.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	When *bFlushState*=0 - Indicates an error condition if one of the specified pointers is NULL, when *bFlushState*=1 - Indicates an error condition if one of the *pDst*, *pDstCurrPos* or *pEncHuffState* pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *nDstLenBytes* parameter has zero or negative value, or if *pDstCurrPos* is out of *nDstLenBytes* limit.

DecodeHuffmanOne_JPEG

Decodes one Huffman coded difference.

Syntax

```
IppStatus ippiDecodeHuffmanOne_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int nSrcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker,
    const IppiDecodeHuffmanSpec* pDecHuffTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

Parameters

pSrc	Pointer to the input bitstream.
nSrcLenBytes	Number of available bytes in the input buffer.
pSrcCurrPos	Pointer to the current byte in the input buffer. This pointer is updated in the function.
pDst	Pointer to the output buffer to store the decoded difference.
pMarker	Pointer to a variable that will receive JPEG marker detected during decoding.
pDecHuffTable	Pointer to the `IppiDecodeHuffmanSpec` structure that contains the Huffman code table.
pDecHuffState	Pointer to the `IppiDecodeHuffmanState` structure that contains the Huffman decoder state.

Description

The function `ippiDecodeHuffmanOne_JPEG` is declared in the `ippj.h` file. This function decodes one Huffman coded difference pointed to in the bitstream by *pSrc* and stores the result in the output buffer *pDst*. The function uses Huffman code table *pDecHuffTable*.

If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *nSrcLenBytes* parameter has zero or negative value, or if *pSrcCurrPos* is out of *nSrcLenBytes* limit.

`ippStsJPEGMarkerWarn` Indicates a warning if a JPEG marker is detected.

EncodeHuffmanRow_JPEG

Performs Huffman encoding of one row of differences for each color component in the JPEG scan.

Syntax

```
IppStatus ippiEncodeHuffmanRow_JPEG_16s1u_P4C1(const Ipp16s* pSrc[4],
    int nSrcLen, int nSrcRows, Ipp8u* pDst, int nDstLenBytes, int*
    pDstCurrPos, const IppiEncodeHuffmanSpec*  pEncHuffTable[4],
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

Parameters

pSrc	Array of pointers to the differences to be encoded. It can be `NULL` if *bFlusfState* = 1.
nSrcLen	Number of elements in the input rows (should be the same for all rows).
nSrcRows	Number of rows (one row for every color component in JPEG scan).
pDst	Pointer to the output bitstream buffer.
nDstLenBytes	Number of available bytes in the output buffer.
pDstCurrPos	Pointer to the current byte in the output buffer. This pointer is updated in the function.
pEncHuffTable	Pointer to the `IppiEncodeHuffmanSpec` structure (one for every color component in JPEG scan) that contains the Huffman code table. This pointer can be `NULL` if *bFlusfState* = 1, or if there is no corresponding color component.
pEncHuffState	Pointer to the `IppiEncodeHuffmanState` structure that contains the Huffman encoder state.
bFlushState	Setting this parameter to 1 forces the function to flush collected bits from the state structure to the bitstream, setting it to 0 forces to perform encoding.

Description

The function ippiEncodeHuffmanRow_JPEG is declared in the ippj.h file. This function encodes *nSrcRows* rows (up to 4, one for every color component) containing *nSrcLen* elements of differences pointed by pointers in the aray *pSrc*[4] using corresponding Huffman code tables *pEncHuffTable*[4]. If number of components is less than 4, corresponding pointers are not specifed.

Only full bytes are written to the output buffer. The IppiEncodeHuffmanState structure collects the bits that do not make up a complete byte. To force adding the bits accumulated in the IppiEncodeHuffmanState to the output buffer, the parameter *bFlushState* should be set to 1 when the last sample in scan or restart interval is encoded. In all other cases this parameter must be set to zero.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	When *bFlushState*=0 - Indicates an error condition if one of the specified pointers is NULL, when *bFlushState*=1 - Indicates an error condition if one of the *pDst*, *pDstCurrPos* or *pEncHuffState* pointers is NULL.
ippStsSizeErr	Indicates an error condition if the *nDstLenBytes* parameter has zero or negative value, or if *pDstCurrPos* is out of *nDstLenBytes* limit.

DecodeHuffmanRow_JPEG

Decodes one row for each color component of Huffman coded differences.

Syntax

```
IppStatus ippiDecodeHuffmanRow_JPEG_1u16s_C1P4(const Ipp8u* pSrc, int
    nSrcLenBytes, int* pSrcCurrPos, Ipp16s* pDst[4], int nDstLen, int
    nDstRows, int* pMarker, const IppiDecodeHuffmanSpec*
    pDecHuffTable[4], IppiDecodeHuffmanState* pDecHuffState);
```

Parameters

pSrc	Pointer to the input bitstream.
nSrcLenBytes	Number of available bytes in the input buffer.

pSrcCurrPos	Pointer to the current byte in the input buffer. This pointer is updated in the function.
pDst	Array of pointers to the output buffer for each color component in the JPEG scan to store the decoded difference.
pDstLen	Number of elements to decode, must be equal to the image width or JPEG restart interval.
pDstRows	Number of output rows, must be equal to the number of color components.
pMarker	Pointer to a variable that will receive JPEG marker detected during decoding.
pDecHuffTable	Array of pointers to the `IppiDecodeHuffmanSpec` structures (one for each color component) that contains the Huffman decode table.
pDecHuffState	Pointer to the `IppiDecodeHuffmanState` structure that contains the Huffman decoder state.

Description

The function `ippiDecodeHuffmanRow_JPEG` is declared in the `ippj.h` file. This function decodes *pDstLen* Huffman coded difference for *nDstRows* rows (one for each color component) using corresponding *pDecHuffTable* table from the bitstream pointed by *pSrc*, and places it to the output buffer pointed by the appropriate pointer in the array *pDst*.

If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *nSrcLenBytes* parameter has zero or negative value, or if *pSrcCurrPos* is out of *nSrcLenBytes* limit.
`ippStsJPEGMarkerWarn`	Indicates a warning if a JPEG marker is detected.

Wavelet Transform Functions

This section describes the wavelet transform functions that are specific for JPEG 2000 image coding system (see [ISO15444]). These functions are listed in the Table 15-10.

Table 15-10 Wavelet Transform Functions

Function Base Name	Description
Low-Level Operations	
WTFwdRow_B53_JPEG2K	Performs a row oriented forward wavelet transform with reversible filter.
WTInvRow_B53_JPEG2K	Performs a row oriented inverse wavelet transform with reversible filter.
WTFwdCol_B53_JPEG2K	Performs a forward wavelet transform with reversible filter of image columns.
WTFwdColLift_B53_JPEG2K	Performs a single step of forward wavelet transform with reversible filter of image columns.
WTInvCol_B53_JPEG2K	Performs a column oriented inverse wavelet transform with reversible filter.
WTInvColLift_B53_JPEG2K	Performs a single step of inverse wavelet transform with reversible filter of image columns.
WTFwdRow_D97_JPEG2K	Performs a row oriented forward wavelet transform with irreversible filter.
WTInvRow_D97_JPEG2K	Performs a row oriented inverse wavelet transform with irreversible filter.
WTFwdCol_D97_JPEG2K	Performs a column oriented forward wavelet transform with irreversible filter.
WTFwdColLift_D97_JPEG2K	Performs a single step of forward wavelet transform with irreversible filter of image columns.
WTInvCol_D97_JPEG2K	Performs a column oriented inverse wavelet transform with irreversible filter.
WTInvColLift_D97_JPEG2K	Performs a single step of inverse wavelet transform with irreversible filter of image columns.
Tile-Oriented Transforms	
WTGetBufSize_B53_JPEG2K	Computes the size of the buffer for a tile-oriented wavelet transform
WTFwd_B53_JPEG2K	Performs a tile-oriented forward wavelet transform.
WTInv_B53_JPEG2K	Performs a tile-oriented inverse wavelet transform.

Table 15-10 Wavelet Transform Functions (continued)

Function Base Name	Description
WTGetBufSize_D97_JPEG2K	Computes the size of the buffer for a tile-oriented wavelet transform
WTFwd_D97_JPEG2K	Performs a tile-oriented forward wavelet transform.
WTInv_D97_JPEG2K	Performs a tile-oriented inverse wavelet transform.

Low-Level Operations

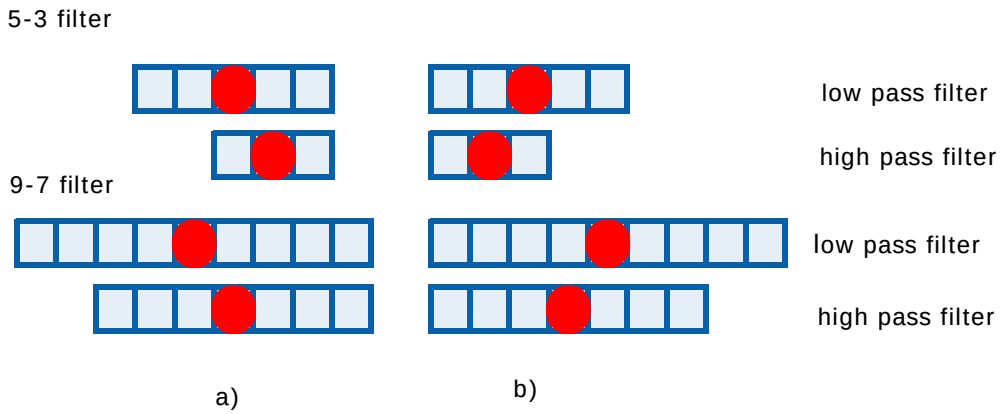
These functions perform one-dimensional reversible (_B53 modifier) and irreversible (_D97 modifier) wavelet transforms of image rows or columns for lossless and lossy compressions, respectively, in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*]. The transformations are the lifting-based implementations of filtering by 5-3 reversible and 9-7 irreversible wavelet filters.

The filters have odd length: 5 (9) - for lowpass filters, and 3 (7) - for highpass filters. There are two possible relative positions of lowpass and highpass filters allowed by the standard, as shown in the Figure 15-2. The relative position is specified by the *phase* argument that has two possible values:

ippWTFilterFirstLow corresponds to positions a),
ippWTFilterFirstHigh corresponds to positions b).

The wavelet transform functions do not apply any fixed border extension (symmetrical, wraparound or other) to the source image ROI, instead they require valid and accessible border data outside of the source image ROI.

Figure 15-2 Possible Relative Positions of Filters



WTFwdRow_B53_JPEG2K

Performs a forward wavelet transform with reversible filter of image rows.

Syntax

```
IppStatus ippiWTFwdRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc,
    int srcStep, Ipp16s* pDstLow,  int dstLowStep, Ipp16s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
IppStatus ippiWTFwdRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc,
    int srcStep, Ipp32s* pDstLow,  int dstLowStep, Ipp32s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDstLow	Pointer to ROI of the low frequency component of a destination image.
dstLowStep	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
pDstHigh	Pointer to ROI of the high frequency component of a destination image.
dstHighStep	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
dstRoiSize	Size of the destination image ROI.
phase	Relative position of the high-pass and low-pass filters (see page 15-117).

Description

The function `ippiWTFwdRow_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a wavelet decomposition of the source image rows using the 5-3 reversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

```
srcRoiSize.width = 2 * dstRoiSize.width;
srcRoiSize.height = dstRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI:

- if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 2 extra pixels on the left and 1 pixel on the right (outside ROI border) for each processed row;
- if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel on the left and 2 pixels on the right (outside ROI border) for each processed row.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the width or height of ROI has zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

WTInvRow_B53_JPEG2K

Performs an inverse wavelet transform with reversible filter of image rows.

Syntax

```
IppStatus ippiWTInvRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow,
    int srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiWTFilterFirst phase);
IppStatus ippiWTInvRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow,
    int srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiWTFilterFirst phase);
```

Parameters

pSrcLow	Pointer to ROI of the low frequency component of a source image.
srcLowStep	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
pSrcHigh	Pointer to ROI of the high frequency component of a source image.
srcHighStep	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
srcRoiSize	Size of the source image ROI.
pDst	Pointer to ROI of the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
phase	Relative position of the high-pass and low-pass filters (see page 15-117).

Description

The function `ippiWTInvRow_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a wavelet reconstruction of the source image rows using the 5-3 reversible filter. Both source ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = 2 * srcRoiSize.width;
dstRoiSize.height = srcRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 1 extra pixel on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel on the left (outside ROI border) for each processed row.

For the high frequency component, the function always requires 1 extra pixel on the left and 1 pixel on the right (outside ROI border) for each processed row.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is NULL.
`ippStsSizeErr`	Indicates an error condition if the width or height of ROI has zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

WTFwdCol_B53_JPEG2K

Performs a forward wavelet transform with reversible filter of image columns.

Syntax

```
IppStatus ippiWTFwdCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc,
    int srcStep, Ipp16s* pDstLow, int dstLowStep, Ipp16s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
IppStatus ippiWTFwdCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc,
    int srcStep, Ipp32s* pDstLow, int dstLowStep, Ipp32s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.

pDstLow	Pointer to ROI of the low frequency component of a destination image.
dstLowStep	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
pDstHigh	Pointer to ROI of the high frequency component of a destination image.
dstHighStep	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
dstRoiSize	Size of the destination image ROI.
phase	Relative position of the high-pass and low-pass filters (see page 15-116).

Description

The function `ippiWTFwdCol_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a wavelet decomposition of the source image columns using the 5-3 reversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

```
srcRoiSize.width = dstRoiSize.width;
srcRoiSize.height = 2 * dstRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 2 extra pixels up and 1 pixel down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel up and 2 pixels down (outside ROI border) for each processed column.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the width or height of ROI has zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

WTFwdColLift_B53_JPEG2K

Performs a single step of forward wavelet transform with reversible filter of image columns.

Syntax

```
IppStatus ippiWTFwdColLift_B53_JPEG2K_16s_C1(const Ipp16s* pSrc0, const
    Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDstLow0, const Ipp16s*
    pSrcHigh0, Ipp16s* pDstHigh1, int width);
IppStatus ippiWTFwdColLift_B53_JPEG2K_32s_C1(const Ipp32s* pSrc0, const
    Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDstLow0, const Ipp32s*
    pSrcHigh0, Ipp32s* pDstHigh1, int width);
```

Parameters

pSrc0	Pointer to the source image row #0.
pSrc1	Pointer to the source image row #1.
pSrc2	Pointer to the source image row #2.
pSrcHigh0	Pointer to the high frequency component row #0 of a source image.
pDstLow0	Pointer to the low frequency component row #0 of a destination image.
pDstHigh1	Pointer to the high frequency component row #1 of a destination image.
width	Width in pixels of rows.

Description

The function `ippiWTFwdColLift_B53_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based wavelet decomposition of the source image columns using the 5-3 reversible filter.

For each i-pixel in the row this function computes the following values:

$$pDstHigh1[i] = pSrc1[i] - \left\lfloor \frac{pSrc0[i] + pSrc2[i]}{2} \right\rfloor$$

$$pDstLow0[i] = pSrc1[i] + \left\lfloor \frac{pSrcHigh0[i] + pDstHigh1[i] + 2}{4} \right\rfloor$$

Here the notation $\lfloor x \rfloor$ designates the floor function, that is, the largest integer not exceeding x.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the width of row has zero or negative value.

WTInvCol_B53_JPEG2K

Performs an inverse wavelet transform with reversible filter of image columns.

Syntax

```
IppStatus ippiWTInvCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow,
    int srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiWTFilterFirst phase);
IppStatus ippiWTInvCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow,
    int srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiWTFilterFirst phase);
```

Parameters

pSrcLow	Pointer to ROI of the low frequency component of a source image.
srcLowStep	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
pSrcHigh	Pointer to ROI of the high frequency component of a source image.
srcHighStep	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
srcRoiSize	Size of the source image ROI.
pDst	Pointer to ROI of the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
phase	Relative position of the high-pass and low-pass filters (see page 15-116).

Description

The function `ippiWTInvCol_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a wavelet reconstruction of the source image columns using the 5-3 reversible filter. Both source image ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = srcRoiSize.width;
dstRoiSize.height = 2 * srcRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 1 extra pixel down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel up (outside ROI border) for each processed column.

For the high frequency component, the function always requires 1 extra pixel up and 1 pixel down (outside ROI border) for each processed column.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the width or height of ROI has zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

WTInvColLift_B53_JPEG2K

Performs a single step of inverse wavelet transform with reversible filter of image columns.

Syntax

```
IppStatus ippiWTInvColLift_B53_JPEG2K_16s_C1(const Ipp16s* pSrcLow0,
    const Ipp16s* pSrcHigh0, const Ipp16s* pSrcHigh1, const Ipp16s*
    pSrc0, Ipp16s* pDst1, Ipp16s* pDst2, int width);
```

```
IppStatus ippiWTInvColLift_B53_JPEG2K_32s_C1(const Ipp32s* pSrcLow0,
    const Ipp32s* pSrcHigh0, const Ipp32s* pSrcHigh1, const Ipp32s*
    pSrc0, Ipp32s* pDst1, Ipp32s* pDst2, int width))
```

Parameters

pSrcLow0	Pointer to the source low frequency component row #0.
pSrcHigh0	Pointer to the source high frequency component row #0.
pSrcHigh1	Pointer to the source high frequency component row #1.
pSrc0	Pointer to the reconstructed image row #0.
pDst1	Pointer to the lreconstructed image row #1.
pDst2	Pointer to the lreconstructed image row #2.
width	Width in pixels of rows.

Description

The function `ippiWTInvColLift_B53_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based wavelet reconstruction of the source image columns using the 5-3 reversible filter.

For each i-pixel in the row this function computes the following values:

$$pDst2[i] = pSrcLow0[i] - \left\lfloor \frac{pSrcHigh0[i] + pSrcHigh1[i] + 2}{4} \right\rfloor$$

$$pDst1[i] = pSrcHigh0[i] + \left\lfloor \frac{pSrc0[i] + pDst2[i]}{2} \right\rfloor$$

Here the notation $\lfloor x \rfloor$ designates the floor function, that is, the largest integer not exceeding x.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *width* is less than or equal to 0.

WTFwdRow_D97_JPEG2K

Performs a forward wavelet transform with irreversible filter of image rows.

Syntax

```
IppStatus ippiWTFwdRow_D97_JPEG2K_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDstLow,  int dstLowStep, Ipp<datatype>*
    pDstHigh, int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst
    phase);
```

Supported values for *mod*:

16s_C1R 32s_C1R 32f_C1R

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstLow	Pointer to ROI of the low frequency component of a destination image.
dstLowStep	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
pDstHigh	Pointer to ROI of the high frequency component of a destination image.
dstHighStep	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
dstRoiSize	Size of the destination image ROI.
phase	Relative position of the high-pass and low-pass filters (see page 15-116).

Description

The function `ippiWTFwdRow_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a wavelet decomposition of the source image rows using the 9-7 irreversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

srcRoiSize.width = 2 * *dstRoiSize.width*;
srcRoiSize.height = *dstRoiSize.height*

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to ippWTFilterFirstLow, the function requires 4 extra pixels on the left and 3 pixels on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to ippWTFilterFirstHigh, the function requires 3 extra pixels on the left and 4 pixels on the right (outside ROI border) for each processed row.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the width or height of ROI has zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

WTInvRow_D97_JPEG2K

Performs an inverse wavelet transform with irreversible filter of image rows.

Syntax

```
IppStatus ippiWTInvRow_D97_JPEG2K_<mod>(const Ipp<datatype>* pSrcLow,
    int srcLowStep, const Ipp<datatype>* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiWTFilterFirst
    phase);
```

Supported values for *mod*:

16s_C1R 32s_C1R 32f_C1R

Parameters

pSrcLow	Pointer to ROI of the low frequency component of a source image.
srcLowStep	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.

pSrcHigh	Pointer to ROI of the high frequency component of a source image.
srcHighStep	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
srcRoiSize	Size of the source image ROI.
pDst	Pointer to ROI of the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
phase	Relative position of the high-pass and low-pass filters (see page 15-116).

Description

The function `ippiWTInvRow_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a wavelet reconstruction of the source image rows using the 9-7 irreversible filter. Both source ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = 2 * srcRoiSize.width;
dstRoiSize.height = srcRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 1 extra pixel on the left and 2 extra pixels on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 2 extra pixels on the left and 1 extra pixel on the right (outside ROI border) for each processed row.

For the high frequency component, the function always requires 2 extra pixels on the left and 2 extra pixels on the right (outside ROI border) for each processed row.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the width or height of ROI has zero or negative value.

ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

WTFwdCol_D97_JPEG2K

Performs a forward wavelet transform with irreversible filter of image columns.

Syntax

```
IppStatus ippiWTFwdCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDstLow, int dstLowStep, Ipp32f* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDstLow	Pointer to ROI of the low frequency component of a destination image.
dstLowStep	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
pDstHigh	Pointer to ROI of the high frequency component of a destination image.
dstHighStep	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
dstRoiSize	Size of the destination image ROI.
phase	Relative position of the high-pass and low-pass filters (see page 15-116).

Description

The function `ippiWTFwdCol_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a wavelet decomposition of the source image columns using the 9-7 irreversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

srcRoiSize.width = dstRoiSize.width;

```
srcRoiSize.height = 2 * dstRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 4 extra pixels up and 3 extra pixels down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 3 extra pixels up and 4 extra pixels down (outside ROI border) for each processed column.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the width or height of ROI has zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

WTFwdColLift_D97_JPEG2K

Performs a single step of forward wavelet transform with irreversible filter of image columns.

Syntax

```
IppStatus ippiWTFwdColLift_D97_JPEG2K_<mod>(const Ipp<datatype>* pSrc0,
    const Ipp<datatype>* pSrc1, const Ipp<datatype>* pSrc2,
    Ipp<datatype>* pSrcDstLow0, Ipp<datatype>* pDstLow1, Ipp<datatype>*
    pSrcDstHigh0, Ipp<datatype>* pSrcDstHigh1, Ipp<datatype>* pDstHigh2,
    int width);
```

Supported values for *mod*:

`16s_C1` `32s_C1` `32f_C1`

Parameters

pSrc0	Pointer to the source image row #0.
pSrc1	Pointer to the source image row #1.
pSrc2	Pointer to the source image row #2.

pSrcDstLow0	Pointer to the low frequency component row #0.
pDstLow1	Pointer to the low frequency component row #1.
pSrcDstHigh0	Pointer to the high frequency component row #0.
pSrcDstHigh1	Pointer to the high frequency component row #1.
pDstHigh2	Pointer to the high frequency component row #2.
width	Width in pixels of rows.

Description

The function `ippiWTFwdColLift_D97_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based 2D wavelet decomposition of the source image columns using the 9-7 irreversible filter.

For each *i*-pixel in the row this function computes the following values:

$$pDstHigh2[i] = pSrc1[i] + \alpha \cdot (pSrc0[i] + pSrc2[i])$$

$$pDstLow1[i] = pSrc0[i] + \beta \cdot (pSrcDstHigh1[i] + pDstHigh2[i])$$

$$pSrcDstHigh1[i] = pSrcDstHigh1[i-1] + \gamma \cdot (pSrcDstLow0[i] + pDstLow1[i])$$

$$pSrcDstLow0[i] = pSrcDstLow0[i-1] + \delta \cdot (pSrcDstHigh0[i] + pSrcDstHigh1[i])$$

$$pSrcDstLow0[i] = pSrcDstLow0[i] \cdot K$$

$$pSrcDstHigh0[i] = (pSrcDstHigh0[i]) / K$$

where α, β, γ, δ, K - are the lifting parameters in accordance with [ISO15444], *Annex F.3, Inverse Discrete Wavelet Transformation*].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *width* is less than or equal to 0.

WTInvCol_D97_JPEG2K

Performs an inverse wavelet transform with irreversible filter of image columns.

Syntax

```
IppStatus ippiWTInvCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrcLow,
    int srcLowStep, const Ipp32f* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp32f* pDst, int dstStep,
    IppiWTFilterFirst phase);
```

Parameters

pSrcLow	Pointer to ROI of the low frequency component of a source image.
srcLowStep	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
pSrcHigh	Pointer to ROI of the high frequency component of a source image.
srcHighStep	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
srcRoiSize	Size of the source image ROI.
pDst	Pointer to ROI of the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
phase	Relative position of the high-pass and low-pass filters (see page 15-116).

Description

The function `ippiWTInvCol_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs a wavelet reconstruction of the source image columns using the 9-7 irreversible filter. Both source image ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

dstRoiSize.width = *srcRoiSize.width*;
dstRoiSize.height = 2 * *srcRoiSize.height*

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to ippWTFilterFirstLow, the function requires 1 extra pixel up and 2 extra pixels down (outside ROI border) for each processed column;

if the *phase* argument is equal to ippWTFilterFirstHigh, the function requires 2 extra pixels up and 1 extra pixel down (outside ROI border) for each processed column.

For the high frequency component, the function always requires 2 extra pixels up and 2 extra pixels down (outside ROI border) for each processed column.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the width or height of ROI has zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

WTInvColLift_D97_JPEG2K

Performs a single step of inverse wavelet transform with irreversible filter of image columns.

Syntax

```
IppStatus ippiWTInvColLift_D97_JPEG2K_<mod>(const Ipp<datatype>*
    pSrcLow0, const Ipp<datatype>* pSrcHigh0, const Ipp<datatype>*
    pSrcHigh1, const Ipp<datatype>* pSrc0, Ipp<datatype>* pSrcDst1,
    Ipp<datatype>* pSrcDst2, Ipp<datatype>* pDst3, Ipp<datatype>* pDst4,
    int width);
```

Supported values for *mod*:

16s_C1 32s_C1 32f_C1

Parameters

pSrcLow0	Pointer to the source low frequency component row #0.
pSrcHigh0	Pointer to the source high frequency component row #0.
pSrcHigh1	Pointer to the source high frequency component row #1.

pSrc0	Pointer to the reconstructed image row #0.
pSrcDst1	Pointer to the reconstructed image row #1.
pSrcDst2	Pointer to the reconstructed image row #2.
pDst3	Pointer to the reconstructed image row #3.
pDst4	Pointer to the reconstructed image row #4.
width	Width in pixels of rows.

Description

The function `ippiWTInvColLift_D97_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based 2D wavelet reconstruction of the source image columns using the 9-7 irreversible filter.

For each *i*-pixel in the row this function computes the following values:

$$pDst4[i] = pSrcLow0[i] \cdot K - (\delta / K \cdot (pSrcHigh0[i] + pSrcHigh1[i]))$$
$$pDst3[i] = (pSrcHigh0[i]) / K - (\gamma \cdot (pSrcDst2[i] + pDst4[i]))$$
$$pSrcDst2[i] = pSrcDst2[i-1] - (\beta \cdot (pSrcDst1[i] + pDst3[i]))$$
$$pSrcDst1[i] = pSrcDst1[i-1] - (\alpha \cdot (pSrc0[i] + pSrcDst2[i]))$$

where α, β, γ, δ, K - are the lifting parameters in accordance with [ISO15444], *Annex F.3, Inverse Discrete Wavelet Transformation*].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the *width* is less than or equal to 0.

Tile-Oriented Transforms

These functions perform high-level tile-oriented reversible (`_B53` modifier) and irreversible (`_D97` modifier) wavelet transforms for lossless and lossy compressions, respectively, in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*]. The transformations are the lifting-based implementations of filtering by 5-3 reversible and 9-7 irreversible wavelet filters.

The functions operate with tile-component and perform one-level decomposition or reconstruction of the image. Each tile-component is transformed into a set of four two-dimensional subbands.

WTGetBufSize_B53_JPEG2K

Computes the size of the buffer for wavelet transform with reversible filter.

Syntax

```
IppStatus ippiWTGetBufSize_B53_JPEG2K_16s_C1R(const IppiRect* pTileRect,
    int* pSize);
IppStatus ippiWTGetBufSize_B53_JPEG2K_32s_C1R(const IppiRect* pTileRect,
    int* pSize);
IppStatus ippiWTGetBufSize_B53_JPEG2K_16s_C1IR(const IppiRect* pTileRect,
    int* pSize);
IppStatus ippiWTGetBufSize_B53_JPEG2K_32s_C1IR(const IppiRect* pTileRect,
    int* pSize);
```

Parameters

pTileRect	Pointer to the tile rectangle structure.
pSize	Pointer to the computed value of the buffer.

Description

The function `ippiWTGetBufSize_B53_JPEG2K` is declared in the `ippj.h` file. This function computes the size in bytes of the work buffer that is required for a tile-oriented wavelet transform functions `ippiWTFwd_B53_JPEG2K` and `ippiWTInv_B53_JPEG2K`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.

WTFwd_B53_JPEG2K

Performs a tile-oriented forward wavelet transform with reversible filter.

Syntax

```
IppStatus ippiWTFwd_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep,
    const IppiRect* pTileRect, Ipp16s* pDst[4], int dstStep[4], Ipp8u* pBuffer);
IppStatus ippiWTFwd_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep,
    const IppiRect* pTileRect, Ipp32s* pDst[4], int dstStep[4], Ipp8u* pBuffer);
IppStatus ippiWTFwd_B53_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int srcDstStep,
    const IppiRect* pTileRect, Ipp8u* pBuffer);
IppStatus ippiWTFwd_B53_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int srcDstStep,
    const IppiRect* pTileRect, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pTileRect	Pointer to the tile rectangle structure.
pDst	Array of pointers to the subbands of the destination image (in the order LxLy, LxHy, HxLy, HxHy).
dstStep	Array of distances in bytes between starts of consecutive lines in the the subbands of the destination image.
pSrcDstTile	Pointer to the source and destination image for in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
pBuffer	Pointer to the work buffer.

Description

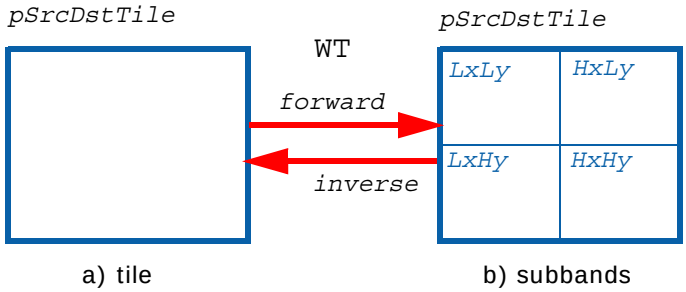
The function ippiWTFwd_B53_JPEG2K is declared in the ippj.h file. This function performs a forward discrete wavelet transformation of the tile-component using a subband decomposition in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*]. The function performs one-level decomposition by downsampling each tile of the source image *pSrc* (*pSrcDstTile* for in-place flavors) into a set of four two-dimensional subbands labelled as LxLy, LxHy, HxLy, HxHy. The subband LxLy is a

downsampled version of source tile that is low-pass filtered horizontally and low-pass filtered vertically. The subband `LxHy` is a downsampled version of source tile that is low-pass filtered horizontally and high-pass filtered vertically. The subband `HxLy` is a downsampled version of source tile that is high-pass filtered horizontally and low-pass filtered vertically. The subband `HxHy` is a downsampled version of source tile that is high-pass filtered horizontally and high-pass filtered vertically. The function uses the 5-3 reversible filter.

For not-in-place flavors, the subbands are stored separately in the array *pDst*. For in-place flavors, the subbands are stored in the *pSrcDstTile* (see Figure 15-3)

.

Figure 15-3 Subband Positions for in-place Operations



The size of the required external buffer should be determined by calling the `ippiWTGetBufSize_B53_JPEG2K` function prior to using the wavelet transform function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the width or height of ROI has zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified step values is zero or negative.

WTInv_B53_JPEG2K

Performs a tile-oriented inverse wavelet transform with reversible filter.

Syntax

```
IppStatus ippiWTInv_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc[4],
    int srcStep[4], Ipp16s* pDst, int dstStep, const IppiRect* pTileRect,
    Ipp8u* pBuffer);
IppStatus ippiWTInv_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc[4],
    int srcStep[4], Ipp32s* pDst, int dstStep, const IppiRect* pTileRect,
    Ipp8u* pBuffer);
IppStatus ippiWTInv_B53_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int
    srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
IppStatus ippiWTInv_B53_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int
    srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

Parameters

pSrc	An array of pointers to the subbands of the source image (in the order LxLy, LxHy, HxLy, HxHy).
srcStep	Array of distances in bytes between starts of consecutive lines in the subbands of the source image.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDstTile	Pointer to the source and destination image for in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
pTileRect	Pointer to the tile rectangle structure.
pBuffer	Pointer to the work buffer.

Description

The function ippiWTInv_B53_JPEG2K is declared in the ippj.h file. This function performs an inverse discrete wavelet transformation using a subband reconstruction in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*]. The function performs one-level reconstruction of the tile of the

destination image *pDst* from the set of four two-dimensional subbands labelled as LxLy, LxHy, HxLy, HxHy. The subband LxLy is a downsampled version of the initial tile that is low-pass filtered horizontally and low-pass filtered vertically. The subband LxHy is a downsampled version of the initial tile that is low-pass filtered horizontally and high-pass filtered vertically. The subband HxLy is a downsampled version of the initial tile that is high-pass filtered horizontally and low-pass filtered vertically. The subband HxHy is a downsampled version of the initial tile that is high-pass filtered horizontally and high-pass filtered vertically. The function uses the 5-3 reversible filter.

For not-in-place flavors, the source subbands are placed separately in the array *pDst*. For in-place flavors, the source subbands are stored in the *pSrcDstTile* (see Figure 15-3)

The size of the required external buffer should be determined by calling the ippiWTGetBufSize_B53_JPEG2K function prior to using the wavelet transform function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the width or height of ROI has zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified step values is zero or negative.

WTGetBufSize_D97_JPEG2K

Computes the size of the buffer wavelet transform with irreversible filter

Syntax

```
IppStatus ippiWTGetBufSize_D97_JPEG2K_16s_C1R(const IppiRect* pTileRect,
    int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_32s_C1R(const IppiRect* pTileRect,
    int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_16s_C1IR(const IppiRect* pTileRect,
    int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_32s_C1IR(const IppiRect* pTileRect,
    int* pSize);
```

Parameters

pTileRect	Pointer to the tile rectangle structure.
pSize	Pointer to the computed value of the buffer.

Description

The function `ippiWTGetBufSize_D97_JPEG2K` is declared in the `ippj.h` file. This function computes the size in bytes of the work buffer that is required for a tile-oriented irreversible wavelet transform functions `ippiWTFwd_D97_JPEG2K` and `ippiWTInv_D97_JPEG2K`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is NULL.

WTFwd_D97_JPEG2K

Performs a tile-oriented forward wavelet transform with irreversible filter.

Syntax

```
IppStatus ippiWTFwd_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep,
    const IppiRect* pTileRect, Ipp16s* pDst[4], int dstStep[4],
    Ipp8u* pBuffer);
IppStatus ippiWTFwd_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep,
    const IppiRect* pTileRect, Ipp32s* pDst[4], int dstStep[4],
    Ipp32s* pBuffer);
IppStatusippiWTFwd_D97_JPEG2K_16s_C1IR(Ipp16s*pSrcDstTile,intsrcDstStep,
    const IppiRect* pTileRect, Ipp8u* pBuffer);
IppStatusippiWTFwd_D97_JPEG2K_32s_C1IR(Ipp32s*pSrcDstTile,intsrcDstStep,
    const IppiRect* pTileRect, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pTileRect	Pointer to the tile rectangle structure.

pDst	Array of pointers to the subbands of the destination image (in the order LxLy, LxHy, HxLy, HxHy).
dstStep	Array of distances in bytes between starts of consecutive lines in the subbands of the destination image.
pSrcDstTile	Pointer to the source and destination image for in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
pBuffer	Pointer to the buffer.

Description

The function ippiWTFwd_D97_JPEG2K is declared in the ippj.h file. This function performs a forward discrete wavelet transformation of the tile-component using a subband decomposition in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*]. The function performs one-level decomposition by downsampling each tile of the source image *pSrc* into a set of four two-dimensional subbands *pDst* labelled as LxLy, LxHy, HxLy, HxHy. The subband LxLy is a downsampled version of the source tile to which the horizontal and vertical low-pass filters were applied. The subband LxHy is a downsampled version of the source tile to which a horizontal low-pass and vertical and high-pass filters were applied. The subband HxLy is a downsampled version of the source tile to which the horizontal and vertical high-pass filters were applied. The subband HxHy is a downsampled version of the source tile to which horizontal high-pass and vertical high-pass filters were applied. The function uses the 9-7 irreversible filter.

For not-in-place flavors, the subbands are stored separately in the array *pDst*. For in-place flavors, the subbands are stored in the *pSrcDstTile* (see Figure 15-3)

The size of the required external buffer should be determined by calling the ippiWTGetBufSize_D97_JPEG2K function prior to using the wavelet transform function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the width or height of ROI has zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified step values is zero or negative.

WTInv_D97_JPEG2K

Performs a tile-oriented inverse wavelet transform with irreversible filter.

Syntax

```
IppStatus ippiWTInv_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc[4],
    int srcStep[4], Ipp16s* pDst, int dstStep, const IppiRect* pTileRect,
    Ipp8u* pBuffer);
IppStatus ippiWTInv_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc[4],
    int srcStep[4], Ipp32s* pDst, int dstStep, const IppiRect* pTileRect,
    Ipp8u* pBuffer);
IppStatus ippiWTInv_D97_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile,
    int srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
IppStatus ippiWTInv_D97_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile,
    int srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

Parameters

pSrc	An array of pointers to the source subbands (in the order LxLy, LxHy, HxLy, HxHy).
srcStep	Array of distances in bytes between starts of consecutive lines in the the source subbands.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the the destination image.
pSrcDstTile	Pointer to the source and destination image for in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
pTileRect	Pointer to the tile rectangle structure.
pBuffer	Pointer to the buffer.

Description

The function ippiWTInv_D97_JPEG2K is declared in the ippj.h file. This function performs an inverse discrete wavelet transformation using a subband reconstruction in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*]. The function performs one-level reconstruction of the tile of the

destination image *pDst* from the set of four two-dimensional subbands *pSrc* labelled as LxLy, LxHy, HxLy, HxHy. The subband LxLy is a downsampled version of the initial tile to which horizontal and vertical low-pass filters were applied. The subband LxHy is a downsampled version of the initial tile to which a horizontal low-pass and vertical high-pass filters were applied. The subband HxLy is a downsampled version of the initial tile to which horizontal high-pass and vertical low-pass filters were applied. The subband HxHy is a downsampled version of the initial tile to which horizontal and vertical high-pass filters were applied. The function uses the 9-7 irreversible filter.

For not-in-place flavors, the source subbands are placed separately in the array *pDst*. For in-place flavors, the source subbands are placed in the *pSrcDstTile* (see Figure 15-3)

The size of the required external buffer should be determined by calling the ippiWTGetBufSize_D97_JPEG2K function prior to using the wavelet transform function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the width or height of ROI has zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified step values is zero or negative.

JPEG2000 Entropy Coding and Decoding Functions

This section describes the image processing functions that perform arithmetic encoding and decoding procedures and are specific for JPEG 2000 image coding system. These functions are listed in the Table 15-11.

Table 15-11 Entropy Coder Functions

Function Base Name	Description
EncodeInitAlloc_JPEG2K	Allocates memory and initializes an entropy encoder state structure
EncodeFree_JPEG2K	Deallocates memory allocated for an entropy encoding state structure
EncodeLoadCodeBlock_JPEG2K	Loads code block and prepares data for entropy encoding
EncodeStoreBits_JPEG2K	Produces output codestream

Table 15-11 Entropy Coder Functions (continued)

Function Base Name	Description
EncodeGetTermPassLen_JPEG2K	Returns the length of the specified terminated coding pass.
EncodeGetRate_JPEG2K	Returns estimated target bit rate for the specified coding pass
EncodeGetDist_JPEG2K	Returns estimated distortion of the image for the specified coding pass
DecodeGetBufSize_JPEG2K	Computes the size of the working buffer for decoding routine
DecodeCodeBlock_JPEG2K	Decodes the compressed code block data
Decoder Functions for Progressive Mode	
DecodeCBProgrGetStateSize_JPEG2K	Computes the size of the external buffer for the decoder state structure.
DecodeCBProgrInit_JPEG2K	Initializes the decoder state structure in the external buffer/
DecodeCBProgrInitAlloc_JPEG2K	Allocates memory and initializes the decoder state structure in the external buffer.
DecodeCBProgrFree_JPEG2K	Frees memory allocated for the decoder state structure.
DecodeCBProgrAttach_JPEG2K	Attaches the buffer with data for progressive decoding.
DecodeCBProgrSetPassCounter_JPEG2K	Sets the value of internal coding pass counter
DecodeCBProgrGetPassCounter_JPEG2K	Returns the value of the internal coding pass counter.
DecodeCBProgrGetCurBitPlane_JPEG2K	Returns the number of the current bit plane
DecodeCBProgrStep_JPEG2K	Performs a single step of decoding.

The implemented adaptive arithmetic encoding is performed as two successive procedures. First, the prepared code block is pre-processed by the function `ippiEncodeLoadCodeBlock_JPEG2K`. After that, the function `ippiEncodeStoreBits_JPEG2K` performs final stages of encoding and produces the output codestream. In the course of encoding, the state structure *pState* is used, which is allocated and initialized by the function `ippiEncodeInitAlloc_JPEG2K`.

EncodeInitAlloc_JPEG2K

Allocates memory and initializes an entropy encoder state structure.

Syntax

```
IppStatus ippiEncodeInitAlloc_JPEG2K(IppiEncodeState_JPEG2K** pState,
    IppiSize codeBlockMaxSize);
```

Parameters

pState	Pointer to variable that returns the pointer to allocated and initialized encoder state structure.
codeBlockMaxSize	Maximum size of code block in pixels.

Description

The function `ippiEncodeInitAlloc_JPEG2K` is declared in the `ippj.h` file. This function allocates and initializes the entropy encoder state structure that is used in the encoding procedures.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsSizeErr`	Indicates an error condition if the width or height of the code block has zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

EncodeFree_JPEG2K

Frees memory allocated for an entropy encoding state structure.

Syntax

```
IppStatus ippiEncodeFree_JPEG2K(IppiEncodeState_JPEG2K* pState);
```

Parameters

pState	Pointer to the allocated and initialized encoder state structure.

Description

The function ippiEncodeFree_JPEG2K is declared in the ippj.h file. This function frees memory allocated for the entropy encoder state structure by the function ippiEncodeInitAlloc_JPEG2K.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if *pState* pointer to a state structure is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid state structure is passed.

EncodeLoadCodeBlock_JPEG2K

Loads a code block and prepares data for entropy encoding.

Syntax

```
IppStatus ippiEncodeLoadCodeBlock_JPEG2K_32s_C1R(const Ipp32s* pSrc,
    int srcStep, IppiSize codeBlockSize, IppiEncodeState_JPEG2K* pState,
    IppiWTSubband subband, int magnBits, IppiMQTermination mqTermType,
    IppiMQRateAppr mqRateAppr, int codeStyleFlags, int* pSfBits,
    int* pNOfPasses, int* pNOfTermPasses);
```

Parameters

pSrc	Pointer to the source code block.
srcStep	Distance in bytes between starts of consecutive lines in the code block.
codeBlockSize	Size of the code block.
pState	Pointer to allocated and initialized encoder state structure.
subband	Wavelet transform subband contained the given code block. Possible values are listed in the Table 15-12.
magnBits	Non-fractional bit count in integer representation.

codeStyleFlags	Specifies the options for encoding procedure. The possible values are listed in the Table 15-13.
mqTermType	Termination mode for MQ-coder. Possible values are listed in the Table 15-14.
mqRateAppr	Specifies the bit rate estimation model. Only one value, `ippMQRateApprGood`, that sets the non-optimal approximation model is available now.
`pSfBits`	Pointer to the variable that returns the number of significant bit planes.
`pNOfPasses`	Pointer to the variable that returns the number of coding passes.
`pNOfTermPasses`	Pointer to the variable that returns the number of terminated coding passes.

Description

The function `ippiEncodeLoadCodeBlock_JPEG2K` is declared in the `ippj.h` file. This function performs the first encoding procedure, that is, loads the specified code block, computes required parameters of the MQ-coder, and prepares data for the second encoding procedure.

The wavelet transform subband should be specified in the *subband* parameter. Possible values are listed in the Table 15-12.

Table 15-12 Subband Parameter for the JPEG2000 Entropy Coder Functions

Value	Description
ippWTSubbandLxLy	Subband obtained by low-pass filtering along x and y directions
ippWTSubbandLxHy	Subband obtained by low-pass filtering along x and high-pass filtering along y direction
ippWTSubbandHxLy	Subband obtained by high-pass filtering along x and low-pass filtering along y direction
ippWTSubbandHxHy	Subband obtained by high-pass filtering along x and y directions

Table 15-13 CodeStyleFlag Argument for the JPEG2000 Entropy Coder Functions

Value	Description
IPP_JPEG2K_VERTICALLY_CAUSAL_CONTEXT	Sets the creation of vertically causal context
IPP_JPEG2K_SELECTIVE_MQ_BYPASS	Sets the selective MQ encoding bypass, which is raw encoding for some coding passes
IPP_JPEG2K_TERMINATE_ON_EVERY_PASS	Sets the termination of MQ coder after every coding pass
IPP_JPEG2K_RESETCTX_ON_EVERY_PASS	Sets the reset of MQ coder context after each coding pass
IPP_JPEG2K_USE_SEGMENTATION_SYMBOLS	Uses the segmentation symbol sequence for error resilience
IPP_JPEG2K_LOSSLESS_MODE	Indicates that the lossless wavelet transforms are used in the rate-distortion estimation
IPP_JPEG2K_DEC_CONCEAL_ERRORS	Allows the error concealment in the last bit-plane where the errors were detected
IPP_JPEG2K_DEC_DO_NOT_CLEAR_CB	Does not clear the code block data before decoding
IPP_JPEG2K_DEC_DO_NOT_RESET_LOW_BITS	Does not reset low-order bits after decoding
IPP_JPEG2K_DEC_DO_NOT_CLEAR_SFBUFFER	Does not clear the buffer before decoding to keep the previous significance state
IPP_JPEG2K_DEC_CHECK_PRED_TERM	Verifies during decoding if the predictable termination is correct; if not - “damage in codeblock” error code is generated. It should be used in predictable termination mode.

Table 15-14 MqTermType Argument for the JPEG2000 Entropy Coder Functions

Value	Description
ippMQTermSimple	Simple termination - some extra bytes are added
ippMQTermNearOptimal	Near optimal termination mode
ippMQTermPredictable	Termination mode for predictable error resilience

The variable *sfBits* returns the number of significant bit planes only. For example, if all source pixels are non-negative and their maximum value is 0xA (binary 1010), then *sfBits* should return 4 significant bits. The higher bits are not coded.

Only significant non-fractional bits in integer representation are coded. *magnBits* specifies the number of non-fractional bits. For example, if *magnBits* = 11, the 20 (31-11) least significant bits will not be coded.

Negative integers in the code block should be presented in direct code with the sign in the most significant bit (31st in the zero-based numeration, when the least significant bit has a zero index). This code may be effectively obtained using the specially designed function ippiComplement.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the width or height of the code block has zero or negative value.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid state structure is passed.
ippStsStepErr	Indicates an error condition if *srcStep* has zero or negative value.

EncodeStoreBits_JPEG2K

Produces output codestream.

Syntax

```
IppStatus ippiEncodeStoreBits_JPEG2K_1u(Ipp8u* pDst, int* pDstLen,
    IppiEncodeState_JPEG2K* pState, int* pIsNotFinish);
```

Parameters

pDst	Pointer to the destination data buffer.
pDstLen	Pointer to the destination buffer length.
pState	Pointer to the allocated and initialized encoder state structure.
pIsNotFinish	Pointer to the variable indicating that the encoding is completed.

Description

The function `ippiEncodeStoreBits_JPEG2K` is declared in the `ippj.h` file. This function performs the second encoding procedure and produces the output codestream.

Application Notes

The parameter *dstLen* is used both for reading and writing. It should be passed to the function with a valid buffer length, and then it will return the length of used (filled) buffer.

Example 15-7 shows how this function may be used.

Example 15-7 Using the `ippiEncodeStoreBits_JPEG2K` Function

```
int isNotFinish = 1;
while(isNotFinish)
{
    int len = BUFFER_LEN;
   ippiEncodeStoreBits_JPEG2K_1u(buffer, &len, state, &isNotFinish);
   // You can write compressed data to the output stream, for example:
    // fwrite(buffer, sizeof(Ipp8u), len, file);
}
```

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the width or height of the destination buffer has zero or negative value.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid state structure is passed.

EncodeGetTermPassLen_JPEG2K

Returns the length of the specified terminated coding pass.

Syntax

```
IppStatus ippiEncodeGetTermPassLen_JPEG2K(IppiEncodeState_JPEG2K* pState,
    int passNumber, int* pPassLen);
```

Parameters

pState	Pointer to the allocated and initialized encoder state structure.
passNumber	Number of the terminated coding pass.
pPassLen	Pointer to the variable that returns the length of the terminated coding pass.

Description

The function `ippiEncodeGetTermPassLen_JPEG2K` is declared in the `ippj.h` file. This function returns the length *pPassLen* of the terminated coding pass of a given number *passNumber* that is stored in the *pState* state structure.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid state structure is passed.
`ippStsJPEG2KBadPassNumber`	Indicates an error condition if *passNumber* exceeds the allowed boundaries.

EncodeGetRate_JPEG2K

Returns estimated bit rate for the specified coding pass.

Syntax

```
IppStatus ippiEncodeGetRate_JPEG2K(IppiEncodeState_JPEG2K* pState,
    int passNumber, int* pRate);
```

Parameters

pState	Pointer to the allocated and initialized encoder state structure.
passNumber	Number of the specified coding pass.
pRate	Pointer to the variable that returns the estimated rate for the specified coding pass.

Description

The function `ippiEncodeGetRate_JPEG2K` is declared in the `ippj.h` file. This function returns the estimated target bit rate *pRate* for the coding pass of a given number *passNumber*. The estimated bit rate is computed during the encoding procedure and then stored in the *pState* state structure. Therefore it is available only after completing both encoding procedures.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid state structure is passed.
`ippStsJPEG2KBadPassNumber`	Indicates an error condition if *passNumber* exceeds the allowed boundaries.

EncodeGetDist_JPEG2K

Returns estimated distortion of the image for the specified coding pass.

Syntax

```
IppStatus ippiEncodeGetDist_JPEG2K(IppiEncodeState_JPEG2K* pState,
    int passNumber, Ipp64f* pDist);
```

Parameters

pState	Pointer to the allocated and initialized encoder state structure.
passNumber	Number of the specified coding pass.
pDist	Pointer to the variable that returns the value of the estimated distortion.

Description

The function `ippiEncodeGetDist_JPEG2K` is declared in the `ippj.h` file. This function returns the cumulative weighted mean square error (MSE) distortion *pDist* of the reconstructed image for the coding pass of a given number *passNumber*. The distortions are computed in accordance with [ISO15444], *Annex J, section J.14*, *Rate Control*] during the encoding procedure and then stored in the *pState* structure.Therefore, it is available only after completing both encoding procedures.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid state structure is passed.
`ippStsJPEG2KBadPassNumber`	Indicates an error condition if *passNumber* exceeds the allowed boundaries.

DecodeGetBufSize_JPEG2K

Computes the size of the work buffer for decoding routine.

Syntax

```
IppStatus ippiDecodeGetBufSize_JPEG2K(IppiSize codeBlockMaxSize,
    int* pSize);
```

Parameters

codeBlockMaxSize	Maximum size of code block in pixels.
pSize	Pointer to the variable that returns the size of the work buffer.

Description

The function `ippiDecodeGetBufSize_JPEG2K` is declared in the `ippj.h` file. This function computes the size (in bytes) of the work buffer that corresponds to the specified maximum size of a code block (in pixels).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the width or height of the code block has zero or negative value.

DecodeCodeBlock_JPEG2K

Decodes the compressed code block data.

Syntax

```
IppStatus ippiDecodeCodeBlock_JPEG2K_1u32s_C1R(const Ipp8u* pSrc,
    Ipp32s* pDst, int dstStep, IppiSize codeBlockSize, IppiWTSubband subband,
    int sfBits, int nOfPasses, const int* pTermPassLen, int nOfTermPasses,
    int codeStyleFlags, int* pErrorBitPlane, Ipp8u* pBuffer);
```

Parameters

pSrc	Pointer to the source compressed code block.
pDst	Pointer to the destination for decoded data.
dstStep	Distance in bytes between starts of consecutive lines in the destination buffer.
codeBlockSize	Size of the code block.
subband	Wavelet transform subband contained the given code block. Possible values are listed in the Table 15-12
sfBits	Number of significant bit planes.
nOfPasses	Number of coding passes.
pTermPassLen	Pointer to the array that contains the length of each terminated coding pass.
nOfTermPasses	Number of terminated coding passes.
codeStyleFlags	Options for the decoding procedure. Possible values are listed in the Table 15-13 .
pErrorBitPlane	Pointer to the bit plane that contains the first error returned when the damaged code block occurs.
pBuffer	Pointer to the work buffer.

Description

The function ippiDecodeCodeBlock_JPEG2K is declared in the ippj.h file. This function decodes the compressed code block *pSrc* of data using adaptive arithmetic decoding procedure and stores the decoded data in the *pDst* buffer.
Negative integers in the code block should be presented in the direct code with the sign in the most significant bit (31st in the zero-based numeration, when the least significant bit has a zero index). This code may be effectively obtained using the specially designed function ippiComplement.

NOTE. *errorBitPlane* should be set to NULL if the corresponding information is not required.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if the width or height of the code block has zero or negative value.
ippStsJPEG2KDamagedCodeBlock	Indicates an error condition if a code block contains damaged data.
ippStsStepErr	Indicates an error condition if *dstStep* has zero or negative value.

DecodeCBProgrGetStateSize_JPEG2K

Computes the size of the external buffer for the decoder state structure.

Syntax

```
IppStatus ippiDecodeCBProgrGetStateSize_JPEG2K(IppiSize codeBlockMaxSize,
    int* pStateSize);
```

Parameters

codeBlockMaxSize	Maximum size of the codeblock.
pStateSize	Pointer to the variable that returns the size of the buffer for the decoder state structure.

Description

The function `ippiDecodeCBProgrGetStateSize_JPEG2K` is declared in the `ippj.h` file. This function computes the size of the external buffer for the decoder state structure. This parameter is depending on the maximum size of the codeblock *codeBlockMaxSize*.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if *pStateSize* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *codeBlockMaxSize* has a field with zero or negative value.

DecodeCBProgrInit_JPEG2K

Initializes the decoder state structure in the external buffer.

Syntax

```
IppStatus ippiDecodeCBProgrInit_JPEG2K(IppiDecodeCBProgrState_JPEG2K* pState);
```

Parameters

pState	Pointer to the decoder state structure.

Description

The function `ippiDecodeCBProgrInit_JPEG2K` is declared in the `ippj.h` file. This function initialize the decoder state structure for progressive mode *pState* in the external buffer which size should be computed previously by calling the function ippiDecodeCBProgrGetStateSize_JPEG2K.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if *pState* pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to the invalid structure is passed.

DecodeCBProgrInitAlloc_JPEG2K

Allocates memory and initializes the decoder state structure in the external buffer.

Syntax

```
IppStatus ippiDecodeCBProgrInitAlloc_JPEG2K(IppiDecodeCBProgrState_JPEG2K**
    ppState, IppiSize codeBlockMaxSize);
```

Parameters

ppState	Pointer to the pointer to the decoder state structure.

codeBlockMaxSize	Maximum size of the codeblock.

Description

The function `ippiDecodeCBProgrInitAlloc_JPEG2K` is declared in the `ippj.h` file. This function allocates memory and initialize the decoder state structure for progressive mode *pState*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pState* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *codeBlockMaxSize* has a field with zero or negative value.

DecodeCBProgrFree_JPEG2K

Frees memory allocated for the decoder state structure.

Syntax

```
IppStatus ippiDecodeCBProgrFree_JPEG2K(IppiDecodeCBProgrState_JPEG2K* pState);
```

Parameters

pState	Pointer to the allocated and initialized decoder state structure.

Description

The function `ippiDecodeCBProgrFree_JPEG2K` is declared in the `ippj.h` file. This function frees memory allocated by the function `DecodeCBProgrInitAlloc_JPEG2K` for the decoder state structure.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pState* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to the invalid structure is passed.

DecodeCBProgrAttach_JPEG2K

Attaches the buffer with data for progressive decoding.

Syntax

```
IppStatus ippiDecodeCBProgrAttach_JPEG2K_32s_C1R(Ipp32s* pDst, int dstStep,
    IppiSize codeBlockSize, IppiDecodeCBProgrState_JPEG2K* pState, IppiWTSubband
    subband, int sfBits, int codeStyleFlags);
```

Parameters

pDst	Pointer to the destination buffer.
dstStep	Step in bytes throught the destination buffer.
codeBlockSize	Size of the codeblock.
pState	Pointer to the decoder state structure.
subband	Wavelet transform subband contained the given code block. Possible values are listed in the Table 15-12.
sfBits	Number of significant bits in code-block.
codeStyleFlags	Specifies the options for encoding procedure. The possible values are listed in the Table 15-13.

Description

The function `ippiDecodeCBProgrAttach_JPEG2K` is declared in the `ippj.h` file. This function attaches a destination buffer *pDst* for the code-block rectangle and fix its parameters to perform the progressive decoding.

CAUTION. The image data buffer should not be moved in memory and should not be freed during whole decoding procedure, that is until the last step of decoding progression and damage correction will be completed.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if the value of *dstStep* is zero or negative.
ippStsContextMatchErr	Indicates an error condition if a pointer to the invalid structure is passed.

DecodeCBProgrSetPassCounter_JPEG2K

Sets the value of internal coding pass counter.

Syntax

```
IppStatus ippiDecodeCBProgrSetPassCounter_JPEG2K(int nOfPasses,
    IppiDecodeCBProgrState_JPEG2K* pState);
```

Parameters

nOfPasses	Number of coding passes.
pState	Pointer to the decode state structure.

Description

The function ippiDecodeCBProgrSetPassCounter_JPEG2K is declared in the ippj.h file. This function sets the value of internal coding pass counter.

The single coding pass is “sub-atom” for coding progression, but their counts in a single terminated segment (for example, the segment filled to the byte boundary) is available from the JPEG 2000 stream or file. During decoding step, pass counter is decremented by one or some other number depending on coding options. If pass counter is set to zero, decoding process will be blocked internally. Externally the pass counter value can be obtained by calling ippiDecodeCodeBlockProgrGetPassCounter_JPEG2K function.

The coding pass counter value can be set before any progression step as well as before series of steps.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if *pState* pointer is NULL.

ippStsContextMatchErr	Indicates an error condition if a pointer to the invalid structure is passed.

DecodeCBProgrGetPassCounter_JPEG2K

Returns the value of the internal coding pass counter.

Syntax

```
IppStatus ippiDecodeCBProgrGetPassCounter_JPEG2K(IppiDecodeCBProgrState_JPEG2K*
    pState, int* pNOfResidualPasses);
```

Parameters

pState	Pointer to the decode state structure.
pNOfResidualPasses	Pointer to the number of residual coding passes.

Description

The function `ippiDecodeCBProgrGetPassCounter_JPEG2K` is declared in the `ippj.h` file. This function returns the value of internal coding pass counter that is received from the decoding state structure *pState*.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if *pState* pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to the invalid structure is passed.

DecodeCBProgrGetCurBitPlane_JPEG2K

Returns the number of the current bit plane.

Syntax

```
IppStatus
    ippiDecodeCBProgrGetCurBitPlaneNum_JPEG2K(IppiDecodeCBProgrState_JPEG2K*
    pState, int* pBitPlaneNum);
```

Parameters

pState	Pointer to the decode state structure.
pBitPlaneNum	Pointer to the variable containing current bit plane number.

Description

The function `ippiDecodeCBProgrGetCurBitPlane_JPEG2K` is declared in the `ippj.h` file. This function returns the current bit plane number. Initially the bit plane counter is set to `(sfBits - 1)` value (see the function ippiDecodeCBProgrAttach_JPEG2K). During successful decoding step, the bit plane counter is decremented by one or some other number depending on coding options.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pState* pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to the invalid structure is passed.

DecodeCBProgrStep_JPEG2K

Performs a single step of decoding.

Syntax

```
IppStatus ippiDecodeCBProgrStep_JPEG2K(const Ipp8u* pSrc, int srcLen,
    IppiDecodeCBProgrState_JPEG2K* pState);
```

Parameters

pSrc	Pointer to the source data.
srcLen	Length of the segment.
pState	Pointer to the decode state structure.

Description

The function `ippiDecodeCBProgrStep_JPEG2K` is declared in the `ippj.h` file. This function performs one step of the decoding progression.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if length of terminated segment is specified incorrectly.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to the invalid structure is passed.

Component Transform Functions

This section describes the functions that perform component transformation specific for JPEG 2000 image coding system. All functions implement the component transformations in accordance with its definition by [ISO15444], *Annex G*, *DC Level Shifting and Multiple Component Transformations*.]

Table 15-15. lists these functions described in more detail later in this section.

Table 15-15 Component Transform Functions

Function Base Name	Description
RCTFwd_JPEG2K	Performs forward reversible component transformation.
RCTInv_JPEG2K	Performs inverse reversible component transformation.
ICTFwd_JPEG2K	Performs forward irreversible component transformation.
ICTInv_JPEG2K	Performs inverse irreversible component transformation.

RCTFwd_JPEG2K

Performs forward reversible component transformation.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRCTFwd_JPEG2K_32s_C3P3R(const Ipp32s* pSrc, int srcStep,
    Ipp32s*  pDst[3], int dstStep, IppiSize roiSize);
```

Case 2: In-place operation on planar data

```
IppStatus ippiRCTFwd_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the array of pointers to ROIs in the planes of the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the array of pointers to the source and destination image ROIs for the in-place operation.
dstSrcStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRCTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs forward reversible component transformation (RCT) of the image buffer *pSrc*. The RCT is applied to all image component samples I_0, I_1, I_2, corresponding to the first, second, and third components, and produces the transform samples Y_0, Y_1, Y_2 in accordance with the following formulas:

$$Y_0 = \left\lfloor \frac{I_0 + 2I_1 + I_2}{4} \right\rfloor$$

$$Y_1 = I_2 - I_1$$

$$Y_2 = I_0 - I_1$$

where the notation $\lfloor x \rfloor$ designates the floor function, that is, the largest integer not exceeding x.

Example 15-8 shows how to use the function `ippiRCTFwd_JPEG2K_32s_C3P3R`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

Example 15-8 Using the function ippiRCTFwd_JPEG2K

```
void func_RCTFwd_JPEG2K()
{
    Ipp32s pSrc[(3*2)*4] = {1, 5, 2, 5, 1, 2,
                            3, 1, 2, 6, 2, 1,
                            1, 3, 5, 3, 3, 1,
                            2, 3, 1, 4, 2, 3};
    int srcStep = (3*2)*sizeof(Ipp32s);
    IppiSize  roiSize = {2, 4};

    Ipp32s* pDst[3];
    int dstStep;

    pDst[0] = ippiMalloc_32s_C1(roiSize.width, roiSize.height, &dstStep);
    pDst[1] = ippiMalloc_32s_C1(roiSize.width, roiSize.height, &dstStep);
    pDst[2] = ippiMalloc_32s_C1(roiSize.width, roiSize.height, &dstStep);

    ippiRCTFwd_JPEG2K_32s_C3P3R(pSrc, srcStep, pDst, dstStep, roiSize);
}
Result:
        pDst[0]             pDst[1]        pDst[2]
         3  2                -3   1         -4  4
         1  2                 1  -1          2  4
         3  2                 2  -2         -2  0
         2  2                -2   1         -1  2
```

RCTInv_JPEG2K

Performs inverse reversible component transformation.

Syntax

Case 1: Operation on planar data

```
IppStatus ippiRCTInv_JPEG2K_32s_P3C3R(const Ipp32s* pSrc[3], int srcStep,
    Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

Case 2: In-place operation on planar data

```
IppStatus ippiRCTInv_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

Parameters

pSrc	Pointer to the array of pointers to the ROIs in the planes of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the array of pointers to ROIs in the planes of the source and destination image for the in-place operation.
dstSrcStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiRCTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs inverse RCT of the image buffer *pSrc* after inverse wavelet transformation. The inverse RCT is applied to transformed image component samples Y_0, Y_1, Y_2 and produces the corresponding samples I_0, I_1, I_2 in accordance with the following formulas:

$$I_1 = Y_0 - \left\lfloor \frac{Y_2 + Y_1}{4} \right\rfloor$$

$$I_0 = Y_2 + I_1$$

$$I_2 = Y_1 + I_1$$

where the notation $\lfloor x \rfloor$ designates the floor function, that is, the largest integer not exceeding x.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

ICTFwd_JPEG2K

Performs forward irreversible component transformation.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiICTFwd_JPEG2K_32f_C3P3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst[3], int dstStep, IppiSize roiSize);
```

Case 2: In-place operation on planar data

```
IppStatus ippiICTFwd_JPEG2K_<mod>(Ipp<dataype>* pSrcDst[3], int
    srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

16s_P3IR 32s_P3IR 32f_P3IR

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the array of pointers to ROIs in the planes of the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.

pSrcDst	Pointer to the array of pointers to the source and destination image ROIs for the in-place operation.
dstSrcStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiICTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs forward irreversible component transformation (ICT) of the image buffer *pSrc*. The ICT is applied to all image component samples I_0, I_1, I_2, corresponding to the first, second, and third components, and produces the transform samples Y_0, Y_1, Y_2 in accordance with the following formulas:

$$Y_0 = 0.299*I_0 + 0.587*I_1 + 0.114*I_2$$
$$Y_1 = -0.16875*I_0 - 0.33126*I_1 + 0.5*I_2$$
$$Y_2 = 0.5*I_0 - 0.41869*I_1 - 0.08131*I_2$$

NOTE. If the first three components are Red, Green and Blue components, then the forward ICT is of a YCbCr transformation.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

ICTInv_JPEG2K

Performs inverse irreversible component transformation.

Syntax

Case 1: Operation on planar data

```
IppStatus ippiICTInv_JPEG2K_32f_P3C3R(const Ipp32f* pSrc[3], int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 2: In-place operation on planar data

```
IppStatus ippiICTInv_JPEG2K_<mod>(Ipp<datatype>* pSrcDst[3], int
    srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

16s_P3IR 32s_P3IR 32f_P3IR

Parameters

pSrc	Pointer to the array of pointers to the ROIs in the planes of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the array of pointers to ROIs in the planes of the source and destination image for the in-place operation.
dstSrcStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

The function `ippiICTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function performs inverse irreversible component transformation (ICT) of the image buffer *pSrc* after inverse wavelet transformation. The inverse ICT is applied to transformed image component samples Y_0, Y_1, Y_2 and produces the corresponding samples I_0, I_1, I_2 in accordance with the following formulas:

$$I_0 = Y_0 + 1.402*Y_2$$
$$I_1 = Y_0 - 0.34413*Y_1 - 0.71414*Y_2$$
$$I_2 = Y_0 + 1.772*Y_1$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

Functions for RLE Coding

This section describes the functions that perform RLE compression of images in accordance with the DICOM "Digital imaging and communication in medicine" specification [DICOM].

Table 15-16. lists these functions described in more detail later in this section.

Table 15-16 RLE Compression Functions

Function Base Name	Description
PackBitsRow_TIFF	Encodes one row of the image.
UnpackBitsRow_TIFF	Decodes one RLE segment.
SplitRow_TIFF	Splits 16-bit monocrome Composite Pixel Code.
JoinRow_TIFF	Restores 16-bit monocrome Composite Pixel Code.

RLE Coding Model

According to the DICOM specification, RLE compression consists of the following steps:

- the image is converted to a sequence of *Composite Pixel Codes*;
- the Composite Pixel Codes are used to generate a set of *Byte Segments*;

- each Byte Segment is RLE compressed to produce a *RLE Segment*;
- the *RLE Header* is appended in front of the concatenated RLE Segments.

PackBitsRow_TIFF

Encodes one row of the image.

Syntax

```
IppStatus ippiPackBitsRow_TIFF_8u_C1(const Ipp8u* pSrc, int
    srcLenBytes, Ipp8u* pDst, int* pDstCurrPos, int dstLenBytes);
```

Parameters

pSrc	Pointer to the input buffer to keep *Byte Segment* or part of B*yte Segment.*
srcLenBytes	Length of the input buffer, bytes.
pDst	Pointer to the output buffer.
pDstCurrPos	Pointer to the current byte position in the output buffer.
dstLenBytes	Length of the output buffer, bytes.

Description

The function `ippiPackBitsRow_TIFF` is declared in the `ippj.h` file. This function encodes *srcLenBytes* data from the input buffer *pSrc*. Encoded data are stored in the output buffer *pDst*. The function uses the encoding scheme called *PackBits* (see pseudo-code below). The input buffer keeps the part of the *Byte Segment* that represens one row of an image. The output buffer is used to keep RLE compressed input data. The size of output buffer must be sufficient to keep at least one row of output *RLE Segment*. Its size can be estimated in accordance with the following formula:

dstLenBytes = *srcLenBytes* + (*srcLenBytes* + 127)/128 for the worst case.

The algorityhm can be expressed in the pseudo-code as follows:

a sequence of identical bytes (called as Replicate Run) is encoded as a two-byte code:

`<-count + 1><byte value>`, where `count` is number of bytes in the run, and `2 <= count <= 128`

and a non repetitive sequence of bytes (called as Literal Run) is encoded as:

`<count - 1><literal sequence of bytes>`, where `count` is number of bytes in the sequence, and `1 <= count <= 128`

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *pDstCurrPos* has an illegal value, or if *srcLenBytes* has a negative value.
`ippStsUnderRunErr`	Indicates an error condition if size of the buffer *pDst* is insufficient to keep encoded row of the *RLE Segment*.

UnpackBitsRow_TIFF

Decodes one RLE Segment.

Syntax

```
IppStatus ippiUnpackBitsRow_TIFF_8u_C1(const Ipp8u* pSrc, int*
    pSrcCurrPos, int srcLenBytes, Ipp8u* pDst, int dstLenBytes);
```

Parameters

pSrc	Pointer to input buffer with RLE segment
srcLenBytes	Size of the input RLE segment, bytes.
pSrcCurrPos	Pointer to the current byte position in the input buffer.
pDst	Pointer to the output buffer.
dstLenBytes	Size of single uncompressed row, bytes.

Description

The function `ippiUnpackBitsRow_TIFF` is declared in the `ippj.h` file. This function decodes *srcLenBytes* data from the input buffer *pSrc* containing *RLE Segment*. Decoded *Byte segment* is stored in the output buffer *pDst*. The size of the input buffer must be sufficient to keep data for at least one row of *Byte Segment*. The function uses the decompression algorithm called *UnPackBits* (TIFF 6.0 specification) that can be expressed in pseudo-code as follows:

```
    LOOP until the number of desired output bytes
  Read the next source byte into N
```

```
    IF N >= 0 and N <= 127 THEN
          output the next N+1 bytes literally
    ELSE IF N <= -1 and N >= -127 THEN
          output the next byte -N+1 times
    ELSE IF N = -128
          output nothing
    ENDIF
ENDLOOP
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *pSrcCurrPos* has an illegal value, or if *dstLenBytes* has a negative value.
ippStsUnderRunErr	Indicates an error condition if size of the buffer *pSrc* is insufficient to decompress row of the *Byte segment.*

SplitRow_TIFF

Splits 16-bit monochrome Composite Pixel Code.

Syntax

```
IppStatus ippiSplitRow_TIFF_16u8u_C1(const Ipp16u* pSrc, Ipp8u pDst[2],
    int srcLen);
```

Parameters

pSrc	Pointer to the input buffer containing 16-bit monochrome Composite Pixel Code.
srcLen	Number of elements in the source buffer.
pDst	Pointer to the output buffers.

Description

The function `ippiSplitRow_TIFF` is declared in the `ippj.h` file. This function takes the most significant and the least significant bytes of each of *srcLen* elements of the input buffer *pSrc*, and stores them sequentially to the output buffers pointed by *pDst*. The *pDst*[0] contains only the most significant bytes, and the *pDst*[1] contains only the least significant bytes.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcLen* has a negative or illegal value.

JoinRow_TIFF

Restores 16-bit monochrome Composite Pixel Code.

Syntax

```
IppStatus ippiJoinRow_TIFF_8u16u_C1(const Ipp8u pSrc[2], Ipp16u* pDst,
    int dstLen);
```

Parameters

pSrc	Pointers to the input buffer containing decoded RLE segments.
pDst	Pointer to the output buffer.
dsten	Number of elements in the destination buffer.

Description

The function `ippiJoinRow_TIFF` is declared in the `ippj.h` file.

This function restores 16-bit elements in the output buffer *pDst* by joining for each element the most significant and the least significant bytes stored in the source buffers *pSrc*[0] and *pSrc*[1] respectively. The number of elements is specified by the parameter *dstLen*.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if *pSrc* or *pDst* pointer is NULL.
ippStsSizeErr	Indicates an error condition if *dstLen* has a negative or illegal value.

Texture Compression Functions

This section describes the functions that perform compression of the textures. Textures are images that are applied to the surface of shapes to add visual detail or color. In the modern computer graphics textures are used extremely wide and can consume large amounts of system and video memory. The compression of textures reduces the size of the required memory.

Intel IPP implements the Microsoft DirectX* texture compression algorithms (DXTC or DXTn) also known as S3 Texture Compression (S3TC). It is a lossy compression with a fixed compression ratio of 4:1 or 8:1. This compression is a form of *block truncating coding* (BTC), where an image is divided into non-overlapping blocks and pixels in each block and quantized to a limited number of values. Intel IPP supports two variations of the S3TC algorithm: DXT1 (compression ratio 8:1) and DXT5 (compression ratio 4:1) (see also more details at http://msdn.microsoft.com/en-us/library/bb204843(VS.85).aspx).

Table 15-17 lists functions for texture compression described in more detail later in this section.

Table 15-17 Texture Compression Functions

Function Base Name	Description
EncodeTextureBlockFromRGBA	Encodes RGBA texture images.
DecodeTextureBlockToRGBA	Decodes compressed RGBA texture images.

EncodeTextureBlockFromRGBA

Encodes RGBA texture images.

Syntax

```
IppStatus ippiEncodeTextureBlockFromRGBA_DXT1_8u_C4C1R(const Ipp8u*
    pSrc, Ipp32u srcStep, IppiSize roiSize, Ipp8u* pDst);
IppStatus ippiEncodeTextureBlockFromRGBA_DXT5_8u_C4C1R(const Ipp8u*
    pSrc, Ipp32u srcStep, IppiSize roiSize, Ipp8u* pDst);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source image ROI in pixels.
pDst	Pointer to the destination array of the encoded blocks.

Description

The function `ippiEncodeTextureBlockFromRGBA` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function encodes the source image ROI *pSrc* from RGBA to DXT1 or DXT5 format in accordance with the descriptor `DXT1` and `DXT5` respectively. The user must ensure that the *pDst* buffer contains enough space for compressed data. For the DXT1 compression (compression ratio 8:1) the size of one compressed block is 8 bytes, for the DXT5 compression (compression ratio 4:1) the size of one compressed block is 16 bytes.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

DecodeTextureBlockToRGBA

Decodes compressed RGBA texture images.

Syntax

```
IppStatus ippiDecodeTextureBlockToRGBA_DXT1_8u_C1C4R(const Ipp8u* pSrc,
    Ipp8u* pDst, Ipp32u dstStep, IppiSize roiSize);
IppStatus ippiDecodeTextureBlockToRGBA_DXT5_8u_C1C4R(const Ipp8u* pSrc,
    Ipp8u* pDst, Ipp32u dstStep, IppiSize roiSize);
```

Parameters

pSrc	Pointer to the source array of the encoded blocks.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destonation image.
roiSize	Size of the destination image ROI in pixels.

Description

The function `ippiDecodeTextureBlockToRGBA` is declared in the `ippj.h` file. It operates with ROI (see Regions of Interest in Intel IPP).

This function decodes the source data *pSrc* and restores the RGBA texture image *pDst*. Source data are the encoded blocks in DXT1 or DXT5 formats in accordance with the descriptor `DXT1` and `DXT5` respectively. The user must ensure that the *pSrc* buffer contains enough compressed data. For the DXT1 compression (compression ratio 8:1) the size of one compressed block is 8 bytes, for the DXT5 compression (compression ratio 4:1) the size of one compressed block is 16 bytes.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if *pSrc* or *pDst* pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

Video Coding 16

This chapter describes the subset of Intel IPP functions designed for video coding applications. This subset is a library of functions for encoding and decoding of video data according to MPEG-1 ([ISO11172]), MPEG-2 ([ISO13818]), MPEG-4 ([ISO14496A]), DV ([IEC61834] and [SMPTE314M]), H.261 ([ITUH261]), H.263 ([ITUH263]), H.264 ([JVTG050] and [ITUH264]), AVS ([AVS]), and VC-1 ([SMPTE 421M]) standards. These functions have a convenient interface and present an appropriate solution for both encoding and decoding pipeline and, as all other parts of Intel IPP, are intended for the development of high-performance cross-platform code.

In addition to common data types used in Intel IPP for image processing (see Data Types in Chapter 2), video coding functions subset defines its specific data type:

Table 16-1 Conventional vs Intel IPP Data Types

Bit Depth	Conventional Type	Intel IPP Type
32	signed int	`Ipp32s`
16	signed short	`Ipp16s`
8	signed char	`Ipp8s`
32	unsigned int	`Ipp32u`
16	unsigned short	`Ipp16u`
8	unsigned char	`Ipp8u`

Descriptors in the names of video coding functions (see Function Naming in Chapter 2) may indicate the standard according to which the function operates (`MPEG1`, `MPEG2`, `MPEG4`, `H263`, `H264`, `DV`) or the size of a block of elements processed together (`16x16`, `16x8`, `8x16`, `8x8`, `8x4`). The first number is the width of the block and the second is its height. For functions that process one channel the descriptor `C1` is used. In-place function names contain the descriptor `I`, while functions are not in-place by default.

The use of video coding functions is demonstrated in Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

Descriptions of Intel IPP video coding functions are grouped into the following sections:

- General Functions
- MPEG-1 and MPEG-2
- DV
- MPEG-4
- H.261
- H.263
- H.264
- AVS
- VC-1

General Functions

These functions are quite universal and can be used in different codecs. This category includes functions of the following types:

- Variable Length Decoding functions that use Variable Length Code tables to decode video data.
- Motion Compensation functions that calculate sum of the residual block and the predicted block for reconstruction of the source block.
- Motion Estimation functions that calculate:
  - residual block - difference between source block and predicted block
  - some characteristics of the residual block
  - some characteristics of the source block. These characteristics can be used for comparison of the blocks.
- Scanning Functions that convert two-dimensional arrays into one-dimensional data and vice versa using the predefined scan pattern.
- Color Conversion functions that convert images with rotation of the destination image.
- Video Processing function that is used to process decompressed video data.

See Table 16-2 for the complete list of Video Coding general functions.

Structures and Enumerators

The structure IppMotionVector can be used in both H.263 and MPEG-4 video processing functions and is defined as follows:

```
typedef struct {
    Ipp16s dx;
    Ipp16s dy;
} IppMotionVector;
```

The following enumerator indicates the type of a decoded macroblock or a macroblock to be encoded. IPPVC_MB_STUFFING indicates that bit stuffing codeword was decoded from the bitstream or should be encoded into the bitstream.

```
enum
{
   IPPVC_MBTYPE_INTER = 0,        /* valid in P-picture or P-VOP*/
   IPPVC_MBTYPE_INTER_Q = 1,      /* P-picture or P-VOP*/
   IPPVC_MBTYPE_INTER4V = 2,      /* P-picture or P-VOP*/
   IPPVC_MBTYPE_INTRA = 3,        /* I- and P-picture, or I- and P-VOP*/
   IPPVC_MBTYPE_INTRA_Q = 4,      /* I- and P-picture, or I- and P-VOP*/
   IPPVC_MBTYPE_INTER4V_Q = 5,    /* P-picture or P-VOP*/
   IPPVC_MBTYPE_DIRECT = 6,       /* B-picture or B-VOP*/
   IPPVC_MBTYPE_INTERPOLATE = 7, /* B-picture or B-VOP*/
   IPPVC_MBTYPE_BACKWARD = 8,     /* B-picture or B-VOP*/
   IPPVC_MBTYPE_FORWARD = 9,      /* B-picture or B-VOP*/
   IPPVC_MB_STUFFING = 255        /* indicates bit stuffing codeword*/
};
```

The following enumerator indicates the type of scan performed on DCT coefficients:

```
enum
{
   IPPVC_SCAN_NONE = -1,       no scan to be performed
   IPPVC_SCAN_ZIGZAG = 0,      classical zigzag scan
   IPPVC_SCAN_VERTICAL = 1,    alternate vertical scan
   IPPVC_SCAN_HORIZONTAL = 2   alternate horizontal scan
};
```

See typical scan patterns in Figure 16-12 and the scanning process according to the classical scanning pattern in Figure 16-16.

The following enumerator indicates the type of interpolation performed on a picture. This enumerator is used, for example, in spatial scalability mode.

```
enum
{
   IPPVC_INTERP_NONE = 0,         no interpolation
```

```
    IPPVC_INTERP_HORIZONTAL = 1,  1-D horizontal interpolation
    IPPVC_INTERP_VERTICAL = 2,    1-D vertical interpolation
    IPPVC_INTERP_2D = 3           2-D interpolation
};
```

The following enumerator indicates the type of rotate operation for general color conversion functions:

```
enum {
IPPVC_ROTATE_DISABLE = 0,
IPPVC_ROTATE_90CCW = 1,
IPPVC_ROTATE_90CW = 2,
IPPVC_ROTATE_180 = 3
};
```

The following enumerator indicates the type of color space conversion for general color conversion functions:

```
enum
{
IPPVC_CbYCr422ToBGR565 = 0,
IPPVC_CbYCr422ToBGR555 = 1
};
```

The enumerator `IPPVC_MC_APX` indicates the type of prediction for motion compensation or motion estimation.

```
typedef enum _IPPVC_MC_APX{
        IPPVC_MC_APX_FF = 0x0,
        IPPVC_MC_APX_FH = 0x4,
        IPPVC_MC_APX_HF = 0x8,
        IPPVC_MC_APX_HH = 0x0c
    }IPPVC_MC_APX;
```

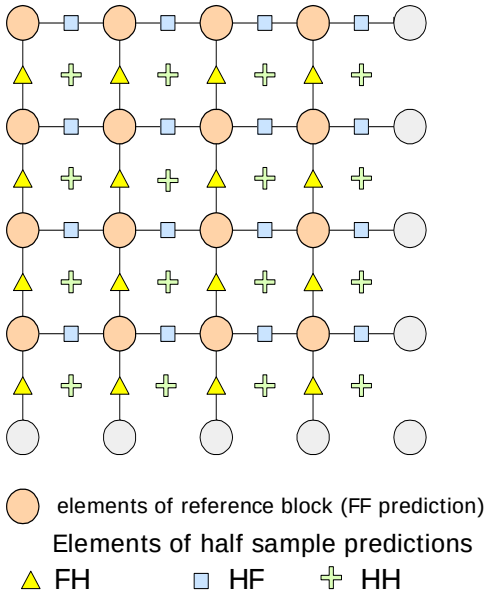
The first descriptors `FF, FH, HF, HH` show whether prediction is accurate to full pel (`F`) or to half a pel (`H`). The first letter indicates the accuracy in horizontal direction and the second — in vertical direction.

Most of general functions use parameter *mcType* (see enumeration `IPPVC_MC_APX`). It is used for calculating prediction on the basis of block in the reference frame. See Figure 16-1.

- If *mcType* = `IPPVC_MC_APX_FF`, the reference block is used as the prediction block.
- If *mcType* = `IPPVC_MC_APX_FH` or `IPPVC_MC_APX_HF`, each element of the prediction block is calculated as average of two elements of the reference block.

- If *mcType* = IPPVC_MC_APX_HH, each element of the prediction block is calculated as average of four elements of the reference block.

Figure 16-1 Predictions for 4x4 Reference Block



UV Block

UV block of size (2**H*)x*W* combines *U* block of size *H*x*W* and *V* block of size *H*x*W*. The formula for *UV* is as follows:

$$UV[2*i+1,\ j] = V[i,\ j]$$

$$UV[2*i,\ j] = U[i,\ j],$$

where *i*=[0, *H*-1] and *j*=[0, *W*-1]. See the figure below for an example of a *UV* block.

Figure 16-2 16x8 UV Block (8x8 U Block and 8x8 V Block)

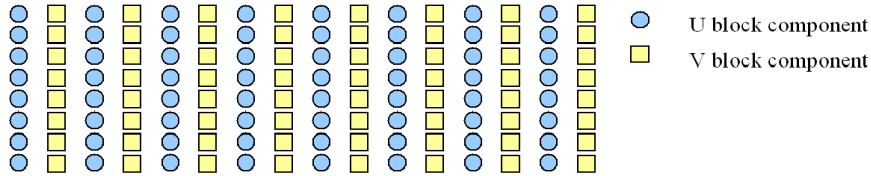


Table 16-2 General Functions

Function Short Name	Description
	Variable Length Decoding
HuffmanTableInitAlloc	Allocates memory and initializes structure for table with one-to-one code/value correspondence.
HuffmanRunLevelTableInitAlloc	Allocates memory and initializes structure for table with one-to-two code/value correspondence.
DecodeHuffmanOne	Decodes one code using specified table and gets one decoded value.
DecodeHuffmanPair	Decodes one code using specified table and gets two decoded values.
HuffmanTableFree	Frees memory allocated for VLC table.
	Motion Compensation
MC16x16	Performs motion compensation for a predicted 16X16 block.
MC16x8	Performs motion compensation for a predicted 16X8 block.
MC8x16	Performs motion compensation for a predicted 8X16 block.
MC8x8	Performs motion compensation for a predicted 8X8 block.

Table 16-2 General Functions (continued)

Function Short Name	Description
MC8x4	Performs motion compensation for a predicted 8X4 block.
MC4x8	Performs motion compensation for a predicted 4X8 block.
MC4x4	Performs motion compensation for a predicted 4X4 block.
MC2x4	Performs motion compensation for a predicted 2X4 block.
MC4x2	Performs motion compensation for a predicted 4X2 block.
MC2x2	Performs motion compensation for a predicted 2X2 block.
MC16x4	Performs motion compensation for predicted UV 16X4 block.
MC16x8UV	Performs motion compensation for predicted UV 16X8 block.
MC16x16B	Performs motion compensation for a bi-predicted 16X16 block.
MC16x8B	Performs motion compensation for a bi-predicted 16X8 block.
MC8x16B	Performs motion compensation for a bi-predicted 8X16 block.
MC8x8B	Performs motion compensation for a bi-predicted 8X8 block.
MC8x4B	Performs motion compensation for a bi-predicted 8X4 block.
MC4x8B	Performs motion compensation for a bi-predicted 4X8 block.
MC4x4B	Performs motion compensation for a bi-predicted 4X4 block.
MC2x4B	Performs motion compensation for a bi-predicted 2X4 block.
MC4x2B	Performs motion compensation for a bi-predicted 4X2 block.
MC2x2B	Performs motion compensation for a bi-predicted 2X2 block.

Table 16-2 General Functions (continued)

Function Short Name	Description
MC16x4B	Performs motion compensation for bi-predicted UV 16X4 block.
MC16x8UVB	Performs motion compensation for bi-predicted UV 16X8 block.
Copy8x8, Copy16x16	Copy the fixed size block to the destination block.
Copy8x4HP, Copy8x8HP, Copy16x8HP, Copy16x16HP	Copy fixed size blocks with half-pixel accuracy.
InterpolateAverage8x4, InterpolateAverage8x8, InterpolateAverage16x8, InterpolateAverage16x16	Interpolate source block according to half-pixel offset and average the result with destination block.
Add8x8	Adds two blocks with saturation.
Add8x8HP	Adds blocks interpolated with half-pixel accuracy prediction to difference with saturation.
AddC8x8	Adds a constant to 8x8 block with saturation.
Average8x8, Average16x16	Average two blocks.
	Motion Estimation
Difference Evaluation	
GetDiff16x16	Evaluates difference between current predicted and reference blocks of 16x16 elements.
GetDiff16x8	Evaluates difference between current predicted and reference blocks of 16x8 elements.
GetDiff8x8	Evaluates difference between current predicted and reference blocks of 8x8 elements.
GetDiff8x16	Evaluates difference between current predicted and reference blocks of 8x16 elements.
GetDiff8x4	Evaluates difference between current predicted and reference blocks of 8x4 elements.
GetDiff4x4	Evaluates difference between current predicted and reference 4x4 blocks.
GetDiff16x16B	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.

Table 16-2 General Functions (continued)

Function Short Name	Description
GetDiff16x8B	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.
GetDiff8x8B	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.
GetDiff8x16B	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.
GetDiff8x4B	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.
Sub4x4, Sub8x8, Sub16x16	Subtract two blocks and store the result in the third block.
SubSAD8x8	Subtracts two block, stores the result in the third block and computes a sum of absolute differences.
Sum of Squares of Differences Evaluation	
SqrDiff16x16	Evaluates sum of squares of differences between current and reference blocks.
SqrDiff16x16B	Evaluates sum of squares of differences between the current bi-predicted block and the mean of two reference blocks.
SSD8x8	Evaluates sum of squares of differences between current and reference 8X8 blocks.
SSD4x4	Evaluates sum of squares of differences between current and reference 4X4 blocks.
Block Variance and Mean Evaluation	
VarMean8x8	Evaluates variance and mean of a 8X8 block of unsigned char or short integer values.
Evaluation of Variances and Means of Blocks of Difference Between Two Blocks	
VarMeanDiff16x16	Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.
VarMeanDiff16x8	Evaluates variances and means of four 8x8 blocks of difference between two 16x8 blocks.

Table 16-2 General Functions (continued)

Function Short Name	Description
Block Variance Evaluation	
Variance16x16	Evaluates variance of current block.
Evaluation of Block Deviation	
MeanAbsDev8x8	Evaluates mean absolute deviation for a 8x8 block.
MeanAbsDev16x16	Evaluates mean absolute deviation for a 16x16 block.
Edges Detection	
EdgesDetect16x16	Detects edges inside a 16X16 block.
SAD Functions	
SAD16x16	Evaluates sum of absolute difference between current and reference 16X16 blocks.
SAD16x8	Evaluates sum of absolute difference between current and reference 16X8 blocks.
SAD8x16	Evaluates sum of absolute difference between current and reference 8X16 blocks.
SAD8x8	Evaluates sum of absolute difference between current and reference 8X8 blocks.
SAD8x4	Evaluates sum of absolute difference between current and reference 8X4 blocks.
SAD4x8	Evaluates sum of absolute difference between current and reference 4X8 blocks.
SAD4x4	Evaluates sum of absolute difference between current and reference 4X4 blocks.
SAD16x16Blocks8x8	Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.
SAD16x16Blocks4x4	Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.
SATD16x16	Evaluates sum of absolute transformed differences between current and reference 16X16 blocks using 4x4 transform.

Table 16-2 General Functions (continued)

Function Short Name	Description
SATD16x8	Evaluates sum of absolute transformed differences between current and reference 16X8 blocks using 4x4 transform.
SATD8x16	Evaluates sum of absolute transformed differences between current and reference 8X16 blocks using 4x4 transform.
SATD8x8	Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 4x4 transform.
SATD8x4	Evaluates sum of absolute transformed differences between current and reference 8X4 blocks using 4x4 transform.
SATD4x8	Evaluates sum of absolute transformed differences between current and reference 4X8 blocks using 4x4 transform.
SATD4x4	Evaluates sum of absolute transformed differences between current and reference 4X4 blocks using 4x4 transform.
SAT8x8D	Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 8x8 transform.
FrameFieldSAD16x16	Calculates SAD between field lines and SAD between frame lines of block 16x16.
Sum of Differences Evaluation	
SumsDiff16x16Blocks4x4	Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 16X16 blocks.
SumsDiff8x8Blocks4x4	Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 8X8 blocks.
	Scanning
ScanInv	Performs classical zigzag, alternate-horizontal, or alternate-vertical inverse scan on a block stored in a compact buffer.
ScanFwd	Performs classical zigzag, alternate-horizontal, or alternate-vertical forward scan on a block.

Table 16-2 General Functions (continued)

Function Short Name	Description
	Color Conversion
CbYCr422ToYCbCr420_Rotate	Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image with rotation.
ResizeCCRotate	Creates a low-resolution preview image for high-resolution video or still capture applications.
	Video Processing
Deinterlacing	
DeinterlaceFilterTriangle	Deinterlaces video plane.
DeinterlaceFilterCAVT	Performs deinterlacing of two-field image.
Median	Creates an image consisting of median values of three source images.
DeinterlaceMedianThreshold	Performs median deinterlacing with threshold.
DeinterlaceEdgeDetect	Generates image field using `EdgeDetect` filter.
DeinterlaceMotionAdaptive	Performs deinterlacing using temporal and spatial interpolations.
DeinterlaceBlendInitAlloc	Allocates and initializes an internal structure for `DeinterlaceBlend` function.
DeinterlaceBlendFree	Releases memory allocated by `DeinterlaceBlendInitAlloc` function.
DeinterlaceBlend	Calculates output pixel as alpha blends of the results of two filters applied to input pixels.
Denoising	
FilterDenoiseCASTInit	Initializes a denoise specification structure for content adaptive spatio-temporal noise reduction filtering.
FilterDenoiseCAST	Performs content adaptive spatio-temporal (CAST) noise reduction filtering.
FilterDenoiseSmooth	Performs spatial noise reduction (SNR) filtering.
FilterDenoiseAdaptiveInitAlloc	Creates and initializes a denoise specification structure for spatio-temporal adaptive noise reduction filtering.

Table 16-2 General Functions (continued)

Function Short Name	Description
FilterDenoiseAdaptiveFree	Closes a denoise specification structure for spatio-temporal adaptive noise reduction filtering.
FilterDenoiseAdaptive	Performs spatio-temporal adaptive noise reduction filtering.
FilterDenoiseMosquitoInitAlloc	Creates and initializes a denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering
FilterDenoiseMosquitoFree	Closes a denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering
FilterDenoiseMosquito	Performs spatio-temporal motion adaptive mosquito noise reduction filtering.

Variable Length Decoding

Video data in the bit stream is encoded with Variable Length Code (VLC) tables so that the shortest codes correspond to the most frequent values and the longer codes correspond to the less frequent values. Tables for each video standard contain specific possible codes and their values.

VLC decoding functions can work with two types of tables:

1. Tables, in which one code corresponds to one value (see Table 16-3 for an example)
2. Tables, in which one code corresponds to two values (see Table 16-4 for an example).

Table 16-3 One-to-One Code/Value Correspondence

Code	Value
0001 1	-3
0011	-2
011	-1
1	0
010	1
0010	2

Table 16-3 One-to-One Code/Value Correspondence (continued)

Code	Value
0001 0	3

Table 16-4 One-to-Two Code/Value Correspondence

Code	Value 1	Value 2
0001 1	1	0
0011	1	1
011	1	2
1	2	0
010	2	2
0010	3	2
0001 0	4	1

Video data VLC-decoding starts with memory allocation and table initialization. Then the decoding proper should be made and the allocated memory must be released.

Table 16-5 VLC Decoding Functions

One-to-One Correspondence	One-to-Two Correspondence	Short Description
HuffmanTableInitAlloc	HuffmanRunLevelTableInitAlloc	Allocates memory and initializes structure that is used for decoding.
DecodeHuffmanOne	DecodeHuffmanPair	Decodes one code using specified table.
HuffmanTableFree		Frees memory allocated for VLC table.

For using `ippiVCHuffmanInitAlloc_32s` and `ippiVCHuffmanInitAllocRL_32s` functions, the source table should have the following structure:

Example 16-1 Source Table Structure for `HuffamanTableInitAlloc` Functions

```
static Ipp32s Table[]=
{
```

Example 16-1 Source Table Structure for `HuffamanTableInitAlloc` Functions (continued)

```
max_bits,                          // The maximum length of code
total_subt,                        // The total number of all subtables
sub_sz1,                           // The sizes of subtables. Their sum must
sub_sz2,                           // be equal to the maximum length of code
…,                                 // max_bits and their number should be
                                   // equal to total_subt.
N1,                                // The number of 1-bit codes
Code1, Value11, [Value21]          //
Code2, Value12, [Value22]          // The 1-bit codes
…,                                 //
N2,                                // The number of 2-bit codes
Code1, Value11, [Value21]          //
Code2, Value12, [Value22]          // The 2-bit codes
…,                                 //
Nm,                                // The number of max_bits, -bit codes
Code1, Value11, [Value21]          //
Code2, Value12, [Value22]          // max_bits codes
…,                                 //
-1                                 // The significant value to indicate the
                                   // end of table
};
```

Notes:

1. The values $Value2_i$ are used for the second type table (one-to-two code/value correspondence) and must be in the range [-32768, 32767] or [0, 65535].
2. The values $Value1_i$ must be in the range:

[-8388608, 8388607] or [0, 16777215] in the case of first type table (one-to-one correspondence)

[-128, 127] or [0, 255] in the case of second type table (one-to-two correspondence).

Using Subtables

Subtables are used for large source tables. Internally, VLC decoding is done by table lookups. A straightforward approach would be to have one lookup of a lookup table of size 2^L, where L is the maximum number of code bits. This approach however is proven to be memory inefficient. Therefore, subtables are used to balance the memory and the lookup speed. Using subtables reduces the memory size but increases the number of lookups. The greater is the number of subtables, the smaller is the structure size but the number of decoding operations increases. Table structure of this kind provides optimal ratio between the table redundancy and the number of decoding operations. Each subtable works on

some numbers of bits out of L. In `ippiVCHuffmanInitAlloc_32s` and `ippiVCHuffmanInitAllocRL_32s` functions subtables sizes are specified manually. Any combinations of subtables are possible as long as their sum adds up to L.

Consider an example of VLC decoding. Table 16-6 gives codes and their values.

Table 16-6 Example of VLC Decoding

Code	Value
0	A
1100	B
101	C
1110	D
100	E
11110	F
11111	G
1101	H

First, let us consider decoding when subtables are not used. In this case the source table should look as follows:

```
static Ipp32s Table[]=
{
5,                                      // The maximum length of code
1,                                      // The total number of all subtables
5,                                      // The size of 1 subtable
1,                                      // The number of 1-bit codes
0/*0*/,A,                               // code1, value1
0,                                      // The number of 2-bit codes
2                                       // The number of 3-bit codes
5/*101*/,C,                             // code1, value1
4/*100*/,E,                             // code2, value2
3                                       // The number of 4-bit codes
12/*1100*/,B,                           // code1, value1
14/*1110*/,D,                           // code2, value2
13/*1101*/,H,                           // code3, value3
2                                       // The number of 4-bit codes
30/*11110*/,F,                          // code1, value1
31/*11111*/,G,                          // code2, value2
-1
};
```

The function `ippiVCHuffmanInitAlloc_32s` is used to form a structure that contains the following data:

Table 16-7 Structure Data For a Case With One Subtable

Code (length=5)	Value	Length
00000	A	1
00000	A	1
...	...	
01111	A	1
10100	C	3
10101	C	3
10110	C	3
10111	C	3
10000	E	3
10000	E	3
10000	E	3
10000	E	3
11000	B	4
11001	B	4
11100	D	4
11101	D	4
11010	H	4
11011	H	4
11110	F	5
11111	G	5

In process of decoding, 5 bits are extracted from the bitstream. They are used to find a corresponding code in Table 16-7 and matching value and length. (5-Length) bits are returned to the bitstream.

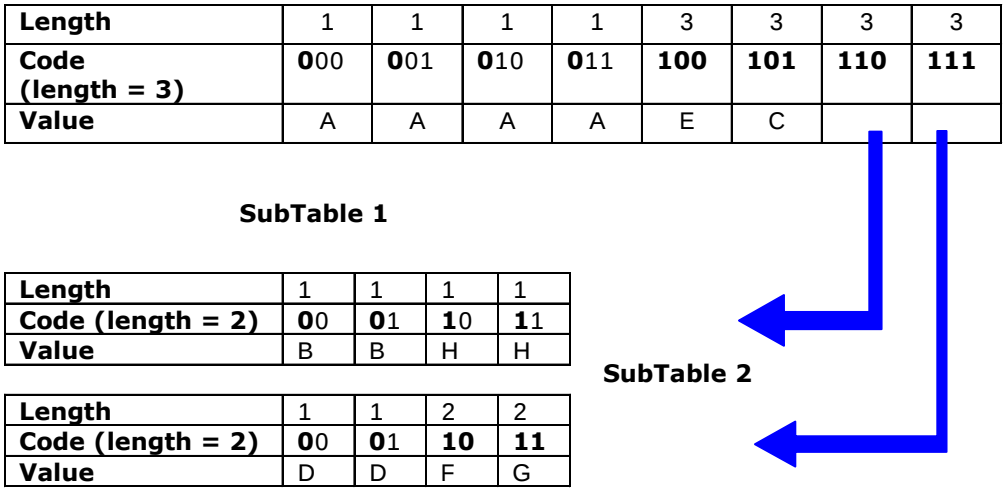
Now, consider an example when two subtables are used for decoding. The source table structure will look as follows:

```
static Ipp32s Table[]=
{
```

```
5,                                      // The maximum length of code
2,                                      // The total number of all subtables
3,                                      // The size of subtable 1
2,                                      // The size of subtable 2
1,                                      // The number of 1-bit codes
0/*0*/,A,                               // code1, value1
0,                                      // The number of 2-bit codes
2                                       // The number of 3-bit codes
5/*101*/,C,                             // code1, value1
4/*100*/,E,                             // code2, value2
3                                       // The number of 4-bit codes
12/*1100*/,B,                           // code1, value1
14/*1110*/,D,                           // code2, value2
13/*1101*/,H,                           // code3, value3
2                                       // The number of 4-bit codes
30/*11110*/,F,                          // code1, value1
31/*11111*/,G,                          // code2, value2
-1
};
```

The function `ippiVCHuffmanInitAlloc_32s` is used to form a structure that contains two subtables:

Figure 16-3 VLC Subtables



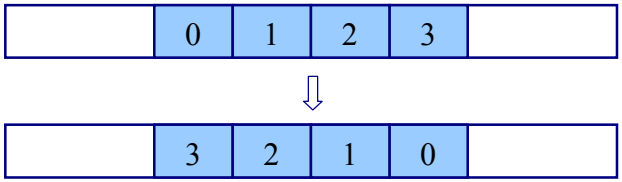
Subtable 1 contains the values with the code length that does not exceed 3. Subtable 2 contains the values with the code length that exceeds 3, with the corresponding code consisting of 3-bit code from Subtable 1 and 2-bit code from Subtable 2.

In process of decoding 3 bits are extracted from the bitstream. They are used to find a corresponding code in Subtable 1. The matching value and length are found if the code is other than 100 or 111. (3-Length) bits are returned to the bitstream. Otherwise, two more bits are extracted from the bitstream to find respective value and length in Subtable 2 and (2-Length) bits are returned to the bitstream afterwards.

`ppBitStream` and `pOffset` Parameters

Before decoding and applying the Intel IPP functions, all bytes in each 32-bit double word in the bit stream must be flipped (Figure 16-4) to improve performance of decoding functions.

Figure 16-4 Flipping of Bytes

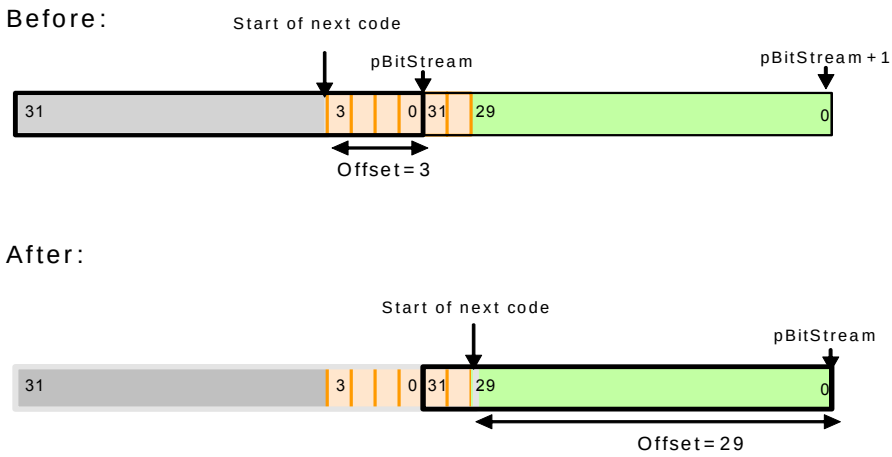


The parameters *ppBitStream* and *pOffset* define start position for a subsequent code: current element of the array and position in this element. These parameters are updated by function.

ppBitStream	Double pointer to the current position in the bitstream.
pOffset	Pointer to the offset between the bit that* *ppBitStream* points to and the start of the code.

See Figure 16-5 for an example of getting one code from the bitstream.

Figure 16-5 Getting One Code From Bitstream



The VLC functions are used in the decoders included into Intel IPP Samples. See introduction to the chapter.

HuffmanTableInitAlloc

Allocates memory and initializes structure for table with one-to-one code/value correspondence.

Syntax

```
IppStatus ippiHuffmanTableInitAlloc_32s(const Ipp32s* pSrcTable,
    IppVCHuffmanSpec_32s** ppDstSpec);
```

Parameters

pSrcTable	Pointer to the source table (see Source Table Structure).
ppDstSpec	Double pointer to the destination decoding table.

Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanTableInitAlloc_32s` allocates memory and initializes structure for a table in which one code corresponds to one value (See Table 16-3). This structure is used for decoding.

See Table 16-11 for details of specific use of the function in MPEG1, MPEG2 standards.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsMemAllocErr`	Indicates an error when no memory is allocated.

HuffmanRunLevelTableInitAlloc

Allocates memory and initializes structure for table with one-to-two code/value correspondence.

Syntax

```
IppStatus ippiHuffmanRunLevelTableInitAlloc_32s(const Ipp32s* pSrcTable,
    IppVCHuffmanSpec_32s** ppDstSpec);
```

pSrcTable	Pointer to the source table (see Source Table Structure).
ppDstSpec	Double pointer to the destination decoding table.

Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanRunLevelTableInitAlloc_32s` allocates memory and initializes structure for a table in which one code corresponds to two value (See Table 16-4). This structure is used for decoding.

See Table 16-11 for details of specific use of the function in MPEG1, MPEG2 standards.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsMemAllocErr`	Indicates an error when no memory is allocated.

Decoding with non-Run-Level Tables

This function decodes all codes except for DCT coefficients.

DecodeHuffmanOne

Decodes one code using specified table and gets one decoded value.

Syntax

```
IppStatus ippiDecodeHuffmanOne_1u32s(Ipp32u** ppBitStream, Ipp32s* pOffset,
    Ipp32s* pDst, const IppVCHuffmanSpec_32s* pDecodeTable);
```

Parameters

ppBitStream	Double pointer to the current position in the bitstream.
pOffset	Pointer to offset between the bit that *ppBitStream* points to and the start of the code. *pOffset* may vary from zero to 31.
pDst	Pointer to the destination result, that is, one value.
pDecodeTable	Pointer to the decoding table.

Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeHuffmanOne_1u32s` decodes one code from the bitstream using a specified table, sends the result (one value) to destination data, and resets the pointers to new positions (See example in Figure 16-5). The function uses the table derived by the `HuffmanTableInitAlloc` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsH263VLCCodeErr`	Indicates an error in process of decoding.

DecodeHuffmanPair

Decodes one code using specified table and gets two decoded values.

Syntax

```
IppStatus ippiDecodeHuffmanPair_1u16s(Ipp32u** ppBitStream, Ipp32s* pOffset,
    const IppVCHuffmanSpec_32s* pDecodeTable, Ipp8s* pFirst, Ipp16s* pSecond);
```

Parameters

ppBitStream	Double pointer to the current position in the bitstream.
pOffset	Pointer to offset between the bit that** *ppBitStream* points to and the start of the code. *pOffset* may vary from zero to 31.
pDecodeTable	Pointer to the decoding table.
pFirst	Pointer to the first destination value.
pSecond	Pointer to the second destination value.

Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeHuffmanPair_1u16s` decodes one code from the bitstream using a specified table, sends the result (two values) to destination data, and resets the pointers to new positions (See example in Figure 16-5). The function uses the table derived by the `HuffmanRunLevelTableInitAlloc` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsH263VLCCodeErr`	Indicates an error in process of decoding.

Memory Release

HuffmanTableFree

Frees memory allocated for VLC table.

Syntax

```
IppStatus ippiHuffmanTableFree_32s(IppVCHuffmanSpec_32s** ppDecodeTable);
```

Parameters

ppDecodeTable	Double pointer to the decoding table.

Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanTableFree_32s` frees memory at *ppDecodeTable* allocated for VLC table.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Motion Compensation

These functions calculate sum of the residual block and the predicted block for reconstruction of the source block. After processing, the data is converted with saturation from `Ipp16s` to `Ipp8u` type.

Motion compensation functions are divided into two groups:

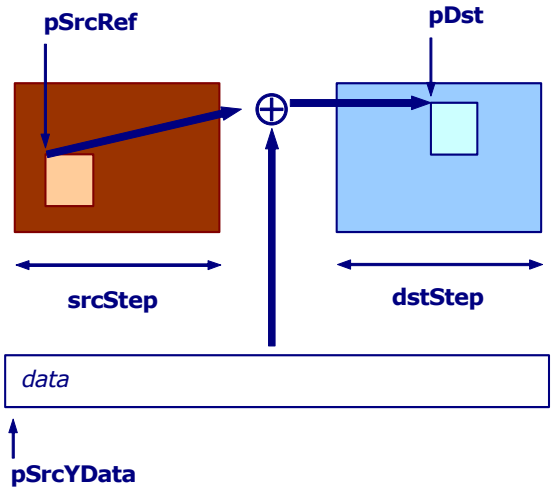
- Functions for predicted block, using one reference block for prediction
- Functions for bi-predicted block, using two reference blocks for prediction; in this case predictions are first calculated for each reference block, and then prediction is calculated as average of two predictions.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

Predicted Blocks

These functions use one reference block for prediction.

Figure 16-6 Motion Compensation Scheme for Predicted Block



MC16x16

Performs motion compensation for predicted 16X16 block.

Syntax

```
IppStatus ippiMC16x16_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC16x16_8u_C1 calculates sum of 16x16 residual block and 16x16 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC16x8

Performs motion compensation for predicted 16X8 block.

Syntax

```
IppStatus ippiMC16x8_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC16x8_8u_C1 calculates sum of 16x8 residual block and 16x8 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC8x16

Performs motion compensation for predicted 8X16 block.

Syntax

```
IppStatus ippiMC8x16_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC8x16_8u_C1 calculates sum of 8x16 residual block and 8x16 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC8x8

Performs motion compensation for predicted 8X8 block.

Syntax

```
IppStatus ippiMC8x8_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC8x8_8u_C1 calculates sum of 8x8 residual block and 8x8 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC8x4

Performs motion compensation for predicted 8X4 block.

Syntax

```
IppStatus ippiMC8x4_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC8x4_8u_C1 calculates sum of 8x4 residual block and 8x4 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC4x8

Performs motion compensation for predicted 4X8 block.

Syntax

```
IppStatus ippiMC4x8_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC4x8_8u_C1 calculates sum of 4x8 residual block and 4x8 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC4x4

Performs motion compensation for predicted 4X4 block.

Syntax

```
IppStatus ippiMC4x4_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC4x4_8u_C1 calculates sum of 4x4 residual block and 4x4 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC2x4

Performs motion compensation for predicted 2X4 block.

Syntax

```
IppStatus ippiMC2x4_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC2x4_8u_C1 calculates sum of 2x4 residual block and 2x4 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC4x2

Performs motion compensation for predicted 4X2 block.

Syntax

```
IppStatus ippiMC4x2_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC4x2_8u_C1 calculates sum of 4x2 residual block and 4x2 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC2x2

Performs motion compensation for predicted 2X2 block.

Syntax

```
IppStatus ippiMC2x2_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC2x2_8u_C1 calculates sum of 2x2 residual block and 2x2 predicted block for reconstruction of the source block (See Figure 16-6). Prediction is calculated on the basis of the reference block and *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC16x4

Performs motion compensation for predicted UV 16X4 block.

Syntax

```
IppStatus ippiMC16x4_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const Ipp16s*
    pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC16x4_8u_C1 calculates sum of 16x4 residual UV block and 16x4 predicted UV block for reconstruction of the source UV block (See Figure 16-2). Prediction is calculated on the basis of the reference UV block and *mcType* (see IPPVC_MC_APX).

Half sample prediction is calculated taking into account structure of UV block. So (horizontally) neighboring elements of U-block (or V-block) have indexes *i* and *i*+2 in UV block.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC16x8UV

Performs motion compensation for predicted UV 16X8 block.

Syntax

```
IppStatus ippiMC16x8UV_8u_C1(const Ipp8u* pSrcRef, Ipp32s srcStep, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    mcType, Ipp32s roundControl);
```

Parameters

pSrcRef	Pointer to the reference intra block.
srcStep	Size of the row in bytes, specifying the aligned reference frame width.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
mcType	MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiMC16x8UV_8u_C1 calculates sum of 16x8 residual UV block and 16x8 predicted UV block for reconstruction of the source UV block (See Figure 16-2). Prediction is calculated on the basis of the reference UV block and *mcType* (see IPPVC_MC_APX).

Half sample prediction is calculated taking into account structure of UV block. So (horizontally) neighboring elements of U-block (or V-block) have indexes *i* and *i*+2 in UV block.

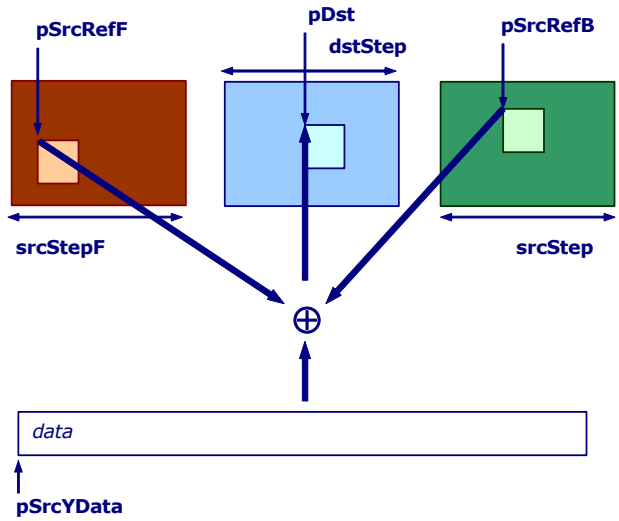
Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Bi-Predicted Blocks

These functions use two reference blocks for prediction. Predictions are calculated for each reference block, and then prediction is calculated as average of two predictions. The prediction calculated as a rounded average of two blocks is the first to be added to the data.

Figure 16-7 Motion Compensation Scheme for Bi-Predicted Block



MC16x16B

Performs motion compensation for bi-predicted block.

Syntax

```
IppStatus ippiMC16x16B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type IPPVC_MC_APX.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of the ow in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type IPPVC_MC_APX.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x16B_8u_C1` calculates sum of 16x16 residual block and 16x16 predicted block, which is calculated as average of two 16x16 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC16x8B

Performs motion compensation for bi-predicted 16X8 block.

Syntax

```
IppStatus ippiMC16x8B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type IPPVC_MC_APX.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of the row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type IPPVC_MC_APX.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x8B_8u_C1` calculates sum of 16x8 residual block and 16x8 predicted block, which is calculated as average of two 16x8 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see `IPPVC_MC_APX`).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

MC8x16B

Performs motion compensation for bi-predicted 8X16 block.

Syntax

```
IppStatus ippiMC8x16B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type `IPPVC_MC_APX`.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of the row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type `IPPVC_MC_APX`.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.

dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x16B_8u_C1` calculates sum of 8x16 residual block and 8x16 predicted block, which is calculated as average of two 8x16 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see `IPPVC_MC_APX`).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

MC8x8B

Performs motion compensation for bi-predicted 8X8 block.

Syntax

```
IppStatus ippiMC8x8B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type `IPPVC_MC_APX`.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of the row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type `IPPVC_MC_APX`.

pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x8B_8u_C1` calculates sum of 8x8 residual block and 8x8 predicted block, which is calculated as average of two 8x8 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see `IPPVC_MC_APX`).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

MC8x4B

Performs motion compensation for bi-predicted 8X4 block.

Syntax

```
IppStatus ippiMC8x4B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type `IPPVC_MC_APX`.

pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type IPPVC_MC_APX.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x4B_8u_C1` calculates sum of 8x4 residual block and 8x4 predicted block, which is calculated as average of two 8x4 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC4x8B

Performs motion compensation for bi-predicted 4X8 block.

Syntax

```
IppStatus ippiMC4x8B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type IPPVC_MC_APX.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type IPPVC_MC_APX.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x8B_8u_C1` calculates sum of 4x8 residual block and 4x8 predicted block, which is calculated as average of two 4x8 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see IPPVC_MC_APX).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

MC4x4B

Performs motion compensation for bi-predicted 4X4 block.

Syntax

```
IppStatus ippiMC4x4B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type IPPVC_MC_APX.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type IPPVC_MC_APX.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x4B_8u_C1` calculates sum of 4x4 residual block and 4x4 predicted block, which is calculated as average of two 4x4 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see IPPVC_MC_APX).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

MC2x4B

Performs motion compensation for bi-predicted 2X4 block.

Syntax

```
IppStatus ippiMC2x4B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type IPPVC_MC_APX.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type IPPVC_MC_APX.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC2x4B_8u_C1` calculates sum of 2x4 residual block and 2x4 predicted block, which is calculated as average of two 2x4 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see `IPPVC_MC_APX`).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

MC4x2B

Performs motion compensation for bi-predicted 4X2 block.

Syntax

```
IppStatus ippiMC4x2B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type `IPPVC_MC_APX`.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type `IPPVC_MC_APX`.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.

dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x2B_8u_C1` calculates sum of 4x2 residual block and 4x2 predicted block, which is calculated as average of two 4x2 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see `IPPVC_MC_APX`).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

MC2x2B

Performs motion compensation for bi-predicted 2X2 block.

Syntax

```
IppStatus ippiMC2x2B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type `IPPVC_MC_APX`.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type `IPPVC_MC_APX`.

pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC2x2B_8u_C1` calculates sum of 2x2 residual block and 2x2 predicted block, which is calculated as average of two 2x2 predictions (F-prediction and B-prediction) (See Figure 16-7). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see `IPPVC_MC_APX`).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

MC16x4B

Performs motion compensation for bi-predicted UV 16X4 block.

Syntax

```
IppStatus ippiMC16x4B_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type `IPPVC_MC_APX`.

pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of the row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type IPPVC_MC_APX.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x4B_8u_C1` calculates sum of 16x4 residual UV block and 16x4 predicted UV block, which is calculated as average of two 16x4 predictions (F-prediction and B-prediction) (See Figure 16-2). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see IPPVC_MC_APX).

Half sample prediction is calculated taking into account structure of UV block. So (horizontally) neighboring elements of U-block (or V-block) have indexes *i* and *i*+2 in UV block.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

MC16x8UVB

Performs motion compensation for bi-predicted 16X8 block.

Syntax

```
IppStatus ippiMC16x8BUV_8u_C1(const Ipp8u* pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u* pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s* pSrcYData, Ipp32s srcYDataStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

Parameters

pSrcRefF	Pointer to the forward reference block.
srcStepF	Size of the row in bytes, specifying the aligned forward reference frame width.
mcTypeF	Forward MC type IPPVC_MC_APX.
pSrcRefB	Pointer to the backward reference block.
srcStepB	Size of row in bytes, specifying the aligned backward reference frame width.
mcTypeB	Backward MC type IPPVC_MC_APX.
pSrcYData	Pointer to the data obtained after inverse DCT.
srcYDataStep	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
pDst	Pointer to the destination predicted block.
dstStep	Size of the row in bytes, specifying the aligned destination frame width.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x8BUV_8u_C1` calculates sum of 16x8 residual UV block and 16x8 predicted UV block, which is calculated as average of two 16x8 predictions (F-prediction and B-prediction) (See Figure 16-2). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see IPPVC_MC_APX).

Half sample prediction is calculated taking into account structure of UV block. So (horizontally) neighboring elements of U-block (or V-block) have indexes *i* and *i*+2 in UV block.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

Copy8x8, Copy16x16

Copy fixed size block.

Syntax

```
IppStatus ippiCopy8x8_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep);
IppStatus ippiCopy16x16_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep);
```

Parameters

pSrc	Pointer to the source block.
srcStep	Distance in bytes between starts of the consecutive lines in the source block.
pDst	Pointer to the destination block.
dstStep	Distance in bytes between starts of the consecutive lines in the destination block.

Description

The functions ippiCopy8x8_8u_C1R and ippiCopy16x16_8u_C1R are declared in the ippvc.h file. These functions copy the source block to the destination block.

These functions are used in the H.261, H.263, and MPEG-4 encoders and decoders included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

Copy8x4HP, Copy8x8HP, Copy16x8HP, Copy16x16HP

Copy fixed size blocks with half-pixel accuracy.

Syntax

```
IppStatus ippiCopy8x4HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, int acc, int rounding);
IppStatus ippiCopy8x8HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, int acc, int rounding);
IppStatus ippiCopy16x8HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, int acc, int rounding);
IppStatus ippiCopy16x16HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, int acc, int rounding);
```

Parameters

pSrc	Pointer to the source block.
srcStep	Distance in bytes between starts of the consecutive lines in the source block.
pDst	Pointer to the destination block.
dstStep	Distance in bytes between starts of the consecutive lines in the destination block.
acc	Parameter that determines half-pixel accuracy.
rounding	Parameter that determines type of rounding for pixel interpolation; may be 0 or 1.

Description

The functions ippiCopy8x4HP_8u_C1R, ippiCopy8x8HP_8u_C1R, ippiCopy16x8HP_8u_C1R, and ippiCopy16x16HP_8u_C1R are declared in the ippvc.h file. These functions copy the source block to the destination block with half-pixel accuracy. Parameter *rounding* has the same meaning as RTYPE in [ITUH263] and

vop_rounding_type in [ISO14496]. Parameter *acc* is a two-bit value. Bit 0 defines half-pixel offset in horizontal direction and bit 1 defines half-pixel offset in vertical direction. The process of half-pixel interpolation is described in [ITUH263] subclause 6.1.2 and in [ISO14496] subclause 7.6.2.1.

The functions ippiCopy8x4HP_8u_C1R and ippiCopy16x8HP_8u_C1R are used in the MPEG-4 encoders and decoders included into Intel IPP Samples. The functions ippiCopy8x8HP_8u_C1R and ippiCopy16x16HP_8u_C1R are used in the H.263 and MPEG-4 encoders and decoders included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

InterpolateAverage8x4, InterpolateAverage8x8, InterpolateAverage16x8, InterpolateAverage16x16

Interpolate source block according to half-pixel offset and average the result with destination block.

Syntax

```
IppStatus ippiInterpolateAverage8x4_8u_C1IR(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
IppStatus ippiInterpolateAverage8x8_8u_C1IR(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
IppStatus ippiInterpolateAverage16x8_8u_C1IR(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
IppStatus ippiInterpolateAverage16x16_8u_C1IR(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
```

Parameters

pSrc	Pointer to the source block.

srcStep	Distance in bytes between starts of the consecutive lines in the source block.
pSrcDst	Pointer to the destination block.
srcDstStep	Distance in bytes between starts of the consecutive lines in the destination block.
acc	Parameter that determines half-pixel offset.
rounding	Parameter that determines type of rounding for pixel interpolation; may be 0 or 1.

Description

The functions `ippiInterpolateAverage8x4_8u_C1IR`, `ippiInterpolateAverage8x8_8u_C1IR`, `ippiInterpolateAverage16x8_8u_C1IR`, and `ippiInterpolateAverage16x16_8u_C1IR` are declared in the `ippvc.h` file. These functions interpolate the source block according to half-pixel offset and average the result with the destination block. Parameters *rounding* has the meaning as RTYPE in [ITUH263] and `vop_rounding_type` in [ISO14496]. Parameter *acc* is a two-bit value. Bit 0 defines half-pixel offset in horizontal direction and bit 1 defines half-pixel offset in vertical direction. These functions are used in MPEG-2 encoder and decoder included into Intel IPP Samples together with functions `Copy8x4HP, Copy8x8HP, Copy16x8HP, Copy16x16HP` for bidirectional motion compensation. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Add8x8

Adds two blocks with saturation.

Syntax

```
IppStatus ippiAdd8x8_16s8u_C1IRS(const Ipp16s* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep);
```

Parameters

pSrc	Pointer to the source block.

srcStep	Distance in bytes between starts of the consecutive lines in the source block.
pSrcDst	Pointer to the second source/destination block.
srcDstStep	Distance in bytes between starts of the consecutive lines in the destination block.

Description

The function `ippiAdd8x8_16s8u_C1IRS` is declared in the `ippvc.h` file. This function adds `16s` data from *pSrc* block to the `8u` data from *pSrcDst* block with saturation and stores the result in *pSrcDst* block. It can be used in motion compensation process to add a reconstructed block of prediction errors to the predictor.

This function is used in the H.261, H.263, and MPEG-4 encoders and decoders included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Add8x8HP

Adds blocks interpolated with half-pixel accuracy prediction to difference with saturation.

Syntax

```
IppStatus ippiAdd8x8HP_16s8u_C1RS(const Ipp16s* pSrc1, int src1Step, Ipp8u* pSrc2,
    int src2Step, Ipp8u* pDst, int dstStep, int acc, int rounding);
```

Parameters

pSrc1	Pointer to the source block of differences.
src1Step	Distance in bytes between starts of the consecutive lines in the *pSrc1* plane.
pSrc2	Pointer to the source block of prediction.
src2Step	Distance in bytes between starts of the consecutive lines in the *pSrc2* plane.
pDst	Pointer to the second source/destination block.

dstStep	Distance in bytes between starts of the consecutive lines in the destination plane.
acc	Parameter that determines half-pixel accuracy.
rounding	Parameter that determines type of rounding for pixel interpolation; may be 0 or 1.

Description

The function `ippiAdd8x8HP_16s8u_C1RS` is declared in the `ippvc.h` file. This function adds `16s` data from *pSrc1* block to the data from *pSrc2* block interpolated with half-pixel accuracy with saturation. It can be used in motion compensation process to add a reconstructed block of prediction errors to the predictor. Parameter *rounding* has the same meaning as RTYPE in [ITUH263] and `vop_rounding_type` in [ISO14496]. Parameter *acc* is a two-bit value. Bit 0 defines half-pixel offset in horizontal direction and bit 1 defines half-pixel offset in vertical direction. The process of half-pixel interpolation is described in [ITUH263] subclause 6.1.2 and in [ISO14496] subclause 7.6.2.1.

This function is used in the H.263 and MPEG-4 encoders and decoders included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

AddC8x8

Adds a constant to 8x8 block with saturation.

Syntax

```
IppStatus ippiAddC8x8_16s8u_C1IR(Ipp16s value, Ipp8u* pSrcDst, int srcDstStep);
```

Parameters

value	Constant value to be added to the block.
pSrcDst	Pointer to the source/destination block.
srcDstStep	Distance in bytes between starts of the consecutive lines in the destination block.

Description

The function `ippiAddC8x8_16s8u_C1IR` is declared in the `ippvc.h` file. This function adds a `16s` value to an `8u` block of 8x8 size with saturation.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Average8x8, Average16x16

Average two blocks.

Syntax

```
IppStatus ippiAverage8x8_8u_C1IR(Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst, int
    srcDstStep);
IppStatus ippiAverage8x8_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp8u* pDst, int dstStep);
IppStatus ippiAverage16x16_8u_C1IR(Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,
    int srcDstStep);
IppStatus ippiAverage16x16_8u_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep);
```

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source blocks.
pDst	Pointer to the destination block.
pSrcDst	Pointer to the source and destination blocks.
srcStep, src1Step,	
src2Step	Width in bytes through the source planes.
dstStep	Width in bytes through the destination planes.
srcDstStep	Width in bytes through the source and destination planes.

Description

The functions ippiAverage8x8_8u_C1IR, ippiAverage8x8_8u_C1R, ippiAverage16x16_8u_C1IR, and ippiAverage16x16_8u_C1R are declared in the ippvc.h file. These functions average two blocks pixel-by-pixel, as specified by [ISO14496], subclause 7.6.9.4. The functions can be used in the B-frame reconstruction.

These functions are used in the MPEG-4 encoders and decoders included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

Motion Estimation

These functions calculate:

- residual block - difference between source block and predicted block (Table 16-8)
- some characteristics of the residual block (Table 16-9)
- some characteristics of the blocks. These characteristics can be used for comparison of the blocks. (Table 16-10).

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

Table 16-8 Evaluation of Residual Block

For predicted blocks	*For bi-predicted blocks*
GetDiff16x16	GetDiff16x16B
GetDiff16x8	GetDiff16x8B
GetDiff8x8	GetDiff8x8B
GetDiff8x16	GetDiff8x16B
GetDiff8x4	GetDiff8x4B
GetDiff4x4	
Sub4x4, Sub8x8, Sub16x16	
SubSAD8x8	

Table 16-9 Evaluation of Residual Block Characteristics

For predicted blocks	*For bi-predicted blocks*
SqrDiff16x16	SqrDiff16x16B
SSD8x8	
SSD4x4	
Evaluation of variances and means of blocks of difference between two blocks	
VarMeanDiff16x16	
VarMeanDiff16x8	
SAD functions	
SAD16x16	
SAD16x8	
SAD8x16	
SAD8x8	
SAD8x4	
SAD4x8	
SAD4x4	
SAD16x16Blocks8x8	
SAD16x16Blocks4x4	
SATD16x16	
SATD16x8	
SATD8x16	
SATD8x8	
SATD8x4	
SATD4x8	
SATD4x4	
SAT8x8D	
FrameFieldSAD16x16	
Sum of differences evaluation	
SumsDiff16x16Blocks4x4	
SumsDiff8x8Blocks4x4	

Table 16-10 Evaluation of Blocks Characteristics

Block variance and mean evaluation
VarMean8x8
Block variance evaluation
Variance16x16
Evaluation of block deviation
MeanAbsDev8x8
MeanAbsDev16x16
Edges detection
EdgesDetect16x16

Difference Evaluation

These functions calculate residual block - difference between the source block and the predicted block.

Difference evaluation functions are divided into two groups:

- Functions for predicted block, using one reference block for prediction
- Functions for bi-predicted block, using two reference blocks for prediction; in this case predictions are first calculated for each reference block, and then prediction is calculated as average of two predictions.

Predicted Blocks

These functions use one reference block for prediction.

GetDiff16x16

Evaluates difference between current predicted and reference blocks of 16x16 elements.

Syntax

```
IppStatus ippiGetDiff16x16_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 16x16 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRef	Pointer to the block of size 16x16 in the reference plane.
srcRefStep	Step of the reference block, specifying width of the plane in bytes.
pDstDiff	Pointer to the block of size 16x16 in the destination plane, which contains difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.
pDstPredictor	Pointer to the destination block of size 16x16 that contains a block of predictors for the current block. The predictor is calculated from the reference block taking into account *mcType*. If predictor is not used, it must be 0.
dstPredictorStep	Step of the *pDstPredictor* block in bytes.
mcType	Type of the following MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x16_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This

information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

GetDiff16x8

Evaluates difference between current predicted and reference blocks of 16x8 elements.

Syntax

```
IppStatus ippiGetDiff16x8_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 16x8 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRef	Pointer to the block of size 16x8 in the reference plane.
srcRefStep	Step of the reference block, specifying width of the plane in bytes.
pDstDiff	Pointer to the block of size 16x8 in the destination plane, which contains difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.
pDstPredictor	Pointer to the destination block of size 16x8 that contains a block of predictors for the current block. The predictor is calculated from the reference block taking into account *mcType*. If predictor is not used, it must be 0.
dstPredictorStep	Step of the *pDstPredictor* block in bytes.
mcType	Type of the following MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x8_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

GetDiff8x8

Evaluates difference between current predicted and reference blocks of 8x8 elements.

Syntax

```
IppStatus ippiGetDiff8x8_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s  mcType,
    Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 8x8 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRef	Pointer to the block of size 8x8 in the reference plane.
srcRefStep	Step of the reference block, specifying width of the plane in bytes.
pDstDiff	Pointer to the block of size 8x8 in the destination plane, which contains difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.

pDstPredictor	Pointer to the destination block of size 8x8 that contains a block of predictors for the current block. The predictor is calculated from the reference block taking into account *mcType*. If predictor is not used, it must be 0.
dstPredictorStep	Step of the *pDstPredictor* block in bytes.
mcType	Type of the following MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x8_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

GetDiff8x16

Evaluates difference between current predicted and reference blocks of 8x16 elements.

Syntax

```
IppStatus ippiGetDiff8x16_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s  mcType,
    Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 8x16 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.

pSrcRef	Pointer to the block of size 8x16 in the reference plane.
srcRefStep	Step of the reference block, specifying width of the plane in bytes.
pDstDiff	Pointer to the block of size 8x16 in the destination plane, which contains difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.
pDstPredictor	Pointer to the destination block of size 8x16 that contains a block of predictors for the current block. The predictor is calculated from the reference block taking into account *mcType*. If predictor is not used, it must be 0.
dstPredictorStep	Step of the *pDstPredictor* block in bytes.
mcType	Type of the following MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the ippvc.h header file. The function ippiGetDiff8x16_8u16s_C1 evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

GetDiff8x4

Evaluates difference between current predicted and reference blocks of 8x4 elements.

Syntax

```
IppStatus ippiGetDiff8x4_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s  mcType,
    Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 8x4 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRef	Pointer to the block of size 8x4 in the reference plane.
srcRefStep	Step of the reference block, specifying width of the plane in bytes.
pDstDiff	Pointer to the block of size 8x4 in the destination plane, which contains difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.
pDstPredictor	Pointer to the destination block of size 8x4 that contains a block of predictors for the current block. The predictor is calculated from the reference block taking into account *mcType*. If predictor is not used, it must be 0.
dstPredictorStep	Step of the *pDstPredictor* block in bytes.
mcType	Type of the following MC type IPPVC_MC_APX.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x4_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This

information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

GetDiff4x4

Computes difference between current predicted and reference 4x4 blocks.

Syntax

```
IppStatus ippiGetDiff4x4_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType,
    Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 4x4 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRef	Pointer to the block of size 4x4 in the reference plane.
srcRefStep	Step of the reference block, specifying width of the plane in bytes.
pDstDiff	Pointer to the block of size 4x4 in the destination plane, which contains difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.
pDstPredictor	Reserved parameter (must be 0).
dstPredictorStep	Reserved parameter (must be 0).
mcType	Reserved parameter (must be 0).
roundControl	Reserved parameter (must be 0).

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff4x4_8u16s_C1` computes the difference between predictor 4x4 block and source 4x4 block.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Bi-Predicted Blocks

These functions use two reference blocks for prediction. At first, predictions are calculated for each reference block, and then prediction is calculated as average of two predictions.

GetDiff16x16B

Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.

Syntax

```
IppStatus ippiGetDiff16x16B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 16x16 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRefF	Pointer to the forward block of size 16x16 in the reference plane.
srcRefStepF	Step of the forward reference block, specifying width of the plane in bytes.
mcTypeF	Type of the following forward MC type `IPPVC_MC_APX`.
pSrcRefB	Pointer to the backward block of size 16x16 in the reference plane.

srcRefStepB	Step of the backward reference block, specifying width of the block in bytes.
mcTypeB	Type of the following backward MC type IPPVC_MC_APX.
pDstDiff	Pointer to the destination block of size 16x16 that contains difference between current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x16B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

GetDiff16x8B

Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.

Syntax

```
IppStatus ippiGetDiff16x8B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 16x8 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRefF	Pointer to the forward block of size 16x8 in the reference plane.
srcRefStepF	Step of the forward reference block, specifying width of the plane in bytes.
mcTypeF	Type of the following forward MC type IPPVC_MC_APX.
pSrcRefB	Pointer to the backward block of size 16x8 in the reference plane.
srcRefStepB	Step of the backward reference block, specifying width of the block in bytes.
mcTypeB	Type of the following backward MC type IPPVC_MC_APX.
pDstDiff	Pointer to the destination block of specified size containing difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x8B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

GetDiff8x8B

Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.

Syntax

```
IppStatus ippiGetDiff8x8B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 8x8 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRefF	Pointer to the forward block of size 8x8 in the reference plane.
srcRefStepF	Step of the forward reference block, specifying width of the plane in bytes.
mcTypeF	Type of the following forward MC type IPPVC_MC_APX.
pSrcRefB	Pointer to the backward block of size 8x8 in the reference plane.
srcRefStepB	Step of the backward reference block, specifying width of the block in bytes.
mcTypeB	Type of the following backward MC type IPPVC_MC_APX.
pDstDiff	Pointer to the destination block of specified size containing difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x8B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion

compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

GetDiff8x16B

Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.

Syntax

```
IppStatus ippiGetDiff8x16B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 8x16 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRefF	Pointer to the forward block of size 8x16 in the reference plane.
srcRefStepF	Step of the forward reference block, specifying width of the plane in bytes.
mcTypeF	Type of the following forward MC type IPPVC_MC_APX.
pSrcRefB	Pointer to the backward block of size 8x16 in the reference plane.
srcRefStepB	Step of the backward reference block, specifying width of the block in bytes.
mcTypeB	Type of the following backward MC type IPPVC_MC_APX.
pDstDiff	Pointer to the destination block of specified size containing difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.

roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x16B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

GetDiff8x4B

Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.

Syntax

```
IppStatus ippiGetDiff8x4B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp32s roundControl);
```

Parameters

pSrcCur	Pointer to the block of size 8x4 in the current plane.
srcCurStep	Step of the current block, specifying width of the plane in bytes.
pSrcRefF	Pointer to the forward block of size 8x4 in the reference plane.
srcRefStepF	Step of the forward reference block, specifying width of the plane in bytes.
mcTypeF	Type of the following forward MC type `IPPVC_MC_APX`.
pSrcRefB	Pointer to the backward block of size 8x4 in the reference plane.

srcRefStepB	Step of the backward reference block, specifying width of the block in bytes.
mcTypeB	Type of the following backward MC type IPPVC_MC_APX.
pDstDiff	Pointer to the destination block of specified size containing difference between the current and reference blocks.
dstDiffStep	Step of the destination block, specifying width of the block in bytes.
roundControl	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x4B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Sub4x4, Sub8x8, Sub16x16

Subtract two blocks and store the result in the third block.

Syntax

```
IppStatus ippiSub4x4_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp16s* pDst, int  dstStep);
IppStatus ippiSub4x4_16u16s_C1R(const Ipp16u* pSrc1, int src1Step, const
    Ipp16u* pSrc2, int src2Step, Ipp16s* pDst, int  dstStep);
```

```
IppStatus ippiSub8x8_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp16s* pDst, int dstStep);
IppStatus ippiSub8x8_16u16s_C1R(const Ipp16u* pSrc1, Ipp32s  src1Step, const
    Ipp16u* pSrc2, Ipp32s  src2Step, Ipp16s* pDst, Ipp32s  dstStep);
IppStatus ippiSub16x16_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp16s* pDst, int dstStep);
```

Parameters

pSrc1	Pointer to the first source block.
src1Step	Step in bytes through the first source plane.
pSrc2	Pointer to the second source block.
src2Step	Step in bytes through the second source plane.
pDst	Pointer to the destination block.
dstStep	Step in bytes through the destination plane.

Description

The functions `ippiSub4x4_8u16s_C1R`, `ippiSub4x4_16u16s_C1R`, `ippiSub8x8_8u16s_C1R`, `ippiSub8x8_16u16s_C1R`, and `ippiSub16x16_8u16s_C1R` are declared in the `ippvc.h` file. These functions subtract two blocks and store the result in *pDst* block. The functions can be used in motion estimation process to get a difference of current and reference blocks.

These functions are used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

SubSAD8x8

Subtracts two blocks, stores the result in the third block and computes a sum of absolute differences.

Syntax

```
IppStatus ippiSubSAD8x8_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, Ipp32s* pSAD);
```

Parameters

pSrc1	Pointer to the first source block.
src1Step	Step in bytes through the first source plane.
pSrc2	Pointer to the second source block.
src2Step	Step in bytes through the second source plane.
pDst	Pointer to the destination block.
dstStep	Step in bytes through the destination plane.
pSAD	Pointer to the resulting SAD.

Description

The function `ippiSubSAD8x8_8u16s_C1R` is declared in the `ippvc.h` file. This function subtracts two blocks and stores the result in *pDst* block and stores SAD of the result in *pSAD*. This function can be used in motion estimation process to get a difference of current and reference blocks.

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Sum of Squares of Differences Evaluation

These functions evaluate sum of squares of differences between current and predicted blocks.

These functions are divided into two groups:

- Functions for predicted block, using one reference block for prediction
- Functions for bi-predicted block, using two reference blocks for prediction; in this case predictions are first calculated for each reference block, and then prediction is calculated as average of two predictions.

SqrDiff16x16

Evaluates sum of squares of differences between current and reference 16X16 blocks.

Syntax

```
IppStatus ippiSqrDiff16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const
    Ipp8u* pRef, Ipp32s refStep, Ipp32s mcType, Ipp32s* pSqrDiff);
```

Parameters

pSrc	Pointer to the current block of specified size.
srcStep	Step of the current block, specifying width of the block in bytes.
pRef	Pointer to the reference block of specified size.
refStep	Step of the reference block, specifying width of the block in bytes.
mcType	MC type IPPVC_MC_APX.
pSqrDiff	Pointer to the sum of square difference between all elements in the current and reference blocks.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSqrDiff16x16_8u32s` evaluates the sum of square difference between all the elements in the current block and corresponding elements in the reference block. The result is stored in integer* *pSqrDiff*.

Let us denote the pel which lies at the crossing of ith row and j^{th} column of the current block as *block[i,j]* and the pel which lies at the crossing of ith row and j^{th} column of the reference block as *ref_block[i,j]*. Then

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (block[i, j] - ref_block[i, j])^2.$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.
ippStsStepErr	Indicates an error when *srcStep* or *refStep* is less than or equal to zero.

SqrDiff16x16B

Evaluates sum of squares of differences between current bi-predicted 16X16 block and mean of two reference blocks.

Syntax

```
IppStatus ippiSqrDiff16x16B_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const
    Ipp8u pRefF, Ipp32s refStepF, Ipp32s mcTypeF, const Ipp8u pRefB, Ipp32s
    refStepB, Ipp32s mcTypeB, Ipp32s* pSqrDiff);
```

Parameters

pSrc	Pointer to the current block of specified size.
srcStep	Step of the current block, specifying width of the block in bytes.
pRefF	Pointer to the forward reference block of specified size.
refStepF	Step of the forward reference block, specifying width of the block in bytes.
mcTypeF	MC type IPPVC_MC_APX for the forward reference block.
pRefB	Pointer to the backward reference block of specified size.
refStepB	Step of the backward reference block, specifying width of the block in bytes.

mcTypeB	MC type IPPVC_MC_APX for the backward reference block.
pSqrDiff	Pointer to the sum of square difference between all elements in the current and reference blocks.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSqrDiff16x16B_8u32s` evaluates the sum of square difference between all the elements in the current block and the mean of corresponding elements in two reference blocks of 8x8 elements. The result is stored in integer *pSqrDiff*.

Let us denote the pel which lies at the crossing of ith row and j^{th} column of the current block as *block[i,j]*, the pel which lies at the crossing of ith row and j^{th} column of the forward reference block as *f_block[i,j]*, and the pel which lies at the crossing of ith row and j^{th} column of the forward reference block as *b_block[i,j]*. Then

$$pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} \left(block[i, j] - \frac{f_block[i, j] + b_block[i, j]}{2} \right)^2$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.
ippStsStepErr	Indicates an error when *srcStep*, *refStepF* or *refStepB* is less than or equal to zero.

SSD8x8

Evaluates sum of squares of differences between current and reference 8X8 blocks.

Syntax

```
IppStatus ippiSSD8x8_8u32s_C1R(const Ipp8u* srcStep, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pDst, Ipp32s mcType);
```

Parameters

pSrc	Pointer to the current block of specified size.
srcCurStep	Step of the current block, specifying width of the block in bytes.

pSrcRef	Pointer to the reference block of specified size.
srcRefStep	Step of the reference block, specifying width of the block in bytes.
pDst	Pointer to the sum of square difference between all elements in the current and reference blocks.
mcType	MC type IPPVC_MC_APX.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSSD8x8_8u32s_C1R` evaluates the sum of square difference between all the elements in the current block and corresponding elements in the reference block. The result is stored in integer* *pDst*.

Let us denote the pel which lies at the crossing of ith row and j^{th} column of the current block as *block[i,j]* and the pel which lies at the crossing of ith row and j^{th} column of the reference block as *ref_block[i,j]*. Then

$$^{*}pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (block[i, j] - ref_block[i, j])^2.$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SSD4x4

Evaluates sum of squares of differences between current and reference 4X4 blocks.

Syntax

```
IppStatus ippiSSD4x4_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pDst, Ipp32s mcType);
```

Parameters

pSrcCur	Pointer to the current block of specified size.
srcCurStep	Step of the current block, specifying width of the block in bytes.

pSrcRef	Pointer to the reference block of specified size.
srcRefStep	Step of the reference block, specifying width of the block in bytes.
pDst	Pointer to the sum of square difference between all elements in the current and reference blocks.
mcType	MC type IPPVC_MC_APX.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSSD4x4_8u32s_C1R` evaluates the sum of square difference between all the elements in the current block and corresponding elements in the reference block. The result is stored in integer* *pDst*.

Let us denote the pel which lies at the crossing of ith row and j^{th} column of the current block as *block[i,j]* and the pel which lies at the crossing of ith row and j^{th} column of the reference block as *ref_block[i,j]*. Then

$$^{k}pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (block[i, j] - ref_block[i, j])^2.$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

Block Variance and Mean Evaluation

These functions evaluate sum and variance of all elements in block.

VarMean8x8

Evaluates variance and mean of 8X8 block.

Syntax

```
IppStatus ippiVarMean8x8_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp32s*
    pVar, Ipp32s* pMean);
```

```
IppStatus ippiVarMean8x8_16s32s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp32s*
    pVar, Ipp32s* pMean);
```

Parameters

pSrc	Pointer to the input block 8x8.
srcStep	Input block step.
pVar	Pointer to the variance.
pMean	Pointer to the mean value.

Description

This function is declared in the `ippvc.h` header file. The function evaluates the mean and variance of an 8x8 block of unsigned char values (`ippiVarMean8x8_8u32s_C1R`) or short integer values (`ippiVarMean8x8_16s32s_C1R`).

Mean is evaluated by formula:

$$M = \sum_{i=0}^{7} \sum_{j=0}^{7} block[i, j]$$

The actual value of the *M* is sum rather than mean yet this designation is preserved for convenience.

Variance is evaluated by formula:

$$V = \frac{\sum_{i=0}^{7} \sum_{j=0}^{7} block[i, j]^2}{64} - \left(\frac{M}{64}\right)^2.$$

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Evaluation of Variances and Means of Blocks of Difference Between Two Blocks

These functions evaluate sum and variance of all elements in block.

VarMeanDiff16x16

Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.

Syntax

```
IppStatus ippiVarMeanDiff16x16_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep,
    const Ipp8u* pRef, Ipp32s srcRefStep, Ipp32s* pSrcSum, Ipp32s* pVar, Ipp32s*
    pMean, Ipp32s mcType);
```

Parameters

pSrc	Pointer to the current block 16x16.
srcStep	Current block step.
pRef	Pointer to the reference block 16x16.
srcRefStep	Reference block step.
pSrcSum	Pointer to the sum of pixel values for the current block.
pVar	Pointer to the variance.
pMean	Pointer to the mean value.
mcType	MC type IPPVC_MC_APX.

Description

This function is declared in the `ippvc.h` header file. The function `ippiVarMeanDiff16x16_8u32s_C1R` evaluates the means and variances of four 8x8 blocks of difference between two 16x16 blocks.

As the function is repeatedly called for the same current block, the parameter *pSrcSum* is used to avoid excessive computation. This parameter should point to an array of four 8x8 blocks comprising the current block. The parameters *pVar*, *pMean* should point to arrays of four elements, since the values of variance and mean are calculated separately for each 8x8 block.

Mean is evaluated by formula:

$$M = \frac{Sum - \sum_{i=0}^{7}\sum_{j=0}^{7} ref_block[i, j]}{8},$$

where *Sum* is the sum of pixel values for the current block taken from *pSrcSum* parameter.

Variance is evaluated by formula:

$$V = \frac{\left(\sum_{i=0}^{7}\sum_{j=0}^{7}(cur_block[i, j] - ref_block[i, j])^2\right) - M^2}{64}.$$

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

VarMeanDiff16x8

Evaluates variances and means of two 8x8 blocks of difference between two 16x8 blocks.

Syntax

```
IppStatus ippiVarMeanDiff16x8_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep,
    const Ipp8u* pRef, Ipp32s srcStep, Ipp32s* pSrcSum, Ipp32s* pVar, Ipp32s*
    pMean, Ipp32s mcType);
```

Parameters

pSrc	Pointer to the current block 16x8.
srcStep	Current block step.
pRef	Pointer to the reference block 16x8.
srcStep	Reference block step.
pSrcSum	Pointer to the sum of pixel values for the current block.
pVar	Pointer to the variance.
pMean	Pointer to the mean value.

mcType	MC type IPPVC_MC_APX.

Description

This function is declared in the ippvc.h header file. The function ippiVarMeanDiff16x8_8u32s_C1R evaluates the means and variances of two 8x8 blocks of difference between two 16x8 blocks.

As the function is repeatedly called for the same current block, the parameter *pSrcSum* is used to avoid excessive computation. This parameter should point to an array of four 8x8 blocks comprising the current block. The parameters *pVar*, *pMean* should point to arrays of four elements, since the values of variance and mean are calculated separately for each 8x8 block.

Mean is evaluated by formula:

$$M = \frac{Sum - \sum_{i=0}^{7}\sum_{j=0}^{7} ref_block[i, j]}{8}$$

Variance is evaluated by formula:

$$V = \frac{\left(\sum_{i=0}^{7}\sum_{j=0}^{7}(cur_block[i, j] - ref_block[i, j])^2\right) - M^2}{64}$$

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

Block Variance Evaluation

This function evaluates variance of all elements in block.

Variance16x16

Evaluates variance of current block.

Syntax

```
IppStatus ippiVariance16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, Ipp32s*
    pVar);
```

Parameters

pSrc	Pointer to the current block of specified size.
srcStep	Step of the current block, specifying width of the block in bytes.
pVar	Pointer to block variance.

Description

This function is declared in the `ippvc.h` header file. The function `ippiVariance16x16_8u32s` evaluates variance of all the elements in the current block. The result is stored in integer *Var*.

Let us denote the pel which lies at the crossing of ith row and j^{th} column of the current block as *block[i,j]*. Then

$$Var = \sum_{i=0}^{15} \sum_{j=0}^{15} block[i, j]^2 - \frac{\left(\sum_{i=0}^{15} \sum_{j=0}^{15} block[i, j] \right)^2}{256}.$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.
ippStsStepErr	Indicates an error when *srcStep* is less than or equal to zero.

Evaluation of Block Deviation

These functions evaluate mean absolute deviation of a block.

MeanAbsDev8x8

Evaluates mean absolute deviation for a 8x8 block.

Syntax

```
IppStatus ippiMeanAbsDev8x8_8u32s_C1R(const Ipp8u* pSrc, int srcStep,  Ipp32s*
    pDst);
```

Parameters

pSrc	Pointer to the block 8x8.
srcStep	Step of the block.
pDst	Pointer to the deviation.

Description

The function `ippiMeanAbsDev8x8_8u32s_C1R` is declared in the `ippvc.h` file. This function evaluates the mean absolute deviation for a 8x8 block by formula:

$$Dev = \sum_{i=0}^{7} \sum_{j=0}^{7} |block[i, j] - Mean|,$$

where *Mean* is evaluated by formula

$$Mean = \frac{\sum_{i=0}^{7} \sum_{j=0}^{7} block[i, j] + 32}{64}.$$

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

MeanAbsDev16x16

Evaluates mean absolute deviation for a 16x16 block.

Syntax

```
IppStatus ippiMeanAbsDev16x16_8u32s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst);
```

Parameters

pSrc	Pointer to the block 16x16.
srcStep	Step of the block.
pDst	Pointer to the deviation.

Description

The function ippiMeanAbsDev16x16_8u32s_C1R is declared in the ippvc.h file. This function evaluates the mean absolute deviation for a 16x16 block by formula:

$$Dev = \sum_{i=0}^{15} \sum_{j=0}^{15} |block[i, j] - Mean|,$$

where *Mean* is evaluated by formula

$$Mean = \frac{\sum_{i=0}^{15} \sum_{j=0}^{15} block[i, j] + 128}{256}.$$

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

Edges Detection

This function detects edges inside block.

EdgesDetect16x16

Detects edges inside 16X16 block.

Syntax

```
IppStatus ippiEdgesDetect16x16_8u_C1R(const Ipp8u* pSrc, Ipp32u srcStep, Ipp8u
    EdgePelDifference, Ipp8u EdgePelCount, Ipp8u* pRes);
```

Parameters

pSrc	Pointer to a 16x16 block in the current plan.
srcStep	Step of the current block, specifying width of the plane in bytes.
EdgePelDifference	The value for estimation of difference between neighboring elements. This value must be within the range of [0, 128].
EdgePelCount	The value for estimation of number of pairs of elements with 'big difference'. This value must be within the range of [0, 128].
pRes	Pointer to output value. (**pRes*) is equal to 1, if edges are detected, and is equal to 0 if edges are not detected.

Description

This function is declared in the ippvc.h header file. The function ippiEdgesDetect16x16_8u_C1R detects edges inside a 16x16 block: finds pair of neighboring (horizontal and vertical) elements with difference greater than *EdgePelDifference*.

If the number of pairs is greater than *EdgePelCount*, edges are detected and flag (**pRes*) is set to 1. Otherwise, edges are not detected ((**pRes*) is set to 0).

$res = 0 \quad count = 0 \quad row \in [0,14] \quad col \in [0,14]$

$if(Src[row][col] - Src[row][col+1]) > EdgePelDifference \rightarrow count{+}{+}$

$if(Src[row][col] - Src[row+1][col]) > EdgePelDifference \rightarrow count{+}{+}$

$if(count > EdgePelCount) \rightarrow res = 1$.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SAD Functions

These functions evaluate sum of absolute difference (SAD) between current and predicted blocks.

SAD16x16

Evaluates sum of absolute difference between current and reference 16X16 blocks.

Syntax

```
IppStatus ippiSAD16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u*
    pRef, Ipp32s refStep, Ipp32s* pSAD, Ipp32s mcType);
IppStatus ippiSAD16x16_16u32s_C1R(const Ipp16u* pSrc, Ipp32s  srcStep, const
    Ipp16u* pRef, Ipp32s  refStep, Ipp32s* pSAD, Ipp32s  mcType);
```

Parameters

pSrc	Pointer to the current block of specified size.
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
pRef	Pointer to the reference block of specified size.
refStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSAD	Pointer to the SAD value.

mcType	MC type IPPVC_MC_APX.

Description

This function is declared in the ippvc.h header file. The functions ippiSAD16x16_8u32s and ippiSAD16x16_16u32s_C1R evaluate the sum of absolute difference between all the elements in the current block and the corresponding elements in the reference block. The result is stored in *pSAD*.

This function is used in the MPEG-2 and H.264 encoders included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.
ippStsStepErr	Indicates an error when *srcCurStep* or *srcRefStep* is less than or equal to zero.

SAD16x8

Evaluates sum of absolute difference between current and reference 16X8 blocks.

Syntax

```
IppStatus ippiSAD16x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

Parameters

pSrcCur	Pointer to a 16x8 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to a 16x8 block in the reference plane.
srcRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSAD	Pointer to the SAD value.
mcType	MC type IPPVC_MC_APX.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD16x8_8u32s_C1R` evaluates the sum of absolute difference of all the elements in the current 16x8 block and the corresponding elements in the reference 16x8 block. The result is stored in *pSAD*.

This function is used in the MPEG-4 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

SAD8x16

Evaluates sum of absolute difference between current and reference 8X16 blocks.

Syntax

```
IppStatus ippiSAD8x16_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

Parameters

pSrcCur	Pointer to an 8x16 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to an 8x16 block in the reference plane.
srcRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSAD	Pointer to the SAD value.
mcType	MC type `IPPVC_MC_APX`; reserved and must be 0.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD8x16_8u32s_C1R` evaluates the sum of absolute difference of all the elements in the current 8x16 block and the corresponding elements in the reference 8x16 block. The result is stored in *pSAD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

SAD8x8

Evaluates sum of absolute difference between current and reference 8X8 blocks.

Syntax

```
IppStatus ippiSAD8x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
IppStatus ippiSAD8x8_16u32s_C1R(const Ipp16u* pSrcCur, Ipp32s  srcCurStep,
    const Ipp16u* pSrcREf, Ipp32s  srcRefStep, Ipp32s* pSAD, Ipp32s  mcType);
```

Parameters

pSrcCur	Pointer to an 8x8 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to an 8x8 block in the reference plane.
srcRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pDst	Pointer to the SAD value.
mcType	MC type `IPPVC_MC_APX`.

Description

This function is declared in the `ippvc.h` header file. The functions `ippiSAD8x8_8u32s_C1R` and `ippiSAD8x8_16u32s_C1R` evaluate the sum of absolute difference of all the elements in the current 8x8 block and the corresponding elements in the reference 8x8 block. The result is stored in *pSAD*.

This function is used in the H.261, H.263, H.264 and MPEG-4 encoders included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

SAD8x4

Evaluates sum of absolute difference between current and reference 8X4 blocks.

Syntax

```
IppStatus ippiSAD8x4_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

Parameters

pSrcCur	Pointer to an 8x4 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to an 8x4 block in the reference plane.
srcRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pDst	Pointer to the SAD value.
mcType	MC type `IPPVC_MC_APX`; reserved and must be 0.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD8x4_8u32s_C1R` evaluates the sum of absolute difference of all the elements in the current 8x4 block and the corresponding elements in the reference 8x4 block. The result is stored in *pSAD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SAD4x8

Evaluates sum of absolute difference between current and reference 4X8 blocks.

Syntax

```
IppStatus ippiSAD4x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pSAD, Ipp32s mcType);
```

Parameters

pSrcCur	Pointer to a 4x8 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to a 4x8 block in the reference plane.
srcRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pDst	Pointer to the SAD value.
mcType	MC type IPPVC_MC_APX; reserved and must be 0.

Description

This function is declared in the ippvc.h header file. The function ippiSAD4x8_8u32s_C1R evaluates the sum of absolute difference of all the elements in the current 4x8 block and the corresponding elements in the reference 4x8 block. The result is stored in *pSAD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SAD4x4

Evaluates sum of absolute difference between current and reference 4X4 blocks.

Syntax

```
IppStatus ippiSAD4x4_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u*
    pRef, Ipp32s refStep, Ipp32s* pSAD, Ipp32s mcType);
IppStatus ippiSAD4x4_16u32s_C1R(const Ipp16u* pSrc, Ipp32s  srcStep, const
    Ipp16u* pRef, Ipp32s  refStep, Ipp32s* pSAD, Ipp32s  mcType);
```

Parameters

pSrc	Pointer to a 4x4 block in the source plane.
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
pRef	Pointer to a 4x4 block in the reference plane.
refStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSAD	Pointer to the SAD value.
mcType	MC type IPPVC_MC_APX; reserved and must be 0.

Description

This function is declared in the ippvc.h header file. The functions ippiSAD4x4_8u32s and ippiSAD4x4_16u32s_C1R evaluate the sum of absolute difference of all the elements in the current 4x4 block and the corresponding elements in the reference 4x4 block. The result is stored in *pSAD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SAD16x16Blocks8x8

Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.

Syntax

```
IppStatus ippiSAD16x16Blocks8x8_8u16u(const Ipp8u* pSrc, Ipp32s srcStep, const
    Ipp8u* pRef, Ipp32s refStep, Ipp16u* pDstSAD, Ipp32s mcType);
```

Parameters

pSrc	Pointer to a 16x16 block in the source plane.
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
pRef	Pointer to a 16x16 block in the reference plane.
refStep	Distance in bytes between starts of the consecutive lines in the reference image.
pDstSAD	Pointer to an array of size 4 to store the SAD values.
mcType	Reserved and must be 0.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD16x16Blocks8x8_8u16u` evaluates the four partial sums of absolute differences of all the elements in the current 16x16 block and the corresponding elements in the reference 16x16 block. The result is stored in *pDstSAD*.

Figure 16-8 Splitting of 16x16 Block Into Four 8x8 Blocks

0	1
2	3

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SAD16x16Blocks4x4

Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.

Syntax

```
IppStatus ippiSAD16x16Blocks4x4_8u16u(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u*
    pRef, Ipp32s refStep, Ipp16u* pDstSAD, Ipp32s mcType);
```

Parameters

pSrc	Pointer to a 16x16 block in the source plane.
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
pRef	Pointer to a 16x16 block in the reference plane.
refStep	Distance in bytes between starts of the consecutive lines in the reference image.
pDstSAD	Pointer to an array of size 16 to store the SAD values.
mcType	Reserved and must be 0.

Description

This function is declared in the ippvc.h header file. The function ippiSAD16x16Blocks4x4_8u16u evaluates 16 partial sums (for each 4x4 block in the order shown in Figure 16-9) of absolute differences between all the elements in the current 16x16 block and the corresponding elements in the reference 16x16 block. The result is stored in *pDstSAD*.

Figure 16-9 Splitting of 16x16 Block Into 4x4 Blocks

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

SATD16x16

Evaluates sum of absolute transformed differences between current and reference 16X16 blocks using 4x4 transform.

Syntax

```
IppStatus ippiSATD16x16_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
IppStatus ippiSATD16x16_16u32s_C1R(const Ipp16u* pSrcCur, int srcCurStep, const
    Ipp16u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

Parameters

pSrcCur	Pointer to a 16x16 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to a 16x16 block in the reference plane.

scrRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSATD	Pointer to the SATD value.

Description

This function is declared in the `ippvc.h` header file. The functions `ippiSATD16x16_8u32s_C1R` and `ippiSATD16x16_16u32s_C1R` evaluate the sum of absolute tranformed differences of all the elements in the current 16x16 block and the corresponding elements in the reference 16x16 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 16x16 block.

If *D* is a 16x16 block of difference, the result of the function is the sum of absolute values of elements of *T**D*T* for each subblock. The result is stored in *pSATD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

SATD16x8

Evaluates sum of absolute transformed differences between current and reference 16X8 blocks using 4x4 transform.

Syntax

```
IppStatus ippiSADT16x8_8u32s(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u*
    pSrcRef, int srcRefStep, Ipp32s* pSATD);
IppStatus ippiSADT16x8_16u32s_C1R(const Ipp16u* pSrcCur, int srcCurStep, const
    Ipp16u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

Parameters

pSrcCur	Pointer to a 16x8 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to a 16x8 block in the reference plane.
scrRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSATD	Pointer to the SATD value.

Description

This function is declared in the `ippvc.h` header file. The functions `ippiSATD16x8_8u32s` and `ippiSATD16x8_16u32s_C1R` evaluate the sum of absolute tranformed differences of all the elements in the current 16x8 block and the corresponding elements in the reference 16x8 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 16x8 block.

If *D* is a 16x8 block of difference, the result of the function is the sum of absolute values of elements of *T***D***T* for each subblock. The result is stored in *pSATD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SATD8x16

Evaluates sum of absolute transformed differences between current and reference 8X16 blocks using 4x4 transform.

Syntax

```
IppStatus ippiSADT8x16_8u32s(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u*
    pSrcRef, int srcRefStep, Ipp32s* pSATD);
IppStatus ippiSADT8x16_16u32s_C1R(const Ipp16u* pSrcCur, int srcCurStep, const
    Ipp16u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

Parameters

pSrcCur	Pointer to an 8x16 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to an 8x16 block in the reference plane.
scrRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSATD	Pointer to the SATD value.

Description

This function is declared in the ippvc.h header file. The functions ippiSATD8x16_8u32s and ippiSATD8x16_16u32s_C1R evaluate the sum of absolute tranformed differences of all the elements in the current 8x16 block and the corresponding elements in the reference 8x16 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 8x16 block.

If *D* is a 8x16 block of difference, the result of the function is the sum of absolute values of elements of *T***D***T* for each subblock. The result is stored in *pSATD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

SATD8x8

Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 4x4 transform.

Syntax

```
IppStatus ippiSADT8x8_8u32s(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u*
    pSrcRef, int srcRefStep, Ipp32s* pSATD);
IppStatus ippiSADT8x8_16u32s_C1R(const Ipp16u* pSrcCur, int srcCurStep, const
    Ipp16u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

Parameters

pSrcCur	Pointer to an 8x8 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to an 8x8 block in the reference plane.

scrRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSATD	Pointer to the SATD value.

Description

This function is declared in the `ippvc.h` header file. The functions `ippiSATD8x8_8u32s` and `ippiSATD8x8_16u32s_C1R` evaluate the sum of absolute tranformed differences of all the elements in the current 8x8 block and the corresponding elements in the reference 8x8 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 8x8 block.

If D is a 8x8 block of difference, the result of the function is the sum of absolute values of elements of $T*D*T$ for each subblock. The result is stored in *pSATD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

SATD8x4

Evaluates sum of absolute transformed differences between current and reference 8X4 blocks using 4x4 transform.

Syntax

```
IppStatus ippiSADT8x4_8u32s(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u*
    pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

```
IppStatus ippiSADT8x4_16u32s_C1R(const Ipp16u* pSrcCur, int srcCurStep, const
    Ipp16u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

Parameters

pSrcCur	Pointer to an 8x4 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to an 8x4 block in the reference plane.
scrRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSATD	Pointer to the SATD value.

Description

This function is declared in the `ippvc.h` header file. The functions `ippiSATD8x4_8u32s` and `ippiSATD8x4_16u32s_C1R` evaluate the sum of absolute tranformed differences of all the elements in the current 8x4 block and the corresponding elements in the reference 8x4 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 8x4 block.

If D is a 8x4 block of difference, the result of the function is the sum of absolute values of elements of $T*D*T$ for each subblock. The result is stored in *pSATD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

SATD4x8

Evaluates sum of absolute transformed differences between current and reference 4X8 blocks using 4x4 transform.

Syntax

```
IppStatus ippiSADT4x8_8u32s(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u*
    pSrcRef, int srcRefStep, Ipp32s* pSATD);
IppStatus ippiSADT4x8_16u32s_C1R(const Ipp16u* pSrcCur, int srcCurStep, const
    Ipp16u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

Parameters

pSrcCur	Pointer to a 4x8 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to a 4x8 block in the reference plane.
scrRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSATD	Pointer to the SATD value.

Description

This function is declared in the `ippvc.h` header file. The functions `ippiSATD4x4_8u32s` and `ippiSATD4x4_16u32s_C1R` evaluate the sum of absolute tranformed differences of all the elements in the current 4x8 block and the corresponding elements in the reference 4x8 block using a 4x4 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

This matrix is applied to each 4x4 subblock of the 8x4 block.

If *D* is a 4x8 block of difference, the result of the function is the sum of absolute values of elements of *T*D*T* for each subblock. The result is stored in *pSATD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SATD4x4

Evaluates sum of absolute transformed differences between current and reference 4X4 blocks.

Syntax

```
IppStatus ippiSADT4x4_8u32s(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u*
    pSrcRef, int srcRefStep, Ipp32s* pSATD);
IppStatus ippiSADT4x4_16u32s_C1R(const Ipp16u* pSrcCur, int srcCurStep, const
    Ipp16u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

Parameters

pSrcCur	Pointer to a 4x4 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to a 4x4 block in the reference plane.
scrRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSATD	Pointer to the SATD value.

Description

This function is declared in the ippvc.h header file. The functions ippiSATD4x4_8u32s and ippiSATD4x4_16u32s_C1R evaluate the sum of absolute tranformed differences of all the elements in the current 4x4 block and the corresponding elements in the reference 4x4 block. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

If *D* is a 4x4 block of difference, the result of the function is the sum of absolute values of elements of *T**D*T*. The result is stored in *pSATD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SAT8x8D

Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 8x8 transform.

Syntax

```
IppStatus ippiSAT8x8D_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
IppStatus ippiSAT8x8D_16u32s_C1R(const Ipp16u* pSrcCur, int srcCurStep, const
    Ipp16u* pSrcRef, int srcRefStep, Ipp32s* pSATD);
```

Parameters

pSrcCur	Pointer to an 8x8 block in the source plane.
srcCurStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcRef	Pointer to an 8x8 block in the reference plane.
scrRefStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSATD	Pointer to the SATD value.

Description

This function is declared in the `ippvc.h` header file. The functions `ippiSAT8x8D_8u32s_C1R` and `ippiSAT8x8D_16u32s_C1R` evaluate the sum of absolute tranformed differences of all the elements in the current 8x8 block and the corresponding elements in the reference 8x8 block using an 8x8 transform. The transform matrix is as follows:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

If D is a 8x8 block of difference, the result of the function is the sum of absolute values of elements of $T*D*T$. The result is stored in *pSATD*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

FrameFieldSAD16x16

Calculates SAD between field lines and SAD between frame lines of block 16x16.

Syntax

```
IppStatus ippiFrameFieldSAD16x16_8u32s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32s* pFrameSAD, Ipp32s* pFieldSAD);
IppStatus ippiFrameFieldSAD16x16_16s32s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp32s* pFrameSAD, Ipp32s* pFieldSAD);
```

Parameters

pSrc	Pointer to the 16x16 block.
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
pFrameSAD	Pointer to the resulting SAD between frame lines of the block.
pFieldSAD	Pointer to the resulting SAD between field lines of the block.

Description

This function is declared in the `ippvc.h` header file. The functions `ippiFrameFieldSAD16x16_8u32s_C1R` and `ippiFrameFieldSAD16x16_16s32s_C1R` calculate SAD between field lines and between frame lines of 16x16 blocks. The functions are used for decision on DCT type (Frame or Field) in encoding process of interlaced video.

The computation formulas are as follows:

$$pFrameSAD = \sum_{i=0}^{6} \sum_{j=0}^{15} \left| pSrc_{i,\ j} - pSrc_{i+1,\ j} \right| + \sum_{i=8}^{14} \sum_{j=0}^{15} \left| pSrc_{i,\ j} - pSrc_{i+1,\ j} \right|$$

$$pFieldSAD = \sum_{i=0}^{6} \sum_{j=0}^{15} \left| pSrc_{i*2,\ j} - pSrc_{i*2+2,\ j} \right| +$$

$$+ \sum_{i=0}^{6} \sum_{j=0}^{15} \left| pSrc_{i*2+1,\ j} - pSrc_{i*2+3,\ j} \right|$$

The functions `ippiFrameFieldSAD16x16_8u32s_C1R` and `ippiFrameFieldSAD16x16_16s32s_C1R` are used in the MPEG-4 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Sum of Differences Evaluation

These functions evaluate difference between current and predicted blocks and calculates sums of elements of all residual blocks.

SumsDiff16x16Blocks4x4

Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 16X16 blocks.

Syntax

```
IppStatus ippiSumsDiff16x16Blocks4x4_8u16s_C1(const Ipp8u* pSrc, Ipp32s
    srcStep, const Ipp8u* pPred, Ipp32s predStep, Ipp16s* pSums, Ipp16s* pDiff);
```

Parameters

pSrc	Pointer to a 16x16 block in the current plane.
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
pPred	Pointer to a 16x16 block in the reference plane.
predStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSums	Pointer to an array of size 16 that contains sums of 4x4 difference blocks coefficients. The array is filled by function.

pDiff	If not NULL, pointer to an array of size 256 that contains a sequence of 4x4 residual blocks. The array is filled by function.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSumsDiff16x16Blocks4x4_8u16s_C1` evaluates difference between current and reference 4x4 blocks and calculates sums of 4x4 residual blocks elements in same order as shown in Figure 16-10.

Figure 16-10 Splitting of 16x16 Block Into Sixteen 4x4 Blocks

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

$$pDiff[(k*4+l)*16+i*4+j]=$$

$$pSrc[(k*4+i)*srcStep+(l*4+j)]-pPred[(k*4+i)*predStep+(l*4+j)]$$

$$:,\ l,\ i,\ j \in [0,\ 3$$

$$pSums[n] = \sum_{i=0}^{15} pDiff[n*16+i]$$

$$n \in [0,15].$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

SumsDiff8x8Blocks4x4

Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 8X8 blocks.

Syntax

```
IppStatus ippiSumsDiff8x8Blocks4x4_8u16s_C1(const Ipp8u* pSrc, Ipp32s srcStep,
    const Ipp8u* pPred, Ipp32s predStep, Ipp16s* pSums, Ipp16s* pDiff);
```

Parameters

pSrc	Pointer to an 8x8 block in the current plane.
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
pPred	Pointer to an 8x8 block in the reference plane.
predStep	Distance in bytes between starts of the consecutive lines in the reference image.
pSums	Pointer to an array of size 4 that contains sums of 4x4 difference blocks coefficients. The array is filled by function.
pDiff	If not NULL, pointer to an array of size 64 that contains a sequence of 4x4 residual blocks. The array is filled by function.

Description

This function is declared in the `ippvc.h` header file. The function `ippiSumsDiff8x8Blocks4x4_8u16s_C1` evaluates difference between current and reference 4x4 blocks and calculates sums of 4x4 residual blocks elements in same order as shown in Figure 16-11.

Figure 16-11 Splitting of 8x8 Block Into Sixteen 4x4 Blocks

0	1
2	3

$$pDiff[(k*2+l)*16+i*4+j]=$$

$$pSrc[(k*4+i)*srcStep+(l*4+j)]-pPred[(k*4+i)*predStep+(l*4+j)]$$

$$i,\ j \in [0,\ 3]$$

$$k,\ l \in [0,1]$$

$$pSums[n] = \sum_{i=0}^{15} pDiff[n*16+i]$$

$$n \in [0,3].$$

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Scanning Functions

Scanning functions convert a two-dimensional 8x8 array into one-dimensional data and vice versa using the predefined scan pattern.

ScanInv

Performs classical zigzag, alternate-horizontal, or alternate-vertical inverse scan on a block stored in a compact buffer.

Syntax

```
ippiScanInv_16s_C1(Ipp16s* pSrc, Ipp16s* pDst, int indxLastNonZero, int scan);
```

Parameters

pSrc	Pointer to the input block (coefficients in the scan order).
pDst	Pointer to the output block (coefficients in the normal order). The output is in a full buffer that contains 64 elements.
indxLastNonZero	Index of the last non-zero coefficient. Valid within the range of 0 to 63. This parameter provides faster operation. If the value is unknown, set to 63.
scan	Type of the scan, takes one of the following values: `IPPVC_SCAN_ZIGZAG`, indicating the classical zigzag scan, `IPPVC_SCAN_HORIZONTAL`, indicating the alternate-horizontal scan, `IPPVC_SCAN_VERTICAL`, indicating the alternate-vertical scan. See the corresponding enumerator on p.16-3.

Description

The function `ippiScanInv_16s_C1` is declared in the `ippvc.h` header file. This function converts a block of coefficients placed in classical zigzag, alternate-horizontal, or alternate-vertical scan order to a block of coefficients placed in the conventional – left-to-right, top-to-bottom raster scan – order. The zigzag and the two alternate scan patterns are shown, for example, in [ITUH263], Figure14 and [ITUH263], Annex I, Figure I.2.

This function is used in the H.263 decoder included into Intel IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	Indicates an error condition if *indxLastNonZero* is out of the range [0, 63].

ScanFwd

Performs classical zigzag, alternate-horizontal, or alternate-vertical forward scan on a block.

Syntax

```
ippiScanFwd_16s_C1(Ipp16s* pSrc, Ipp16s* pDst, int countNonZero, int scan);
```

Parameters

pSrc	Pointer to the input block (coefficients in the normal order).
pDst	Pointer to the output block (coefficients in the scan order).
countNonZero	Number of non-zero coefficients in the block. Valid within the range of 1 to 64. This parameter provides faster operation. If the value is unknown, set to 64.
scan	Type of the scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan. See the corresponding enumerator on p.16-3.

Description

The function ippiScanFwd_16s_C1 is declared in the ippvc.h header file. This function converts a block of coefficients placed in the conventional – left-to-right, top-to-bottom raster scan – order to a block of coefficients placed in classical zigzag, alternate-horizontal,

or alternate-vertical scan order. See Figure 16-16 for the scanning process in line with the classical zigzag pattern. The zigzag and the two alternate scan patterns are shown, for example, in [ITUH263], Figure14 and [ITUH263], Annex I, Figure I.2.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	Indicates an error condition if *countNonZero* is out of the range [1, 64].

Color Conversion

CbYCr422ToYCbCr420_Rotate

Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image with rotation.

Syntax

```
IppStatus ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoi, Ipp8u* pDst[3], int dstStep[3], int rotation);
IppStatus ippiCbYCr422ToYCbCr420_Rotate_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], IppiSize srcRoi, Ipp8u* pDst[3], int dstStep[3], int rotation);
```

Parameters

pSrc	Pointer to the source image for pixel-order data. Array of pointers to the separate source image planes for planar data.
srcStep	Distance in bytes between starts of the consecutive lines in the source image. Array of such values in bytes through the source image.
srcRoi	ROI of the source image, in pixels. The ROI of the destination image is calculated by user.
pDst	Array of pointers to the destination image planes.

dstStep	Distance in bytes between starts of the consecutive lines in the destination image. Array of such values through the destination image planes .
rotation	Rotation control parameter. Possible values:
	IPPVC_ROTATE_DISABLE no rotation
	IPPVC_ROTATE_90CCW rotating by 90° counterclockwise
	IPPVC_ROTATE_90CW rotating by 90° clockwise
	IPPVC_ROTATE_180 rotating by 180°
	See the corresponding enumerator in the introduction to the General Functions section.

Description

The function ippiCbYCr422ToYCbCr420_Rotate is declared in the ippvc.h file. This function converts 4:2:2 two-channel or three-plane CbYCr image *pSrc* to the 4:2:0 YCbCr three-plane image *pDst*. The two-channel source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... Three-plane source image has the following order of pointers: Cb-plane, Y-plane, Cr-plane. The destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see Table 6-2 and Table 6-3).

The function additionally rotates or flips an image in accordance with the value of the parameter *rotation*.

Example 16-2 shows how to call ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R.

Example 16-2 Call of ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R

```
#define ROTATE IPPVC_ROTATE_90CCW
  {
    Ipp8u* pSrcIm;
    Ipp8u* pDstIm[3];
    int    stepSrc;
    int    stepDst[3];
    IppiSize srcRoi = {32,32};
    IppiSize dstRoi;

    switch( ROTATE )
    {
    case IPPVC_ROTATE_DISABLE:
    case IPPVC_ROTATE_180:
      dstRoi.width  = srcRoi.width ;
      dstRoi.height = srcRoi.height;
      break;
    case IPPVC_ROTATE_90CCW:
    case IPPVC_ROTATE_90CW:
      dstRoi.width  = srcRoi.height;
```

Example 16-2 Call of `ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R` (continued)

```
        dstRoi.height = srcRoi.width ;
        break;
      }
      pSrcIm     = ippiMalloc_8u_C2(srcRoi.width, srcRoi.height, &stepSrc);
      pDstIm[0] = ippiMalloc_8u_C1(dstRoi.width, dstRoi.height,
&(stepDst[0]));
      pDstIm[1] = ippiMalloc_8u_C1(dstRoi.width/2, dstRoi.height/2,
&(stepDst[1]));
      pDstIm[2] = ippiMalloc_8u_C1(dstRoi.width/2, dstRoi.height/2,
&(stepDst[2]));
      ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R(pSrcIm, stepSrc, srcRoi,
pDstIm, stepDst, ROTATE);
      ippiFree(pDstIm[0]);
      ippiFree(pDstIm[1]);
      ippiFree(pDstIm[2]);
      ippiFree(pSrcIm);
    }
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any field of the *`srcRoi`* is less than 2.
`ippStsDoubleSize`	Indicates the condition when the values *`srcRoi`*`.width` and *`srcRoi`*`.height` are not multiples of 2. The function reduces their original values to the nearest multiples of 2 and continues operation.

NOTE. If *`srcRoi`*`.width` and *`srcRoi`*`.height` are not multiples of 2, the function reduces the values to the nearest multiples of 2. For example, if the size value is 7, the function approximates this value to 6.

ResizeCCRotate

Creates a low-resolution preview image for high-resolution video or still capture applications.

Syntax

```
IppStatus ippiResizeCCRotate_8u_C2R(const Ipp8u* pSrc, int srcStep, IppiSize
    srcRoi, Ipp16u * pDst, int dstStep, int zoomFactor, int interpolation, int
    colorConversion, int rotation);
```

Parameters

pSrc	Pointer to the source image. Input byte order is Cb Y Cr Y.
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
srcRoi	ROI of the source image, in pixels. The ROI of the destination image is calculated by user.
pDst	Pointer to the destination image. As an output parameter, indicates a pointer to the start of the buffer containing the resized and rotated image with completed color conversion.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.
zoomFactor	Parameter, indicating downscale factor; takes values 2, 4 or 8 for 2:1, 4:1, and 8:1 downscale respectively.
interpolation	Type of interpolation to perform resampling of the input image. The following types are currently supported: `IPPI_INTER_NN` nearest neighbor interpolation `IPPI_INTER_LINEAR` linear interpolation.
colorConversion	Color conversion control parameter, must be set to one of the following pre-defined values: `IPPVC_CbYCr422ToBGR565` `IPPVC_CbYCr422ToBGR555`. See the corresponding enumerator in the introduction to the General Functions section.
rotation	Rotation control parameter. Possible values: `IPPVC_ROTATE_DISABLE` no rotation `IPPVC_ROTATE_90CCW` rotating by 90° counterclockwise `IPPVC_ROTATE_90CW` rotating by 90° clockwise

IPPVC_ROTATE_180	rotating by 180°

See the corresponding enumerator in the introduction to the General Functions section.

Description

The function ippiResizeCCRotate_8u_C2R is declared in the ippvc.h file. This function synthesizes a low-resolution preview image for high-resolution video or still capture applications. The function combines scale reduction 2:1, 4:1 or 8:1, color space conversion, and rotation of an image.

Example 16-3 shows how to call ippiResizeCCRotate_8u_C2R.

Example 16-3 Call of ippiResizeCCRotate_8u_C2R

```
#define ROTATE IPPVC_ROTATE_90CW
#define ZOOMOUT  8

  {
    Ipp8u*  pSrcIm;
    Ipp16u* pDstIm;
    int    stepSrc;
    int    stepDst;
    IppiSize srcRoi = {64,64};
    IppiSize dstRoi;
    /* you should calculate dstROI for memory allocation only.*/
    /* dstROI is a function of parameters zoomFactor and rotation.*/
    switch (ROTATE){
      case IPPVC_ROTATE_DISABLE:
      case IPPVC_ROTATE_180:
        dstRoi.width  = srcRoi.width /ZOOMOUT;
        dstRoi.height = srcRoi.height/ZOOMOUT;
                break;
             case IPPVC_ROTATE_90CCW:
             case IPPVC_ROTATE_90CW:
        dstRoi.width  = srcRoi.height/ZOOMOUT;
        dstRoi.height = srcRoi.width /ZOOMOUT;
                break;
    }
    pSrcIm = ippiMalloc_8u_C2 (srcRoi.width, srcRoi.height, &stepSrc);
    pDstIm = ippiMalloc_16u_C1(dstRoi.width, dstRoi.height, &stepDst);
    ippiResizeCCRotate_8u_C2R (pSrcIm, stepSrc, srcRoi, pDstIm, stepDst,
ZOOMOUT, IPPI_INTER_NN, IPPVC_CbYCr422ToBGR555, ROTATE);
    ippiFree(pDstIm);
    ippiFree(pSrcIm);
  }
```

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *srcRoi*.width or *srcRoi*.height is less than *zoomFactor*.
ippStsInterpolationErr	Indicates an invalid value of *interpolation* control parameter.
ippStsResizeFactorErr	Indicates an invalid value of *zoomFactor* control parameter.
ippStsBadArgErr	Indicates an invalid value of *rotation* control parameter.
ippStsDoubleSize	Indicates the condition when the values *srcRoi*.width and *srcRoi*.height are not multiples of 2. The function reduces their original values to the nearest multiples of 2 and continues operation.

NOTE. If *srcRoi*.width and *srcRoi*.height are not multiples of 2, the function reduces the values to the nearest multiples of 2. For example, if the size value is 7, the function approximates this value to 6.

Video Processing

This section describes functions that are used to process decompressed video data.

Deinterlacing Functions

DeinterlaceFilterTriangle

Deinterlaces video plane.

Syntax

```
IppStatus ippiDeinterlaceFilterTriangle_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, IppiSize roiSize, Ipp32u centerWeight,
    Ipp32u layout);
```

Parameters

pSrc	Pointer to the source video plane.
srcStep	Distance in bytes between starts of the consecutive lines in the source video plane.
pDst	Pointer to the destination video plane.
dstStep	Distance in bytes between starts of the consecutive lines in the destination video plane.
roiSize	Size of ROI; height should be greater than 3.
centerWeight	Weight of filtered pixel, must lie within the range from 0 to 256.
layout	Plane layout, required when the plane is only a part of the frame. Takes the following values: `IPP_UPPER` for the first slice `IPP_CENTER` for the middle slices `IPP_LOWER` for the last slice `IPP_LOWER&&IPP_UPPER&&IPP_CENTER` for the image that is not sliced.

Description

The function `ippiDeinterlaceFilterTriangle_8u_C1R` is declared in the `ippvc.h` header file. This function deinterlaces video plane. The function performs triangle filtering of the image to remove interlacing flicker effect that arises when analogue interlaced TV data is viewed on a computer monitor.

Pixels are filtered by the following formula:

```
pixel_new = [pixel_above*(256 - centerWeight)/2 + pixel*centerWeight +
pixel_below*(256 - centerWeight)/2]/256.
```

It is recommended to use values in range from 128 to 256 as `centerWeight`. Otherwise the final result may be unpredictable.

When *`layout`* takes the value of `IPP_CENTER` or `IPP_LOWER`, the last data row from the previous slice should be accessible, that is, *`pSrc`*`[-`*`srcStep`*`]` should also be a valid pointer.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is NULL.
`ippStsSizeErr`	Indicates an error when *`roiSize`* has a field with zero or negative value.
`ippStsBadArgErr`	Indicates invalid argument.

DeinterlaceFilterCAVT

Performs deinterlacing of two-field image.

Syntax

```
IppStatus ippiDeinterlaceFilterCAVT_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, Ipp16u threshold, IppiSize roiSize);
```

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`threshold`	Threshold level value.
`roiSize`	Size of the source and destination image ROI.

Description

The function `ippiDeinterlaceFilterCAVT_8u_C1R` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function performs deinterlacing of a two-field image using content adaptive vertical temporal (CAVT) filtering.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`roiSize.width`* is less than or equal to 0, or *`roiSize.height`* is odd or less than 8.

Median

Creates an image consisting of median values of three source images.

Syntax

```
IppStatus ippiMedian_8u_P3C1R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
```

Parameters

`pSrc`	Array of pointers to the ROI in each plane of the source image.
`srcStep`	Distance in bytes between starts of consecutive lines in each plane of the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI.

Description

The function `ippiMedian_8u_P3C1R` is declared in the `ippi.h` file. It operates with ROI (see Regions of Interest in Intel IPP). This function sets each pixel in the destination image ROI as the median value of correspondent pixels in the each plane of the source image.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

DeinterlaceMedianThreshold

Performs median deinterlacing with threshold.

Syntax

```
IppStatus ippiDeinterlaceMedianThreshold_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int threshold, int fieldNum, int
    bCopyBorder);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
threshold	Threshold level value.
fieldNum	Specifies field to process. 0 means the top field, 1 means the bottom field.
bCopyBorder	Controls how to process the border. If its value is not equal to 0, the border line is copied to the destination image, otherwise the border line is processed like internal lines.

Description

The function ippiDeinterlaceMedianThreshold_8u_C1R is declared in the ippvc.h file. It operates with ROI (see Regions of Interest in Intel IPP). This function sets each pixel in the output buffer in the following way:

1. Copy an unprocessed field from the source to the destination.
2. Copy the border lines from the source to the destination in the case *bCopyBorder* parameter is set.
3. Perform processing of the field, chosen by parameter *fieldNum*. If the absolute difference between the current pixel value and the median value (M) of three input pixel values, taken from the current position, upper and lower pixels adjacent to the current one, is greater or equal to the threshold, the output pixel value is set to the median value M, otherwise it is set to the current pixel.

The size of the source image ROI is equal to the size *roiSize* of the destination image ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* has a field with zero or negative value.

DeinterlaceEdgeDetect

Generates image field using EdgeDetect filter.

Syntax

```
IppStatus ippiDeinterlaceEdgeDetect_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, int fieldNum, int bCopyBound);
```

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
fieldNum	Specifies field to process. 0 means the top field, 1 means the bottom field.

bCopyBorder	Controls how to process the border. If its value is not equal to 0, the border line is copied to the destination image, otherwise the border line is processed like internal lines.

Description

The function `ippiDeinterlaceEdgeDetect_8u_C1R` is declared in the `ippvc.h` file. It operates with ROI (see Regions of Interest in Intel IPP). The input of the function must be the field chosen to process. The filter outputs a field of the same parity as the source field.

Parameter *fieldNum* shows the field to generate; the other field must be copied by the user. Border lines are copied if the *bCopyBorder* parameter is enabled.

Processing of the input field is done by the Edge Detect algorithm.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.

DeinterlaceMotionAdaptive

Performs deinterlacing using temporal and spatial interpolations.

Syntax

```
IppStatus ippiDeinterlaceMotionAdaptive_8u_C1(const Ipp8u* pSrcPlane[4], int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize planeSize, int threshold, int
    topFirst, int topField, int copyField, int artifactProtection);
```

Parameters

pSrcPlane	Array of pointers to source images: four consecutive source plane pointers.
srcStep	Distance in bytes between starts of the consecutive lines in the source image buffer.
pDst	Pointer to the destination image.

dstStep	Distance in bytes between starts of the consecutive lines in the destination image buffer.
planeSize	Size of the image plane in pixels.
threshold	Threshold level value. The default value is 12. A tradeoff between flickering and residual combing artifacts. You may decrease the value of the threshold to reduce combing artifacts in moving objects, but note that flickering in the static region may increase. Zero value corresponds to the bob-deinterlacing.
topFirst	Defines the field order for the video sequence; can be 0 - bottom field is the first field, 1 - top field is the first field.
topField	Defines the field to process; can be 0 - bottom field, 1 - top field.
copyField	Defines copying of unprocessed field from the source frame to the destination; can be 0 - the unprocessed field is not copied to the destination frame, 1 - the unprocessed field is copied from the source frame to the destination.
artifactProtection	Sets the additional artifact protection to suppress distortion; can be 0 - disabling artifact protection, 1 - enabling artifact protection.

Description

The function `ippiDeinterlaceMotionAdaptive_8u_C1` is declared in the `ippvc.h` file. The function performs deinterlacing using two consecutive steps - *temporal* and *spatial* interpolations. The first step is used to detect significant temporal difference between pixels from different consecutive frames. The second step is used to detect minor spatial changes between vertical neighbor pixels from opposite fields.

The input sequence of source planes must be set according to the value of the parameters *topFirst* and *topField*. If the processed field is the second field in the frame, the input source planes must be from the previous, current, next, and next after next frames. Otherwise, if the first field is processed, the input source planes must be from the pre-previous, previous, current, and next source frames.

The function can be used to process the source image by parts (slices). To keep the equivalence of slice processing to the processing of the whole image, the following method should be used depending on the parameter *topField*:

— For the top field, the upper part of the image, which begins from the zero line and contains an even number of lines processed by the function in an ordinary way, *topField* is set to 1.
Other slices of the source image must contain all necessary neighbor lines to perform deinterlacing. Because of this requirement, other slices of the image that begin from an even line and contain an even number of lines should be extended with one upper neighbor odd line, and *topField* should be set to 0.

— For the bottom field, the lower part of the image, which ends with the last odd line and contains an even number of lines processed by the function in an ordinary way, *topField* is set to 0.
Other slices of the source image that end with the odd line and contain an even number of lines should be extended with one lower neighbor even line, and *topField* should be set to 0.

The unprocessed field should be copied to the destination frame separately, if necessary.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if *planeSize* has a field with zero or negative value.
ippStsMemAllocErr	Indicates a memory allocation error.

DeinterlaceBlendInitAlloc

Allocates and initializes internal structure for DeinterlaceBlend *function.*

Syntax

```
IppStatus ippiDeinterlaceBlendInitAlloc_8u_C1(IppiSize planeSize, int
    blendThresh[2], double blendConstants[2], IppiDeinterlaceBlendState_8u_C1**
    ppState);
```

Parameters

planeSize	Size of the image plane in pixels.
blendThresh	Array of two thresholds to determine *alpha*-coefficients; parameter values must be in the range [0,255]. Recommended values are 5, 9.
blendConstants	Array of two values tfor *alpha*-coefficients; parameter values must be in the range [0.0, 0.1]. Recommended values are 0.3, 0.7. The *blendConstants* parameter corresponds to *blendThresh*.
ppState	Pointer to pointer to an instance of the IppiDeinterlaceBlendState_8u_C1 structure containing internal table to store calculated *alpha*-coefficients.

Description

The function `ippiDeinterlaceBlendInitAlloc_8u_C1` is declared in the `ippvc.h` file. The function performs the following sequence of steps:

1. Allocating memory for the internal structure `IppiDeinterlaceBlendState_8u_C1`
2. Setting a pointer** *ppState* to the instance of the internal structure.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition when the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *planeSize* has a field with a value smaller than 3.
`ippStsMemAllocErr`	Indicates a memory allocation error.

DeinterlaceBlendFree

Releases memory allocated by `DeinterlaceBlendInitAlloc` *function*.

Syntax

```
IppStatus ippiDeinterlaceBlendFree_8u_C1(IppiDeinterlaceBlendState_8u_C1*
    pState);
```

Parameters

pState	Pointer to the internal structure `IppiDeinterlaceBlendState_8u_C1`.

Description

The function `ippiDeinterlaceBlendFree_8u_C1` is declared in the `ippvc.h` file. The function releases memory that was allocated by the `ippiDeinterlaceBlendInitAlloc_8u_C1` function. Call this function after finishing deinterlacing of the video sequence.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.

ippStsNullPtrErr	Indicates an error condition when the specified pointer is NULL.

DeinterlaceBlend

Calculates output pixels as alpha blends of the results of two filters applied to input pixels.

Syntax

```
IppStatus ippiDeinterlaceBlend_8u_C1(const Ipp8u* pSrcPlane[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize planeSize, int topFirst, int topField,
    int copyField, IppiDeinterlaceBlendState_8u_C1* pState);
```

Parameters

pSrcPlane	Array of pointers to source images: previous, current, and next source plane pointers.
srcStep	Distance in bytes between starts of the consecutive lines in the source image buffer.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image buffer.
planeSize	Size of the source and desctination image planes in pixels.
topFirst	Defines the field order for the video sequence; can be 0 - bottom field is the first field, 1 - top field is the first field.
topField	Defines the field to process; can be 0 - bottom field, 1 - top field.
copyField	Defines copying of unprocessed field from the source frame to the destination; can be 0 - the unprocessed field is not copied to the destination frame, 1 - the unprocessed field is copied from the source frame to the destination.
pState	Pointer to the internal structure IppiDeinterlaceBlendState_8u_C1R.

Description

The function `ippiDeinterlaceBlend_8u_C1` is declared in the `ippvc.h` file. The function calculates each output pixel as an alpha blend of the results of two filters - EdgeDetect and Median 3 - applied to the input pixels. The blending coefficient *alpha* is determined by differences of the corresponding pixels in the previous, current, and subsequent source frames.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition when the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`planeSize`* has a field with a value smaller than 3.
`ippStsMemAllocErr`	Indicates a memory allocation error.

Denoising Functions

This subsection describes denoising filtering functions. Note that the FilterDenoiseCASTInit and FilterDenoiseCAST functions use the following structure:

```
typedef struct {
    Ipp8u TemporalDifferenceThreshold;      // default  16 - range [0, 255]
    Ipp8u NumberOfMotionPixelsThreshold;    // default   0 - range [0,  16]
    Ipp8u StrongEdgeThreshold;              // default   8 - range [0, 255]
    Ipp8u BlockWidth;                       // default 4 - range [1, 16]
    Ipp8u BlockHeight;                      // default 4 - range [1, 16]
    Ipp8u EdgePixelWeight;                  // default 128 - range [0, 255]
    Ipp8u NonEdgePixelWeight;               // default  16 - range [0, 255]
    Ipp8u GaussianThresholdY;               // default 12
    Ipp8u GaussianThresholdUV;              // default 6
    Ipp8u HistoryWeight;                    // default 192 - range [0, 255]
} IppDenoiseCAST;
```

FilterDenoiseCASTInit

Initializes a denoise specification structure for content adaptive spatio-temporal noise reduction filtering.

Syntax

```
IppStatus ippiFilterDenoiseCASTInit(IppiDenoiseCAST* pParam);
```

Parameters

pParam	Pointer to an instance of the IppDenoiseCAST structure that contains input parameters for ippiFilterDenoiseCAST().

Description

The function ippiFilterDenoiseCASTInit is declared in the ippvc.h file. The function initializes the denoise specification structure for content adaptive spatio-temporal (CAST) noise reduction filtering.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition when the specified pointer is NULL.

FilterDenoiseCAST

Performs content adaptive spatio-temporal noise reduction filtering.

Syntax

```
IppStatus ippiFilterDenoiseCAST_8u_C1R(const Ipp8u* pSrcCur, const Ipp8u*
    pSrcPrev, Ipp32s srcStep, const Ipp8u* pSrcEdge, Ipp32s srcEdgeStep,
    IppiSize srcRoiSize, Ipp8u* pDst, Ipp32s dstStep, Ipp8u* pHistoryWeight,
    IppiDenoiseCAST* pParam);
IppStatus ippiFilterDenoiseCASTYUV422_8u_C2R(const Ipp8u* pSrcCur, const Ipp8u*
    pSrcPrev, Ipp32s srcStep, const Ipp8u* pSrcEdge, Ipp32s srcEdgeStep,
    IppiSize srcRoiSize, Ipp8u* pDst, Ipp32s dstStep, Ipp8u* pHistoryWeight,
    IppiDenoiseCAST* pParam);
```

Parameters

pSrcCur	Pointer to the current source image frame.
pSrcPrev	Pointer to the previous processed image frame; can be NULL (see Description below).
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
pSrcEdge	Pointer to the edge-detection filtered source image frame.
srcEdgeStep	Distance in bytes between starts of the consecutive lines in the edge-detection filtered source image frame.
pDst	Pointer to the destination image frame.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.
srcRoiSize	Size of the source and destination ROI.

pHistoryWeight	Pointer to the array of per-block history weights. Values correspond to blocks in raster order and are modified by the function. If `pHistoryWeight == NULL`, `HistoryWeight` from the structure pointed to by `pParam`, or the default value, is used for all blocks instead.
pParam	Pointer to the structure containing input parameters. If `pParam == NULL`, the default parameters are used.

Description

The functions `ippiFilterDenoiseCAST_8u_C1R` and `ippiFilterDenoiseCASTYUV422_8u_C2R` are declared in the `ippvc.h` file. The functions perform content adaptive spatio-temporal (CAST) noise reduction filtering for the packed YUY2 format (`ippiFilterDenoiseCASTYUV422_8u_C2R`) and for any planar format (`ippiFilterDenoiseCAST_8u_C1R`).

When called with `pSrcPrev == NULL` (no previous frame available), the functions perform spatial denoising only. In this case, the values pointed to by `pHistoryWeight` are not used but updated (if `pHistoryWeight != NULL`) by the function.

If the first call of the function is performed with `pSrcPrev != NULL` and `pHistoryWeight != NULL`, the user should fill the array with desired initial values.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition when `pSrcCur` or `pSrcEdge` pointers are `NULL`.
`ippStsSizeErr`	Indicates an incorrect input size of the image, that is, `srcRoiSize` has a field with zero or negative value or, in the case of `ippiFilterDenoiseCASTYUV422_8u_C2R`, `srcRoiSize.width` is odd.

FilterDenoiseSmooth

Performs spatial noise reduction filtering.

Syntax

```
IppStatus ippiFilterDenoiseSmooth_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize size, IppiRect roi, Ipp8u
    threshold, Ipp8* pWorkBuffer);
```

Parameters

pSrc	Pointer to the source image origin.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image origin.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
size	Size of the source image. The destination image is of the same size.
roi	Region of interest in the source image (of the `IppiRect` type). The destination image has the same ROI.
threshold	Parameter of the denoise algorithm for pixel comparison. If difference between the compared pixels is below the *threshold* value, the pixels are considered to be noised and will be smoothed in line with values of the neighboring pixels. Possible range is (0, 255).
pWorkBuffer	Pointer to the external work buffer of the size `2*(roi.height*roi.width)`.

Description

The function `ippiFilterDenoiseSmooth_8u_C1R` is declared in the `ippvc.h` file. The function performs Spatial Noise Reduction (SNR) filtering.

The algorithm applies a lowpass filter to an image by using a 3x3 mask and analyzes difference between pixels in accordance with *threshold*. If the difference is smaller than the threshold, the corresponding pixels are noised and will be smoothed. The 3x3 mask is also used for smoothing.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an incorrect input size of the image.

FilterDenoiseAdaptiveInitAlloc

Creates and initializes a denoise specification structure for spatio-temporal adaptive noise reduction filtering.

Syntax

```
IppStatus
    ippiFilterDenoiseAdaptiveInitAlloc_8u_C1(IppiDenoiseAdaptiveState_8u_C1**
    ppState, IppiSize roiSize, IppiSize maskSize);
```

Parameters

ppState	Pointer to pointer to the denoise specification structure to be created.
roiSize	Size of the source image ROI in pixels that will be processed.
maskSize	Parameter that defines the region used in current pixel transformation.

Description

The function ippiFilterDenoiseAdaptiveInitAlloc_8u_C1 is declared in the ippvc.h file. The function creates and initializes the denoise specification structure for spatio-temporal adaptive noise reduction filtering.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition when the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if *roiSize* or *maskSize* have a field with zero or negative value.
ippStsMemAllocErr	Indicates an error condition when no memory is allocated.

FilterDenoiseAdaptiveFree

Closes a denoise specification structure for spatio-temporal adaptive noise reduction filtering.

Syntax

```
IppStatus ippiFilterDenoiseAdaptiveFree_8u_C1(IppiDenoiseAdaptiveState_8u_C1*
    pState);
```

Parameters

pState	Pointer to the denoise specification structure to be closed.

Description

The function `ippiFilterDenoiseAdaptiveFree_8u_C1` is declared in the `ippvc.h` file. The function closes the denoise specification structure for spatio-temporal adaptive noise reduction filtering.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition when the specified pointer is `NULL`.

FilterDenoiseAdaptive

Performs spatio-temporal adaptive noise reduction filtering.

Syntax

```
IppStatus ippiFilterDenoiseAdaptive_8u_C1R(const Ipp8u* pSrcPlane[3], int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize size, IppiRect roi, IppiSize
    maskSize, Ipp8u threshold, IppvcNoiseBlurFlag blurType,
    IppiDenoiseAdaptiveState_8u_C1* pState);
```

Parameters

pSrcPlane	Array of pointers to frames that incorporates three source plane pointers: *pSrcPlane*[0] points to the previous source image origin, *pSrcPlane*[1] points to the current source image origin, *pSrcPlane*[2] points to the next source image origin.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image origin.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
size	Size of the source image. The destination image is of the same size.
roi	Region of interest in the source image (of the `IppiRect` type). The destination image has the same ROI. Size of ROI must be the same as *roiSize* in `ippFilterDenoiseAdaptiveInitAlloc_8u_C1R`.
maskSize	Parameter that defines the region used in current pixel transformation; must be the same as *maskSize* in `ippFilterDenoiseAdaptiveInitAlloc_8u_C1R`.
threshold	Parameter of the denoise algorithm for pixel comparison. If difference between the compared pixels is below the *threshold* value, the pixels are considered to be noised.
blurType	Type of blurring of the noised pixel. Possible modes are as follows: `IPPVC_NOISE_BLUR0`, `IPPVC_NOISE_BLUR1`, `IPPVC_NOISE_BLUR2`, `IPPVC_NOISE_BLUR3`. See Description for details.
pState	Pointer to the `IppiDenoiseAdaptiveState` specification structure.

Description

The function `ippiFilterDenoiseAdaptive_8u_C1R` is declared in the `ippvc.h` file. The function performs spatio-temporal adaptive noise reduction filtering. It requires information about previous, current and next source planes. The algorithm compares differences between the pixel of interest of the current frame and neighbors of the pixel of interest of the previous frame with the threshold value as well as the same differences between the pixel of interest of the current frame and the neighbors of the pixel of interest of the next frame. The neighbors of the pixel of interest are described by *maskSize*. The

pixel is detected as noised if one of the differences is smaller than *threshold*. In that case, the noised pixel will be blurred by using information about the neighbors. There are 4 methods of blurring depending on *blurType*:

```
case IPPVC_NOISE_BLUR0:
          prev = pSrcROIPrev[posCur];
          next = pSrcROINext[posCur];
          break;
case IPPVC_NOISE_BLUR1:
          prev = sumPrev;
          next = pSrcROINext[posCur];
          break;
case IPPVC_NOISE_BLUR2:
          prev = pSrcROIPrev[posCur];
          next = sumNext;
          break;
case IPPVC_NOISE_BLUR3:
          prev = sumPrev;
          next = sumNext;
          break;

pDstROI[posCur] = (prev + next + sumCur)*  scaleFactor,
```

where

`posCur` - position of the interesting pixel

`sumPrev` - sum of neighbours of the interesing pixel of the previous frame

`sumNext` - sum of neighbours of the interesing pixel of the next frame

`scaleFactor` - some normalization factor excluding overflow.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition when one of the the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an incorrect input size of the image, mask, or ROI.

FilterDenoiseMosquitoInitAlloc

Creates and initializes a denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.

Syntax

```
IppStatus
    ippiFilterDenoiseMosquitoInitAlloc_8u_C1(IppiDenoiseMosquitoState_8u_C1**
    ppState, IppiSize roiSize);
```

Parameters

ppState	Pointer to pointer to the denoise specification structure to be created.
roiSize	Size of the source image ROI in pixels that will be processed.

Description

The function `ippiFilterDenoiseMosquitoInitAlloc_8u_C1` is declared in the `ippvc.h` file. The function creates and initializes the denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition when the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roiSize* has a field with zero or negative value.
`ippStsMemAllocErr`	Indicates an error condition when no memory is allocated.

FilterDenoiseMosquitoFree

Closes a denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.

Syntax

```
IppStatus ippiFilterDenoiseMosquitoFree_8u_C1(IppiDenoiseMosquitoState_8u_C1*
    pState);
```

Parameters

pState	Pointer to the denoise specification structure to be closed.

Description

The function `ippiFilterDenoiseMosquitoFree_8u_C1` is declared in the `ippvc.h` file. The function closes the denoise specification structure for spatio-temporal motion adaptive mosquito noise reduction filtering.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition when the specified pointer is `NULL`.

FilterDenoiseMosquito

Performs spatio-temporal motion adaptive mosquito noise reduction filtering.

Syntax

```
IppStatus ippiFilterDenoiseMosquito_8u_C1R(const Ipp8u* pSrcPlane[2], int
    srcStep, Ipp8u* pDst, int dstStep, IppiSize size, IppiRect roi,
    IppiDenoiseMosquitoState_8u_C1* pState);
```

Parameters

pSrcPlane	Array of pointers to frames that incorporates two source plane pointers: *pSrcPlane*[0] points to the previous source image origin, *pSrcPlane*[1] points to the current source image origin.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image origin.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
size	Size of the source image. The destination image is of the same size.
roi	Region of interest in the source image (of the `IppiRect` type). The destination image has the same ROI. Size of ROI must be the same as *roiSize* in `ippFilterDenoiseMosquitoInitAlloc_8u_C1R`.
pState	Pointer to the denoise specification structure.

Description

The function `ippiFilterDenoiseMosquito_8u_C1R` is declared in the `ippvc.h` file. The function performs spatio-temporal motion adaptive mosquito noise reduction filtering. It requires information about previous and current source planes.

The Mosquito Noise (MN) manifests itself as fluctuations in luminance/chrominance level around edges and moving objects in a video sequence. MN is mainly the result of the ringing effect and of the inability of the translational motion compensation technique to find the best match.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an incorrect memory address.
`ippStsSizeErr`	Indicates an incorrect input size of the image.

MPEG-1 and MPEG-2

This section contains functions for encoding and decoding of video data according to MPEG-1([ISO11172]) and MPEG-2([ISO13818]) standards.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

Structures and Enumerations

The enumeration `IPPVC_MV_TYPE` indicates motion vector types (`FIELD`, `FRAME`).

```
typedef enum _IPPVC_MV_TYPE{
        IPPVC_MV_FIELD =      0x0,
        IPPVC_MV_FRAME =      0x1,
    }IPPVC_ MV_TYPE;
```

Table 16-11 MPEG-1 and MPEG-2 Video Decoding Functions

Function Short Name	Description
Variable Length Decoding Functions	
HuffmanTableInitAlloc	Allocates memory and initializes a table that contains codes for macroblock address increment, macroblock type, macroblock pattern, or motion vectors.
HuffmanRunLevelTableInitAlloc	Allocates memory and initializes a table that contains codes for DCT coefficients, that is, Run-Level codes.
DecodeHuffmanOne	Decodes a code from the bitstream using a specified table.
DecodeHuffmanPair	Decodes one code using specified table and gets two decoded values.
ReconstructDCTBlock_MPEG1	Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.
ReconstructDCTBlockIntra_MPEG1	Decodes 8X8 intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.

Table 16-11 MPEG-1 and MPEG-2 Video Decoding Functions (continued)

Function Short Name	Description
ReconstructDCTBlock_MPEG2	Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.
ReconstructDCTBlockIntra_MPEG2	Decodes 8X8 intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.
HuffmanTableFree	Frees memory allocated for VLC table.
Inverse Quantization Functions	
QuantInvIntra_MPEG2	Performs inverse quantization for intra frames according to MPEG-2 standard.
QuantInv_MPEG2	Performs inverse quantization for non-intra frames according to MPEG-2 standard.
Inverse Discrete Cosine Transformation	
DCT8x8Inv_AANTransposed_16s_C1R	Performs inverse DCT on pre-transposed block.
DCT8x8Inv_AANTransposed_16s8u_C1R	Performs inverse DCT on pre-transposed block and converts output to unsigned char format.
DCT8x8Inv_AANTransposed_16s_P2C2R	Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one array.
DCT8x8Inv_AANTransposed_16s8u_P2C2R	Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one unsigned char array.
Motion Compensation Functions	
MC16x16	Performs motion compensation for a predicted 16X16 block.
MC16x8	Performs motion compensation for a predicted 16X8 block.
MC8x16	Performs motion compensation for a predicted 8X16 block.
MC8x8	Performs motion compensation for a predicted 8X8 block.
MC8x4	Performs motion compensation for a predicted 8X4 block.
MC4x8	Performs motion compensation for a predicted 4X8 block.

Table 16-11 MPEG-1 and MPEG-2 Video Decoding Functions (continued)

Function Short Name	Description
MC16x4	Performs motion compensation for predicted UV 16X4 block.
MC16x8UV	Performs motion compensation for predicted UV 16X8 block.
MC16x16B	Performs motion compensation for a bi-predicted 16X16 block.
MC16x8B	Performs motion compensation for a bi-predicted 16X8 block.
MC8x16B	Performs motion compensation for a bi-predicted 8X16 block.
MC8x8B	Performs motion compensation for a bi-predicted 8X8 block.
MC8x4B	Performs motion compensation for a bi-predicted 8X4 block.
MC16x4B	Performs motion compensation for bi-predicted UV 16X4 block.
MC16x8UV	Performs motion compensation for bi-predicted UV 16X8 block.

Table 16-12 MPEG-1 and MPEG-2 Video Encoding Functions

Function Short Name	Description
Motion Estimation Functions	
GetDiff16x16	Evaluates difference between current predicted and reference blocks of 16x16 elements.
GetDiff16x8	Evaluates difference between current predicted and reference blocks of 16x8 elements.
GetDiff8x8	Evaluates difference between current predicted and reference blocks of 8x8 elements.
GetDiff8x16	Evaluates difference between current predicted and reference blocks of 8x16 elements.

Table 16-12 MPEG-1 and MPEG-2 Video Encoding Functions (continued)

Function Short Name	Description
GetDiff8x4	Evaluates difference between current predicted and reference blocks of 8x4 elements.
GetDiff16x16B	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.
GetDiff16x8B	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.
GetDiff8x8B	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.
GetDiff8x16B	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.
GetDiff8x4B	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.
SqrDiff16x16	Evaluates sum of squares of differences between current and reference blocks.
SqrDiff16x16B	Evaluates the sum of squares of differences between the current bi-predicted block and the mean of two reference blocks.
VarMean8x8	Evaluates variance and mean of 8X8 block.
VarMeanDiff16x16	Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.
VarMeanDiff16x8	Evaluates variances and means of four 8x8 blocks of difference between two 16x8 blocks.
SAD16x16	Evaluates sum of absolute difference between current and reference blocks.
Quantization Functions	
QuantIntra_MPEG2	Performs quantization on DCT coefficients for intra blocks in-place with specified quantization matrix according to MPEG-2 standard.

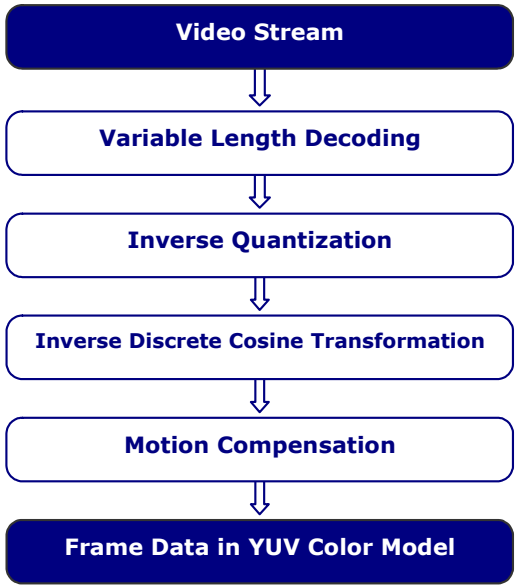
Table 16-12 MPEG-1 and MPEG-2 Video Encoding Functions (continued)

Function Short Name	Description
Quant_MPEG2	Performs quantization on DCT coefficients for non-intra blocks in-place with specified quantization matrix according to MPEG-2 standard.
Huffman Encoding Functions	
CreateRLEncodeTable	Creates Run-Level Encode Table.
PutIntraBlock	Encodes, rearranges and puts intra block into the bit stream.
PutNonIntraBlock	Encodes, rearranges and puts non-intra block into the bit stream.

Video Data Decoding

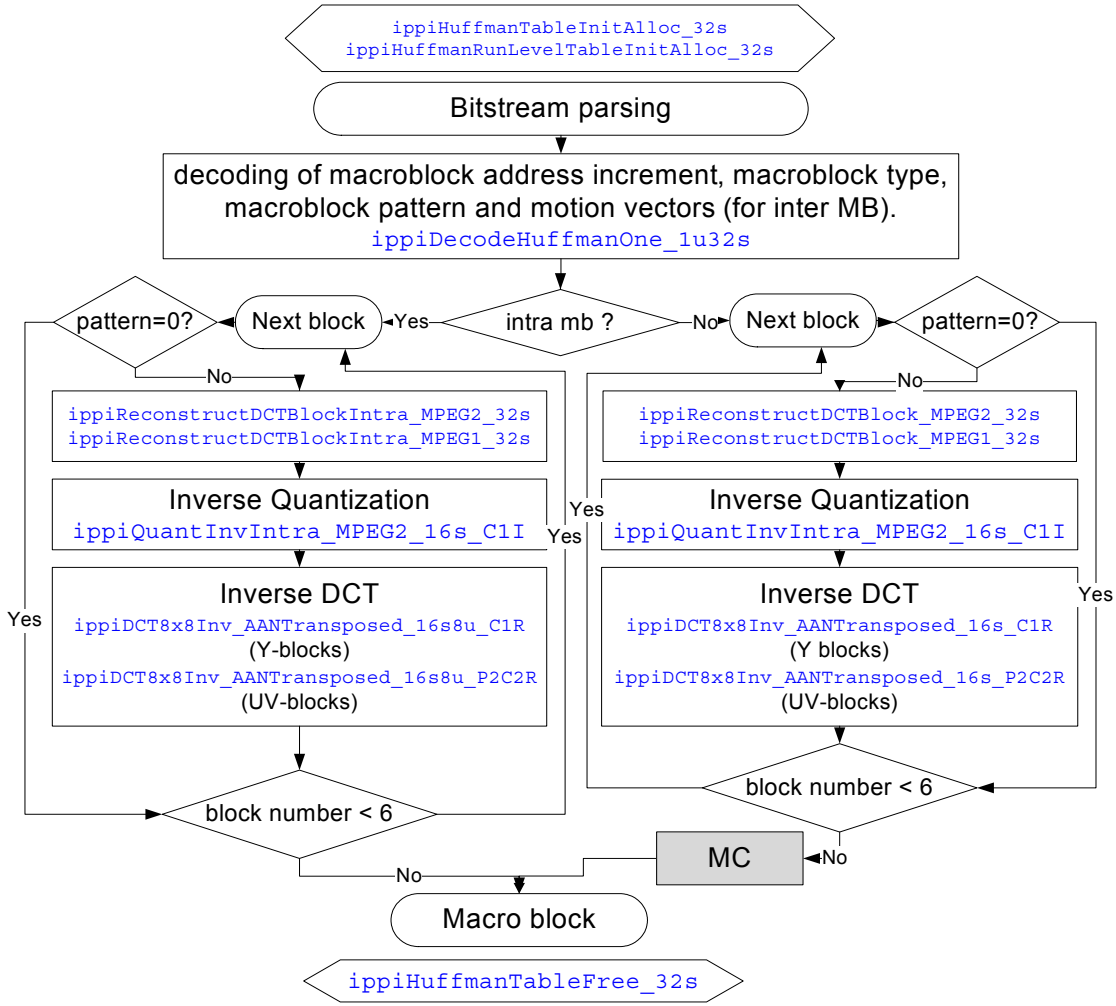
This section describes principal steps of MPEG-1 and MPEG-2 video decoding in accordance with the standard decoding pipeline shown in Figure 16-12.

Figure 16-12 Standard Decoding Pipeline



NOTE. The inverse DCT is one of the most common transformations. It may be performed using Ipp image processing functions: `ippiDCT8x8Inv_16s_C1R` for 8x8 blocks of non-intra type and `ippiDCT8x8Inv_16s8u_C1R` for 8x8 blocks of intra type.

Figure 16-13 Macroblock Decoding Scheme



NOTE. High-level functions `ippiReconstructDCTBlockIntra_MPEG2_32s` and `ippiReconstructDCTBlockIntra_MPEG1_32s` may be replaced by low-level functions according to the scheme presented in Figure 16-14 and high-level functions `ippiReconstructDCTBlock_MPEG2_32s` and `ippiReconstructDCTBlock_MPEG1_32s` may be replaced by low-level functions according to the scheme presented in Figure 16-15.

Figure 16-14 DCT Intra Block Reconstruction

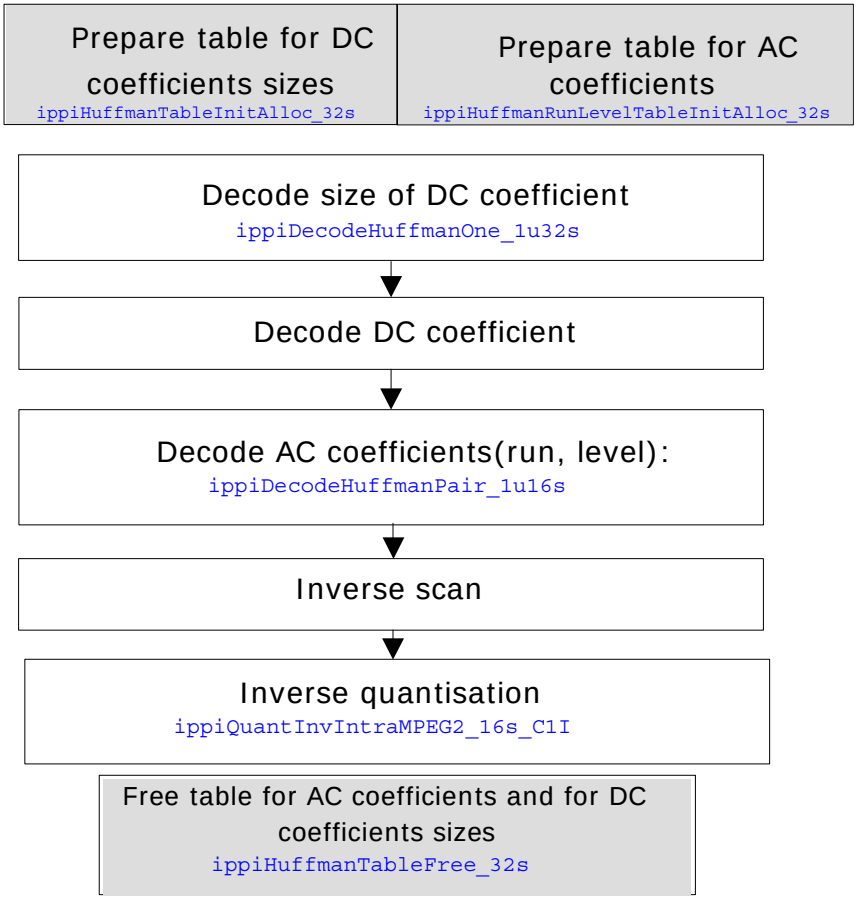
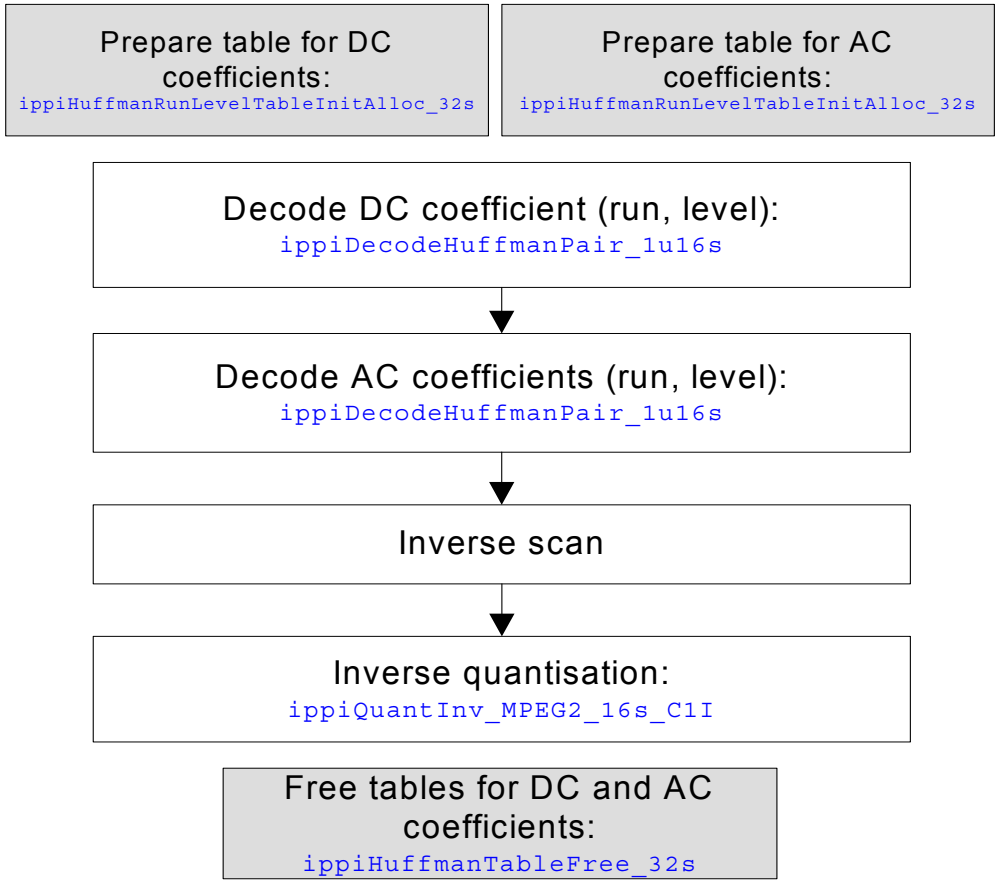


Figure 16-15 DCT Non-Intra Block Reconstruction



Variable Length Decoding

Video data in the bitstream is encoded with Variable Length Code (VLC) tables so that the shortest codes correspond to the most frequent values and the longer codes correspond to the less frequent values. MPEG standard tables contain possible codes and their values.

Decoding with Run-Level Tables

These functions decode, rearrange, and inverse quantize non-intra and intra 8x8 blocks of DCT coefficients using Run-Level Tables.

ReconstructDCTBlock_MPEG1

Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.

Syntax

```
IppStatus ippiReconstructDCTBlock_MPEG1_32s(Ipp32u** ppBitStream, int *
    pOffset, const Ipp32s* pDCSizeTable, const Ipp32s* pACTable, Ipp32s*
    pScanMatrix, int QP, Ipp16s* pQPMatrix, Ipp16s* pDstBlock, Ipp32s*
    pDstSize);
```

Parameters

ppBitStream	Double pointer to the current position in the bit stream.
pOffset	Pointer to the offset between the bit that *ppBitStream* points to and the start of the code.
pDCSizeTable	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
pACTable	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
pScanMatrix	Pointer to the matrix containing indices of elements in scanning sequence.
QP	Quantizer scale factor read from the bit stream.
pQPMatrix	Pointer to the weighting matrix imposed by standard or user-defined.
pDstBlock	Pointer to decoded elements.
pDstSize	Position of the last non-zero block coefficient in scanning sequence.

Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlock_MPEG1_32s` is applied to predicted and bi-predicted blocks and processes all the DCT coefficients in block using the tables of Run-Level codes: *pDCSizeTable* for DC coefficient and *pACTable* for AC coefficients.

The function uses the tables derived with `HuffmanRunLevelTableInitAlloc` function.

The pointer *pScanMatrix* points to the scanning matrix described in MPEG standard. Figure 16-16 illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence. Then inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix *pQPMatrix* and by the quantizer scale factor, which are read from the bit stream.

The function decodes the block of 8x8 DCT coefficients. The result is sent to *pDstBlock* and* *pDstSize* is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in Figure 16-5.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is NULL.
`ippStsH263VLCCodeErr`	Indicates an error when decoding in accordance with H263 standard.

ReconstructDCTBlockIntra_MPEG1

Decodes 8X8 intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.

Syntax

```
IppStatus ippiReconstructDCTBlockIntra_MPEG1_32s(Ipp32u** ppBitStream, int *
    pOffset, const Ipp32s* pDCSizeTable, const Ipp32s* pACTable, Ipp32s*
    pScanMatrix, int QP, Ipp16s* pQPMatrix, Ipp16s* pDCPred, Ipp16s* pDstBlock,
    Ipp32s* pDstSize);
```

Parameters

ppBitStream	Double pointer to the current position in the bit stream.

pOffset	Pointer to the offset between the bit that *ppBitStream* points to and the start of the code.
pDCSizeTable	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
pACTable	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
pScanMatrix	Pointer to the scanning matrix imposed by standard or user-defined.
QP	Quantizer scale factor read from the bit stream.
pQPMatrix	Pointer to the weighting matrix imposed by standard or user-defined.
pDCPred	Pointer to the value to be added to the DC coefficient. This value is read from the special table imposed by standard.
pDstBlock	Pointer to the decoded elements.
pDstSize	Position of the last non-zero block coefficient in scanning sequence.

Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlockIntra_MPEG1_32s` is applied to intra blocks and processes all the DCT coefficients in block using *pACTable*, which contains Run-Level codes for AC coefficients, except for DC coefficient that is to be processed using *pDCTable*.

The function uses the tables derived with `HuffmanTableInitAlloc` and `HuffmanRunLevelTableInitAlloc` functions.

The pointer *pScanMatrix* points to the scanning matrix described in MPEG standard. Figure 16-16 illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence. Then inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix *pQPMatrix* and by the quantizer scale factor, which are read from the bit stream.

The DC coefficient is increased by *pDCPred*. After performing the function, the *pDCPred* stores the sum of its initial value and DC coefficient.

The function decodes the block of 8x8 DCT coefficients. The result is sent to *pDstBlock* and* *pDstSize* is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in Figure 16-5.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.
ippStsH263VLCCodeErr	Indicates an error when decoding in accordance with H263 standard.

ReconstructDCTBlock_MPEG2

Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.

Syntax

```
IppStatus ippiReconstructDCTBlock_MPEG2_32s(Ipp32u** ppBitStream, int *
    pOffset, const IppVCHuffmanSpec_32s* pDCTable, const IppVCHuffmanSpec_32s*
    pACTable, Ipp32s* pScanMatrix, int QP, Ipp16s* pQPMatrix, Ipp16s* pDstBlock,
    Ipp32s* pDstSize);
```

Parameters

ppBitStream	Double pointer to the current position in the bit stream.
pOffset	Pointer to the offset between the bit that *ppBitStream* points to and the start of the code.
pDCTable	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
pACTable	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
pScanMatrix	Pointer to the matrix containing indices of elements in scanning sequence.
QP	Quantizer scale factor read from the bit stream.
pQPMatrix	Pointer to the weighting matrix imposed by standard or user-defined; may be NULL.
pDstBlock	Pointer to decoded elements.
pDstSize	Position of the last non-zero block coefficient in scanning sequence.

Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlock_MPEG2_32s` decodes an 8x8 block using tables of Run-Level codes, rearranges the block and performs inverse quantization. The function is applied to predicted and bi-predicted blocks and processes all the DCT coefficients in block.

The function uses the tables derived with `HuffmanRunLevelTableInitAlloc` function.

The function decodes the block of 8x8 DCT coefficients. Simultaneous processing of the coefficients enhances performance. The result is sent to *pDstBlock* and* *pDstSize* is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in Figure 16-5.

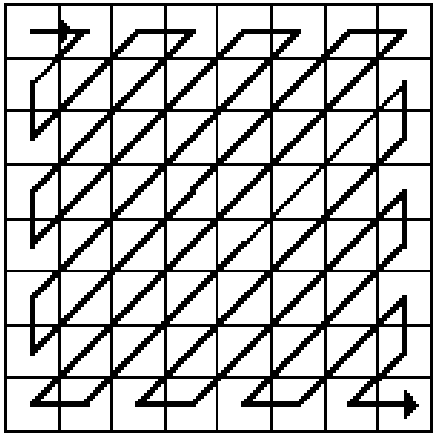
The pointer *pScanMatrix* points to one of the pre-defined scanning matrices, depending on the bitstream. Figure 16-16 illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence.

Then inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix *pQPMatrix* and by the quantizer scale factor, which are read from the bit stream. If *pQPMatrix* is NULL, the default matrix `non_intra_quantizer_matrix` is used, which is described in subclause 6.3.11 of [ISO13818-2].

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See introduction to this section.

Figure 16-16 Scanning Matrix and Sequence

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63



Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsH263VLCCodeErr`	Indicates an error when decoding in accordance with H263 standard.

ReconstructDCTBlockIntra_MPEG2

Decodes 8X8 intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.

Syntax

```
IppStatus ippiReconstructDCTBlockIntra_MPEG2_32s(Ipp32u** ppBitStream, int *
    pOffset, const IppVCHuffmanSpec_32s* pDCSizeTable, const
    IppVCHuffmanSpec_32s* pACTable, Ipp32s* pScanMatrix, int QP, Ipp16s*
    pQPMatrix, Ipp16s* pDCPred, Ipp32s shiftDCVal, Ipp16s* pDstBlock, Ipp32s*
    pDstSize);
```

Parameters

ppBitStream	Double pointer to the current position in the bit stream.
pOffset	Pointer to the offset between the bit that *ppBitStream* points to and the start of the code.
pDCSizeTable	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
pACTable	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
pScanMatrix	Pointer to the scanning matrix imposed by standard or user-defined.
QP	Quantizer scale factor read from the bit stream.
pQPMatrix	Pointer to the weighting matrix imposed by standard or user-defined.
pDCPred	Pointer to the value to be added to the DC coefficient. This value is read from the special table imposed by standard.
shiftDCVal	Integer value. The DC coefficient must be multiplied by $2^{shiftDCVal}$.
pDstBlock	Pointer to the decoded elements.
pDstSize	Position of the last non-zero block coefficient in scanning sequence.

Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlockIntra_MPEG2_32s` decodes an 8x8 intra block using tables of Run-Level codes, rearranges the block and performs inverse quantization. The function is applied to intra blocks and processes all the DCT coefficients in block, except for DC coefficient that is to be processed using the table *pDCSizeTable*.

The function uses the tables derived with `HuffmanTableInitAlloc` and `HuffmanRunLevelTableInitAlloc` functions.

The function decodes the block of 8x8 DCT coefficients. Simultaneous processing of the coefficients enhances performance. The result is sent to *pDstBlock* and* *pDstSize* is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in Figure 16-5.

The pointer *pScanMatrix* points to the scanning matrix that is read from the bit stream. Figure 16-16 illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence.

Then the inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix *pQPMatrix* and by the quantizer scale factor, which are read from the bit stream.

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsH263VLCCodeErr`	Indicates an error when decoding in accordance with H263 standard.

Inverse Quantization

Each of the decoded DCT coefficients in block should be inverse quantized through multiplication by the corresponding value from the weighting matrix and by the quantizer scale factor, which are read from the bit stream. Before this operation is performed, the DCT coefficients should be rearranged from zigzag scanning sequence to linear sequence.

For intra-frames all the DCT coefficients in block are processed, except the DC coefficient that should be processed separately in decoder, as described in MPEG standard. The functions for non-intra frames process all the DCT coefficients in block.

QuantInvIntra_MPEG2

Performs inverse quantization for intra frames according to MPEG-2 standard.

Syntax

```
IppStatus ippiQuantInvIntra_MPEG2_16s_C1I(Ipp16s* pSrcDst, int QP, Ipp16s*
    pQPMatrix);
```

Parameters

pSrcDst	Pointer to the start of the block.
QP	Quantizer scale factor read from the bit stream.
pQPMatrix	Pointer to the weighting matrix imposed by MPEG standard or user-defined.

Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantInvIntra_MPEG2_16s_C1I` multiplies the DCT coefficients from *pSrcDst* by *QP* and by corresponding values from *pQPMatrix* and sends the result back to *pSrcDst*.

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

QuantInv_MPEG2

Performs inverse quantization for non-intra frames according to MPEG-2 standard.

Syntax

```
IppStatus ippiQuantInv_MPEG2_16s_C1I(Ipp16s* pSrcDst, int QP, Ipp16s*
    pQPMatrix);
```

Parameters

pSrcDst	Pointer to the start of the block.
QP	Quantizer read from the bit stream.
pQPMatrix	Pointer to the quantizing matrix imposed by MPEG standard or user-defined.

Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantInv_MPEG2_16s_C1I` multiplies the DCT coefficients from *pSrcDst* by *QP* and by corresponding values from *pQPMatrix* and sends the result back to *pSrcDst*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Inverse Discrete Cosine Transformation

DCT8x8Inv_AANTransposed_16s_C1R

Performs inverse DCT on pre-transposed block.

Syntax

```
ippiDCT8x8Inv_AANTransposed_16s_C1R(const Ipp16s* pSrc, Ipp16s* pDst, Ipp32s
    dstStep, Ipp32s count);
```

Parameters

pSrc	Pointer to the block of DCT coefficients.
pDst	Pointer to the destination array.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.
count	Number of the last non-zero coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s_C1R` is used for non-intra macroblocks and performs inverse DCT on a transposed block. The block is transposed during the rearranging stage of VLC decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions `ReconstructDCTBlock_MPEG1` and `ReconstructDCTBlock_MPEG2`.

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

DCT8x8Inv_AANTransposed_16s8u_C1R

Performs inverse DCT on pre-transposed block and converts output to unsigned char format.

Syntax

```
ippiDCT8x8Inv_AANTransposed_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, Ipp32s
    dstStep, Ipp32s count);
```

Parameters

pSrc	Pointer to the block of DCT coefficients.
pDst	Pointer to the destination array.

dstStep	Distance in bytes between starts of the consecutive lines in the destination image.
count	Number of the last non-zero coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s8u_C1R` is used for intra macroblocks. The function performs inverse DCT on a transposed block and converts the output to unsigned char format. The block is transposed during the rearranging stage of VCL decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions `ReconstructDCTBlockIntra_MPEG1` and `ReconstructDCTBlockIntra_MPEG2`.

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

DCT8x8Inv_AANTransposed_16s_P2C2R

Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one array.

Syntax

```
ippiDCT8x8Inv_AANTransposed_16s_P2C2R(const Ipp16s* pSrcU, const Ipp16s* pSrcV,
    Ipp16s* pDstUV, Ipp32s dstStep, Ipp32s countU, Ipp32s countV);
```

Parameters

pSrcU	Pointer to the block of DCT coefficients for *U* component.
pSrcV	Pointer to the block of DCT coefficients for *V* component.
pDstUV	Pointer to the destination array.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

countU	Number of the last non-zero *U* coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.
countV	Number of the last non-zero *V* coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s_P2C2R` is used for non-intra macroblocks. The function performs inverse DCT on a transposed *U*-block and a transposed *V*-block and joins the output data into one UV Block. The blocks are transposed during the rearranging stage of VCL decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions `ReconstructDCTBlock_MPEG1` and `ReconstructDCTBlock_MPEG2`.

This function is used in the MPEG-2 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

DCT8x8Inv_AANTransposed_16s8u_P2C2R

Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one unsigned char array.

Syntax

```
ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R(const Ipp16s* pSrcU, const Ipp16s*
    pSrcV, Ipp8u* pDstUV, Ipp32s dstStep, Ipp32s countU, Ipp32s countV);
```

Parameters

pSrcU	Pointer to the block of DCT coefficients for *U* component.
pSrcV	Pointer to the block of DCT coefficients for *V* component.
pDstUV	Pointer to the destination array.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

countU	Number of the last non-zero *U* coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.
countV	Number of the last non-zero *V* coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R` is used for intra macroblocks. The function performs inverse DCT on a transposed U-block and a transposed V-block and joins the output data into one unsigned char UV Block. The blocks are transposed during the rearranging stage of VCL decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions `ReconstructDCTBlockIntra_MPEG1` and `ReconstructDCTBlockIntra_MPEG2`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Motion Compensation

This operation is applied to each block of non-intra type. The values of data obtained after inverse DCT are added to the data of predicted block and sent to the destination block. Prediction for half-pixel interpolation is performed by averaging of neighboring pixels with rounding. After processing, the data is converted from `Ipp16s` to `Ipp8u` type.

These functions are divided into two groups:

- Functions for predicted block (for block of P-type)
- Functions for bi-predicted block (for block of B-type).

See the detailed descriptions of these functions in General Functions section.

Table 16-13 lists MC functions used in Video Data decoding.

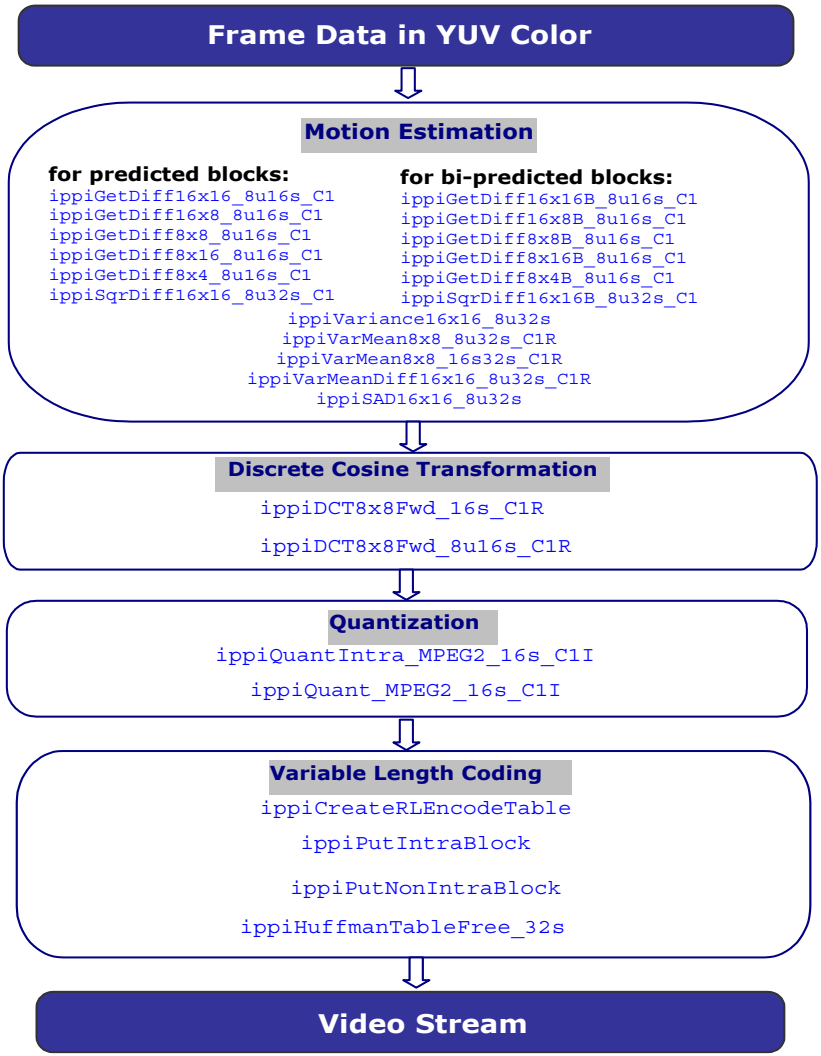
Table 16-13 Motion Compensation Functions

for predicted blocks:	for bi-predicted blocks:
MC16x4	MC16x4B
MC16x16	MC16x16B
MC16x8	MC16x8B
MC16x8UV	MC16x8UVB
MC8x16	MC8x16B
MC8x8	MC8x8B
MC8x4	MC8x4B

Video Data Encoding

This section describes the main steps of MPEG video encoding according to the standard encoding pipeline shown in Figure 16-17:

Figure 16-17 Encoding Pipeline



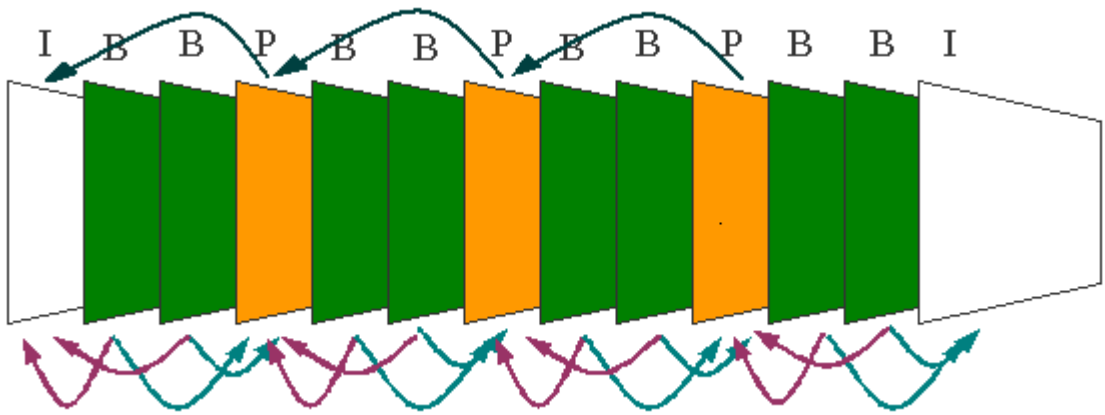
Note that for I and P frames after Quantization phase reconstruction may be implemented through inverse quantization (`QuantInv_MPEG2`), inverse discrete cosine transformation (`ippiDCT8x8Inv_16s_C1R`, see "DCT8x8Inv, DCT8x8Inv_A10" on page 10-53), and, for nonintra macroblocks, motion compensation in the same way as in the decoder (see Figure 16-13). The result is further used for motion estimation and compensation.

Motion Estimation

The process of encoding starts with motion estimation (see Figure 16-6 and Figure 16-7). Block *SrcRef* is found for each non-intra block *Dst* to be predicted, or two blocks *SrcRefF* and *SrcRefB* are found if block *Dst* is bi-predicted. The reference blocks are supposed to be similar to block *Dst*.

Figure 16-18 shows how reference frames are selected.

Figure 16-18 Intra, Predicted and Bi-Predicted Frames With Forward and Backward References



The motion estimation process is performed by the following general Motion Estimation functions:

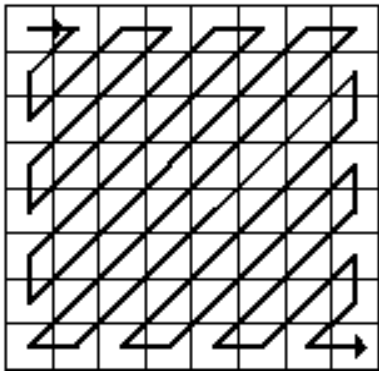
Table 16-14 MPEG Video Motion Estimation Functions

Evaluation of difference between current predicted and reference blocks	
For predicted blocks	*For bi-predicted blocks*
GetDiff16x16	GetDiff16x16B
GetDiff16x8	GetDiff16x8B
GetDiff8x8	GetDiff8x8B
GetDiff8x16	GetDiff8x16B
GetDiff8x4	GetDiff8x4B
Evaluation of sum of squares of differences between current and reference blocks	
For predicted blocks	*For bi-predicted blocks*
SqrDiff16x16	SqrDiff16x16B
Evaluation of variance and mean of 8x8 block	
VarMean8x8	
Evaluation of variances and means of difference between two blocks	
VarMeanDiff16x16	
VarMeanDiff16x8	
Evaluation of variance of current block	
Variance16x16	
Evaluation of sum of absolute difference between current and reference blocks	
SAD16x16	
SAD16x8	
SAD8x8	
SAD4x4	

Quantization

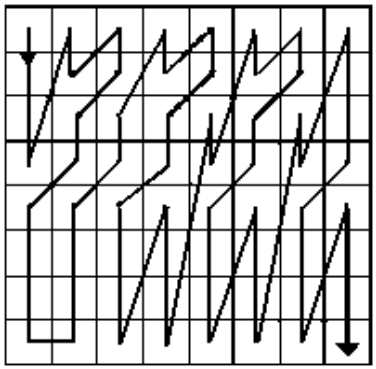
Quantization is applied to DCT coefficients to decrease their precision. You may use quatization matrices to vary precision coefficients with different positions. Scale factor is applied to all coefficients in the macroblock in the same way. Quantization values are often inverted to increase performance. Scan matrices specify the way to reorder 8x8 quantized DCT coefficients to an array of 64, so that statistically bigger values are located earlier; the further the element is from the first AC coefficient in the scanning sequence (Figure 16-19) the less important its value is for representing image and for further decoding.

Figure 16-19 Default Scanning Sequence



The alternate scanning sequence is used for interlaced video and provides better compression for field-coded blocks:

Figure 16-20 Alternate Scanning Sequence



QuantIntra_MPEG2

Performs quantization on DCT coefficients for intra blocks in-place with specified quantization matrix according to MPEG-2 standard.

Syntax

```
IppStatus ippiQuantIntra_MPEG2_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP, const
    Ipp32f* pQPMatrix, Ipp32s* pCount);
```

Parameters

pSrcDst	Pointer to the block of DCT coefficients.
QP	Quantizer.
pQPMatrix	Pointer to the matrix of inverted quantization coefficients.
pCount	Number of the non-zero AC coefficients after quantization.

Description

This function is declared in the `ippvc.h` header file. The in-place function `ippiQuantIntra_MPEG2_16s_C1I` multiplies all DCT coefficients in block except DC coefficients by the elements of the inverted quantization matrix and divide them by quantizer. The number of non-zero coefficients after quantization is stored in *pCount* for future considerations. If the pointer *pQPMatrix* is `NULL`, then the default matrix is used.

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsDivByZeroErr`	Indicates an error when quantizer *QP* is equal to zero.

Quant_MPEG2

Performs quantization on DCT coefficients for non-intra blocks in-place with specified quantization matrix according to MPEG-2 standard.

Syntax

```
IppStatus ippiQuant_MPEG2_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP, const Ipp32f*
    pQPMatrix, Ipp32s* pCount);
```

Parameters

pSrcDst	Pointer to the block of DCT coefficients.
QP	Quantizer.
pQPMatrix	Pointer to the matrix of inverted quantization coefficients.
pCount	Number of the non-zero coefficients after quantization.

Description

This function is declared in the `ippvc.h` header file. The in-place function `ippiQuant_MPEG2_16s_C1I` multiplies DCT coefficients in block by the elements of the inverted quantization matrix and divide them by quantizer. The number of non-zero coefficients after quantization is stored in *pCount* for future considerations. If the pointer *pQPMatrix* is `NULL`, then the default matrix is used.

This function is used in the MPEG-2 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsDivByZeroErr`	Indicates an error when quantizer *QP* is equal to zero.

Huffman Encoding Functions

After quantization the DCT coefficients in block should be encoded. Encoding is realized by Variable Length Coding (VLC), which uses standard code tables. These tables are entropy-constrained, that is, non-downloadable and optimized for a limited range of bit rates.

VCL Table Building

CreateRLEncodeTable

Creates Run-Level Encode Table.

Syntax

```
IppStatus ippiCreateRLEncodeTable (const Ipp32s* pSrcTable,
    IppVCHuffmanSpec_32s** ppDstSpec);
```

Parameters

pSrcTable	Pointer to the source table.
ppDstSpec	Double pointer to the destination encode table.

Description

This function is declared in the `ippvc.h` header file. The function `ippiCreateRLEncodeTable` creates the Run-Level Encode Table. The result is stored in block *ppDstSpec*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Block Encoding Functions

PutIntraBlock

Encodes, rearranges and puts intra block into bit stream.

Syntax

```
IppStatus ippiPutIntraBlock(Ipp32u** ppBitStream, int* pOffset, Ipp16s*
    pSrcBlock, Ipp32s* pDCPred, IppVCHuffmanSpec_32u* pDCTable,
    IppVCHuffmanSpec_32s* pACTable, Ipp32s* pScanMatrix, Ipp32s EOBLen, Ipp32s
    EOBCode, Ipp32s count);
```

Parameters

ppBitStream	Double pointer to the current position in the bit stream.
pOffset	Pointer to the offset between the bit that *ppBitStream* points to and the start of the code.
pSrcBlock	Pointer to the block.
pDCPred	Pointer to the value to be added to the DC coefficient; this value is read from the special table imposed by standard.
pDCTable	Pointer to the table containing codes for DC coefficient, that is, the first coefficient among the DCT coefficients.
pACTable	Pointer to the table containing Run-Level codes for AC coefficients, that is, all DCT coefficients except the first coefficient.
pScanMatrix	Pointer to the scanning matrix.
EOBLen	Length of the block end code.

EOBCode	Value of the block end code.
count	Number of the non-zero AC coefficients.

Description

This function is declared in the `ippvc.h` header file. The function `ippiPutIntraBlock` is applied to intra blocks and encodes a block of 8x8 DCT coefficients.

The `ippiPutIntraBlock` function encodes DC coefficient using the value* *pDCPred* for value computation and *pDCTable* for its encoding. Then it encodes AC coefficients using the *pACTable*.

The pointer *pScanMatrix* points to the scanning matrix that is described in MPEG standard. See Figure 16-19 for the simple scanning matrix and sequence. After encoding, the data should be rearranged from linear sequence to scanning sequence.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

PutNonIntraBlock

Encodes, rearranges and puts non-intra block into bit stream.

Syntax

```
IppStatus ippiPutNonIntraBlock(Ipp32u** ppBitStream, int* pOffset, Ipp16s*
    pSrcBlock, IppVCHuffmanSpec_32s* pACTable, Ipp32s* pScanMatrix, Ipp32s
    EOBLen, Ipp32s EOBCode, Ipp32s count);
```

Parameters

ppBitStream	Double pointer to the current position in the bit stream.
pOffset	Pointer to the offset between the bit that *ppBitStream* points to and the start of the code.
pSrcBlock	Pointer to the block.
pACTable	Pointer to the table containing Run-Level codes for AC coefficients, that is, all DCT coefficients except the first coefficient.
pScanMatrix	Pointer to the scanning matrix.

EOBLen	Length of the block end code.
EOBCode	Value of the block end code.
count	Number of the non-zero coefficients.

Description

This function is declared in the `ippvc.h` header file. The function `ippiPutNonIntraBlock` is applied to predicted and bi-predicted non-intra blocks and encodes a block of 8x8 DCT coefficients. The function encodes both DC and AC coefficients using the *pACTable*.

The pointer *pScanMatrix* points to the scanning matrix that is described in MPEG standard. See Figure 16-19 for the simple scanning matrix and sequence. After encoding, the data should be rearranged from linear sequence to scanning sequence.

Return Values

`IppStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

DV

DV is an international standard for a consumer digital video format.

DV uses a 1/4 inch (6.35mm) metal evaporate tape to record very high quality digital video. The video is sampled at the same rate as D-1, D-5, or Digital Betacam video – 720 pixels per scanline – although the color information is sampled at half the D-1 rate: 4:1:1 in 525-line (NTSC), and 4:2:0 in 625-line (PAL) formats.

DV uses intraframe compression: each compressed frame depends entirely on itself, and not on any data from preceding or following frames. However, it also uses adaptive *interfield* compression; if the compressor detects little difference between the two interlaced fields of a frame, it will compress them together, freeing up some of the "bit budget" to allow for higher overall quality. In theory, this means that static areas of images will be more accurately represented than areas with a lot of motion; in practice, this can sometimes be observed as a slight degree of "blockiness" in the immediate vicinity of moving objects.

The use of some functions described in this section is demonstrated of Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

The following section discusses the basic DV notions.

DCT Block

The Y, U, V pixels in one frame shall be divided into *DCT blocks*. All DCT blocks for 625-50 system and DCT blocks except for rightmost DCT blocks in *U* and *V* for 525-60 system are structured with a rectangular area of eight vertical lines and eight horizontal pixels in a frame.

For 525-60 system, the rightmost DCT blocks in *U* and *V* are structured with 16 vertical lines and four horizontal pixels. The rightmost DCT block is reconstructed to eight vertical lines and eight horizontal pixels by moving the lower part of eight vertical lines and four horizontal pixels to the higher part of eight vertical lines and four horizontal pixels.

Figure 16-21 DCT Block Structure

		x								
		0	1	2	3	4	5	6	7	
y	0									Field 2
	1									Field 1
	2									Field 2
	3									Field 1
	4									Field 2
	5									Field 1
	6									Field 2
	7									Field 1

Six DCT blocks Y0, Y1, Y2, Y3, U, V form a *macroblock*.

Figure 16-22 Macroblock Structure for 525-60 System (Except Rightmost Macroblock)

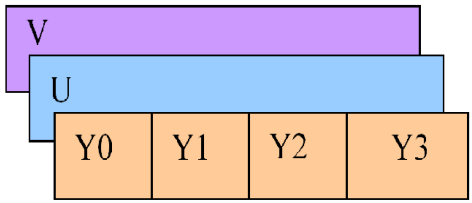
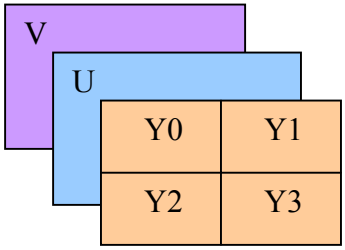


Figure 16-23 Macroblock Structure for 625-50 System and Rightmost Macroblock of 525-60 System



27 DCT macroblocks form a *superblock*. (Each cell stands for one macroblock).

Figure 16-24 DCT Superblock Structure for 525-60 System

0	11	12	23	24										8	9	20	21	0	11	12	23	24
1	10	13	22	25										7	10	19	22	1	10	13	22	
2	9	14	21	26										6	11	18	23	2	9	14	21	25
3	8	15	20										0	5	12	17	24	3	8	15	20	
4	7	16	19										1	4	13	16	25	4	7	16	19	26
5	6	17	18										2	3	14	15	26	5	6	17	18	

System 525-60 has three types of superblocks.

Figure 16-25 DCT Superblock Structure for 625-50 System

0	5	6	11	12	17	18	23	24
1	4	7	10	13	16	19	22	25
2	3	8	9	14	15	20	21	26

All superblocks of 625-50 system are of the same type.

A video *segment* consists of five macroblocks, which are gathered from various areas as below:

Figure 16-26 Macroblock Absolute Coordinates

Number of macroblock in the superblock	k	k	k	k	k
Superblock coordinates	(2,(j+2)% n)	(1,(j+6)% n)	(3,(j+8)% n)	(0,(j)% n)	(4,(j+4)% n)

where *k* is within the interval [0,26], *n* = 10 for 525-60 system, *n* = 12 for 625-50 system, and *j* is the vertical order in the superblock.

Data in a video segment is compressed and transformed to the data of 385 bytes (*compressed segment*). A compressed segment consists of five compressed macroblocks. Each compressed macroblock consists of 77 bytes.

Figure 16-27 Arrangement of Compressed Macroblock

Byte sequence	H	cbY0	cbY1	cbY2	cbY3	cbU	cbV
Length	4	14	14	14	14	10	10

Figure 16-28 Arrangement of Compressed Macroblock Header (H)

Section name	Not used		qn0
Number of bits		4	
Number of bytes	3 bytes	1 byte	

Figure 16-29 Arrangement of Y compressed blocks cbY0, cbY1, cbY2, cbY3

Section name	DC		m0	cl	AC	
Number of bits		1	1	2	4	
Number of bytes	1 byte	1 byte				12 bytes

Figure 16-30 Arrangement Cr compressed blocks cbU, cbV

Section name	DC		m0	cl	AC	
Number of bits		1	1	2	4	
Number of bytes	1 byte	1 byte				8 bytes

Type of block is equal to 1 when the difference between two fields is small (m0=0). Type of block is equal to 2 when the difference between two fields is big (m0=1).

Each DCT block is classified into four classes. For selecting quantization step, class number is used.

Table 16-15 DV Functions

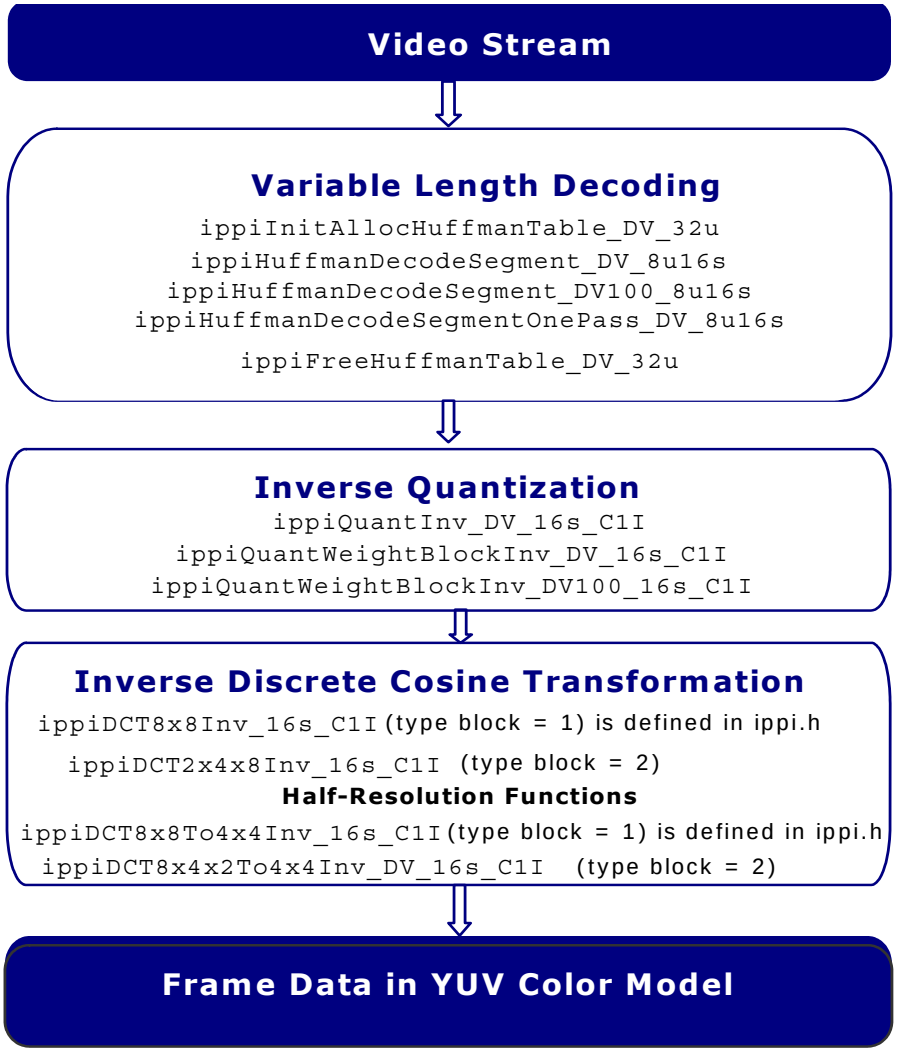
Function Short Name	Description
	Decoding
Variable Length Decoding	
InitAllocHuffmanTable_DV	Allocates memory and initializes the table that contains codes for DCT coefficients (Run-Level codes).
HuffmanDecodeSegment_DV	Decodes and rearranges a segment block, multiplies the first block element by 2.
HuffmanDecodeSegment_DV100	Decodes and rearranges a DV100 segment block, multiplies the first block element by 128.
HuffmanDecodeSegmentOnePass_DV	Performs the first pass of video segment decoding, rearranges the segment block, multiplies the first block element by 128.
FreeHuffmanTable_DV	Frees the memory allocated for VLC table.
Inverse Quantization	
QuantInv_DV	Performs inverse quantization on a block.
QuantWeightBlockInv_DV	Performs inverse quantization and inverse weighting on a block.
QuantWeightBlockInv_DV100	Performs inverse quantization and inverse weighting on a block according to DV100 standard.
Inverse Discrete Cosine Transformation	
DCT2x4x8Inv	Performs the inverse DCT for block of type 2 ($m0$=1).
DCT8x4x2To4x4Inv_DV	Performs the inverse DCT for block 2x4x8 and then creates block 4x4.
	Encoding
Discrete Cosine Transformation	
DCT2x4x8Frw	Performs DCT for a block of type 2.
CountZeros8x8	Evaluates number of zeros in a block.

Table 16-15 DV Functions (continued)

Function Short Name	Description
	Color Conversion
YCrCb411ToYCbCr422_5MBDV, YCrCb411ToYCbCr422_ZoomOut2_5MBDV, YCrCb411ToYCbCr422_ZoomOut4_5MBDV, YCrCb411ToYCbCr422_ZoomOut8_5MBDV,	Convert five macroblocks from YUV411 format into YUV2 format.
YCrCb411ToYCbCr422_16x4x5MB_DV YCrCb411ToYCbCr422_8x8MB_DV	Convert five reduced YCrCb411 macroblocks into YCrCb422 reduced macroblocks.
YCrCb411ToYCbCr422_EdgeDV, YCrCb411ToYCbCr422_ZoomOut2_EdgeDV, YCrCb411ToYCbCr422_ZoomOut4_EdgeDV, YCrCb411ToYCbCr422_ZoomOut8_EdgeDV	Convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image.
YCrCb420ToYCbCr422_5MBDV, YCrCb420ToYCbCr422_ZoomOut2_5MBDV, YCrCb420ToYCbCr422_ZoomOut4_5MBDV, YCrCb420ToYCbCr422_ZoomOut8_5MBDV	Convert five YCrCb420 macroblocks into YCrCb422 macroblocks.
YCrCb420ToYCbCr422_8x8x5MB_DV	Converts five reduced YCrCb420 macroblocks into YCrCb422 reduced macroblocks.
YCrCb422ToYCbCr422_5MBDV, YCrCb422ToYCbCr422_ZoomOut2_5MBDV, YCrCb422ToYCbCr422_ZoomOut4_5MBDV, YCrCb422ToYCbCr422_ZoomOut8_5MBDV	Convert five YCrCb422 macroblocks into YCbCr422 macroblocks.
YCrCb422ToYCbCr422_8x4x5MB_DV	Converts five reduced YCrCb422 macroblocks into YCrCb422 reduced macroblocks.
YCrCb422ToYCbCr422_10HalvesMB16x8_DV100	Converts ten YCrCb422 half-macroblocks into YCrCb422 half-macroblocks.

DV Decoding Functions

Figure 16-31 DV Decoding Pipeline



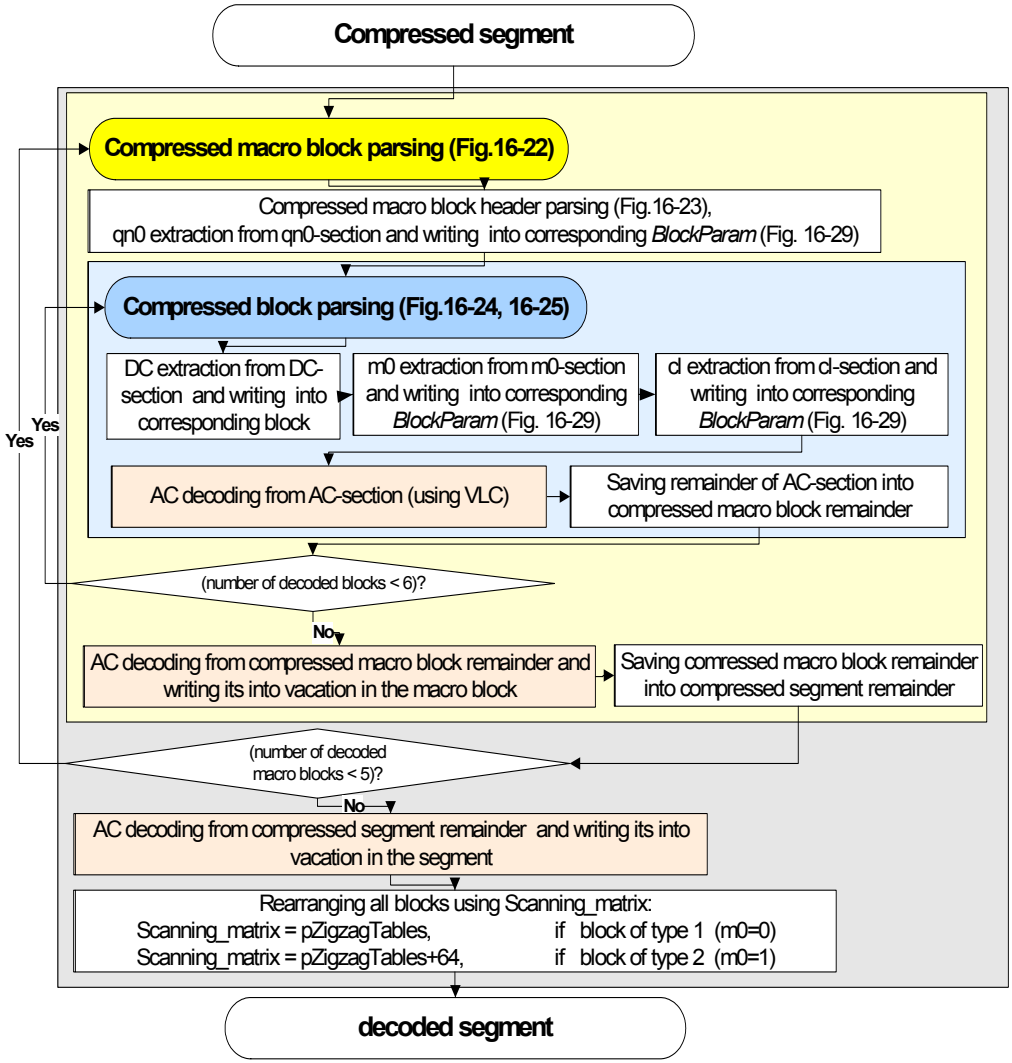
Variable Length Decoding

Structure of the source tables must be as follows:

Example 16-4 Source Table Structure

```
static Ipp32s Table[]=
{
max_bits;                                The maximum length of code
sub_sz1;                                 Not used

sub_sz2;
N1;                                      The number of 1-bit codes
(code1, run1, level1);                   The 1-bit codes, run values and level
                                         values.
(code2, run2, level2);
…;
(codeN1, runN1, levelN1);
N2;                                      The number of 2-bit codes
(code1, run1, level1);                   The 2-bit codes, run values and level
                                         values.
(code2, run2, level2);
…;
(codeN2, runN2, levelN2);
…;                                       …
Nm;                                      The number of maximum length codes
(code1, run1, level1);                   The maximum length codes, run values
                                         and level values.
(code2, run2, level2);
…;
(codeNm, runNm, levelNm);
-1;                                      The significant value to indicate the
                                         end of table
};
```

Figure 16-32 Compressed Segment Decoding



InitAllocHuffmanTable_DV

Allocates memory and initializes table.

Syntax

```
IppStatus ippiInitAllocHuffmanTable_DV_32u(Ipp32s* pSrcTable1, Ipp32s*
    pSrcTable2, Ipp32u** ppHuffTable);
```

Parameters

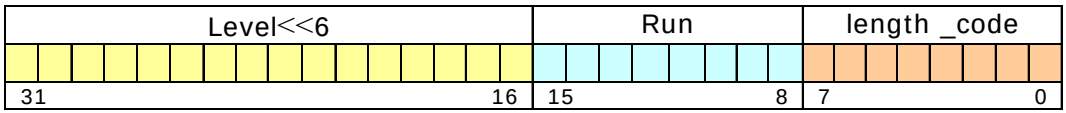
pSrcTable1	Pointer to Source Table 1.
pSrcTable2	Pointer to Source Table 2.
ppHuffTable	Double pointer to the destination decoding table.

Description

This function is declared in the `ippvc.h` header file. The function `ippiInitAllocHuffmanTable_DV_32u` allocates memory and initializes the table that contains codes for DCT coefficients (Run-Level codes). The function allocates memory at the address that* *ppHuffTable* points to, and fills it with data from Source Table 1 that *pSrcTable1* points to and from Source Table 2 that *pSrcTable2* points to. See Example 16-4 for the source table structure. When filling the destination decoding table the function multiplies the value of `level` by 64 (shift left by 6).

While decoding, the search for code is performed through the first subtable (Figure 16-3), which contains the most frequent and shortest codes. If the code value is not found, then the search continues through the following subtables, containing additional bits for longer codes. Those subtables also forward the search to the next ones, if the code value is not found.

Figure 16-33 *HuffT* Structure



`length _code` = length code (if source tables == *pSrcTable1*),

`length _code` = length code + 4 (if source tables == *pSrcTable2*)

Example 16-5 `ippiInitAllocHuffmanTable_DV_32u` Usage

```
Ipp32s HuffSrcTable1[] =
{
    9, /* Maximum length of code in this table*/
    0, /* Not used*/
    0, /* Not used*/
    0, /* Number of 1-bit codes*/
    0, /* Number of 2-bit codes*/
    2, /* Number of 3-bit codes*/
    0x0000, 0, 1,       0x0001, 0, -1,
    3, /* Number of  4-bit codes*/
    0x0004, 0,  2,      0x0005, 0, -2,

    0x0006, 100, 100,  /* EOB*/

    6, /* Number of 5-bit codes*/
    0x000e, 1, 1,       0x000f, 1, -1,
    0x0010, 0, 3,       0x0011, 0, -3,
    0x0012, 0, 4,       0x0013, 0, -4,
    8, /* Number of 6-bit codes*/
    0x0028, 2, 1,       0x0029, 2, -1,
    0x002a, 1, 2,       0x002b, 1, -2,
    0x002c, 0, 5,       0x002d, 0, -5,
    0x002e, 0, 6,       0x002f, 0, -6,
    8, /* Number of 7-bit codes*/
    0x0060, 3, 1,       0x0061, 3, -1,
    0x0062, 4, 1,       0x0063, 4, -1,
    0x0064, 0, 7,       0x0065, 0, -7,
    0x0066, 0, 8,       0x0067, 0, -8,
    16, /* Number of 8-bit codes*/
    0x00d0, 5, 1,       0x00d1, 5, -1,
    0x00d2, 6, 1,       0x00d3, 6, -1,
    0x00d4, 2, 2,       0x00d5, 2, -2,
    0x00d6, 1, 3,       0x00d7, 1, -3,
    0x00d8, 1, 4,       0x00d9, 1, -4,
    0x00da, 0, 9,       0x00db, 0, -9,
    0x00dc, 0, 10,      0x00dd, 0, -10,
    0x00de, 0, 11,      0x00df, 0, -11,
    32, /* Number of 9-bit codes*/
    0x01c0, 7, 1,       0x01c1, 7, -1,
    0x01c2, 8, 1,       0x01c3, 8, -1,
    0x01c4, 9, 1,       0x01c5, 9, -1,
    0x01c6, 10, 1,      0x01c7, 10, -1,
    0x01c8, 3, 2,       0x01c9, 3, -2,
    0x01ca, 4, 2,       0x01cb, 4, -2,
    0x01cc, 2, 3,       0x01cd, 2, -3,
    0x01ce, 1, 5,       0x01cf, 1, -5,
```

Example 16-5 ippiInitAllocHuffmanTable_DV_32u Usage (continued)

```
    0x01d0, 1, 6,       0x01d1, 1, -6,
    0x01d2, 1, 7,       0x01d3, 1, -7,
    0x01d4, 0, 12,      0x01d5, 0, -12,
    0x01d6, 0, 13,      0x01d7, 0, -13,
    0x01d8, 0, 14,      0x01d9, 0, -14,
    0x01da, 0, 15,      0x01db, 0, -15,
    0x01dc, 0, 16,      0x01dd, 0, -16,
    0x01de, 0, 17,      0x01df, 0, -17,
    -1 /* end of table*/
};

Ipp32s HuffSrcTable2[] =
{
    9, /* Maximum length of code. Note that 4 is added to
          all code lengths in the second table*/
    0, /* Not used*/
    0, /* Not used*/

    0,  /* Number of 1(5)-bit codes*/
    0,  /* Number of 2(6)-bit codes*/
    0,  /* Number of 3(7)-bit codes*/
    0,  /* Number of 4(8)-bit codes*/
    0,  /* Number of 5(9)-bit codes*/
    32, /* Number of 6(10)-bit codes*/
    0x0000, 11, 1,      0x0001, 11, -1,
    0x0002, 12, 1,      0x0003, 12, -1,
    0x0004, 13, 1,      0x0005, 13, -1,
    0x0006, 14, 1,      0x0007, 14, -1,
    0x0008, 5,  2,      0x0009, 5 , -2,
    0x000a, 6 , 2,      0x000b, 6 , -2,
    0x000c, 3,  3,      0x000d, 3 , -3,
    0x000e, 4 , 3,      0x000f, 4 , -3,

    0x0010, 2, 4 ,      0x0011, 2, -4 ,
    0x0012, 2, 5 ,      0x0013, 2, -5,
    0x0014, 1, 8 ,      0x0015, 1, -8 ,
    0x0016, 0, 18,      0x0017, 0, -18,
    0x0018, 0, 19,      0x0019, 0, -19,
    0x001a, 0, 20,      0x001b, 0, -20,
    0x001c, 0, 21,      0x001d, 0, -21,
    0x001e, 0, 22,      0x001f, 0, -22,
    16, /* Number of 7(11)-bit codes*/
    0x0040, 5, 3,       0x0041, 5, -3,
    0x0042, 3, 4,       0x0043, 3, -4,
    0x0044, 3, 5,       0x0045, 3, -5,
    0x0046, 2, 6,       0x0047, 2, -6,
    0x0048, 1, 9,       0x0049, 1, -9,
    0x004a, 1,10,       0x004b, 1, -10,
    0x004c, 1, 11,      0x004d, 1, -11,
    0x004e, 0, 0,       0x004f, 1, 0,
```

Example 16-5 `ippiInitAllocHuffmanTable_DV_32u` Usage (continued)

```
    16,  /* Number of 8(12)-bit codes*/
    0x00a0, 6, 3,        0x00a1, 6, -3,
    0x00a2, 4, 4,        0x00a3, 4, -4,
    0x00a4, 3, 6,        0x00a5, 3, -6,
    0x00a6, 1,12,        0x00a7, 1,-12,
    0x00a8, 1,13,        0x00a9, 1,-13,
    0x00aa, 1,14,        0x00ab, 1,-14,
    0x00ac, 2, 0,        0x00ad, 3, 0 ,
    0x00ae, 4, 0,        0x00af, 5, 0,

    16*2 + 56 + 6 + 2,  /* Number of 9(13)-bit codes*/
    0x0160, 7, 2,        0x0161, 7, -2,
    0x0162, 8, 2,        0x0163, 8, -2,
    0x0164, 9, 2,        0x0165, 9, -2,
    0x0166, 10, 2,       0x0167, 10, -2,
    0x0168, 7, 3,        0x0169, 7, -3,
    0x016a, 8, 3,        0x016b, 8, -3,
    0x016c, 4, 5,        0x016d, 4, -5,
    0x016e, 3, 7,        0x016f, 3, -7,

    0x0170, 2, 7,        0x0171, 2, -7,
    0x0172, 2, 8,        0x0173, 2, -8,
    0x0174, 2, 9,        0x0175, 2, -9,
    0x0176, 2, 10,       0x0177, 2, -10,
    0x0178, 2, 11,       0x0179, 2, -11,
    0x017a, 1, 15,       0x017b, 1, -15,
    0x017c, 1, 16,       0x017d, 1, -16,
    0x017e, 1, 17,       0x017f, 1, -17,

    /* Combinations (0,0) through (6,0) have special codes*/
    0x0180 + 0, 0, 0,     0x0180 + 1, 0, 0,
    0x0180 + 2, 0, 0,     0x0180 + 3, 0, 0,
    0x0180 + 4, 0, 0,     0x0180 + 5, 0, 0, /* 1*/

   0x0180 + 6,  6,  0,   0x0180 + 7,  7, 0,  0x0180 +  8,  8, 0,  0x0180 +  9,  9, 0,/*  1*/
   0x0180 + 10, 10, 0,   0x0180 + 11, 11, 0,  0x0180 + 12, 12, 0,  0x0180 + 13, 13, 0,/*  2*/
   0x0180 + 14, 14, 0,   0x0180 + 15, 15, 0,  0x0180 + 16, 16, 0,  0x0180 + 17, 17, 0,/*  3*/
   0x0180 + 18, 18, 0,   0x0180 + 19, 19, 0,  0x0180 + 20, 20, 0,  0x0180 + 21, 21, 0,/*  4*/
   0x0180 + 22, 22, 0,   0x0180 + 23, 23, 0,  0x0180 + 24, 24, 0,  0x0180 + 25, 25, 0,/*  5*/
   0x0180 + 26, 26, 0,   0x0180 + 27, 27, 0,  0x0180 + 28, 28, 0,  0x0180 + 29, 29, 0,/*  6*/
   0x0180 + 30, 30, 0,   0x0180 + 31, 31, 0,  0x0180 + 32, 32, 0,  0x0180 + 33, 33, 0,/*  7*/
   0x0180 + 34, 34, 0,   0x0180 + 35, 35, 0,  0x0180 + 36, 36, 0,  0x0180 + 37, 37, 0,/*  8*/
   0x0180 + 38, 38, 0,   0x0180 + 39, 39, 0,  0x0180 + 40, 40, 0,  0x0180 + 41, 41, 0,/*  9*/
   0x0180 + 42, 42, 0,   0x0180 + 43, 43, 0,  0x0180 + 44, 44, 0,  0x0180 + 45, 45, 0,/* 10*/
   0x0180 + 46, 46, 0,   0x0180 + 47, 47, 0,  0x0180 + 48, 48, 0,  0x0180 + 49, 49, 0,/* 11*/
   0x0180 + 50, 50, 0,   0x0180 + 51, 51, 0,  0x0180 + 52, 52, 0,  0x0180 + 53, 53, 0,/* 12*/
   0x0180 + 54, 54, 0,   0x0180 + 55, 55, 0,  0x0180 + 56, 56, 0,  0x0180 + 57, 57, 0,/* 13*/
   0x0180 + 58, 58, 0,   0x0180 + 59, 59, 0,  0x0180 + 60, 60, 0,  0x0180 + 61, 61, 0,/* 14
(14*4=56)*/
```

Example 16-5 ippiInitAllocHuffmanTable_DV_32u Usage (continued)

```
  0x0180 + 62, 0, 0,    0x0180 + 63, 0, 0, /* combinations (62,0) and (63,0) are not
anticipated*/
  -1 /* end of table*/
};

{
    if (ippStsOk != ippiInitAllocHuffmanTable_DV_32u(SrcTable1,
                                                     SrcTable2,
                                                     &pHuffTable))
    {
        /* Process errors*/
    }
}
```

This function is used in the DV decoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.
ippStsMemAllocErr	Indicates an error when no memory is allocated.

HuffmanDecodeSegment_DV

Decodes and rearranges segment block, multiplies first block element by 128.

Syntax

```
IppStatus ippiHuffmanDecodeSegment_DV_8u16s(const Ipp8u* pStream, const Ipp32u*
    pZigzagTables, const Ipp32u* pHuffTable, Ipp16s* pBlock, Ipp32u*
    pBlockParam);
```

Parameters

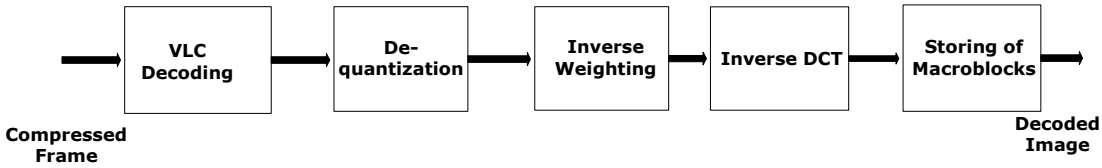
pStream	Pointer to bitstream (compressed segment).
pZigzagTables	Pointer to the array of two scanning matrices.
pHuffTable	Pointer to the decoding Huffman table.
pBlock	Pointer to decoded elements.
pBlockParam	Pointer to output parameters array [30].

Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanDecodeSegment_DV_8u16s` decodes and rearranges segment, multiplies the first block element by 128.

Figure 16-34 depicts the DV frame decoding process.

Figure 16-34 DV Frame Decoding



At the first step, `ippiHuffmanDecodeSegment_DV_8u16s` function decodes the compressed frame. The output of the function is 6 decompressed macro blocks, which are then de-quantized, de-weighted, processed by inverse DCT, and stored to the appropriate places of the destination image.

AC coefficients are decoded using Variable Length Codes with the help of the destination decoding table created by `InitAllocHuffmanTable_DV`. The pointer to this table is passed to the `ippiHuffmanDecodeSegment_DV` function through the *pHuffTable* pointer. The table contains level values multiplied by 64, and the `ippiHuffmanDecodeSegment_DV` function also multiplies output values of AC coefficients by 64.

DC coefficients are obtained from the first 2 bytes of the compressed blocks (see Figure 16-29 and Figure 16-30) with zeroing of the last 7 bits. The output values of DC coefficients are multiplied by 128 (shift left by 7).

The pointer *pStream* points to a compressed segment, which is a sequence of 5 compressed macroblocks. Each compressed macroblock occupies 80 bytes of data, hence the total length of the compressed video segment is 400 bytes.

The *pZigzagTables* pointer refers to two scanning matrices which are used to de-zigzag AC coefficients.

Scanning_matrix = *pZigzagTables*, if the block is of type 1(`m0` = 0).
Scanning_matrix = *pZigzagTables* + 64, if the block is of type 2 (`m0` = 1).

The argument* *pBlockParam* (Figure 16-35) stores values `m0` (block type), `cl` (class number), `index` (index of the last decoded element), and `eob` (end of block indicator) for every block.

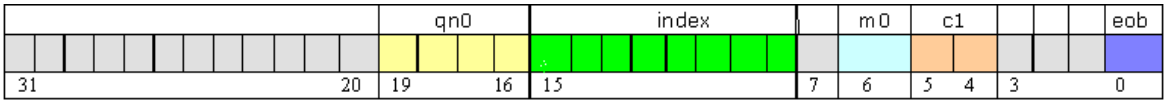
Value `qn0` (quantization number) is obtained from the first block of the macroblock. The same value `qn0` is used for all blocks of the macroblock.

With 64 elements in an 8x8 block, if only the DC coefficient (that is, the first block element) is decoded, `index`=0. If all the elements are decoded, `index`=63.

`eob`=1 (“true”), if the symbol of the block end EOB is present in the block, otherwise `eob`=0 (“false”).

See Example 16-6 for general usage and Example 16-7 for usage with `index` and `eob` values

Figure 16-35 *BlockParam* Structure



Example 16-6 Example of `HuffmanDecodeSegment_DV` Usage

```
/*  This table was designed to avoid coincidences with any GPL code*/
Ipp32u DeZigzagTables[] =
{

    0,
    1,

    8,          16, 9,  2,                  3, 10,  17,
    24,         32,         25,             18,         11,
    4,          5,              12,         19,             26,
    33,         40,             48,         41,             34,
    27,         20,             13,         6,              7,
    14,         21,             28,         35,             42,
    49,         56,         57,             50,         43,
    36,         29, 22, 15,                 23, 30, 37,
    44,         51,                         58,
    59,         52,                         45,
    38,         31,                         39,
    46,         53,                         60,
    61,         54,                         47,
```

Example 16-6 Example of `HuffmanDecodeSegment_DV` Usage (continued)

```
    55,         62,                         63,

    0,  32,  1, 33,  8, 40,  2, 34,
    9,  41, 16, 48, 24, 56, 17, 49,
    10, 42,  3, 35,  4, 36, 11, 43,
    18, 50, 25, 57, 26, 58, 19, 51,
    12, 44,  5, 37,  6, 38, 13, 45,
    20, 52, 27, 59, 28, 60, 21, 53,
    14, 46,  7, 39, 15, 47, 22, 54,
     29, 61, 30, 62, 23, 55, 31, 63
};

{
    /* Create Huffman Decoding table*/
    ippiInitAllocHuffmanTable_DV_32u( ..., ..., &pHuffTable);

    ...

    /* Huffman Decode video segment

       pEncodedVideoSegment - (Ipp8u*) pointer to compresses video segment
       DeZigzagTables       - (Ipp32u*) pointer to array of two de-zigzag tables
      pHuffTable         - (Ipp32u*) pointer to Huffman decoding table created
                                 by the ippiInitAllocHuffmanTable_DV_32u function
      lpsDecodedBlocks     - (Ipp16s*) pointer to 30 decoded DCT blocks (5 macro
                                          blocks of 6 DCT blocks)
       BlockParamBuffer     - (Ipp32u*) pointer to array of 30 unsigned double
                                       words, containing DC, m0 and cl values for
                                          DCT blocks
      */
       ippiHuffmanDecodeSegment_DV_8u16s(pEncodedVideoSegment,
                                          DeZigzagTables,
                                          pHuffTable,
                                          lpsDecodedBlocks,
                                          BlockParamBuffer);

}
```

Example 16-7 Example of `HuffmanDecodeSegment_DV` Usage with `index` and `eob` values

```
void test_huff2(void)
{
/* Create Huffman Decoding table*/
ippiInitAllocHuffmanTable_DV_32u( ..., ..., &pHuffTable);
...
/* Huffman Decode video segment
pEncodedVideoSegment - (Ipp8u*) pointer to compresses video segment
DeZigzagTables - (Ipp32u*) pointer to array of two de-zigzag tables
pHuffTable - (Ipp32u*) pointer to Huffman decoding table created
by the ippiInitAllocHuffmanTable_DV_32u function
lpsDecodedBlocks - (Ipp16s*) pointer to 30 decoded DCT blocks (5 macro
```

Example 16-7 Example of HuffmanDecodeSegment_DV Usage with index and eob values

```
blocks of 6 DCT blocks)
BlockParamBuffer - (Ipp32u*) pointer to array of 30 unsigned double
words, containing DC, m0 and cl values for
DCT blocks
*/
ippiHuffmanDecodeSegment_DV_8u16s(pEncodedVideoSegment,
DeZigzagTables,
pHuffTable,
lpsDecodedBlocks,
BlockParamBuffer);
for (mb_num = 0;mb_num < 5;mb_num++)
{
    // get quantization number
    qno = (BlParamBuffer[mb_num*  6] >> 16) & 0x0F;

    // decompress each block
    for (b_num = 0;b_num < 6;b_num++)
    {
        block_type = (BlParamBuffer[mb_num*  6 + b_num] >> 6) & 0x01;
        bl_index = (BlParamBuffer[mb_num*  6 + b_num] >> 8) & 0xff;
        eob = BlParamBuffer[mb_num*  6 + b_num]  & 0x01;

        // do inverse and adaptive dequantization
        ippiQuantInv_DV_16s_C1I(lpsDecodedBlocks,...);

        // do iDCT
        if (0 == block_type)
        {
            if ((eob == 0)||( 10 > bl_index))
                ippiDCT8x8Inv_4x4_16s_C1I((Ipp16s* ) (lpsDecodedBlocks));
            else
                ippiDCT8x8Inv_16s_C1I((Ipp16s* ) (lpsDecodedBlocks));
        }
        else
        {
            ippiDCT2x4x8Inv_16s_C1I((Ipp16s* ) (lpsDecodedBlocks));
        }
    }
}

}
```

This function is used in the DV decoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

HuffmanDecodeSegment_DV100

Decodes and rearranges DV100 segment block, multiplies first block element by 128.

Syntax

```
IppStatus ippiHuffmanDecodeSegment_DV100_8u16s(const Ipp8u* pStream, const
    Ipp32u* pZigzagTables, const Ipp32u* pHuffTable, Ipp16s* pBlock, Ipp32u*
    pBlockParam);
```

Parameters

pStream	Pointer to bitstream (compressed video segment).
pZigzagTables	Pointer to the de-zigzag table.
pHuffTable	Pointer to the decoding Huffman table.
pBlock	Pointer to the array where DCT decoded blocks should be stored.
pBlockParam	Pointer to output parameters array [40].

Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanDecodeSegment_DV100_8u16s` decodes and rearranges DV100 video segment, multiplies the first block element by 128.

AC coefficients are decoded using Variable Length Codes with the help of the destination decoding table created by `InitAllocHuffmanTable_DV`. Since this table contains level values multiplied by 64, the output values of AC coefficients are also multiplied by 64.

DC coefficients are obtained from the first 2 bytes of the compressed blocks (see Figure 16-29 and Figure 16-30) with zeroing of the last 7 bits. The output values of DC coefficients are multiplied by 128 (shift left by 7).

The pointer *pStream* points to a compressed video segment, which is a sequence of 5 compressed macroblocks. Each macroblock consists of 8 DCT blocks (4 luma and 4 color difference blocks).

The *pZigzagTables* pointer refers to the array that contains the de-zigzag index.

The argument* *pBlockParam* (Figure 16-35) stores values `m0` (block type), `cl` (class number) for every block. In DV100, the same DCT mode (8-8-frame or 8-8-field) is applied to all DCT blocks in a macroblock. The DCT mode is stored in the first DCT block of

the macro block; m0 values of the other DCT blocks are not used. Value qn0 (quantization number) is obtained from the first block of the macroblock. The same value qn0 is used for all blocks of the macroblock.

See Example for the usage example for ippiHuffmanDecodeSegment_DV100_8u16s.

Example 16-8 Usage of HuffmanDecodeSegment_DV100

```
{
    /* Create Huffman Decoding table*/
    /*Dv and DV100 use same Huffman Decoding table*/
    ippiInitAllocHuffmanTable_DV_32u( ..., ..., &pHuffTable);
    /*
    ...
   */
    /* Huffman Decode video segment
    pEncodedVideoSegment - (Ipp8u*) pointer to compresses video segment
    DeZigzagTables - (Ipp32u*) pointer to array of de-zigzag table
    pHuffTable - (Ipp32u*) pointer to Huffman decoding table created
    by the ippiInitAllocHuffmanTable_DV_32u function
    lpsDecodedBlocks - (Ipp16s*) pointer to 40 decoded DCT blocks (5 macro
    blocks of 8 DCT blocks, 8*  5 = 40)
    BlockParamBuffer - (Ipp32u*) pointer to array of 40 unsigned double
    words, containing DC, m0 and cl values for DCT blocks
   */
    ippiHuffmanDecodeSegment_DV100_8u16s(pEncodedVideoSegment,
                                         DeZigzagTable,
                                         pHuffTable,
                                         lpsDecodedBlocks,
                                         BlockParamBuffer);
    /*
    ...
   */
};
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

HuffmanDecodeSegmentOnePass_DV

Performs first pass of video segment decoding, rearranges segment block, multiplies first block element by 128.

Syntax

```
IppStatus ippiHuffmanDecodeSegmentOnePass_DV_8u16s(const Ipp8u* pStream, const
    Ipp32u* pZigzagTables, const Ipp32u* pHuffTable, Ipp16s* pBlock, Ipp32u*
    pBlockParam, Ipp32s nNumCoeffs);
```

Parameters

pStream	Pointer to bitstream (compressed video segment).
pZigzagTables	Pointer to the array of two scanning matrices.
pHuffTable	Pointer to the decoding Huffman table.
pBlock	Pointer to the array where DCT decoded blocks should be stored.
pBlockParam	Pointer to output parameters array [30].
nNumCoeffs	Maximum number of DCT coefficients to extract from the stream.

Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanDecodeSegmentOnePass_DV_8u16s` is designed to be used in DV video decoder in a quarter resolution mode. The function performs only the first pass of video segment VLC decoding and extracts not more than specified number of DCT coefficients.

The function accepts the same arguments as the `HuffmanDecodeSegment_DV` function and one additional parameter – the maximum number of DCT coefficients to extract from the stream, which is passed through the *nNumCoeffs* argument. If the limit of coefficients is reached during the first pass, the function goes over to the next DCT block. If the limit is not reached during the first pass, the function does not perform the second and the third passes and extracts only the number of DCT coefficients available in the first pass.

The *nNumCoeffs* parameter must fall within the range of 1 to 64.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

ippStsBadArgErr	Indicates an error when *nNumCoeffs* does not fall within the range of 1 through 64.

FreeHuffmanTable_DV

Frees the memory allocated for VLC table.

Syntax

```
IppStatus ippiFreeHuffmanTable_DV_32u(Ipp32u* pHuffTable);
```

Parameters

pHuffTable	Pointer to the decoding table.

Description

This function is declared in the `ippvc.h` header file. The function `ippiFreeHuffmanTable_DV_32u` frees the memory at *pHuffTable* allocated for VLC table.

This function is used in the DV decoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when *pHuffTable* is NULL.

Inverse Quantization

QuantInv_DV

Performs inverse quantization on a block.

Syntax

```
IppStatus ippiQuantInv_DV_16s_C1I(Ipp16s* pSrcDst, Ipp16s* pDequantTable);
```

Parameters

pSrcDst	Pointer to the block.
pDequantTable	Pointer to the dequantization table.

Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantInv_DV_16s_C1I` performs inverse quantization on a block. Each of the decoded DCT coefficients in the block should be inverse quantized through multiplying by a corresponding value from the weighting matrix and division on 2^{14}. Selection of dequantization table depends of the block type.

See Example 16-9 and Example 16-10 for an example of the function usage.

This function is used in the DV decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

QuantWeightBlockInv_DV

Performs inverse quantization and inverse weighting on a block.

Syntax

```
IppStatus ippiQuantWeightBlockInv_DV_16s_C1I(Ipp16s* pSrcDst, const Ipp16s*
    pQuantInvTable, const Ipp16s* pWeightInvTable);
```

Parameters

pSrcDst	Pointer to the source and destination 8x8 (or 2x4x8) block.
pQuantInvTable	Pointer to an array that contains values of quantification table for the current block.
pWeightInvTable	Pointer to an array that contains values of weight table for the current block.

NOTE. There are two different types of blocks: 8x8 and 2x4x8. Select respective weighting tables for 8x8 or 2x4x8 blocks.

Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantWeightBlockInv_DV_16s_C1I` performs inverse quantization and inverse weighting on a block in accordance with [SMPTE314M] standard.

The DV frame decoding process is depicted in Figure 16-34. The `ippiQuantWeightBlockInv_DV_16s_C1I` function covers de-quantization and inverse weighting steps. Choose appropriate quantization and weighting tables, that is, dequantization and deweighting tables for decoding. To choose the quantization table, use the values of two parameters - *qno* (quantization number) and the *block quant class*. For details, see [SMPTE314M], Table 23 - Quantization step. Use different weighting tables for 8x8 and 2x4x8 blocks.

The `ippiQuantWeightBlockInv_DV_16s_C1I` function uses the following three pointers as input arguments:

pDataBlock	Pointer to the current block
pQuantTable	Pointer to the quantization table
pWeightTable	Pointer to the weighting table

The function performs the following operations:

```
for (Ipp32s i = 0;i < 64;i += 1)
        pDataBlock[i] = (Ipp16s) ( (
                                   ((Ipp16s) pDataBlock [1]>>3      )
                                *  ((Ipp16s) pQuantTable[i] )
                                 *  ((Ipp16s) pWeightTable[i] ) )>>16);
```

The *pWeightTable* weighting table contains floating point data with values varying from 2.0 to 4.0. Scaling is required for integer multiplication through multiplying the table values by 8192 (1<<13). Then back shift is required through division by 65536 to provide for REAL range, that is, to perform inverse scaling.

After ippiQuantWeightBlockInv_DV_16s_C1I completes dequantization and inverse weighing, iDCT is performed before creating the macroblock.

Compare QuantInv_DV and QuantWeightBlockInv_DV. The ippiQuantInv_DV_16s_C1I function has two arguments: *pDataBlock*, which is a pointer to the current block, and *pQuantTable*, a modification of the quantization table with values multiplied by 256 (1<< 8) and divided by weight coefficients for Ipp16s datatype. The ippiQuantWeightBlockInv_DV_16s_C1I function has separate quantization and weighting tables that contain input arguments for the function.

See Example 16-9, Example 16-10, and Example 16-11 of ippiQuantInv_DV_16s_C1I and ippiQuantWeightBlockInv_DV_16s_C1I usage.

Example 16-9 Inverse Quantization Tables

```
{

//-----------------------------------------------------------------------------
//Several necessary tables

#define _INTERNAL_CS1 0.980785  //cos(1*PI/16)
#define _INTERNAL_CS2 0.923880  //cos(2*PI/16)
#define _INTERNAL_CS3 0.831470  //cos(3*PI/16)
#define _INTERNAL_CS4 0.707107  //cos(4*PI/16)
#define _INTERNAL_CS5 0.555570  //cos(5*PI/16)
#define _INTERNAL_CS6 0.382683  //cos(6*PI/16)
#define _INTERNAL_CS7 0.195090  //cos(7*PI/16)

#define _W0 (1.0000001)
#define _W1 (_INTERNAL_CS4 / (4.0*  _INTERNAL_CS7*  _INTERNAL_CS2))
#define _W2 (_INTERNAL_CS4 / (2.0*  _INTERNAL_CS6))
#define _W3 (1.0 / (2.0*  _INTERNAL_CS5))
#define _W4 (7.0 / 8.0)
#define _W5 (_INTERNAL_CS4 / _INTERNAL_CS3)
#define _W6 (_INTERNAL_CS4 / _INTERNAL_CS2)
#define _W7 (_INTERNAL_CS4 / _INTERNAL_CS1)

 tabl1[] =
    {
        0,1,8,16,    9,2,3,10,
        17,24,32,25, 18,11,4,5,
        12,19,26,33, 40,48,41,34,
        27,20,13,6,  7,14,21,28,
        35,42,49,56, 57,50,43,36,
        29,22,15,23, 30,37,44,51,
        58,59,52,45, 38,31,39,46,
        53,60,61,54, 47,55,62,63
    };
```

Example 16-9 Inverse Quantization Tables (continued)

```
    Ipp16s tabl2[] =
    {
        0,  32,  1, 33,  8, 40,  2, 34,
        9,  41, 16, 48, 24, 56, 17, 49,
        10, 42,  3, 35,  4, 36, 11, 43,
        18, 50, 25, 57, 26, 58, 19, 51,
        12, 44,  5, 37,  6, 38, 13, 45,
        20, 52, 27, 59, 28, 60, 21, 53,
        14, 46,  7, 39, 15, 47, 22, 54,
        29, 61, 30, 62, 23, 55, 31, 63
    };

    static double TablW0[] =
    {

(0.5/*2500076*/),(_W0*_W1/2),(_W0*_W2/2),(_W0*_W3/2),(_W0*_W4/2),(_W0*_W5/2),(_W0*_W6/2),(_W0*_W7/2),

(_W1*_W0/2),(_W1*_W1/2),(_W1*_W2/2),(_W1*_W3/2),(_W1*_W4/2),(_W1*_W5/2),(_W1*_W6/2),(_W1*_W7/2),

(_W2*_W0/2),(_W2*_W1/2),(_W2*_W2/2),(_W2*_W3/2),(_W2*_W4/2),(_W2*_W5/2),(_W2*_W6/2),(_W2*_W7/2),

(_W3*_W0/2),(_W3*_W1/2),(_W3*_W2/2),(_W3*_W3/2),(_W3*_W4/2),(_W3*_W5/2),(_W3*_W6/2),(_W3*_W7/2),


(_W4*_W0/2),(_W4*_W1/2),(_W4*_W2/2),(_W4*_W3/2),(_W4*_W4/2),(_W4*_W5/2),(_W4*_W6/2),(_W4*_W7/2),

(_W5*_W0/2),(_W5*_W1/2),(_W5*_W2/2),(_W5*_W3/2),(_W5*_W4/2),(_W5*_W5/2),(_W5*_W6/2),(_W5*_W7/2),

(_W6*_W0/2),(_W6*_W1/2),(_W6*_W2/2),(_W6*_W3/2),(_W6*_W4/2),(_W6*_W5/2),(_W6*_W6/2),(_W6*_W7/2),

(_W7*_W0/2),(_W7*_W1/2),(_W7*_W2/2),(_W7*_W3/2),(_W7*_W4/2),(_W7*_W5/2),(_W7*_W6/2),(_W7*_W7/2)
    };


    static Ipp16s TablW0_Scale_ZigZag[] =
{
(Ipp16s)(8192.0/TablW0[0]), (Ipp16s)(8192.0/TablW0[1]), (Ipp16s)(8192.0/TablW0[8]), (Ipp16s)(8192.0/TablW0[16]),
(Ipp16s)(8192.0/TablW0[9]), (Ipp16s)(8192.0/TablW0[2]), (Ipp16s)(8192.0/TablW0[3]), (Ipp16s)(8192.0/TablW0[10]),
(Ipp16s)(8192.0/TablW0[17]),(Ipp16s)(8192.0/TablW0[24]),(Ipp16s)(8192.0/TablW0[32]),(Ipp16s)(8192.0/TablW0[25]),
(Ipp16s)(8192.0/TablW0[18]),(Ipp16s)(8192.0/TablW0[11]),(Ipp16s)(8192.0/TablW0[4]), (Ipp16s)(8192.0/TablW0[5]),
(Ipp16s)(8192.0/TablW0[12]),(Ipp16s)(8192.0/TablW0[19]),(Ipp16s)(8192.0/TablW0[26]),(Ipp16s)(8192.0/TablW0[33]),
(Ipp16s)(8192.0/TablW0[40]),(Ipp16s)(8192.0/TablW0[48]),(Ipp16s)(8192.0/TablW0[41]),(Ipp16s)(8192.0/TablW0[34]),
(Ipp16s)(8192.0/TablW0[27]),(Ipp16s)(8192.0/TablW0[20]),(Ipp16s)(8192.0/TablW0[13]),(Ipp16s)(8192.0/TablW0[6]),
(Ipp16s)(8192.0/TablW0[7]), (Ipp16s)(8192.0/TablW0[14]),(Ipp16s)(8192.0/TablW0[21]),(Ipp16s)(8192.0/TablW0[28]),
(Ipp16s)(8192.0/TablW0[35]),(Ipp16s)(8192.0/TablW0[42]),(Ipp16s)(8192.0/TablW0[49]),(Ipp16s)(8192.0/TablW0[56]),
(Ipp16s)(8192.0/TablW0[57]),(Ipp16s)(8192.0/TablW0[50]),(Ipp16s)(8192.0/TablW0[43]),(Ipp16s)(8192.0/TablW0[36]),
(Ipp16s)(8192.0/TablW0[29]),(Ipp16s)(8192.0/TablW0[22]),(Ipp16s)(8192.0/TablW0[15]),(Ipp16s)(8192.0/TablW0[23]),
(Ipp16s)(8192.0/TablW0[30]),(Ipp16s)(8192.0/TablW0[37]),(Ipp16s)(8192.0/TablW0[44]),(Ipp16s)(8192.0/TablW0[51]),
(Ipp16s)(8192.0/TablW0[58]),(Ipp16s)(8192.0/TablW0[59]),(Ipp16s)(8192.0/TablW0[52]),(Ipp16s)(8192.0/TablW0[45]),
(Ipp16s)(8192.0/TablW0[38]),(Ipp16s)(8192.0/TablW0[31]),(Ipp16s)(8192.0/TablW0[39]),(Ipp16s)(8192.0/TablW0[46]),
(Ipp16s)(8192.0/TablW0[53]),(Ipp16s)(8192.0/TablW0[60]),(Ipp16s)(8192.0/TablW0[61]),(Ipp16s)(8192.0/TablW0[54]),
(Ipp16s)(8192.0/TablW0[47]),(Ipp16s)(8192.0/TablW0[55]), (Ipp16s)(8192.0/TablW0[62]),(Ipp16s)(8192.0/TablW0[63])
};

    static double TablW1[] =
```

Example 16-9 Inverse Quantization Tables (continued)

```
    {

(0.5/*2500076*/),(_W0*_W1/2),(_W0*_W2/2),(_W0*_W3/2),(_W0*_W4/2),(_W0*_W5/2),(_W0*_W6/2),(_W0*_W7/2),

(_W2*_W0/2),(_W2*_W1/2),(_W2*_W2/2),(_W2*_W3/2),(_W2*_W4/2),(_W2*_W5/2),(_W2*_W6/2),(_W2*_W7/2),

(_W4*_W0/2),(_W4*_W1/2),(_W4*_W2/2),(_W4*_W3/2),(_W4*_W4/2),(_W4*_W5/2),(_W4*_W6/2),(_W4*_W7/2),

(_W6*_W0/2),(_W6*_W1/2),(_W6*_W2/2),(_W6*_W3/2),(_W6*_W4/2),(_W6*_W5/2),(_W6*_W6/2),(_W6*_W7/2),

(_W0*_W0/2),(_W0*_W1/2),(_W0*_W2/2),(_W0*_W3/2),(_W0*_W4/2),(_W0*_W5/2),(_W0*_W6/2),(_W0*_W7/2),

(_W2*_W0/2),(_W2*_W1/2),(_W2*_W2/2),(_W2*_W3/2),(_W2*_W4/2),(_W2*_W5/2),(_W2*_W6/2),(_W2*_W7/2),

(_W4*_W0/2),(_W4*_W1/2),(_W4*_W2/2),(_W4*_W3/2),(_W4*_W4/2),(_W4*_W5/2),(_W4*_W6/2),(_W4*_W7/2),

(_W6*_W0/2),(_W6*_W1/2),(_W6*_W2/2),(_W6*_W3/2),(_W6*_W4/2),(_W6*_W5/2),(_W6*_W6/2),(_W6*_W7/2),
    };

    static Ipp16s TablW1_Scale_ZigZag[] =
{
(Ipp16s)(8192.0/TablW1[0]), (Ipp16s)(8192.0/TablW1[32]),(Ipp16s)(8192.0/TablW1[1]), (Ipp16s)(8192.0/TablW1[33]),
(Ipp16s)(8192.0/TablW1[8]), (Ipp16s)(8192.0/TablW1[40]),(Ipp16s)(8192.0/TablW1[2]), (Ipp16s)(8192.0/TablW1[34]),
(Ipp16s)(8192.0/TablW1[9]), (Ipp16s)(8192.0/TablW1[41]),(Ipp16s)(8192.0/TablW1[16]),(Ipp16s)(8192.0/TablW1[48]),
(Ipp16s)(8192.0/TablW1[24]),(Ipp16s)(8192.0/TablW1[56]),(Ipp16s)(8192.0/TablW1[17]),(Ipp16s)(8192.0/TablW1[49]),
(Ipp16s)(8192.0/TablW1[10]),(Ipp16s)(8192.0/TablW1[42]),(Ipp16s)(8192.0/TablW1[3]), (Ipp16s)(8192.0/TablW1[35]),
(Ipp16s)(8192.0/TablW1[4]), (Ipp16s)(8192.0/TablW1[36]),(Ipp16s)(8192.0/TablW1[11]),(Ipp16s)(8192.0/TablW1[43]),
(Ipp16s)(8192.0/TablW1[18]),(Ipp16s)(8192.0/TablW1[50]),(Ipp16s)(8192.0/TablW1[25]),(Ipp16s)(8192.0/TablW1[57]),
(Ipp16s)(8192.0/TablW1[26]),(Ipp16s)(8192.0/TablW1[58]),(Ipp16s)(8192.0/TablW1[19]),(Ipp16s)(8192.0/TablW1[51]),
(Ipp16s)(8192.0/TablW1[12]),(Ipp16s)(8192.0/TablW1[44]),(Ipp16s)(8192.0/TablW1[5]), (Ipp16s)(8192.0/TablW1[37]),
(Ipp16s)(8192.0/TablW1[6]), (Ipp16s)(8192.0/TablW1[38]),(Ipp16s)(8192.0/TablW1[13]),(Ipp16s)(8192.0/TablW1[45]),
(Ipp16s)(8192.0/TablW1[20]),(Ipp16s)(8192.0/TablW1[52]),(Ipp16s)(8192.0/TablW1[27]),(Ipp16s)(8192.0/TablW1[59]),
(Ipp16s)(8192.0/TablW1[28]),(Ipp16s)(8192.0/TablW1[60]),(Ipp16s)(8192.0/TablW1[21]),(Ipp16s)(8192.0/TablW1[53]),
(Ipp16s)(8192.0/TablW1[14]),(Ipp16s)(8192.0/TablW1[46]),(Ipp16s)(8192.0/TablW1[7]), (Ipp16s)(8192.0/TablW1[39]),
(Ipp16s)(8192.0/TablW1[15]),(Ipp16s)(8192.0/TablW1[47]),(Ipp16s)(8192.0/TablW1[22]),(Ipp16s)(8192.0/TablW1[54]),
(Ipp16s)(8192.0/TablW1[29]),(Ipp16s)(8192.0/TablW1[61]),(Ipp16s)(8192.0/TablW1[30]),(Ipp16s)(8192.0/TablW1[62]),
(Ipp16s)(8192.0/TablW1[23]),(Ipp16s)(8192.0/TablW1[55]),(Ipp16s)(8192.0/TablW1[31]),(Ipp16s)(8192.0/TablW1[63])
};
}
```

Example 16-10 ippiQuantInv_DV_16s_C1I Usage

```
{
      ippiHuffmanDecodeSegment_DV_8u16s((short* ) pDataBlock, ...);
      // get pointer to corresponding dequantize table
      // depends on qno & block quant class
      pQuantTable = (...);
      if (0 == block_type)
    {
        for (Ipp32s iI = 0; iI < 64; iI++)
            pQuantTable [iI] = pQuntTable [iI]* (Ipp16s)((double) (1 << 8) /
  TablW0[tabl1[iI]);

      // do inverse and adaptive dequantization
      ippiQuantInv_DV_16s_C1I((short* ) pDataBlock), (short* ) pQuantTable);
```

Example 16-10 ippiQuantInv_DV_16s_C1I Usage (continued)

```
      // do iDCT
      ippiDCT8x8Inv_16s_C1I((short* ) (pDataBlock));
    }

    else
    {
      for (Ipp32s iI = 0; iI < 64; iI++)
           pQuantTable [iI] = pQuantTable [iI]* (Ipp16s)(((double) (1 << 8)) /
  TablW1[tabl2[iI]]);

      // do inverse and adaptive dequantization
      ippiQuantInv_DV_16s_C1I((short* ) pDataBlock), (short* ) pQuantTable);
      // do iDCT
      ippiDCT2x4x8Inv_16s_C1I((short* ) (pDataBlock) );
    }
};
```

Example 16-11 ippiQuantWeightBlockInv_DV_16s_C1I Usage

```
{
      ippiHuffmanDecodeSegment_DV_8u16s((short* ) lpsBlocks,...);
      // get pointer to corresponding dequantization table
      // depends on qno & block quant class
      pQauntTable = (...);

      if (0 == block_type)
      {
            ippiQuantWeightInv_DV_16s_C1I( (Ipp16s* ) (pDataBlock),
                                           (Ipp16s* ) pQuantTable,
                                           (Ipp16s* ) TablW0_Scale_ZigZag );
            // do iDCT
            ippiDCT8x8Inv_16s_C1I((short* ) (pDataBlock));
      }

      else
      {
            ippiQuantWeightInv_DV_16s_C1I( (Ipp16s* ) (pDataBlock),
                                           (Ipp16s* ) pQuantTable,
                                           (Ipp16s* ) TablW0_Scale_ZigZag );
            // do iDCT
            ippiDCT2x4x8Inv_16s_C1I((short* ) (pDataBlock));
      }
    };
```

This function is used in the DV decoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.

`ippStsNullPtrErr`	Indicates an error when the *pSrcDst* pointer is NULL.

QuantWeightBlockInv_DV100

Performs inverse quantization and inverse weighting on a block according to DV100 standard.

Syntax

```
IppStatus ippiQuantWeightBlockInv_DV100_16s_C1I(Ipp16s* pSrcDst, const Ipp16s*
    pWeightInvTable, Ipp32s quantValue);
```

Parameters

pSrcDst	Pointer to the source and destination 8x8 block.
pWeightInvTable	Pointer to an array that contains values of weight table for the current block.
quantValue	Quantification step for the current block.

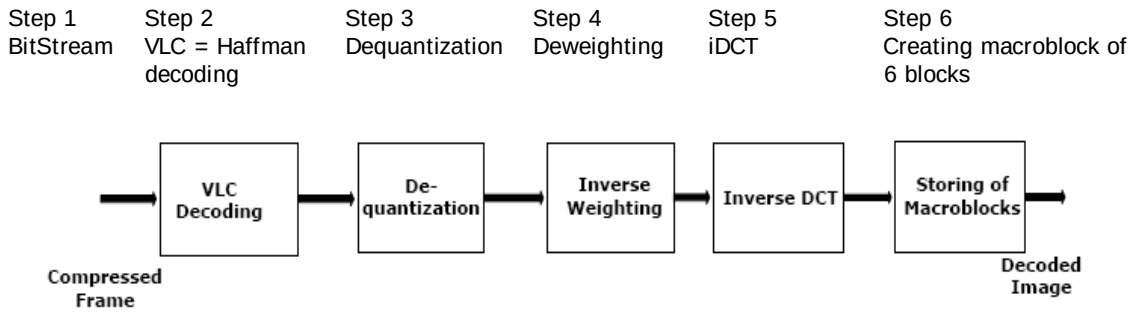
NOTE. This standard deals with only one block type - 8x8 block.

Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantWeightBlockInv_DV100_16s_C1I` performs inverse quantization and inverse weighting on a block in accordance with SMPTE 370M-2006 (DV100) [SMPTE370M-06] standard.

See Figure 16-36 for the DV100 frame decoding process.

Figure 16-36 DV100 One Block Decoding



Step 1 passes the pointer to the required stream. Step 2 is segment decoding. The `HuffmanDecodeSegment_DV(…)` function decodes the block along with multiplying its elements by 64, that is, performs <<6 shift. The `ippiQuantWeightBlockInv_DV100_16s_C1I(…)` function combines Steps 3 and 4. Choose appropriate quantization and weighting tables, that is, dequantization and deweighting tables for decoding. To choose the quantization table, use the values of two parameters - *QNO* (quantization number) and the appropriate block quant class number (see Example 16-12). For details, see [SMPTE370M-06], Table 25 - Quantization step.

The `ippiQuantWeightBlockInv_DV100_16s_C1I(…)` function uses the following three pointers as input arguments:

pDataBlock	Pointer to the current block
quantValue	Quantification step for the current block
pWeightTable	Pointer to the weighting table

The function performs the following operations:

```
for (Ipp32s i = 0;i < 64;i += 1)
                          pDataBlock[i] = (Ipp16s) (
                                ( pDataBlock[i]*  pWeightTable[i] ) )
                            *  quantValue )>>11 );
```

After `ippiQuantWeightBlockInv_DV100_16s_C1I(…)` completes Steps 3 and 4, Step 5 of iDCT is performed before Step 6.

See Example 16-12 of `ippiQuantWeightBlockInv_DV100_16s_C1I` usage.

Example 16-12 `ippiQuantWeightBlockInv_DV100_16s_C1I` Usage

```
{

//For more details see SMPTE 370M-2002 standard.
//You can similarly get tables LumaQuantizeMatrix_720System
//and ChromaQuantizeMatrix_720System.
const Ipp32s LumaQuantizeMatrix_1080System[64] =
{
    128,  16,  17,  18,  18,  19,  42,  44,
     16,  17,  18,  18,  19,  38,  43,  45,
     17,  18,  19,  19,  40,  41,  45,  48,
     18,  18,  19,  40,  41,  42,  46,  49,
     18,  19,  40,  41,  42,  43,  48, 101,
     19,  38,  41,  42,  43,  44,  98, 104,
     42,  43,  45,  46,  48,  98, 109, 116,
     44,  45,  48,  49, 101, 104, 116, 123
};

const Ipp32s ChromaQuantizeMatrix_1080System[64]=
{

    128,  16,  17,  25,  26,  26,  42,  44,
     16,  17,  25,  25,  26,  38,  43,  91,
     17,  25,  26,  27,  40,  41,  91,  96,
     25,  25,  27,  40,  41,  84,  93, 197,
     26,  26,  40,  41,  84,  86, 191, 203,
     26,  38,  41,  84,  86, 177, 197, 209,
     42,  43,  91,  93, 191, 197, 219, 232,
     44,  91,  96, 197, 203, 209, 232, 246
};

//For more details see SMPTE 370M-2002 standard
static const Ipp32s QuantizationSteps[] =
{
     //Class Number
     //0     1     2     3   //Quantization Number (QNO)
       0,    0,    0,    0,  //  0
       1,    2,    4,    8,  //  1
       2,    4,    8,    0,  //  2
       3,    6,   12,    0,  //  3
```

Example 16-12 ippiQuantWeightBlockInv_DV100_16s_C1I Usage (continued)

```
       4,    8,    0,    0,  //  4
       5,   10,    0,    0,  //  5
       6,   12,    0,    0,  //  6
       7,   14,    0,    0,  //  7
       8,    0,    0,    0,  //  8
      16,   32,   64,    0,  //  9
      18,   36,   72,    0,  // 10
      20,   40,   80,    0,  // 11
      22,   44,   88,    0,  // 12
      24,   48,   96,    0,  // 13
      28,   56,  112,    0,  // 14
      52,  104,    0,    0   // 15
};

/*
.....
Before quantization and weighting, you should perform Huffman decoding
.....
*/
    //nMBNum - number of the current macroblock
    int nMBNum;
    //curent DCT block in macroblock
    int nDCTBlockNum;
    //array of BlockParam for each block from 5 macroblock (8* 5 = 40)
    BlockParam BlParam[40];
    //Pointer to current DCT block in macroblock
    Ipp16s* pCurrDCTBlock;

    for(nMBNum = 0; nMBNum < 5; nMBNum++)
    {
        Ipp8u QNO = BlParam[nMBNum*  8].qno;

        for(nDCTBlockNum = 0; nDCTBlockNum < 8; nDCTBlockNum++)
        {
            Ipp8u ClassNumber = BlParam[nMBNum*  8 + nDCTBlockNum].cl;
            Ipp32s QuantStep = QuantizationSteps[QNO*4 + ClassNumber];
            const Ipp16s* pQuantizeTable;

            //Dequantize block
            //Select quantization table that depends on system type and type
            //of block (luma or chroma)
            if(m_System == System720_60p)
                pQuantizeTable = (nDCTBlockNum < 4) ?
                LumaQuantizeMatrix_720System : ChromaQuantizeMatrix_720System;
            else
                pQuantizeTable = (nDCTBlockNum < 4) ?
               LumaQuantizeMatrix_1080System : ChromaQuantizeMatrix_1080System;

            //Do quantization and weighting
            //For more performance you should align pCurrDCTBlock and
            //pQuantizeTable pointers on a 16-byte boundary.
```

Example 16-12 `ippiQuantWeightBlockInv_DV100_16s_C1I` Usage (continued)

```
            ippiQuantWeightBlockInv_DV100_16s_C1I(pCurrDCTBlock, (Ipp16s
           * )pQuantizeTable, QuantStep);
            //Do inverse DCT
            ippiDCT8x8Inv_16s_C1I(pCurrDCTBlock);

            pCurrDCTBlock += 64;
         }//for nDCTBlockNum = 0 to 7
     }
};
```

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the *`pSrcDst`* pointer is `NULL`.

Inverse Discrete Cosine Transformation

DCT2x4x8Inv

Performs the inverse DCT for block of type 2 (m0=1).

Syntax

```
IppStatus ippiDCT2x4x8Inv_16s_C1I(Ipp16s* pSrcDst);
```

Parameters

`pSrcDst`	Pointer to the block.

Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT2x4x8Inv_16s_C1I` performs the inverse DCT for a block of type 2 (*m0* = 1). Formula for inverse DCT is as follows:

$$'(x,\ 2*z) = \sum_{u=0}^{3} \sum_{h=0}^{7} (c(u)c(h)(c(h,\ u) + c(h,\ u+4))KC)$$

$$\varphi(x,\ 2*z+1) = \sum_{u=0}^{3} \sum_{h=0}^{7} (c(u)c(h)(c(h,\ u) - c(h,\ u+4))KC),$$

where

x in [0, 7]

y in [0, 7]

$z = \mathrm{int}(\frac{y}{2})$

$$KC = \cos\left(\frac{\pi u(2z+1)}{8}\right) \cos\left(\frac{\pi h(2x+1)}{16}\right)$$

$c(h) = \frac{0.5}{\mathrm{sqr}(2)}$ for $h = 0$,

$c(h) = 0.5$ for h in [1, 7],

$c(u) = \frac{0.5}{\mathrm{sqr}(2)}$ for $u = 0$,

$c(u) = 0.5$ for u in [1, 7]

This function is used in the DV decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when *`pSrcDst`* is `NULL`.

DCT8x4x2To4x4Inv_DV

Performs the inverse DCT for block 2x4x8 and then creates block 4x4.

Syntax

```
IppStatus ippiDCT8x4x2To4x4Inv_DV_16s_C1I(Ipp16s* pSrcDst);
```

Parameters

pSrcDst	Pointer to the source and destination block.

NOTE. There are two different types of blocks here: 8x8 (or 2x4x8) block as source and 4x4 (or 2x2) as destination.

Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x4x2To4x4Inv_DV_16s_C1I` performs the inverse DCT for a 2x4x8 block, then creates a 4x4 block: first, values in rows are averaged in pairs, then values in columns are averaged in pairs too. See Figure 16-37 for the new block values.

The *pSrcDst* parameter is a pointer to a block of 2x4x8 type. The function performs inverse DCT of the block and averages, first, two adjacent row elements and then two adjacent column elements. The averaging produces 16 elements all in all.

The resulting elements are stored in the memory in a row and without gaps (see Figure 16-38, AverageXX =AveXX) so that after the function is called the *pSrcDst* pointer points to the first element of the sixteen averaged elements.

Figure 16-37 Creating 4x4 Block from 8x8 Block

	0	1	2	3	4	5	6	7
0	Average00		Average01		Average02		Average03	
1								
2	Average04		Average05		Average06		Average07	
3								
4	Average08		Average09		Average10		Average11	
5								
6	Average12		Average13		Average14		Average15	
7								

Figure 16-38 Storing New 4x4 Values

↓pSrcDst						
Ave00	Ave01	Ave02	Ave03	Ave04	...	Ave15

Example 16-13 ippiDCT8x4x2To4x4Inv_DV_16s_C1I Usage

```
…
//mb_num = <current MacroBlock>;
//b_num = <current Block>;
//block_type - type of block
//lpBlocks - pointer on current block (8x8 or 2x4x8)
Ipp32s block_type = BlParam[mb_num*  6 + b_num].m0;

// do iDCT, normal decoding, full resolution
```

Example 16-13 ippiDCT8x4x2To4x4Inv_DV_16s_C1I Usage (continued)

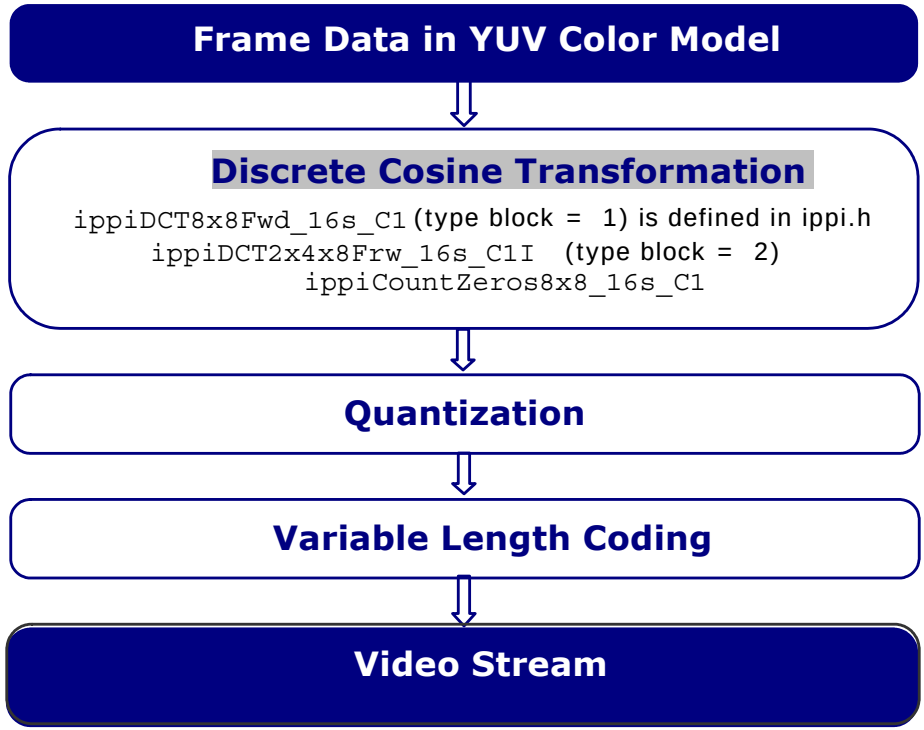
```
if (0 == block_type)
    ippiDCT8x8Inv_16s_C1I((short* ) (lpsBlocks));
else
    ippiDCT2x4x8Inv_16s_C1I((short* ) (lpsBlocks));
…

//If you want to get half resolution (1/2), use these functions:

// do iDCT, half (1/2)resolution
    if (0 == block_type)
        //ippiDCT8x8Inv_16s_C1I((short* ) (lpsBlocks));
        ippiDCT8x8To4x4Inv_16s_C1I((short* ) (lpsBlocks));
        // function ippiDCT8x8To4x4Inv_16s_C1I() defined in ippi domain,
and not ippvc!
    else
        //ippiDCT8x8Inv_16s_C1I((short* ) (lpsBlocks));
        ippiDCT8x4x2To4x4Inv_DV_16s_C1I ((short* ) (lpsBlocks));

    //After that, you increase performance and get half resolution
picture.
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when *pSrcDst* is NULL.

DV Encoding Functions

Figure 16-39 DV Encoding Pipeline



Discrete Cosine Transformation

DCT2x4x8Frw

Performs DCT for a block of type 2.

Syntax

```
IppStatus ippiDCT2x4x8Frw_16s_C1I(Ipp16s* pSrcDst);
```

Parameters

pSrcDst	Pointer to the block.

Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT2x4x8Frw_16s_C1I` performs the DCT for block of type 2. Formula for DCT is as follows:

$$X(h,\ u) = c(u)c(h) \sum_{z=0}^{3} \sum_{x=0}^{7} ((P(x,\ 2z) + P(x,\ 2z+1))KC)$$

$$X(h,\ u+4) = c(u)c(h) \sum_{z=0}^{3} \sum_{x=0}^{7} ((P(x,\ 2z) - P(x,\ 2z+1))KC),$$

where

h in [0, 7]

u in [0, 7]

$z = \mathrm{int}(\frac{y}{2})$

$$KC = \cos\left(\frac{\pi u(2z+1)}{8}\right) \cos\left(\frac{\pi h(2x+1)}{16}\right)$$

$$c(h) = \frac{0.5}{\mathrm{sqr}(2)} \qquad \text{for } h = 0,$$

$c(h) = 0.5$ for h in [1, 7],

$c(u) = \frac{0.5}{\mathrm{sqr}(2)}$ for $u = 0$,

$c(u) = 0.5$ for u in [1, 7].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when *pSrcDst* is `NULL`.

CountZeros8x8

Evaluates number of zeros in a block.

Syntax

```
IppStatus ippiCountZeros8x8_16s_C1(Ipp16s* pSrc, Ipp32u* pCount);
```

Parameters

pSrc	Pointer to the block.
pCount	Pointer to the number of zeros.

Description

This function is declared in the `ippvc.h` header file. The function `ippiCountZeros8x8_16s_C1` is used to identify the type of a block by comparing the results from DCT of the first type and DCT of the second type performed on the block.

See Example 16-14, where `pBlock` is the pointer to the current block and `WeightTableType0` represents tables of the weight coefficients for discrete cosine transform (see [SMPTE 314M-2005] Chapters 5.2 DCT Mode and 5.2.2 Weighting). The coefficients are real numbers that are multiplied by 32767 and reduced to short type through scaling by 2^{15}.

Example 16-14Usage of `ippiCountZeros8x8`

```
Ipp16s sTmp[64];
    Ipp32u countzero, countzero1;
```

Example 16-14Usage of `ippiCountZeros8x8` (continued)

```
    memcpy(sTmp, pBlock->m_psData, 64*  sizeof(Ipp16s));
    ippiDCT8x8Fwd_16s_C1I(pBlock->m_psData);
    ippsMul_16s_ISfs(WeightTableType0, pBlock->m_psData, 64, 16);

    ippiDCT2x4x8Frw_16s_C1I(sTmp);
    ippsMul_16s_ISfs(WeightTableType1, sTmp, 64, 16);

    ippiCountZeros8x8_16s_C1((Ipp16s* )(pBlock->m_psData),
&countzero);
    ippiCountZeros8x8_16s_C1((Ipp16s* )sTmp, &countzero1);
    if (countzero >= countzero1)
    {
        pBlock->m_cM0 = 0;
    }
    else
    {
        pBlock->m_cM0 = 1;
        memcpy(pBlock->m_psData, sTmp, 64*  sizeof(Ipp16s));
    }
```

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one of the pointers is `NULL`.

DV Color Conversion Functions

These functions convert color format in DV video processing.

The minimal coding unit in DV is a set of five macroblocks, with each macroblock comprising four Y-blocks, one Cb block, and one Cr block. The macroblocks are taken from different parts of the processed frame to decrease loss of quality in case the film is damaged. The data format for a DV frame is

- YUV411 in system 525, or NTSC,
- interlaced YUV420 in system 625, or PAL.

These formats are converted into a more popular YUY2 format after inverse discrete cosine transformation.

The functions described in this subsection are used in the DV decoder included into Intel IPP Samples. See introduction to DV section.

YCrCb411ToYCbCr422_5MBDV, YCrCb411ToYCbCr422_ZoomOut2_5MBDV, YCrCb411ToYCbCr422_ZoomOut4_5MBDV, YCrCb411ToYCbCr422_ZoomOut8_5MBDV,

Convert five YCrCb411 macroblocks into YCrCb422 macroblocks.

Syntax

```
IppStatus ippiYCrCb411ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5],
    Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5]);
```

Parameters

pSrc	Array of pointers to the five source macroblocks.
pDst	Array of pointers to the five destination macroblocks.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

Description

These functions are declared in the `ippvc.h` header file. The functions convert YCrCb411 macroblocks to YCbCr422 macroblocks, that is, they transfer the five macroblocks of uncompressed DV data after IDCT processing from the internal format YUV411 to the destination buffer in YUY2 format with saturation.

The function `ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times, `ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R` - by 4 times, and `ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R` - by 8 times.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.

YCrCb411ToYCbCr422_16x4x5MB_DV
YCrCb411ToYCbCr422_8x8MB_DV

Convert five reduced YCrCb411 macroblocks into YCrCb422 reduced macroblocks.

Syntax

```
IppStatus ippiYCrCb411ToYCbCr422_16x4x5MB_DV_16s8u_P3C2R(const Ipp16s* pSrc[5],
    Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb411ToYCbCr422_8x8MB_DV_16s8u_P3C2R(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstStep);
```

Parameters

pSrc	Array of pointers to the five reduced source macroblocks. For the 8x8MB function, *pSrc* is a single pointer to the reduced source macroblock.
pDst	Array of pointers to the five destination macroblocks. For the 8x8MB function, *pDst* is a single pointer to the destination macroblock.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

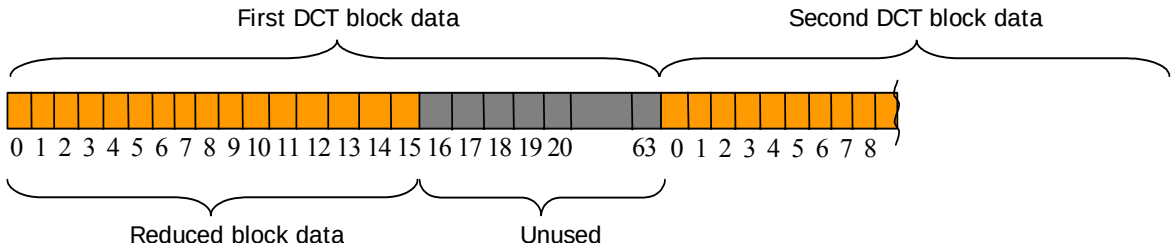
Description

These functions are declared in the ippvc.h header file. The functions convert reduced (quarter resolution) YCrCb411 macroblocks to YCbCr422 macroblocks of the same size. Specifically, the functions transfer the five reduced macroblocks of uncompressed DV data after IDCT processing and block size reduction from the internal format YUV411 to the destination buffer in YUY2 format with saturation.

The function `ippiYCrCb411ToYCbCr422_16x4x5MB_DV_16s8u_P3C2R` converts macroblocks of 16x4 size, and `ippiYCrCb411ToYCbCr422_8x8MB_DV_16s8u_P3C2R` converts a macroblock of 8x8 size located at the right edge of the destination image. The latter function is intended for NTSC standard conversion since all right edge macroblocks have slightly different shapes.

Note that each pointer in the *pSrc* array refers to the uncompressed macroblock data, which consists of 6 uncompressed DCT block data areas. Each DCT block data follows one after another and occupies 64 elements (128 bytes). Reduced uncompressed blocks of size 4x4 occupy first 16 elements of each DCT block data area. See Figure 16-40 for the layout of the reduced blocks in the macroblock data.

Figure 16-40 Layout of Reduced DCT Blocks in Macroblock Data



Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one of the pointers is `NULL`.

YCrCb411ToYCbCr422_EdgeDV, YCrCb411ToYCbCr422_ZoomOut2_EdgeDV, YCrCb411ToYCbCr422_ZoomOut4_EdgeDV, YCrCb411ToYCbCr422_ZoomOut8_EdgeDV

Convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image.

Syntax

```
IppStatus ippiYCrCb411ToYCbCr422_EdgeDV_16s8u_P3C2R(const Ipp16s* pSrc, Ipp8u*
    pDst, int dstStep);
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstStep);
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV_16s8u_P3C2R(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstStep);
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstStep);
```

Parameters

pSrc	Pointer to the source macroblock.
pDst	Pointer to the destination macroblock.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

Description

These functions are declared in the `ippvc.h` header file. They convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image. These functions are intended for NTSC standard conversion as all right edge macroblocks in NTSC have slightly different locations.

The function `ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times, `ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV_16s8u_P3C2R` - by 4 times, and `ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R` - by 8 times.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.

YCrCb420ToYCbCr422_5MBDV, YCrCb420ToYCbCr422_ZoomOut2_5MBDV, YCrCb420ToYCbCr422_ZoomOut4_5MBDV, YCrCb420ToYCbCr422_ZoomOut8_5MBDV

Convert five YCrCb420 macroblocks into YCrCb422 macroblocks.

Syntax

```
IppStatus ippiYCrCb420ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5],
    Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5], int dstStep);
```

Parameters

pSrc	Array of pointers to the five source macroblocks.
pDst	Array of pointers to the five destination macroblocks.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

Description

These functions are declared in the ippvc.h header file. The functions convert YCrCb420 macroblocks to YCbCr422 macroblocks, that is, they transfer the five macroblocks of uncompressed DV data after IDCT processing from the internal format YUV420 (PAL format) to the destination buffer in YUY2 format with saturation.

The function `ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times, `ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R` - by 4 times, and `ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R` - by 8 times.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one of the pointers is `NULL`.

YCrCb420ToYCbCr422_8x8x5MB_DV

Converts five reduced YCrCb420 macroblocks into YCrCb422 reduced macroblocks.

Syntax

```
IppStatus ippiYCrCb420ToYCbCr422_8x8x5MB_DV_16s8u_P3C2R(const Ipp16s* pSrc[5],
    Ipp8u* pDst[5], int dstStep);
```

Parameters

pSrc	Array of pointers to the five reduced source macroblocks.
pDst	Array of pointers to the five destination macroblocks.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

Description

This function is declared in the `ippvc.h` header file. The function converts reduced (quarter resolution) YCrCb420 macroblocks of 8x8 size to YCbCr422 macroblocks of the same size. Specifically, the function transfers the five reduced macroblocks of uncompressed DV data after IDCT processing and block size reduction from the internal format YUV420 to the destination buffer in YUY2 format with saturation.

Note that each pointer in the *pSrc* array refers to the uncompressed macroblock data, which consists of 6 uncompressed DCT block data areas. Each DCT block data follows one after another and occupies 64 elements (128 bytes). Reduced uncompressed blocks of size 4x4 occupy first 16 elements of each DCT block data area. See Figure 16-40 for the layout of the reduced blocks in the macroblock data.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.

YCrCb422ToYCbCr422_5MBDV, YCrCb422ToYCbCr422_ZoomOut2_5MBDV, YCrCb422ToYCbCr422_ZoomOut4_5MBDV, YCrCb422ToYCbCr422_ZoomOut8_5MBDV

Convert five YCrCb422 macroblocks into YCrCb422 macroblocks.

Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5],
    Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5]);
```

Parameters

pSrc	Array of pointers to the five source macroblocks.
pDst	Array of pointers to the five destination macroblocks.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

Description

These functions are declared in the ippvc.h header file. They convert YCrCb422 macroblocks to YCbCr422 macroblocks. These functions may be used in DV50 decoders.

The function ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R also reduces the size of the destination image by 2 times,
ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R - by 4 times, and
ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R - by 8 times.

Note a specific feature of decoding in this case. In spite of the fact that the input data format is YUV422, the *pSrc* pointer should point to the macroblocks that contain the six blocks in the following order:

`[Y0 block][empty block][Y1 block][empty block][V block][U block].`

See below for `ippiYCrCb422ToYCbCr422_5MBDV_16s8u_P3C2R` usage example.

Example 16-15 `ippiYCrCb422ToYCbCr422_5MBDV` Usage

```
...

const Ipp32s lMacroBlockWidth422 = 8*  2;
const Ipp32s lMacroBlockHeight422 = 8;
const Ipp32s lSuperBlockWidth422 = (lMacroBlockWidth422*  9);
const Ipp32s lSuperBlockHeight422 = (lMacroBlockHeight422*  3);

...
{
    // Storing DV50 decompressed video segment

    // i is the DIF sequence number (from 0 to 19 in a NTSC mode or
    // from 0 to 23 in a PAL mode)
    // k is the compressed video segment number in the DIF sequence (from 0
    // to 26)
    // NumDIFSeq is a number of DIF sequences in a channel (10 in NTSC or
    // 12 in PAL mode)
    // Pitch is the destination image step
    // lpDstImage is the (Ipp8u*) pointer to the destination image buffer
    // lpsBlocks is the (Ipp16s*) pointer to the array of 30 decompressed DCT
    // blocks

    Ipp32s lRow, lCol, lBytesPerPixel;
    Ipp16s* lpSrc[5];
    Ipp8u * lpDst[5];

    // set source pointers
    lpSrc[0] = lpsBlocks + (6*  64*  0);
    lpSrc[1] = lpsBlocks + (6*  64*  1);
    lpSrc[2] = lpsBlocks + (6*  64*  2);
    lpSrc[3] = lpsBlocks + (6*  64*  3);
    lpSrc[4] = lpsBlocks + (6*  64*  4);

    // get current column
    lCol = k / 3;

    // get current row
    if (lCol & 0x01)
        lRow = 2 - (k % 3);
    else
        lRow = (k % 3);
```

Example 16-15 ippiYCrCb422ToYCbCr422_5MBDV Usage

```
// set destination pointers
// we use the following formula:
// destination image buffer +
//     required super block row offset +
//     required super block column offset +
//     required macro block row offset +
//     required macro block column offset
lBytesPerPixel = 2; // We use YUV2 destination format
lpDst[0] = lpDstImage +
               (i / NumDIFSeq)*  Pitch*  lSuperBlockHeight422 +
               ((i + 2) % NumDIFSeq)*  Pitch*  lSuperBlockHeight422*  2 +
               Pitch*  lMacroBlockHeight422*  lRow +
               ((2)*  lSuperBlockWidth422 +
               lMacroBlockWidth422*  lCol)*  lBytesPerPixel;
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.

YCrCb422ToYCbCr422_8x4x5MB_DV

Converts five reduced YCrCb422 macroblocks into YCrCb422 reduced macroblocks.

Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_8x4x5MB_DV_16s8u_P3C2R(const Ipp16s* pSrc[5],
    Ipp8u* pDst[5], int dstStep);
```

Parameters

pSrc	Array of pointers to the five reduced source macroblocks.
pDst	Array of pointers to the five destination macroblocks.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

Description

This function is declared in the `ippvc.h` header file. The function converts reduced (quarter resolution) YCrCb422 macroblocks of 8x4 size to YCbCr422 macroblocks of the same size. Specifically, the function transfers the five reduced macroblocks of uncompressed DV data after IDCT processing and block size reduction from the internal format YUV422 to the destination buffer in YUY2 format with saturation.

Note that each pointer in the *pSrc* array refers to the uncompressed macroblock data, which consists of 6 uncompressed DCT block data areas. Each DCT block data follows one after another and occupies 64 elements (128 bytes) but the second and the fourth areas are empty and not used. The block data are arranged in the following order:

`[Y0 block][empty block][Y1 block][empty block][V block][U block].`

Reduced uncompressed blocks of size 4x4 occupy first 16 elements of corresponding DCT block data area. See Figure 16-40 for the layout of the reduced blocks in the macroblock data.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one of the pointers is `NULL`.

YCrCb422ToYCbCr422_10HalvesMB16x8_DV100

Converts ten YCrCb422 half-macroblocks into YCrCb422 half-macroblocks.

Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100_16s8u_P3C2R(const Ipp16s*
    pSrc, Ipp8u* pDst[10], Ipp32s dstStep);
```

Parameters

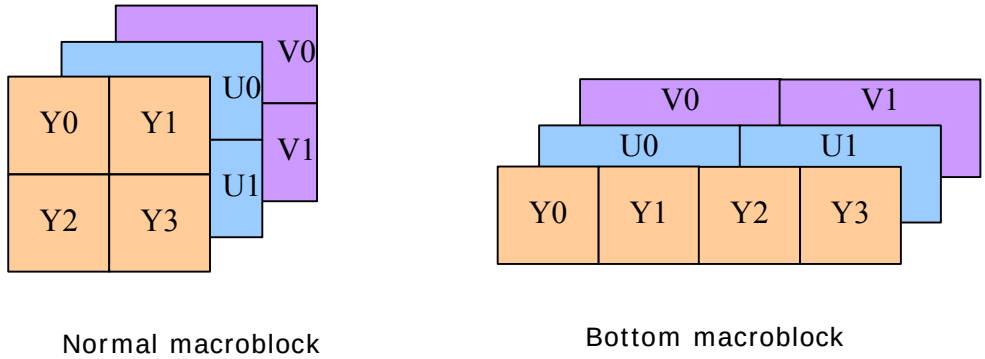
pSrc	Pointer to decompressed DV100 video segment consisting of five macroblocks.
pDst	Array of pointers to ten destinations, where half-macroblocks should be stored.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.

Description

This function is declared in the `ippvc.h` header file. The `ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100_16s8u_P3C2R` function is intended to be used in DV100 video decoder. It converts ten YCrCb422 half-macroblocks of size 16x8 to YCbCr422 half-macroblocks of the same size. Specifically, the function transfers macroblocks of uncompressed DV data after IDCT processing from the internal format (YUV422) to the destination buffer in YUY2 format with saturation.

There are two types of macro blocks in DV100 standard: normal and bottom macroblock. Figure 16-41 depicts these two types. A bottom macroblock is obtained from a normal macroblock by moving its bottom half to the right.

Figure 16-41 DV 100 Macroblocks



The `ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100_16s8u_P3C2R` function can process both types of macroblocks by specifying ten destination pointers that are passed to the function through *pDst* argument. Each pair of pointers (even and odd) is the places on the destination image where two halves of the corresponding macroblock should be stored. For normal macroblock, the odd pointer should refer to the point 8 pixels below the point the even pointer refers to. For the bottom macroblock, the odd pointer should refer to the point 16 pixels to the right of the point the even pointer refers to.

pSrc is the pointer to uncompressed DV100 video segment, which consists of 40 DCT blocks following one after another: 4 luma and 4 color difference DCT blocks for each of 5 macro blocks. Each DCT block consists of 64 elements. DCT blocks in each macroblock are arranged in the following order:

`Y0, Y1, Y2, Y3, Cr0, Cr1, Cb0, Cb1.`

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one of the pointers is `NULL`.

MPEG-4

This section contains functions for encoding and decoding of video data according to ISO/IEC 14496-2 MPEG-4 standard (see [ISO14496A]).

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

Table 16-16 MPEG-4 Video Decoder Functions

Function Short Name	Description
Motion Compensation	
Copy8x8QP_MPEG4, Copy16x8QP_MPEG4, Copy16x16QP_MPEG4	Copy a block with quarter-pixel accuracy.
OBMC8x8HP_MPEG4, OBMC16x16HP_MPEG4, OBMC8x8QP_MPEG4,	Perform the overlapped block motion compensation (OBMC).
Sprite and Global Motion Compensation	
WarpInit_MPEG4	Initializes IppiWarpSpec_MPEG4 structure for further usage in GMC or Sprite reconstruction.
WarpGetSize_MPEG4	Returns size of IppiWarpSpec_MPEG4 structure.
WarpLuma_MPEG4	Warps arbitrary rectangular luminance region.
WarpChroma_MPEG4	Warps arbitrary rectangular chrominance region.
CalcGlobalMV_MPEG4	Calculates Global Motion Vector for one macroblock.
ChangeSpriteBrightness_MPEG4	Change brightness after sprite warping.
Inverse Quantization	
QuantInvIntraInit_MPEG4, QuantInvInterInit_MPEG4	Initialize specification structures for QuantInvIntra_MPEG4 and QuantInvInter_MPEG4 respectively.
QuantInvIntraGetSize_MPEG4, QuantInvInterGetSize_MPEG4	Return the size of the initialized specification structures.
QuantInvIntra_MPEG4, QuantInvInter_MPEG4	Perform inverse quantization on intra/inter coded block.

Table 16-16 MPEG-4 Video Decoder Functions (continued)

Function Short Name	Description
VLC Decoding	
DecodeDCIntra_MPEG4	Decodes one DC coefficient for intra coded block.
DecodeCoeffsIntra_MPEG4	Decodes DCT coefficients for intra coded block.
DecodeCoeffsIntraRVLCBack_MPEG4	Decodes DCT coefficients in backward direction for intra coded block using RVLC.
DecodeCoeffsInter_MPEG4	Decodes DCT coefficients for inter coded block.
DecodeCoeffsInterRVLCBack_MPEG4	Decodes DCT coefficients in backward direction for inter coded block using RVLC.
ReconstructCoeffsInter_MPEG4	Decodes DCT coefficients, performs inverse scan and inverse quantization for inter coded block.
Postprocessing	
FilterDeblocking8x8HorEdge_MPEG4, FilterDeblocking8x8VerEdge_MPEG4	Perform deblocking filtering on a horizontal or vertical edge of two adjacent blocks.
FilterDeringingThreshold_MPEG4	Computes threshold values for the deringing filtering through a macroblock.
FilterDeringingSmooth8x8_MPEG4	Performs deringing filtering of a block.

Table 16-17 MPEG-4 Video Encoder Functions

Function Short Name	Description
Quantization	
QuantIntraInit_MPEG4, QuantInterInit_MPEG4	Initialize specification structures for QuantIntra_MPEG4 and QuantInter_MPEG4 respectively.
QuantIntraGetSize_MPEG4, QuantInterGetSize_MPEG4	Return the size of the initialized specification structures.
QuantIntra_MPEG4, QuantInter_MPEG4	Perform quantization on intra/inter coded block.
VLC Encoding	
EncodeDCIntra_MPEG4	Encodes one DC coefficient for intra coded block.
EncodeCoeffsIntra_MPEG4	Encodes DCT coefficients for intra coded block.
EncodeCoeffsInter_MPEG4	Encodes DCT coefficients for inter coded block.

MPEG-4 Video Decoder Functions

This section describes Intel IPP functions that are built to support the ISO/IEC 14496-2 MPEG-4 video decoder. MPEG-4 (see [ISO14496]) is a widely used coding method for video signals in various applications such as digital storage media, Internet, various forms of wired or wireless communications, etc.

The functions, along with those described in the General Functions section, cover the following aspects of MPEG-4 decoder:

- block-based variable length code (VLC) decoding and inverse zigzag scan
- Inverse quantization, inverse zigzag positioning, and IDCT
- Block layer coefficient reconstruction, including bitstream parsing, VLC decoding, inverse quantization, inverse zigzag positioning, with appropriate clipping on each step
- overlapped block motion compensation
- motion compensation
- postprocessing for coding noise reduction including deblocking and deringing filters.

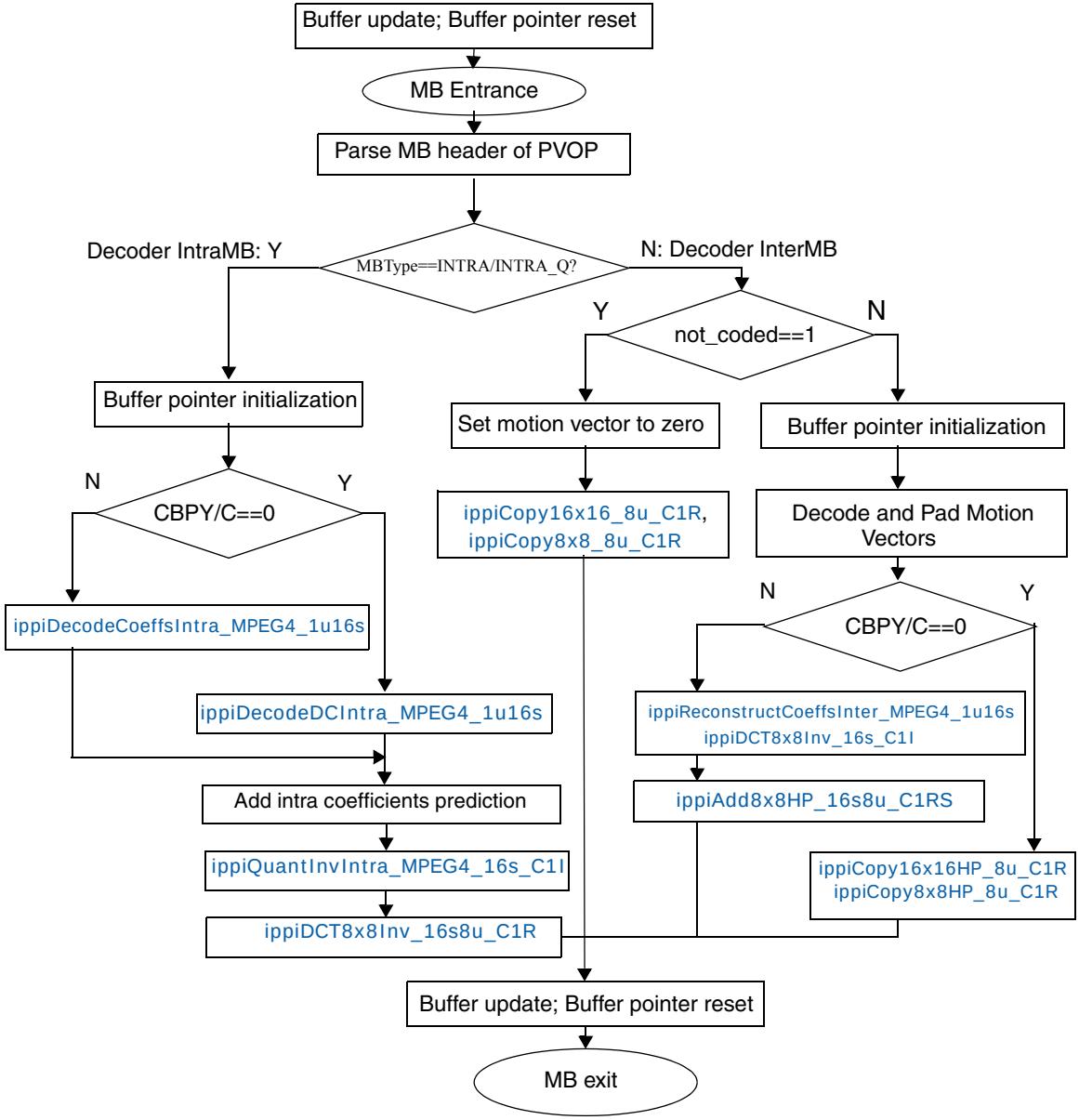
NOTE. All MPEG-4 functions can be applied if `short_video_header` equals 0. If `short_video_header` equals 1, use H.263 decoder functions.

Below a high-level description of the MPEG-4 functions usage is given, followed by Intel IPP macro/data structures definition and the detailed descriptions of individual functions.

High Level Description

Figure 16-42 shows the general steps that are needed to decode a MacroBlock (MB) in Predictive coded Video Object Plane (P-VOP) and the associated Intel IPP functions.

Figure 16-42 Flowchart Decoding an MB in P-VOP



Data Types and Structures

Rectangle Plane

The following structure is defined in Intel IPP to represent a rectangle:

```
typedef structure _IppiRect {
    int        x;
    int        y;
    int        width;
    int        height;
}
```

Block Type

The following enumeration indicates the block type:

```
enum {
    IPPVC_BLOCK_LUMA,
    IPPVC_BLOCK_CHROMA
};
```

Sprite Type

The following enumeration indicates the sprite type:

```
enum {
    IPPVC_SPRITE_STATIC = 1,
    IPPVC_SPRITE_GMC = 2
};
```

Motion Compensation

Copy8x8QP_MPEG4, Copy16x8QP_MPEG4, Copy16x16QP_MPEG4

Copy fixed size blocks with quarter-pixel accuracy.

Syntax

```
IppStatus ippiCopy8x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, int acc, int rounding);
IppStatus ippiCopy16x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, int acc, int rounding);
IppStatus ippiCopy16x16QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, int acc, int rounding);
```

Parameters

pSrc	Pointer to the source block.
srcStep	Distance in bytes between starts of the consecutive lines in the source image.
pDst	Pointer to the destination block.
dstStep	Distance in bytes between starts of the consecutive lines in the destination image.
acc	Parameter that determines quarter-pixel accuracy.
rounding	Parameter that determines type of rounding for pixel approximation; may be 0 or 1.

Description

The functions `ippiCopy8x8QP_MPEG4_8u_C1R`, `ippiCopy16x8QP_MPEG4_8u_C1R`, and`ippiCopy16x16QP_MPEG4_8u_C1R` are declared in the `ippvc.h` file. These functions copy the source block to the destination block with half-pixel accuracy. Parameter rounding has the same meaning as `vop_rounding_type` in [ISO14496]. Parameter *acc* is a four-bit value, the bits 0-1 define quarter-pixel offset in horizontal direction and bits 2-3 define quarter-pixel offset in vertical direction. The process of quarter-pixel interpolation is described in [ISO14496] subclause 7.6.2.2.

These functions are used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

OBMC8x8HP_MPEG4, OBMC16x16HP_MPEG4, OBMC8x8QP_MPEG4,

Perform overlapped block motion compensation (OBMC) for one luminance block.

Syntax

```
IppStatus ippiOBMC8x8HP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, const IppMotionVector* pMVCur, const IppMotionVector*
    pMVLeft, const IppMotionVector* pMVRight, const IppMotionVector* pMVAbove,
    const IppMotionVector* pMVBelow, int rounding);
IppStatus ippiOBMC16x16HP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, const IppMotionVector* pMVCur, const IppMotionVector*
    pMVLeft, const IppMotionVector* pMVRight, const IppMotionVector* pMVAbove,
    const IppMotionVector* pMVBelow, int rounding);
IppStatus ippiOBMC8x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, const IppMotionVector* pMVCur, const IppMotionVector*
    pMVLeft, const IppMotionVector* pMVRight, const IppMotionVector* pMVAbove,
    const IppMotionVector* pMVBelow, int rounding);
```

Parameters

pSrc	Pointer to the source blockcoallocated with the destination block.
srcStep	Width in bytes of the source plane.
pDst	Pointer to the destination block.
dstStep	Width in bytes of the destination plane.
pMVCur	Pointer to the motion vector for the current block.
pMVLeft	Pointer to the motion vector for the left block.

pMVRight	Pointer to the motion vector for the right block.
pMVAbove	Pointer to the motion vector for the above block.
pMVBelow	Pointer to the motion vector for the below block.
rounding	Parameter that determines type of rounding for half pixel approximation; may be 0 or 1.

Description

The functions `ippiOBMC8x8HP_MPEG4_8u_C1R`, `ippiOBMC16x16HP_MPEG4_8u_C1R`, and `ippiOBMC8x8QP_MPEG4_8u_C1R` are declared in the `ippvc.h` file. The functions `ippiOBMC8x8HP_MPEG4_8u_C1R` and `ippiOBMC16x16HP_MPEG4_8u_C1R` perform the overlapped block motion compensation of the *pSrc* with the half-pixel accuracy ([ISO14496], subclause 7.6.2.1) and `ippiOBMC8x8QP_MPEG4_8u_C1R` - with quarter-pixel accuracy ([ISO14496], subclause 7.6.2.2) and store the results in the destination block *pDst*. Overlapped motion compensation is specified in [ISO14496], subclause 7.6.6. `ippiOBMC16x16HP_MPEG4_8u_C1R` should be used for overlapped motion compensation in Reduced Resolution VOPs as specified in [ISO14496], subclause 7.6.10.

The function `ippiOBMC8x8HP_MPEG4_8u_C1R` is used in the H.263 and MPEG-4 encoders and decoders included into Intel IPP Samples. The function `ippiOBMC8x8QP_MPEG4_8u_C1R` is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

Sprite and Global Motion Compensation

Functions for sprite and Global Motion Compensation (GMC) are used in decoding processes of static sprites and in decoding/encoding processes of Video Object Layers with GMC.

WarpInit_MPEG4

Initializes `IppiWarpSpec_MPEG4` *structure for further usage in GMC or Sprite reconstruction.*

Syntax

```
IppStatus ippiWarpInit_MPEG4(IppiWarpSpec_MPEG4* pSpec, const int* pDU, const
    int* pDV, int numWarpingPoints, int spriteType, int warpingAccuracy, int
    roundingType, int quarterSample, int fcode, const IppiRect* pSpriteRect,
    const IppiRect* pVopRect);
```

Parameters

pSpec	Pointer to `IppiWarpSpec_MPEG4` structure.
pDU	Pointer to an array of x-coordinate of warping points.
pDV	Pointer to an array of y-coordinate of warping points.
numWarpingPoints	Number of warping points, valid in [0-4].
spriteType	Indicates a sprite coding mode, takes values `IPPVC_SPRITE_STATIC`, indicating static sprites `IPPVC_SPRITE_GMC`, indicating GMC.
warpingAccuracy	Accuracy of warping, valid in [0-3].
roundingType	Parameter that determines type of rounding for pixel approximation; may be 0 or 1.
quarterSample	Parameter that indicates a quarter sample mode; may be 0 or 1.
fcode	Parameter that determines the range of motion vector, valid in the range [1,7].
pSpriteRect	Parameter that determines rectangle region for Sprite (or reference VOP for GMC).
pVopRect	Parameter that determines rectangle region for current VOP.

Description

The function `ippiWarpInit_MPEG4` is declared in the `ippvc.h` file. This function initializes `IppiWarpSpec_MPEG4structure` for further usage in GMC or Sprite reconstruction. The meaning of parameters is described in [ISO14496] subclause 7.8.

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsSizeErr`	Indicates no error when width or height of images is less than or equal to zero.
`ippStsOutOfRangeErr`	Indicates an error when *`numWarpingPoints`*, *`warpingAccuracy`*, or *`fcode`* are out of the valid range.

WarpGetSize_MPEG4

Returns size of `IppiWarpSpec_MPEG4` *structure.*

Syntax

```
IppStatus ippiWarpGetSize_MPEG4(int* pSpecSize);
```

Parameters

`pSpecSize`	Pointer to the resulting size of the structure `IppiWarpSpec_MPEG4`.

Description

The function `ippiWarpGetSize_MPEG4` is declared in the `ippvc.h` file. This function returns a size of structure `IppiWarpSpec_MPEG4`.

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pointer *pSpecSize* is NULL.

WarpLuma_MPEG4

Warps arbitrary rectangular luminance region.

Syntax

```
IppStatus ippiWarpLuma_MPEG4_8u_C1R(const Ipp8u* pSrcY, int srcStepY, Ipp8u*
    pDstY, int dstStepY, const IppiRect* pDstRect, const IppiWarpSpec_MPEG4*
    pSpec);
```

Parameters

pSrcY	Pointer to the origin of the source plane.
srcStep	Distance in bytes between starts of the consecutive lines in the destination plane.
pDst	Pointer to the destination region.
dstStep	Distance in bytes between starts of the consecutive lines in the destination plane.
pDstRect	Rectangular destination region.
pSpec	Pointer to the structure with motion parameters.

Description

The function ippiWarpLuma_MPEG4_8u_C1R is declared in the ippvc.h file. This function warps an arbitrary rectangular area using motion information from IppiWarpSpec_MPEG4 structure. This function may be used in GMC or Sprite reconstruction ([ISO14496] subclause 7.8).

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

WarpChroma_MPEG4

Warps arbitrary rectangular chrominance region.

Syntax

```
IppStatus ippiWarpChroma_MPEG4_8u_P2R(const Ipp8u* pSrcCb, int srcStepCb, const
    Ipp8u* pSrcCr, int srcStepCr, Ipp8u* pDstCb, int dstStepCb, Ipp8u* pDstCr,
    int dstStepCr, const IppiRect* pDstRect, const IppiWarpSpec_MPEG4* pSpec);
```

Parameters

pSrcCb	Pointer to the origin of the first source plane.
srcStepCb	Distance in bytes between starts of the consecutive lines in the first source plane.
pSrcCr	Pointer to the origin of the second source plane.
srcStepCr	Distance in bytes between starts of the consecutive lines in the second source plane.
pDstCb	Pointer to the first destination plane.
dstStepCb	Distance in bytes between starts of the consecutive lines in the first destination plane.
pDstCr	Pointer to the second destination plane.
dstStepCr	Distance in bytes between starts of the consecutive lines in the second destination plane.
pDstRect	Rectangular destination region.
pSpec	Pointer to the structure with motion parameters.

Description

The function `ippiWarpChroma_MPEG4_8u_P2R` is declared in the `ippvc.h` file. This function warps an arbitrary rectangular chrominance area using motion information from `IppiWarpSpec_MPEG4` structure. This function may be used in GMC or Sprite reconstruction ([ISO14496] subclause 7.8).

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

CalcGlobalMV_MPEG4

Calculates Global Motion Vector for one macroblock.

Syntax

```
IppStatus ippiCalcGlobalMV_MPEG4(int xOffset, int yOffset, IppMotionVector*
    pGMV, const IppiWarpSpec_MPEG4* pSpec);
```

Parameters

xOffset	The left coordinate of top-left corner of luma 16x16 block.
yOffset	The top coordinate of top-left corner of luma 16x16 block.
pGMV	Pointer to the resulting motion vector.
pSpec	Pointer to the structure with motion parameters.

Description

The function ippiCalcGlobalMV_MPEG4 is declared in the ippvc.h file. This function calculates a global motion vector for a macroblock and clips it to lie in range $[-2^{fcode+4};2^{fcode+4}-1]$ as it described in [ISO14496] subclause 7.8.7.3.

This function is used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

ChangeSpriteBrightness_MPEG4

Change brightness after sprite warping.

Syntax

```
IppStatus ippiChangeSpriteBrightness_MPEG4_8u_C1IR(const Ipp8u* pSrcDst, int
    srcDstStep, int width, int height, int brightnessChangeFactor);
```

Parameters

pSrcDst	Pointer to the video plane.
srcDstStep	Distance in bytes between starts of the consecutive lines in the video plane.
width	The width of the video plane.
height	The height of the video plane.
brightnessChangeFactor	Factor for changing brightness; valid in the range [-112; 1648].

Description

The function `ippiChangeSpriteBrightness_MPEG4_8u_C1IR` is declared in the `ippvc.h` file. This function changes brightness of video plane after sprite warping ([ISO14496] subclause 7.8).

This function is used in the MPEG-4 decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsOutOfRangeErr`	Indicates an error when *brightnessChangeFactor* is out of the valid range.

Inverse Quantization

QuantInvIntraInit_MPEG4, QuantInvInterInit_MPEG4

Initialize specification structures.

Syntax

```
IppStatus ippiQuantInvIntraInit_MPEG4(const Ipp8u* pQuantMatrix,
    IppiQuantInvIntraSpec_MPEG4* pSpec, int bitsPerPixel);
IppStatus ippiQuantInvInterInit_MPEG4(const Ipp8u* pQuantMatrix,
    IppiQuantInvInterSpec_MPEG4* pSpec, int bitsPerPixel);
```

Parameters

pQuantMatrix	Pointer to quantization matrix size of 64.
pSpec	Pointer to the initialized specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4`.
bitsPerPixel	Video data precision used for saturation of the result. Valid within the range [4, 12].

Description

The functions `ippiQuantInvIntraInit_MPEG4` and `ippiQuantInvInterInit_MPEG4` are declared in the `ippvc.h` file. These functions initialize specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4`. If *pQuantMatrix* is `NULL`, the second quantization method is used; otherwise, the first method is used.

These functions are used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error pointer *pSpec* is `NULL`.
`ippStsOutOfRangeErr`	Indicates an error condition when *bitsPerPixel* is out of the range [4, 12].

QuantInvIntraGetSize_MPEG4, QuantInvInterGetSize_MPEG4

Return size of specification structures.

Syntax

```
IppStatus ippiQuantInvIntraGetSize_MPEG4(int* pSpecSize);
IppStatus ippiQuantInvInterGetSize_MPEG4(int* pSpecSize);
```

Parameters

pSpecSize	Pointer to the resulting size of the initialized specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4`.

Description

The functions `ippiQuantInvIntraGetSize_MPEG4` and `ippiQuantInvInterGetSize_MPEG4` are declared in the `ippvc.h` file. These functions return a size of specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4`.

These functions are used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error pointer *pSpecSize* is `NULL`.

QuantInvIntra_MPEG4, QuantInvInter_MPEG4

Perform inverse quantization on intra/inter coded block.

Syntax

```
IppStatus ippiQuantInvIntra_MPEG4_16s_C1I(Ipp16s* pCoeffs, int indxLastNonZero,
    const IppiQuantIntraSpec_MPEG4* pSpec, int QP, int blockType);
IppStatus ippiQuantInvInter_MPEG4_16s_C1I(Ipp16s* pCoeffs, int indxLastNonZero,
    const IppiQuantInterSpec_MPEG4* pSpec, int QP);
```

Parameters

pCoeffs	Pointer to the quantized DCT coefficients.
indxLastNonZero	Index of the last non-zero coefficient. This parameter provides faster operation. If the value is unknown, set to 63.
pSpec	Pointer to the initialized specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4` initialized by `ippiQuantInvIntraInit_MPEG4` or `ippiQuantInvInterInit_MPEG4` respectively.
QP	Quantization parameter.
blockType	Indicates the block type. Takes one of the following values: `IPPVC_BLOCK_LUMA` - for lume and alpha blocks, `IPPVC_BLOCK_CHROMA` - for chroma blocks.
pSrcDst	Pointer to the coefficient buffer of intra/inter block. Contains quantized coefficients (QF) on input and dequantized coefficients (F) on output.
pQPMatrix	Pointer to the quantization weighting matrix. If *pQPMatrix* is `NULL`, this function will use the second inverse quantization method; otherwise, it will use the first method.
videoComp	(Intra version only) Video component type of the current block. It is used for DC coefficient dequantizing. Takes one of the following flags: `IPP_VIDEO_LUMINANCE`, `IPP_VIDEO_CHROMINANCE`, `IPP_VIDEO_ALPHA`.

Description

The functions `ippiQuantInvIntra_MPEG4_16s_C1I` and `ippiQuantInvInter_MPEG4_16s_C1I` are declared in the `ippvc.h` file. These functions perform inverse quantization on intra/inter coded block, saturation and mismatch control (for the first inverse quantization method only) as it is specified in [ISO14496], subclause 7.4.4.

For `ippiQuantInvIntra_MPEG4_16s_C1I` and `ippiQuantInvInter_MPEG4_16s_C1I` output coefficients are saturated to lie in range $[-2^{bitsPerPixel+3}; 2^{bitsPerPixel+3} - 1]$.

These functions are used in the MPEG-4 encoder and decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one pointer is `NULL`.
`ippStsQPErr`	Indicates an error condition if *QP* is out of the range $[1, 2^{bitsPerPixel-3} - 1]$.

VLC Decoding

DecodeDCIntra_MPEG4

Decodes one DC coefficient for intra coded block.

Syntax

```
IppStatus ippiDecodeDCIntra_MPEG4_1u16s(Ipp8u** ppBitStream, int* pBitOffset,
    Ipp16s* pDC, int blockType);
```

Parameters

ppBitStream	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.

pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. The pointer is updated by the function.
pDC	Pointer to the output coefficient.
blockType	Indicates the block type, takes one of the following values: `IPPVC_BLOCK_LUMA` - for luma and alpha blocks, `IPPVC_BLOCK_CHROMA` - for chroma blocks.

Description

The function `ippiDecodeDCIntra_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding of the DC coefficient only for one intra coded block using VLC decoding process as specified in [ISO14496], subclause 7.4.1.1.

This function is used in the MPEG-4 decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
`ippStsVLCCodeErr`	Indicates an error condition if an illegal code is detected through the DC decoding.

DecodeCoeffsIntra_MPEG4

Decodes DCT coefficients for intra coded block.

Syntax

```
IppStatus ippiDecodeCoeffsIntra_MPEG4_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoeffs, int* pIndxLastNonZero, int rvlcFlag, int
    noDCFlag, int scan);
```

Parameters

ppBitStream	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.

pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. The pointer is updated by the function.
pCoeffs	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.
rvlcFlag	Flag, when set to '0' indicates that VLC tables B.16, B.18, B.19, and B.21 [ISO14496] are used in decoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
noDCFlag	Flag, when set to '0' indicates that *pCoeffs* is set starting with zero element, otherwise - with the first element.
scan	Type of the scan, takes one of the following values: `IPPVC_SCAN_NONE`, indicating that no inverse scan is performed, `IPPVC_SCAN_ZIGZAG`, indicating the classical zigzag scan. `IPPVC_SCAN_HORIZONTAL`, indicating the alternate-horizontal scan, `IPPVC_SCAN_VERTICAL`, indicating the alternate-vertical scan, See the corresponding enumerator on p.16-3.

Description

The function `ippiDecodeCoeffsIntra_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding of the DC coefficient (if *noDCFlag* is 0) and AC coefficients for one intra coded block using VLC decoding process as specified in [ISO14496], subclause 7.4.1. Additionally, the function can perform inverse scan.

This function is used in the MPEG-4 decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
`ippStsVLCCodeErr`	Indicates an error condition if an illegal code is detected through the decoding.

DecodeCoeffsIntraRVLCBack_MPEG4

Decodes DCT coefficients in backward direction for intra coded block using RVLC.

Syntax

```
IppStatus ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoeffs, int* pIndxLastNonZero, int noDCFlag);
```

Parameters

ppBitStream	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. The pointer is updated by the function.
pCoeffs	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.
noDCFlag	Flag, when set to ‘0’ indicates that *pCoeffs* is set starting with zero element, otherwise - with the first element.

Description

The function `ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs backward decoding of the DC coefficient (if *noDCFlag* is 0) and AC coefficients using variable length tables B.23, B.24, and B.25 [ISO14496] for one intra coded block. The backward RVLC decoding process is specified in [ISO14496], subclauses E.1.3 and E.1.4.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
`ippStsVLCCodeErr`	Indicates an error condition if an illegal code is detected through the decoding.

DecodeCoeffsInter_MPEG4

Decodes DCT coefficients for inter coded block.

Syntax

```
IppStatus ippiDecodeCoeffsInter_MPEG4_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoeffs, int* pIndxLastNonZero, int rvlcFlagint, int
    scan);
```

Parameters

ppBitStream	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. The pointer is updated by the function.
pCoeffs	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.
rvlcFlag	Flag, when set to ‘0’ indicates that VLC tables B.17, B.18, B.19, and B.21 [ISO14496] are used in decoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
scan	Type of the scan, takes one of the following values: `IPPVC_SCAN_NONE`, indicating that no inverse scan is performed, `IPPVC_SCAN_ZIGZAG`, indicating the classical zigzag scan. `IPPVC_SCAN_VERTICAL`, indicating the alternate-vertical scan, See the corresponding enumerator on p.16-3.

Description

The function `ippiDecodeCoefInter_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding of the DCT coefficients for one inter coded block. Inter DCT VLC decoding process is specified in [ISO14496], subclause 7.4.1. Additionally, the function can perform inverse scan.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
ippStsVLCCodeErr	Indicates an error condition if an illegal code is detected through the DCT decoding.

DecodeCoeffsInterRVLCBack_MPEG4

Decodes DCT coefficients in backward direction for inter coded block using RVLC.

Syntax

```
IppStatus ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoeffs, int* pIndxLastNonZero);
```

Parameters

ppBitStream	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. The pointer is updated by the function.
pCoeffs	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.

Description

The function `ippiDecodeCoeffsInterRVLCBack_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs backward decoding of the DC coefficient coefficients using variable length tables B.23, B.24 and B.25 [ISO14496] for one inter coded block. The backward RVLC decoding process is specified in [ISO14496], subclauses E.1.3 and E.1.4.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.
ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].

ippStsVLCCodeErr	Indicates an error condition if an illegal code is detected through the decoding.

ReconstructCoeffsInter_MPEG4

Decodes DCT coefficients, performs inverse scan and inverse quantization for inter coded block.

Syntax

```
IppStatus ippiReconstructCoeffsInter_MPEG4_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoeffs, int* pIndxLastNonZero, int rvlcFlag, int scan,
    const IppiQuantInvInterSpec_MPEG4* pQuantInvInterSpec, int QP);
```

Parameters

ppBitStream	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. The pointer is updated by the function.
pCoeffs	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in this parameter.
rvlcFlag	Flag, which when set to ‘0’ indicates that VLC tables B.17, B.18, B.20 and B.22 [ISO14496] are used when decoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24 and B.25 [ISO14496] are used.
scan	Type of the inverse scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan. See the corresponding enumerator on p.16-3.
pQuantInvInterSpec	Pointer to the IppiQuantInvInterSpec_MPEG4 that was initialized by ippiQuantInvInterInit_MPEG4.
QP	Quantization parameter

Description

The function `ippiReconstructCoeffsInter_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding, inverse scan and inverse quantization of the DCT coefficients for one inter coded block. Inter DCT VLC decoding process is specified in [ISO14496], subclause 7.4.1. Inverse scan process is specified in [ISO14496], subclause 7.4.2. Inverse quantization process is specified in [ISO14496], subclause 7.4.4. This function supports video data precision 4-12 bits per pixel, that is, after inverse quantization the coefficients are saturated to lie in the range: $[-2^{bitsPerPixel+3}; 2^{bitsPerPixel+3} - 1]$.

This function is used in the MPEG-4 decoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer (except *pQuantMatrix*) is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0; 7].
`ippStsVLCCodeErr`	Indicates an error condition if an illegal code is detected through the DCT decoding.
`ippStsQPErr`	Indicates an error condition if QP is out of the range $[1; 2^{bitsPerPixel-3} - 1]$.

Postprocessing

FilterDeblocking8x8HorEdge_MPEG4, FilterDeblocking8x8VerEdge_MPEG4

Perform deblocking filtering on a horizontal or vertical edge of two adjacent blocks.

Syntax

```
IppStatus ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR(Ipp8u* pSrcDst, int
    step, int QP, int THR1, int THR2);
```

```
IppStatus ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR(Ipp8u* pSrcDst, int
    step, int QP, int THR1, int THR2);
```

Parameters

pSrcDst	Pointer to the first pixel of lower (HorEdge) or right (VerEdge) block.
step	Width in bytes of the source plane.
QP	Quantization parameter.
THR1, THR2	Threshold values that specify the filter mode ([ISO14496] Annex F.3.1).

Description

The functions ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR and ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR are declared in the ippvc.h file. These functions perform deblocking filtering of two adjacent blocks on horizontal and vertical edges, respectively ([ISO14496], Annex F.3.1).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if the pointer *pSrcDst* is NULL.
ippStsMP4QPErr	Indicates an error condition if *QP* is out of range [1, 31]

FilterDeringingThreshold_MPEG4

Computes threshold values for the deringing filtering through a macroblock.

Syntax

```
IppStatus ippiFilterDeringingThreshold_MPEG4_8u_P3R(Ipp8u* pSrcY, int stepY,
    const Ipp8u* pSrcCb, int stepCb, const Ipp8u* pSrcCr, int stepCr, int
    threshold[6]);
```

Parameters

pSrcY	Pointer to the left upper Y-block in the current macroblock.
stepY	Width in bytes of the luminance plane.

pSrcCb	Pointer to the Cb-block in the current macroblock.
stepCb	Width in bytes of the chrominance (Cb) plane.
pSrcCr	Pointer to the Cr-block in the current macroblock.
stepCr	Width in bytes of the chrominance (Cr) plane.
threshold	Array of the threshold values for all blocks.

Description

The function `ippiFilterDeringingThreshold_MPEG4_8u` is declared in the `ippvc.h` file. This function performs threshold determination ([ISO14496], Annex F.3.2.1), which is the first subprocess of the deringing filtering. The obtained threshold values are required for the function `FilterDeringingSmooth8x8_MPEG4`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

FilterDeringingSmooth8x8_MPEG4

Performs deringing filtering of a block.

Syntax

```
IppStatus ippiFilterDeringingSmooth8x8_MPEG4_8u_C1R(Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, int QP, int threshold);
```

Parameters

pSrc	Pointer to the source block.
srcStep	Width in bytes of the source plane.
QP	Quantization parameter (`quantiser_scale`).
threshold	Threshold value for the block.
pDst	Pointer to the destination block.
dstStep	Width in bytes of the destination plane.

Description

The function `ippiFilterDeringingSmooth8x8_MPEG4_8u` is declared in the `ippvc.h` file. This function performs deringing filtering, specifically, index acquisition and adaptive smoothing ([ISO14496], Annex F.3.2.1, 3.2.2) of the source block *pSrc*, and stores the result in the destination block *pDst*. The threshold value *threshold* is returned by the auxiliary function `FilterDeringingThreshold_MPEG4` that should be called beforehand.

Return Values

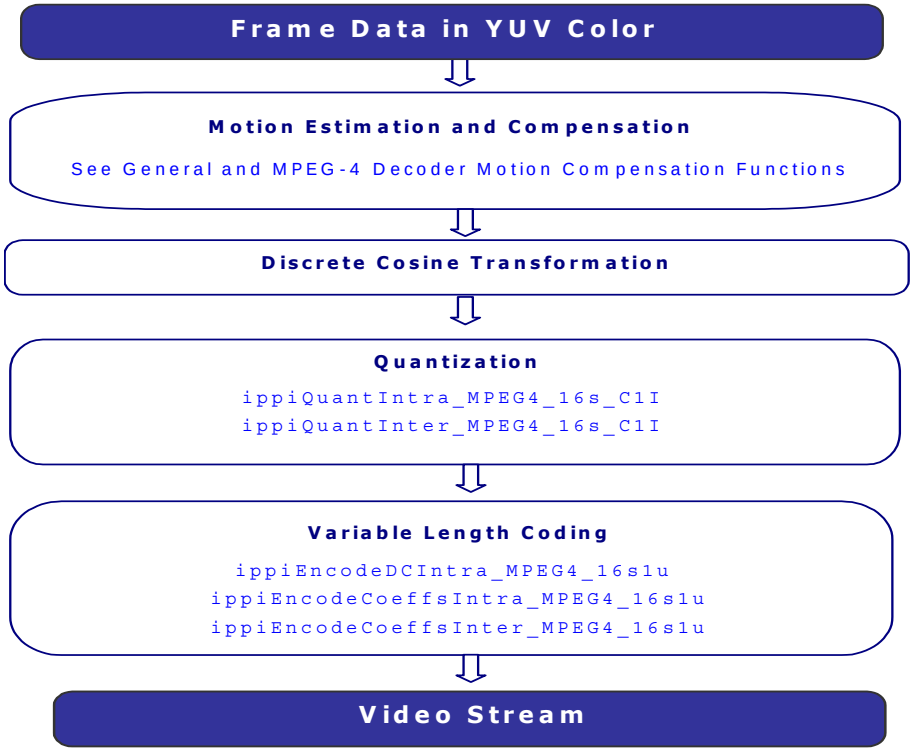
`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsMP4QPErr`	Indicates an error condition if *QP* is out of range [1, 31]

MPEG-4 Video Encoder Functions

This section describes the main steps of MPEG-4 ([ISO14496]) video encoding in accordance with the pipeline shown in Figure 16-43.

Figure 16-43 MPEG-4 Encoding Pipeline

Figure 16-43 MPEG-4 Encoding Pipeline



MPEG-4 is a widely used coding method for video signals in various applications such as digital storage media, internet, various forms of wired or wireless communications, etc.

Data Types and Structures

The basic data types and structures for MPEG-4 encoder are the same as described above for the IPP MPEG-4 decoder. See the corresponding subsections in MPEG-4 Decoder Data Types and Structures.

Quantization

QuantIntraInit_MPEG4, QuantInterInit_MPEG4

Initialize specification structures.

Syntax

```
IppStatus ippiQuantIntraInit_MPEG4(const Ipp8u* pQuantMatrix,
    IppiQuantIntraSpec_MPEG4* pSpec, int bitsPerPixel);
IppStatus ippiQuantInterInit_MPEG4(const Ipp8u* pQuantMatrix,
    IppiQuantInterSpec_MPEG4* pSpec, int bitsPerPixel);
```

Parameters

pQuantMatrix	Pointer to quantization matrix size of 64.
pSpec	Pointer to the initialized specification structure `IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4`.
bitsPerPixel	Video data precision used for saturation of the result. Valid within the range [4, 12].

Description

The functions `ippiQuantIntraInit_MPEG4` and `ippiQuantInterInit_MPEG4` are declared in the `ippvc.h` file. These functions initialize specification structure `IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4`. If *pQuantMatrix* is `NULL`, the second quantization method is used; otherwise, the first method is used.

These functions are used in the MPEG-4 encoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error pointer *pSpec* is `NULL`.
`ippStsOutOfRangeErr`	Indicates an error condition when *bitsPerPixel* is out of the range [4, 12].

QuantIntraGetSize_MPEG4, QuantInterGetSize_MPEG4

Return size of specification structures.

Syntax

```
IppStatus ippiQuantIntraGetSize_MPEG4(int* pSpecSize);
IppStatus ippiQuantInterGetSize_MPEG4(int* pSpecSize);
```

Parameters

pSpecSize	Pointer to the resulting size of the initialized specification structure `IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4`.

Description

The functions `ippiQuantIntraGetSize_MPEG4` and `ippiQuantInterGetSize_MPEG4` are declared in the `ippvc.h` file. These functions return a size of specification structure `IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4`.

These functions are used in the MPEG-4 encoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error pointer *pSpecSize* is `NULL`.

QuantIntra_MPEG4, QuantInter_MPEG4

Perform inverse quantization on intra/inter coded block.

Syntax

```
IppStatus ippiQuantIntra_MPEG4_16s_C1I(Ipp16s* pCoeffs, const
    IppiQuantIntraSpec_MPEG4* pSpec, int QP, int* pCountNonZero, int blockType);
```

```
IppStatus ippiQuantIntra_MPEG4_16s_C1I(Ipp16s*  pSrcDst, Ipp8u QP, int
    blockIndex, const int*  pQPMatrix);
```

Parameters

pCoeffs	Pointer to the quantized DCT coefficients.
pSpec	Pointer to the initialized specification structure `IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4` initialized by `ippiQuantIntraInit_MPEG4` or `ippiQuantInterInit_MPEG4` respectively.
QP	Quantization parameter.
pCountNonZero	Pointer to the count of non-zero coefficients.
blockType	Indicates the block type. Takes one of the following values: `IPPVC_BLOCK_LUMA` - for lume and alpha blocks, `IPPVC_BLOCK_CHROMA` - for chroma blocks.
pSrcDst	Pointer to the input intra block coefficients; as an output argument points to the quantized intra block coefficients.
blockIndex	Block index indicating the component type and position as defined in subclause 6.1.3.8 of [ISO14496A]. Furthermore, indexes 6 to 9 indicate the alpha blocks spatially corresponding to luminance blocks 0 to 3 in the same macroblock.
pQPMatrix	The pointer, which is NULL, if the second inverse quantization method is used. If the first inverse quantization method is used, it points to the quantization weighting coefficient's buffer (for inter MB) whose first 64 elements are the quantization weighting matrix in Q0. The second 64 elements are their reciprocals in Q21.

Description

The functions `ippiQuantIntra_MPEG4_16s_C1I` and `ippiQuantInter_MPEG4_16s_C1I` are declared in the `ippvc.h` file. These functions perform quantization on intra/inter coded block.

For `ippiQuantIntra_MPEG4_16s_C1I` and `ippiQuantInter_MPEG4_16s_C1I` output coefficients are saturated to lie in range [-2047; 2047]. Also these functions calculate the number of non-zero coefficients after quantization.

These functions are used in the MPEG-4 encoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if at least one pointer is NULL.
ippStsQPErr	Indicates an error condition if *QP* is less than or equal to 0.
ippStsBadArgErr	Indicates bad arguments.

VLC Encoding

EncodeDCIntra_MPEG4

Encodes one DC coefficient for intra coded block.

Syntax

```
IppStatus ippiEncodeDCIntra_MPEG4_16s1u(Ipp16s DC, Ipp8u** ppBitStream, int*
    pBitOffset, int blockType);
```

Parameters

DC	DC coefficient to be encoded.
ppBitStream	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. The pointer is updated by the function.
blockType	Indicates the block type, takes one of the following values: IPPVC_BLOCK_LUMA - for luma and alpha blocks, IPPVC_BLOCK_CHROMA - for chroma blocks.

Description

The function ippiEncodeDCIntra_MPEG4_16s1u is declared in the ippvc.h header file. This function performs VLC encoding of the DC coefficient only for one intra coded block using tables B.13, B14, B.15 specified in [ISO14496].

This function is used in the MPEG-4 encoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.
ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].

EncodeCoeffsIntra_MPEG4

Encodes DCT coefficients for intra coded block.

Syntax

```
IppStatus ippiEncodeCoeffsIntra_MPEG4_16s1u(const Ipp16s* pCoeffs, Ipp8u**
    ppBitStream, int* pBitOffset, int countNonZero, int rvlcFlag, int noDCFlag,
    int scan);
```

Parameters

pCoeffs	Pointer to the input coefficients.
ppBitStream	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. The pointer is updated by the function.
countNonZero	Number of non-zero coefficients.
rvlcFlag	Flag, when set to ‘0’ indicates that VLC tables B.16, B.18, B.19, and B.21 [ISO14496] are used in encoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
noDCFlag	Flag, when set to ‘0’ indicates that *pCoeffs* is set starting with zero element, otherwise - with the first element.
scan	Type of the scan, takes one of the following values: IPPVC_SCAN_NONE, indicating that no scan is performed, IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan. IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan, See the corresponding enumerator on p.16-3.

Description

The function `ippiEncodeCoeffsIntra_MPEG4_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the DC coefficient (if *noDCFlag* is 0) and AC coefficients for one intra coded block in scan order defined by *scan*.

This function is used in the MPEG-4 encoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].

EncodeCoeffsInter_MPEG4

Encodes DCT coefficients for inter coded block.

Syntax

```
IppStatus ippiEncodeCoeffsInter_MPEG4_16s1u(const Ipp16s* pCoeffs, Ipp8u**
    ppBitStream, int* pBitOffset, int* countNonZero, int rvlc, int scan);
```

Parameters

pCoeffs	Pointer to the input coefficients.
ppBitStream	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. The pointer is updated by the function.
countNonZero	Number of non-zero coefficients.
rvlc	Flag, when set to ‘0’ indicates that VLC tables B.17, B.18, B.20, and B.22 [ISO14496] are used in encoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
scan	Type of the scan, takes one of the following values:

	IPPVC_SCAN_NONE, indicating that no scan is performed, IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan. IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan, See the corresponding enumerator on p.16-3.

Description

The function ippiEncodeCoefInter_MPEG4_16s1u is declared in the ippvc.h header file. This function performs VLC encoding of the DCT coefficients for one inter coded block in scan order defined by *scan*.

This function is used in the MPEG-4 encoder included into Intel IPP Samples. See introduction to MPEG-4 section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.
ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].

H.261

This section contains functions for encoding and decoding video data according to ITU-T Recommendation H.261 ([ITUH261]).

Table 16-18 H.261 Functions

Function Short Name	Description
Video Decoding Functions	
DecodeCoeffsIntra_H261	Decodes DCT coefficients for intra coded block.
DecodeCoeffsInter_H261	Decodes DCT coefficients for inter coded block.
ReconstructCoeffsIntra_H261	Reconstructs DCT coefficients for intra coded block.
ReconstructCoeffsInter_H261	Reconstructs DCT coefficients for inter coded block.
Video Encoding Functions	
EncodeCoeffsIntra_H261	Encodes and puts quantized DCT coefficients for intra coded block into bitstream.
EncodeCoeffsInter_H261	Encodes and puts quantized DCT coefficients for inter coded block into bitstream.
Filter8x8_H261	Performs two-dimensional filtering on a block.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

H.261 Decoder Functions

This section describes IPP functions that support the decoder and common operations specific to ITU-T Recommendation H.261.

The implemented IPP functions, along with those described in the General Functions section, cover the following aspects of H.261 Decoder:

- Block-based variable length code (VLC) decoding and inverse zigzag scan
- Inverse quantization, inverse zigzag positioning, and IDCT
- Block layer coefficient reconstruction, including bitstream parsing, VLC decoding, inverse quantization, inverse zigzag positioning, with appropriate clipping on each step
- Motion compensation
- Two-dimensional loop filtering.

The following subsections of this section give a high-level description of the H.261 Decoder functions hierarchy, followed by macro/data structures used in the functions and detailed descriptions of individual functions.

Decoding INTRA and INTER Macroblocks

Figure 16-44 INTRA Macroblock Decoding

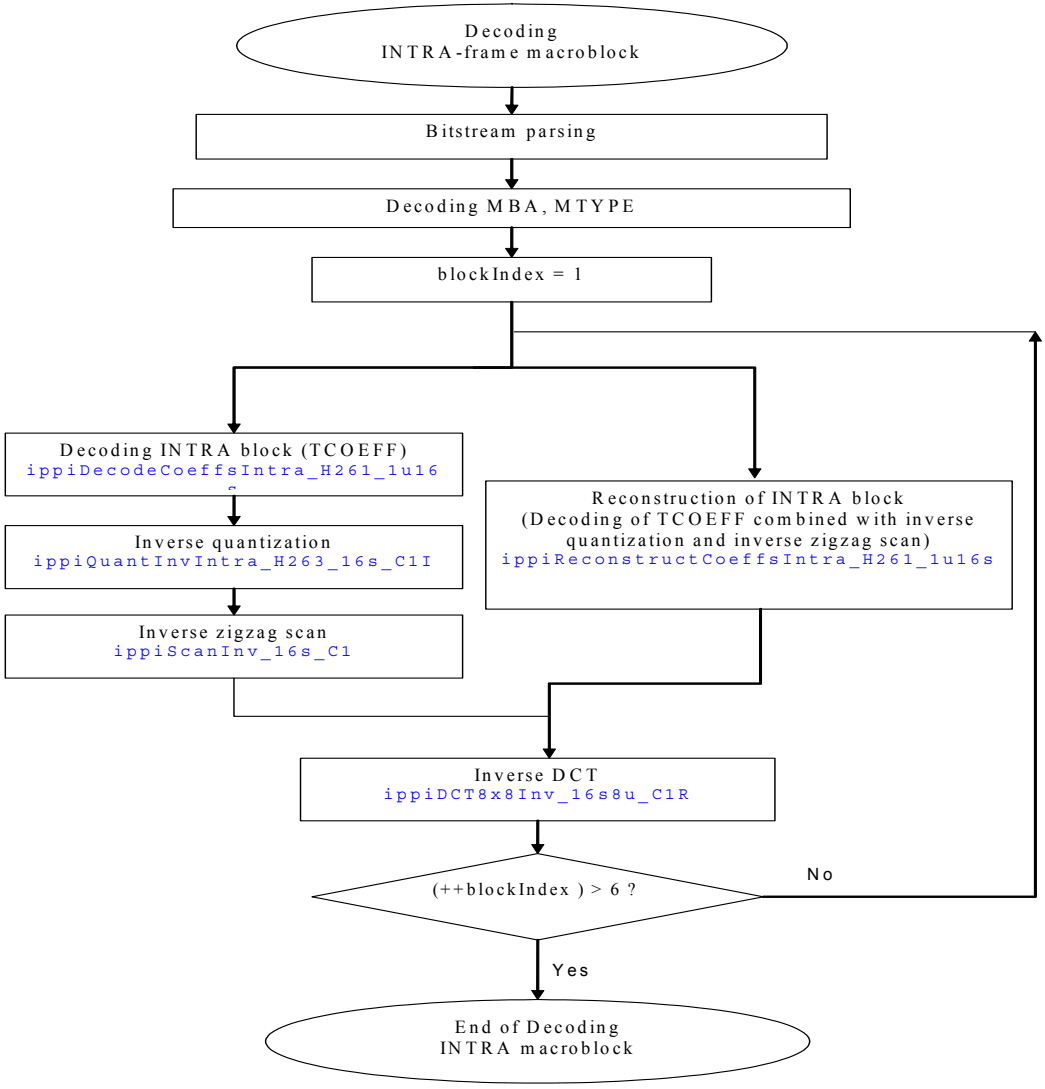
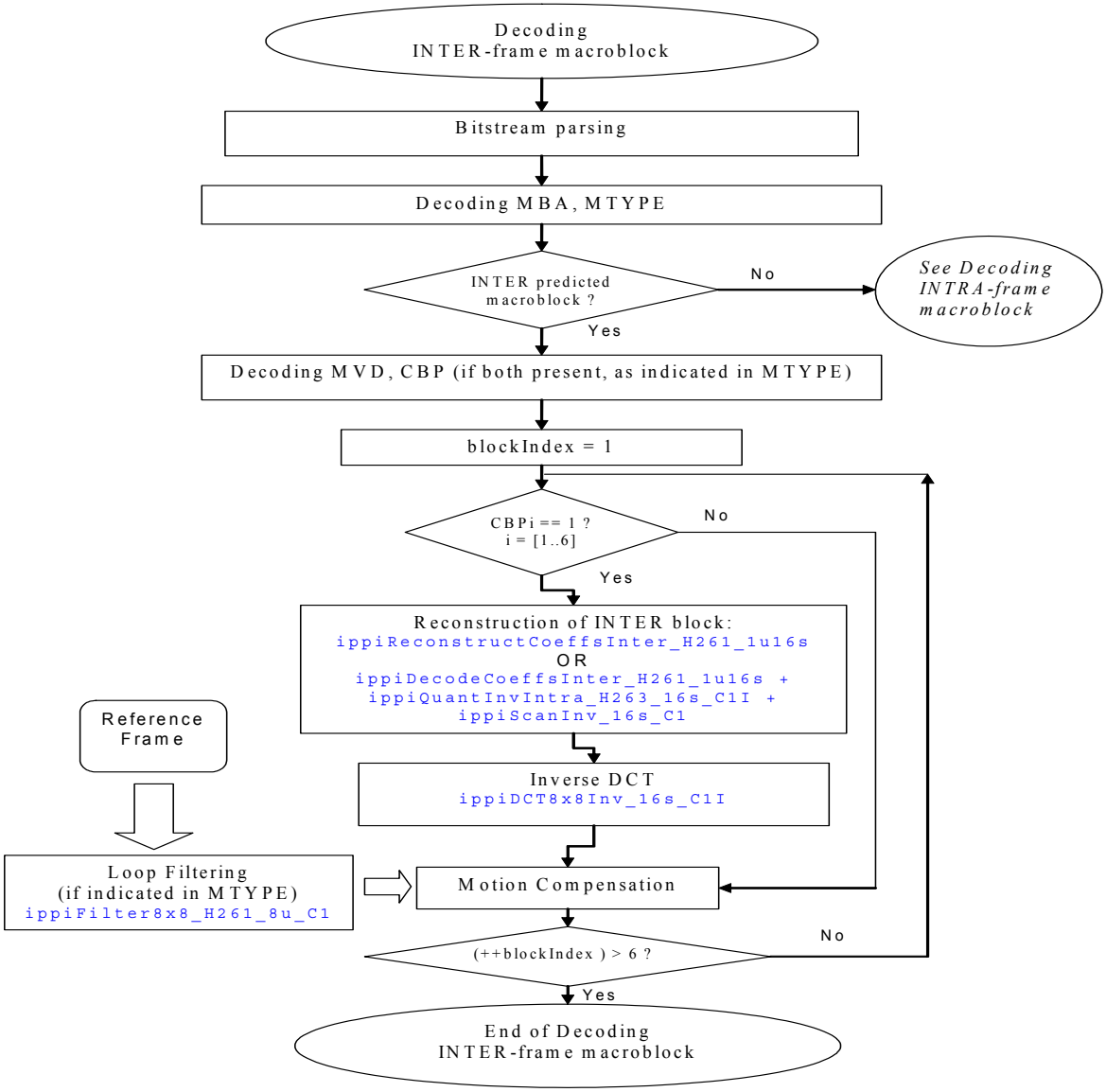


Figure 16-45 shows the process of INTER-frame macroblock decoding.

Figure 16-45 INTER Macroblock Decoding



Structure and Macro Definitions

Analogous to H.263 in the default prediction mode, horizontal and vertical components of a motion vector are integer values in the range [-15, 15]. Frames are never expanded in H.261, as motion vectors are always restricted to a point within the coded picture area.

DecodeCoeffsIntra_H261

Decodes DCT coefficients for intra coded block.

Syntax

```
ippiDecodeCoeffsIntra_H261_1u16s(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s*
    pCoef, int* pIndxLastNonZero, int scan);
```

Parameters

ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.
pCoef	Pointer to the output coefficients. *pCoef*[0] is the DC coefficient.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in* *pIndxLastNonZero*. If the block has no correctly decoded coefficients,* *pIndxLastNonZero* is set to -1.
scan	Type of the inverse scan, takes one of the following values: `IPPVC_SCAN_ZIGZAG`, indicating the classical zigzag scan, `IPPVC_SCAN_NONE`, indicating that no inverse scan is to be performed. See the corresponding enumerator on p.16-3.

Description

The function `ippiDecodeCoeffsIntra_H261_1u16s` is declared in the `ippvc.h` header file. This function performs decoding and, optionally, inverse scan of the DCT coefficients (DC and AC) for one intra coded block. DC fixed length and AC VLC decoding processes are

specified in [ITUH261], subclause 4.2.4.1. If *scan* is not set to IPPVC_SCAN_NONE, the DCT coefficients, encoded in the bitstream in the classical zigzag scan order ([ITUH261], Figure12), are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
ippStsVLCCodeErr	Indicates an error condition if an illegal code is detected through the stream processing.

DecodeCoeffsInter_H261

Decodes DCT coefficients for inter coded block.

Syntax

```
ippiDecodeCoeffsInter_H261_1u16s(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s*
    pCoef, int* pIndxLastNonZero, int scan);
```

Parameters

ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.
pCoef	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in* *pIndxLastNonZero*. If the block has no correctly decoded coefficients,* *pIndxLastNonZero* is set to -1.
scan	Type of the inverse scan, takes one of the following values:

IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no inverse scan is to be performed.

See the corresponding enumerator on p.16-3.

Description

The function ippiDecodeCoeffsInter_H261_1u16s is declared in the ippvc.h header file. This function performs VLC decoding and, optionally, inverse scan of the DCT coefficients for one inter coded block as specified in [ITUH261], subclause 4.2.4.1 (Table 4). If *scan* is not set to IPPVC_SCAN_NONE, the DCT coefficients, encoded in the bitstream in the classical zigzag scan order ([ITUH261], Figure12), are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
ippStsVLCCodeErr	Indicates an error condition if an illegal code is detected through the VLC stream processing.

ReconstructCoeffsIntra_H261

Reconstructs DCT coefficients for intra coded block.

Syntax

```
ippiReconstructCoeffsIntra_H261_1u16s(Ipp8u** ppBitStream, int* pBitOffset,
    Ipp16s* pCoef, int* pIndxLastNonZero, int QP);
```

Parameters

ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.

pCoef	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in* *pIndxLastNonZero*. If the block has no correctly decoded coefficients,* *pIndxLastNonZero* is set to -1.
QP	Quantization parameter.

Description

The function `ippiReconstructCoeffsIntra_H261_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization, and inverse scanning of the DCT coefficients for one intra coded block.

DCT coefficients decoding and dequantization processes are specified in [ITUH261], subclause 4.2.4.1. The DCT coefficients, encoded in the bitstream in classical zigzag scan order ([ITUH261], Figure 12), are reordered in the function into the normal order, that is, the order, in which the coefficients are arranged on DCT output.

This function is used in the H.261 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
`ippStsVLCCodeErr`	Indicates an error condition if an illegal code is detected through the stream processing.
`ippStsQPErr`	Indicates an error condition if *QP* is out of the range [1, 31].

ReconstructCoeffsInter_H261

Reconstructs DCT coefficients for inter coded block.

Syntax

```
ippiReconstructCoeffsInter_H261_1u16s(Ipp8u** ppBitStream, int* pBitOffset,
    Ipp16s* pCoef, int* pIndxLastNonZero, int QP);
```

Parameters

ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.
pCoef	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in* *pIndxLastNonZero*. If the block has no correctly decoded coefficients,* *pIndxLastNonZero* is set to -1.
QP	Quantization parameter.

Description

The function `ippiReconstructCoeffsInter_H261_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization and inverse scanning of the DCT coefficients for one inter coded block.

DCT coefficients decoding and dequantization processes are specified in [ITUH261], subclause 4.2.4.1. The DCT coefficients, encoded in the bitstream in classical zigzag scan order ([ITUH261], Figure 12), are reordered in the function into the normal order, that is, the order, in which the coefficients are arranged on DCT output.

This function is used in the H.261 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

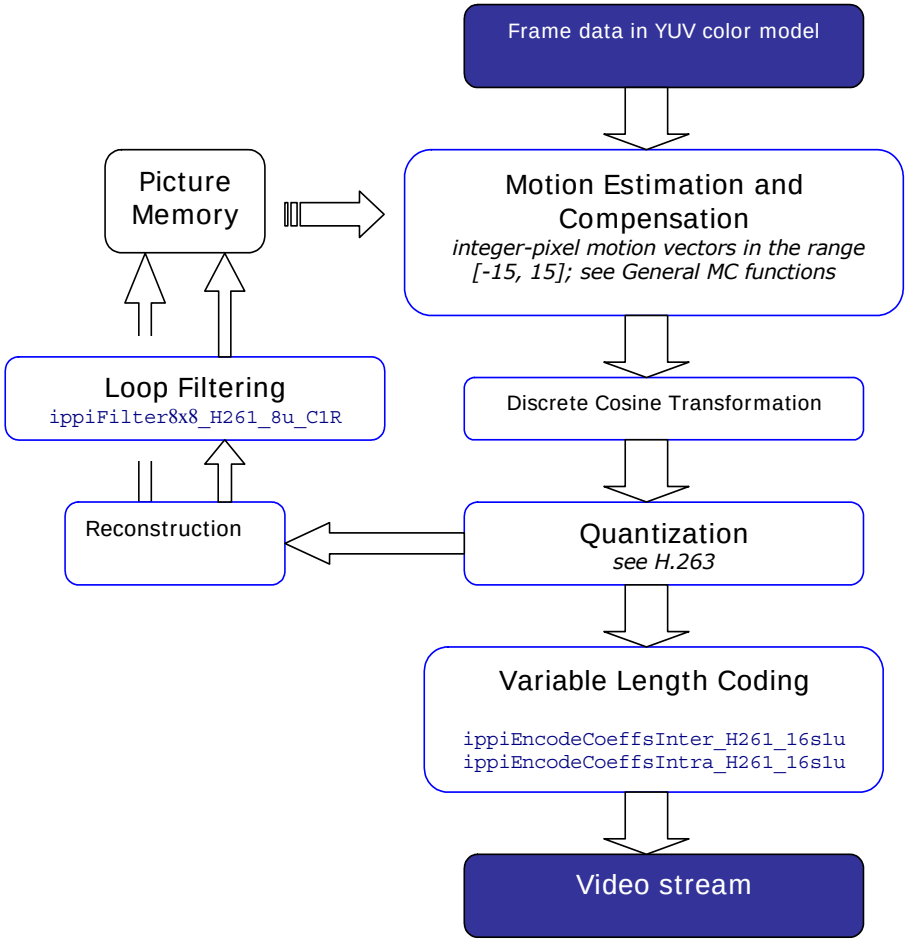
`ippStsNoErr`	Indicates no error.

`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *`pBitOffset`* is out of the range [0, 7].
`ippStsVLCCodeErr`	Indicates an error condition if an illegal code is detected through the stream processing.
`ippStsQPErr`	Indicates an error condition if *`QP`* is out of the range [1, 31].

H.261 Encoder Functions

This section describes the main steps of H.261 ([ITUH261]) video encoding in accordance with the pipeline shown in Figure 16-46:

Figure 16-46 H.261 Encoding Pipeline.



EncodeCoeffsIntra_H261

Encodes and puts quantized DCT coefficients for intra coded block into bitstream.

Syntax

```
IppStatus ippiEncodeCoeffsIntra_H261_16s1u(Ipp16s* pQCoef, Ipp8u** ppBitStream,
    int* pBitOffset, int countNonZero, int scan);
```

Parameters

pQCoef	Pointer to the array of quantized DCT coefficients. *pQCoef*[0] is the DC coefficient.
ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7. * *pBitOffset* is updated by the function.
countNonZero	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
scan	Type of the scan to be performed on the coefficients before encoding, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no scan is to be performed, that is, the input coefficients are already in the scan order. See the corresponding enumerator on p.16-3.

Description

The function ippiEncodeCoeffsIntra_H261_16s1u is declared in the ippvc.h header file. This function performs encoding of the quantized DCT coefficients in a scan order for one intra coded block and puts the codes into the bitstream. DC fixed length and AC VLC encoding processes are specified in [ITUH261], subclause 4.2.4.1.

This function is used in the H.261 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
ippStsOutOfRangeErr	Indicates an error condition if *countNonZero* is out of the range [1, 64].

EncodeCoeffsInter_H261

Encodes and puts quantized DCT coefficients for inter coded block into bitstream.

Syntax

```
IppStatus ippiEncodeCoeffsInter_H261_16s1u(Ipp16s* pQCoef, Ipp8u** ppBitStream,
    int* pBitOffset, int countNonZero, int scan);
```

Parameters

pQCoef	Pointer to the array of quantized DCT coefficients. *pQCoef*[0] is the DC coefficient.
ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7. * *pBitOffset* is updated by the function.
countNonZero	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
scan	Type of the scan to be performed on the coefficients before encoding, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no scan is to be performed, that is, the input coefficients are already in the scan order. See the corresponding enumerator on p.16-3.

Description

The function `ippiEncodeCoeffsInter_H261_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the quantized DCT coefficients in a scan order for one inter coded block and puts the codes into the bitstream. The encoding process is specified in [ITUH261], subclause 4.2.4.1.

This function is used in the H.261 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
`ippStsOutOfRangeErr`	Indicates an error condition if *countNonZero* is out of the range [1, 64].

Filter8x8_H261

Performs two-dimensional filtering on a block.

Syntax

```
ippiFilter8x8_H261_8u_C1R(Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep);
```

Parameters

pSrc	Pointer to the origin of the source block.
srcStep	Width in bytes of the source image plane.
pDst	Pointer to the origin of the destination block.
dstStep	Width in bytes of the destination image plane.

Description

The function `ippiFilter8x8_H261_8u_C1R` is declared in the `ippvc.h` header file. This function performs filtering on an 8x8 block as specified in [ITUH261], subclause 3.2.3. The two-dimensional filter may be used in both encoder and decoder to modify the prediction. The application of the filter to a macroblock (to all six blocks) is signaled in the MTYPE (macroblock type) codeword.

This function is used in the H.261 encoder and decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

H.263

This section contains functions for encoding and decoding of video data according to ITU-T Recommendation H.263 ([ITUH263]) and Annexes, which is most often denoted by the "H263 + decoder" acronym.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

Table 16-19 H.263 Video Decoder Functions

Function Short Name	Description
VLC Decoding	
DecodeDCIntra_H263	Decodes DC coefficient for intra coded block.
DecodeCoeffsIntra_H263	Decodes AC coefficients for intra coded block.
DecodeCoeffsInter_H263	Decodes DCT coefficients for inter coded block.
Quantization	
QuantInvIntra_H263	Performs inverse quantization on an intra coded block stored in a one-dimensional buffer.
QuantInvInter_H263	Performs inverse quantization on an inter coded block stored in a one-dimensional buffer.
Prediction	
AddBackPredPB_H263	Performs bidirectional prediction for a B-block in a PB-frame.
Resampling	
Resample_H263	Performs picture resampling defined in terms of picture area corner displacements.
UpsampleFour_H263	Performs factor-of-4 picture upsampling.
DownsampleFour_H263	Performs factor-of-4 picture downsampling.

Table 16-19 H.263 Video Decoder Functions (continued)

Function Short Name	Description
UpsampleFour8x8_H263	Performs factor-of-4 upsampling on an 8x8 block.
SpatialInterpolation_H263	Interpolates a picture by a factor of two horizontally, vertically, or both horizontally and vertically.
Boundary Filtering	
FilterBlockBoundaryHorEdge_H263, FilterBlockBoundaryVerEdge_H263	Perform block boundary filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.
FilterDeblocking8x8HorEdge_H263, FilterDeblocking8x8VerEdge_H263	Perform deblocking filtering of one block edge on the reconstructed frames.
FilterDeblocking16x16HorEdge_H263, FilterDeblocking16x16VerEdge_H263	Perform deblocking filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.
Middle Level Functions	
ReconstructCoeffsIntra_H263	Reconstructs DCT coefficients for an intra coded block.
ReconstructCoeffsInter_H263	Reconstructs DCT coefficients for an inter coded block.

Table 16-20 H.263 Video Encoder Functions

Function Short Name	Description
VLC Encoding	
EncodeDCIntra_H263	Encodes and puts a quantized DC coefficient for an intra coded block into bitstream.
EncodeCoeffsIntra_H263	Encodes and puts quantized DCT coefficients for an intra coded block into bitstream.
EncodeCoeffsInter_H263	Encodes and puts quantized DCT coefficients for inter coded block into bitstream.
Quantization	
QuantIntra_H263	Performs quantization on an intra coded block.
QuantInter_H263	Performs quantization on an inter coded block.
Resampling	
DownsampleFour16x16_H263	Performs factor-of-4 downsampling on a 16x16 block.

H.263 Decoder Functions

This section describes Intel IPP functions that support general video processing and the decoder part of ITU-T Recommendation H.263 (see ([ITUH263]) and Annexes, which is often denoted by the “H263+ decoder” acronym.

The implemented Intel IPP functions, along with those described in the General Functions and MPEG-4 sections, cover the following aspects of H.263+ Decoder:

- Block-based variable length code (VLC) decoding and inverse zigzag scan
- Inverse quantization, inverse zigzag positioning, and IDCT
- Block layer coefficient reconstruction, including bitstream parsing, VLC decoding, inverse quantization, inverse zigzag positioning, with appropriate clipping on each step
- Motion Compensation
- Unrestricted Motion Vectors mode (Annex D/H.263+)
- Advanced Prediction mode (Annex F/H.263+)
- PB-frames mode (Annex G/H.263+)
- Advanced Intra-Coding mode (Annex I/H.263+)
- Deblocking Filter mode (Annex J/H.263+)
- Temporal, SNR, and Spatial Scalability mode (AnnexO/H.263+)
- Reference Picture Resampling (Annex P/H.263+)
- Reduced-Resolution Update mode (Annex Q/H.263+)
- Alternative INTER VLC mode (Annex S/H.263+)
- Modified Quantization mode (Annex T/H.263+).

The following subsections of this section give a high-level description of the H.263+ Decoder functions hierarchy, followed by macro/data structures used in the functions and the detailed descriptions of individual functions.

INTRA and INTER Macroblocks Decoding

Figure 16-47 INTRA Macroblock Decoding

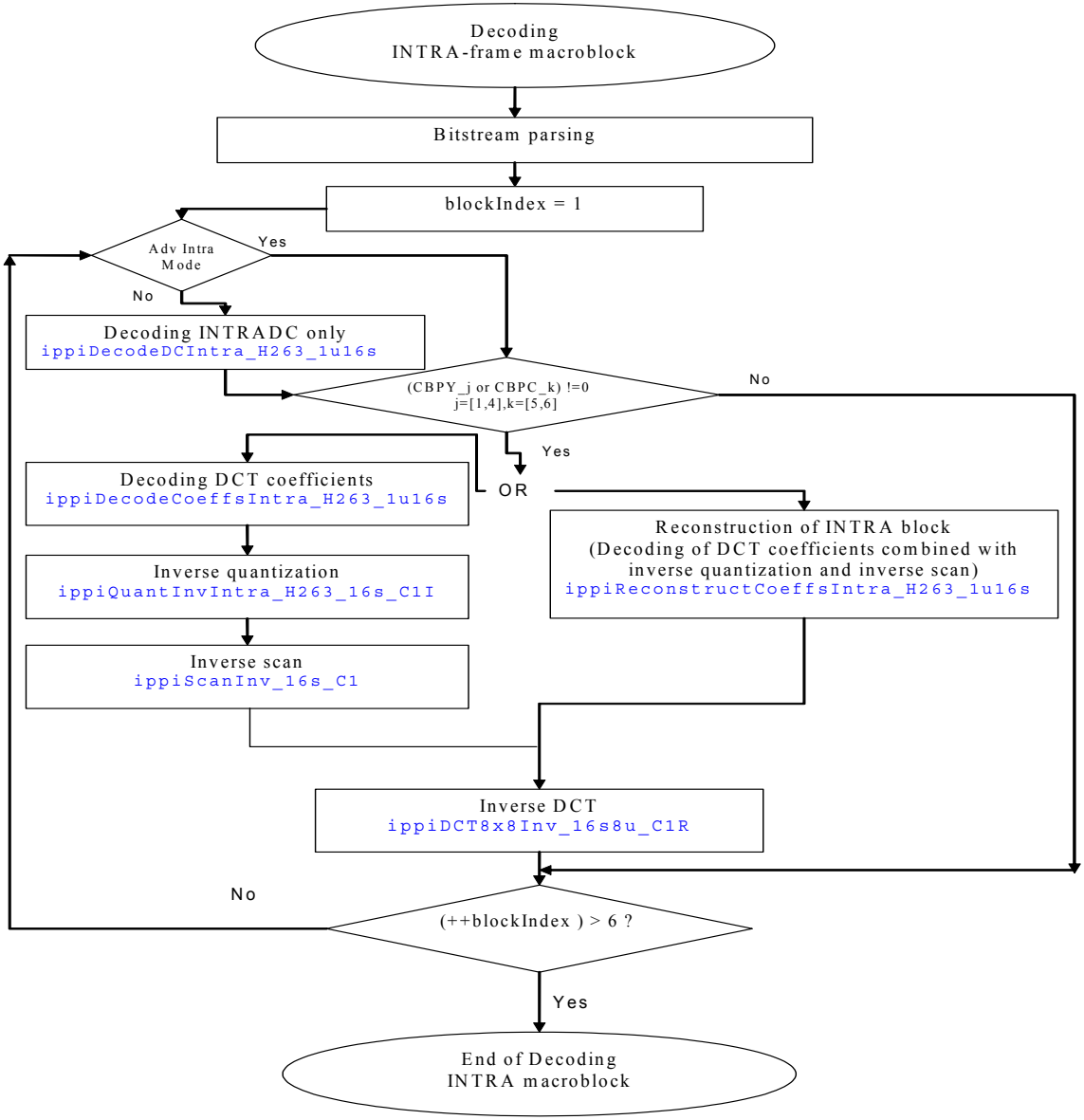
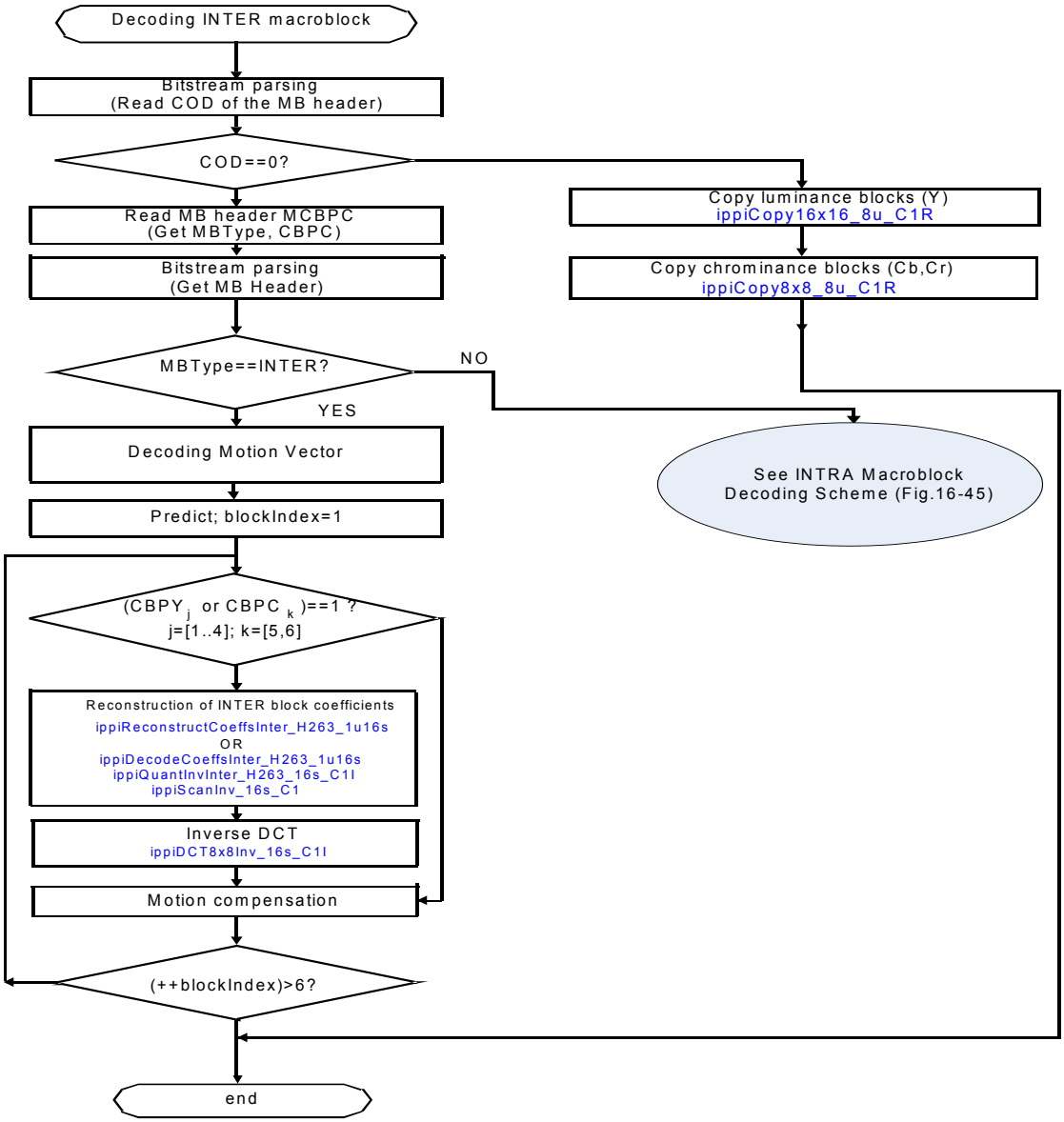


Figure 16-48 shows the process of inter macroblock decoding.

Figure 16-48 INTER Macroblock Decoding



VLC Decoding

DecodeDCIntra_H263

Decodes DC coefficient for intra coded block.

Syntax

```
IppStatus ippiDecodeDCIntra_H263_1u16s(Ipp8u** ppBitStream, int* pBitOffset,
    Ipp16s* pDC);
```

Parameters

ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.
pDC	Pointer to the output coefficient.

Description

The function `ippiDecodeDCIntra_H263_1u16s` is declared in the `ippvc.h` header file. This function performs fixed length decoding of the DC coefficient for one intra coded block. Intra DC decoding process is specified in [ITUH263], subclause 5.4.1.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
`ippStsVLCErr`	Indicates an error condition if an illegal code is detected through the DC stream processing.

DecodeCoeffsIntra_H263

Decodes AC coefficients for intra coded block.

Syntax

```
IppStatus ippiDecodeCoeffsIntra_H263_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoef, int* pIndxLastNonZero, int advIntraFlag, int
    modQuantFlag, int scan);
```

Parameters

ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.
pCoef	Pointer to the output coefficients. *pCoef*[0] is the DC coefficient.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in* *pIndxLastNonZero*. If the block has no correctly decoded coefficients,* *pIndxLastNonZero* is set to -1 when in Advanced Intra Coding mode, and to 0 otherwise.
advIntraFlag	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
scan	Type of the inverse scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan, IPPVC_SCAN_NONE, indicating that no inverse scan is to be performed. See the corresponding enumerator on p.16-3.

Description

The function ippiDecodeCoeffsIntra_H263_1u16s is declared in the ippvc.h header file. This function performs VLC decoding and, optionally, inverse scan of the AC coefficients for one intra coded block. Intra AC VLC decoding process is specified in

[ITUH263], subclause 5.4.2, and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. When in Advanced Intra Coding mode, VLC Table I.2 from [ITUH263] Annex I is used for all intra DC and intra AC coefficients, otherwise Table 16 [ITUH263] is used to decode AC coefficients (starting from *pCoef*[1]) only. If *scan* is not set to IPPVC_SCAN_NONE, the DCT coefficients, encoded in the bitstream in the classical zigzag, alternate-horizontal, or alternate-vertical scan order, are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output. The three scan patterns are shown in [ITUH263], Figure14 and [ITUH263], Annex I, Figure I.2.

This function is used in the H.263 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
ippStsVLCErr	Indicates an error condition if an illegal code is detected through the VLC stream processing.

DecodeCoeffsInter_H263

Decodes DCT coefficients for inter coded block.

Syntax

```
IppStatus ippiDecodeCoeffsInter_H263_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoef, int* pIndxLastNonZero, int modQuantFlag, int
    scan);
```

Parameters

ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.

pCoef	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in* *pIndxLastNonZero*. If the block has no correctly decoded coefficients,* *pIndxLastNonZero* is set to -1.
modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
scan	Type of the inverse scan, takes one of the following values: `IPPVC_SCAN_ZIGZAG`, indicating the classical zigzag scan, `IPPVC_SCAN_NONE`, indicating that no inverse scan is to be performed. See the corresponding enumerator on p.16-3.

Description

The function `ippiDecodeCoeffsInter_H263_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding and, optionally, inverse scan of the DCT coefficients for one inter coded block. Inter DCT VLC decoding process is specified in [ITUH263], subclause 5.4.2 (Table 16), and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. If *scan* is not set to `IPPVC_SCAN_NONE`, the DCT coefficients, encoded in the bitstream in the classical zigzag scan order, are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output. The zigzag scan pattern is shown in [ITUH263], Figure14.

This function is used in the H.263 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
`ippStsVLCErr`	Indicates an error condition if an illegal code is detected through the VLC stream processing.

Inverse Quantization

QuantInvIntra_H263

Performs inverse quantization on an intra coded block stored in a one-dimensional buffer.

Syntax

```
IppStatus ippiQuantInvIntra_H263_16s_C1I(Ipp16s* pSrcDst, int indxLastNonZero,
    int QP, int advIntraFlag, int modQuantFlag);
```

Parameters

pSrcDst	Pointer to the coefficient buffer of the block. The parameter contains quantized coefficients on input and dequantized coefficients on output.
indxLastNonZero	Index of the last non-zero coefficient. This parameter provides faster operation. If the value is unknown, set to 63.
QP	Quantization parameter.
advIntraFlag	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

Description

The function `ippiQuantInvIntra_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs inverse quantization on intra coded block. When not in Advanced Intra Coding mode, the dequantization processes for the intra DC and for all other non-zero coefficients are specified in [ITUH263], subclause 6.2.1, otherwise all the coefficients are dequantized as specified in [ITUH263] Annex I, clause I.3. When not in Advanced Intra Coding mode and not in Modified Quantization mode, the output coefficients other than the intra DC are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2). The overall procedure is defined in [ITUH263] as:

if (*advIntraFlag* == 0):

$$pDst[i] = \begin{cases} 0, & pSrc[i] = 0 \\ Sign(pSrc[i])*QP*(2*|pSrc[i]|+1), & pSrc[i] \neq 0, \quad QP \text{ is odd} \\ Sign(pSrc[i])*(QP*(2*|pSrc[i]|+1)-1), & pSrc[i] \neq 0, \quad QP \text{ is even} \end{cases}$$

where i = 1, 2, ..., *indxLastNonZero*

$pDst[0] = 8*pSrc[i]$

if (*advIntraFlag* != 0):

$pDst[i] = 2*QP*pSrc[i]$

where i = 0, 1, 2, ..., *indxLastNonZero*

if (*advIntraFlag* == 0 && *modQuantFlag* == 0):

$pDst[i] = MIN(MAX(pDst[i], -2048), 2047).$

NOTE. The function `ippiQuantInvIntra_H263_16s_C1I` can be applied to a buffer of arbitrary size (*indxLastNonZero* can be any positive number), and can thus be used, for example, to process multiple blocks in one call. In this case, for any intra block following the first one, the intra DC should be processed separately, if not in Advanced Intra mode.

This function is used in the H.261, H.263, and MPEG-4 encoders and decoders included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrcDst* is NULL.
`ippStsQPErr`	Indicates an error condition if *QP* is out of the range [1, 31].
`ippStsOutOfRangeErr`	Indicates an error condition if *indxLastNonZero* is negative.

QuantInvInter_H263

Performs inverse quantization on an inter coded block stored in a one-dimensional buffer.

Syntax

```
IppStatus ippiQuantInvInter_H263_16s_C1I(Ipp16s* pSrcDst, int indxLastNonZero,
    int QP, int modQuantFlag);
```

Parameters

pSrcDst	Pointer to the coefficient buffer of the block. The parameter contains quantized coefficients on input and dequantized coefficients on output.
indxLastNonZero	Index of the last non-zero coefficient. This parameter provides faster operation. If the value is unknown, set to 63.
QP	Quantization parameter.
modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

Description

The function `ippiQuantInvInter_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs inverse quantization on inter coded block. The dequantization process is specified in [ITUH263], subclause 6.2.1. When not in Modified Quantization mode, the output coefficients are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2). The overall procedure is defined in [ITUH263] as:

$$pDst[i] = \begin{cases} 0, & pSrc[i] = 0 \\ Sign(pSrc[i])*QP*(2*|pSrc[i]|+1), & pSrc[i] \neq 0, \quad QP \text{ is odd} \\ Sign(pSrc[i])*(QP*(2*|pSrc[i]|+1)-1), & pSrc[i] \neq 0, \quad QP \text{ is even} \end{cases}$$

where $i = 0, 1, 2, \ldots, indxLastNonZero$

if ($modQuantFlag == 0$):

$$pDst[i] = MIN(MAX(pDst[i], -2048), 2047).$$

NOTE. The function `ippiQuantInvInter_H263_16s_C1I` can be applied to a buffer of arbitrary size (*indxLastNonZero* can be any positive number), and can thus be used, for example, to process multiple blocks in one call. In this case, for any intra block following the first one, the intra DC should be processed separately, if not in Advanced Intra mode.

This function is used in the H.261, H.263, and MPEG-4 encoders and decoders included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if *pSrcDst* is NULL.
`ippStsQPErr`	Indicates an error condition if *QP* is out of the range [1, 31].
`ippStsOutOfRangeErr`	Indicates an error condition if *indxLastNonZero* is negative.

Prediction

AddBackPredPB_H263

Performs bidirectional prediction for a B-block in a PB-frame.

Syntax

```
IppStatus ippiAddBackPredPB_H263_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pSrcDst, int srcDstStep, int acc);
```

Parameters

pSrc	Pointer to the origin of the source image (P-macroblock) region of interest (ROI).

srcStep	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
srcRoiSize	Size of the source ROI.
pSrcDst	Pointer to the origin of the source-destination image ROI, that is, bidirectionally-predicted part of the block.
srcDstStep	Width in bytes of the source-destination image plane.
acc	Parameter that defines pixel accuracy for backward prediction: bit 0 (the least significant bit) contains the half-pixel offset in horizontal direction, bit 1 – the offset in vertical direction.

Description

The function `ippiAddBackPredPB_H263_8u_C1R` is declared in the `ippvc.h` header file. This function calculates backward prediction for a B-block of a PB-frame and adds it to the block, previously reconstructed with forward prediction. All the operations are restricted to the bidirectionally-predicted part of the B-block. The parameter *srcRoiSize* defines the area size. The backward prediction is performed with pixel accuracy defined by *acc*, and the sum of the forward and backward predictions for every pixel within *srcRoiSize* is divided by 2 (division by truncation). The bidirectional prediction procedure is specified in [ITUH263], Annex G, clause G.5, bidirectionally- and forward-predicted areas for a B-block are demonstrated in [ITUH263], Annex G, Figure G.2.

This function is used in the H.263 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcRoiSize* has a field with zero or negative value.

Resampling

Resample_H263

Performs picture resampling defined in terms of picture area corner displacements.

Syntax

```
IppStatus ippiResample_H263_8u_P3R(const Ipp8u* pSrcY, int srcYStep, IppiSize
    ySrcRoiSize, const Ipp8u* pSrcCb, int srcCbStep, const Ipp8u* pSrcCr, int
    srcCrStep, Ipp8u* pDstY, int dstYStep, IppiSize dstYRoiSize, Ipp8u* pDstCb,
    int dstCbStep, Ipp8u* pDstCr, int dstCrStep, IppMotionVector warpParams[4],
    int wda, int rounding, int fillMode, int fillColor[3]);
```

Parameters

pSrcY	Pointer to the origin of the source image region of interest (ROI) in the luminance plane.
srcYStep	Width in bytes of the source image luminance (Y) plane, that is, distance in bytes between the starting ends of consecutive lines of the source image in the luminance plane.
ySrcRoiSize	Size of the source ROI in the luminance plane.
pSrcCb	Pointer to the origin of the source ROI in Cb chrominance plane.
srcCbStep	Width in bytes of the source image Cb chrominance plane.
pSrcCr	Pointer to the origin of the source image ROI in Cr chrominance plane.
srcCrStep	Width in bytes of the source image Cr chrominance plane.
pDstY	Pointer to the origin of the destination image ROI in the luminance plane.
dstYStep	Width in bytes of the destination image luminance plane.
yDstRoiSize	Size of the destination ROI in th eluminance plane.
pDstCb	Pointer to the origin of the destination image ROI in Cb chrominance plane.
dstCbStep	Width in bytes of the destination image Cb chrominance plane.
pDstCr	Pointer to the origin of the destination image ROI in Cr chrominance plane.

dstCrStep	Width in bytes of the destination image Cr chrominance plane.
warpParams	Array of warping parameters – 4 pairs of motion vectors, describing, in the order they are stored in the array, how the upper left, upper right, lower left, and lower right corners of the destination ROI are mapped onto the source image.
wda	Warping displacement accuracy flag, if set to 0, pixel displacements are quantized to half-pixel accuracy, otherwise – to 1/16-pixel accuracy.
rounding	Rounding value that is used in pixel iterpolation, can be 0 or 1.
fillMode	Flag that defines the fill-mode action for the values of the source pixels for which the calculated location in the source image lies outside of the source image ROI. This parameter takes one of the following values: 0, indicating color fill mode, the “outside” Y, Cb, and Cr pixel values are set to *fillColor*[0], *fillColor*[1], and *fillColor*[2], respectively. 1 – *black* fill mode, the “outside” pixel values are set as follows: Y = 16, Cb = Cr = 128. 2 – *gray* fill mode, the “outside” pixel values are all set to 128. 3 – *clip* fill mode, the “outside” pixel values are extrapolated from the values of pixels at the ROI border, as specified in [ITUH263], Annex D.
fillColor	Array of fill color values used in color fill mode.

Description

The function `ippiResample_H263_8u_P3R` is declared in the `ippvc.h` header file. This function resamples a YCbCr picture as specified in [ITUH263], Annex P. The destination picture ROI is mapped onto the source picture ROI as defined by *warpParams*. The pixels falling outside the source image ROI are estimated according to *fillMode*.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *yRoiSize* or *yDstRoiSize* have a field that is odd or less than 4.

UpsampleFour_H263

Performs factor-of-4 picture upsampling.

Syntax

```
IppStatus ippiUpsampleFour_H263_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize
    srcRoiSize, Ipp8u* pDst, int dstStep, int rounding, int fillColor);
```

Parameters

pSrc	Pointer to the origin of the source image region of interest (ROI).
srcStep	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
srcRoiSize	Size of the source ROI.
pDst	Pointer to the origin of the destination image ROI.
dstStep	Width in bytes of the destination image plane.
rounding	Rounding value used in pixel interpolation, can be 0 or 1.
fillColor	Fill color value used for the source pixels for which the calculated location in the source image lies outside of the source image ROI. When *fillColor* is negative, *clip* fill-mode action is employed – the “outside” pixel values are extrapolated from the values of pixels at the ROI border, as specified in [ITUH263], Annex D.

Description

The function `ippiUpsampleFour_H263_8u_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 picture upsampling as specified in [ITUH263], Annex P, subclause P.5.1.

This function is used in the H.263 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcRoiSize* has a field that is odd or less than 4.

DownsampleFour_H263

Performs factor-of-4 picture downsampling.

Syntax

```
IppStatus ippiDownsampleFour_H263_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, int rounding);
```

Parameters

pSrc	Pointer to the origin of the source image region of interest (ROI).
srcStep	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
srcRoiSize	Size of the source ROI.
pDst	Pointer to the origin of the destination image ROI.
dstStep	Width in bytes of the destination image plane.
rounding	Rounding value used in pixel interpolation, can be 0 or 1.

Description

The function `ippiDownsampleFour_H263_8u_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 picture downsampling as specified in [ITUH263], Annex P, subclause P.5.2.

This function is used in the H.263 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcRoiSize* has a field with zero or negative value.

UpsampleFour8x8_H263

Performs factor-of-4 upsampling on an 8x8 block.

Syntax

```
IppStatus ippiUpsampleFour8x8_H263_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep);
```

Parameters

pSrc	Pointer to the origin of the source 8x8 block.
srcStep	Width in bytes of the source image plane, thta is, distance in bytes between the starting ends of consecutive lines of the source block.
pDst	Pointer to the origin of the destination 16x16 block.
dstStep	Width in bytes of the destination image plane.

Description

The function `ippiUpsampleFour8x8_H263_16s_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 upsampling of an 8x8 source block to a 16x16 destination block, as specified in [ITUH263], Annex Q, clause Q.6.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

SpatialInterpolation_H263

Interpolates a picture by a factor of two horizontally, vertically, or both horizontally and vertically.

Syntax

```
IppStatus ippiSpatialInterpolation_H263_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, int interpType);
```

pSrc	Pointer to the origin of the source image region of interest (ROI).
srcStep	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
srcRoiSize	Size of the source ROI.
pDst	Pointer to the origin of the destination image ROI.
dstStep	Width in bytes of the destination image plane.
interpType	Interpolation type, takes one of the following values: `IPPVC_INTERP_HORIZONTAL`, indicating one-dimensional (1-D) horizontal interpolation, `IPPVC_INTERP_VERTICAL`, indicating 1-D vertical interpolation, `IPPVC_INTERP_2D`, indicating 2-D interpolation. See the corresponding enumerator on p.16-3.

Description

The function `ippiSpatialInterpolation_H263_8u_C1R` is declared in the `ippvc.h` header file. This function performs picture interpolation for 1-D or 2-D spatial scalability, as specified in [ITUH263], Annex O, clause O.6. Depending on *interpType*, the source image ROI is interpolated by a factor of 2 either horizontally, vertically, or both horizontally and vertically, the size of the destination image ROI being equal to the size of the source image ROI multiplied by 2 in the direction(s) for which the interpolation is performed.

This function is used in the H.263 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *srcRoiSize* has a field that is odd or less than 4.

Boundary Filtering

FilterBlockBoundaryHorEdge_H263, FilterBlockBoundaryVerEdge_H263

Perform block boundary filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.

Syntax

```
IppStatus ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int
    srcDstStep);
IppStatus ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int
    srcDstStep);
```

Parameters

pSrcDst	Pointer to the origin of the lower (HorEdge) or the right (VerEdge) 16x16 block.
srcDstStep	Width in bytes of the image plane.

Description

The functions ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR and ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR are declared in the ippvc.h header file. These functions perform block boundary filtering on bordering edges, horizontal and vertical respectively, of two adjacent 16x16 blocks, as specified in [ITUH263], Annex Q, subclause Q.7.1.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if *pSrcDst* is NULL.

FilterDeblocking8x8HorEdge_H263, FilterDeblocking8x8VerEdge_H263

Perform deblocking filtering of one block edge on the reconstructed frames.

Syntax

```
IppStatus ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR(Ipp8u* pSrcDst,
    int srcDstStep, int QP);
IppStatus ippiFilterDeblocking8x8VerEdge_H263_8u_C1IR(Ipp8u* pSrcDst,
    int srcDstStep, int QP);
```

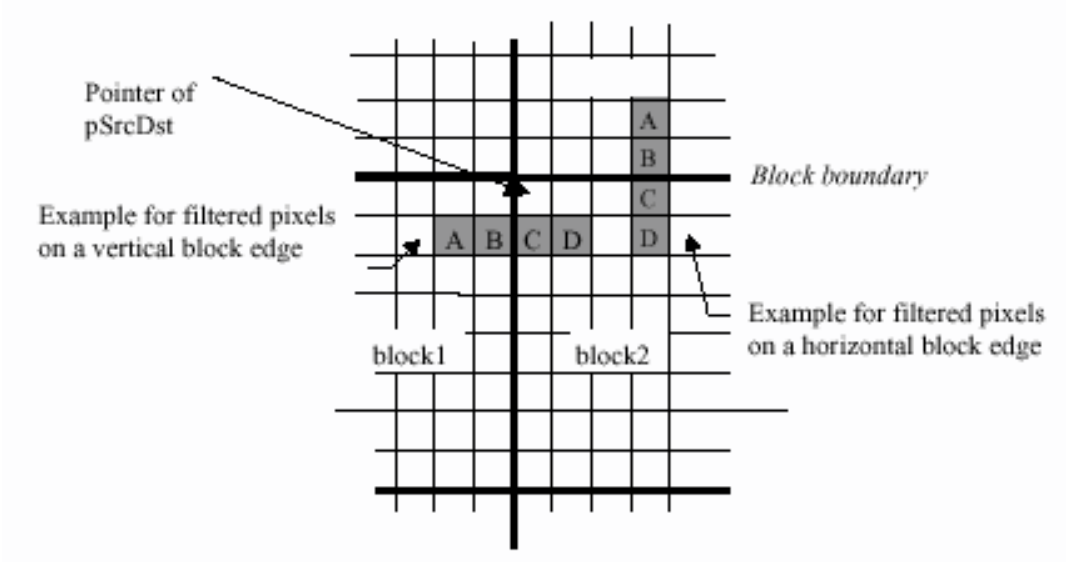
Parameters

pSrcDst	Pointer to the first pixel of the second block (block 2) of the two applied blocks.
srcDstStep	Width of the source and destination plane.
QP	Quantization parameter. The value of *QP* is found as described in Section J.3 of Annex J/H.263+.

Description

The functions `ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR` and `ippiFilterDeblocking_VerEdge8x8H263_8u_C1IR` are declared in the `ippvc.h` file. These functions perform deblocking filtering of one block edge (horizontal or vertical, respectively) on the reconstructed frames. The pointer *pSrcDst* points to the first pixel of the block 2 as shown in Figure 16-49.

Figure 16-49 Deblocking Filtering Layout



These functions are used in the H.263 encoder and decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if *`pSrcDst`* pointer is `NULL`.
`ippStsQPErr`	Indicates an error condition if the quantizer value has a zero or negative value, or if it is greater than 31.

FilterDeblocking16x16HorEdge_H263, FilterDeblocking16x16VerEdge_H263

Perform deblocking filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.

Syntax

```
IppStatus ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int
    srcDstStep);
IppStatus ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR(Ipp8u* pSrcDst, int
    srcDstStep);
```

Parameters

pSrcDst	Pointer to the origin of the lower (HorEdge) or the right (VerEdge) 16x16 block.
step	Width in bytes of the image plane.

Description

The functions ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR and ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR are declared in the ippvc.h header file. These functions perform deblocking filtering on bordering edges, horizontal and vertical respectively, of two adjacent 16x16 blocks, as specified in [ITUH263], Annex Q, subclause Q.7.2 and Annex J, clause J.3.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if *pSrcDst* is NULL.

Middle Level Functions

ReconstructCoeffsIntra_H263

Reconstructs DCT coefficients for an intra coded block.

Syntax

```
IppStatus ippiReconstructCoeffsIntra_H263_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoef, int* pIndxLastNonZero, int cbp, int QP, int
    advIntraFlag, int scan, int modQuantFlag);
```

Parameters

ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.
pCoef	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in* *pIndxLastNonZero*. If the block has no correctly decoded coefficients,* *pIndxLastNonZero* is set to -1.
cbp	Coded block pattern, when set to 0 indicates that the block contains only intra DC coefficient.
QP	Quantization parameter.
advIntraFlag	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
scan	Type of the inverse scan, takes one of the following values: `IPPVC_SCAN_ZIGZAG`, indicating the classical zigzag scan, `IPPVC_SCAN_HORIZONTAL`, indicating the alternate-horizontal scan, `IPPVC_SCAN_VERTICAL`, indicating alternate-vertical scan. See the corresponding enumerator on p.16-3.

modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

Description

The function `ippiReconstructCoeffsIntra_H263_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization, and inverse scan of the DCT coefficients for one intra coded block.

Intra DC decoding process is specified in [ITUH263], subclause 5.4.1. Intra AC VLC decoding process is specified in [ITUH263], subclause 5.4.2, and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. When in Advanced Intra Coding mode, VLC Table I.2 from [ITUH263] Annex I is used for all intra DC and intra AC coefficients, otherwise Table 16 [ITUH263] is used to decode AC coefficients only (for blocks with non-zero *cbp*). When not in Advanced Intra Coding mode, the dequantization processes for the intra DC and for all other non-zero coefficients are specified in [ITUH263], subclause 6.2.1, otherwise all the coefficients are dequantized as specified in [ITUH263] Annex I, clause I.3. When not in Advanced Intra Coding mode and not in Modified Quantization mode, the output coefficients other than the intra DC one are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2).

The DCT coefficients, encoded in the bitstream in the classical zigzag, alternate-horizontal, or alternate-vertical scan order, are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output. The three scan patterns are shown in [ITUH263], Figure14 and [ITUH263], Annex I, Figure I.2.

This function is used in the H.263 and MPEG-4 decoders included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
`ippStsVLCErr`	Indicates an error condition if an illegal code is detected through the stream processing.
`ippStsQPErr`	Indicates an error condition if *QP* is out of the range [1, 31].

ReconstructCoeffsInter_H263

Reconstructs DCT coefficients for an inter coded block.

Syntax

```
IppStatus ippiReconstructCoeffsInter_H263_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoef, int* pIndxLastNonZero, int QP, int modQuantFlag);
```

Parameters

ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.
pCoef	Pointer to the output coefficients.
pIndxLastNonZero	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in* *pIndxLastNonZero*. If the block has no correctly decoded coefficients,* *pIndxLastNonZero* is set to -1.
QP	Quantization parameter.
modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

Description

The function `ippiReconstructCoeffsInter_H263_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization, and inverse scan of the DCT coefficients for one inter coded block.

Inter DCT VLC decoding process is specified in [ITUH263], subclause 5.4.2 (Table 16), and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. The dequantization process is specified in [ITUH263], subclause 6.2.1. When not in Modified Quantization mode, the output coefficients are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2). The DCT coefficients, encoded in the

bitstream in the classical zigzag scan order ([ITUH263], Figure14), are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output.

This function is used in the H.263 and MPEG-4 decoders included into Intel IPP Samples. See introduction to this section.

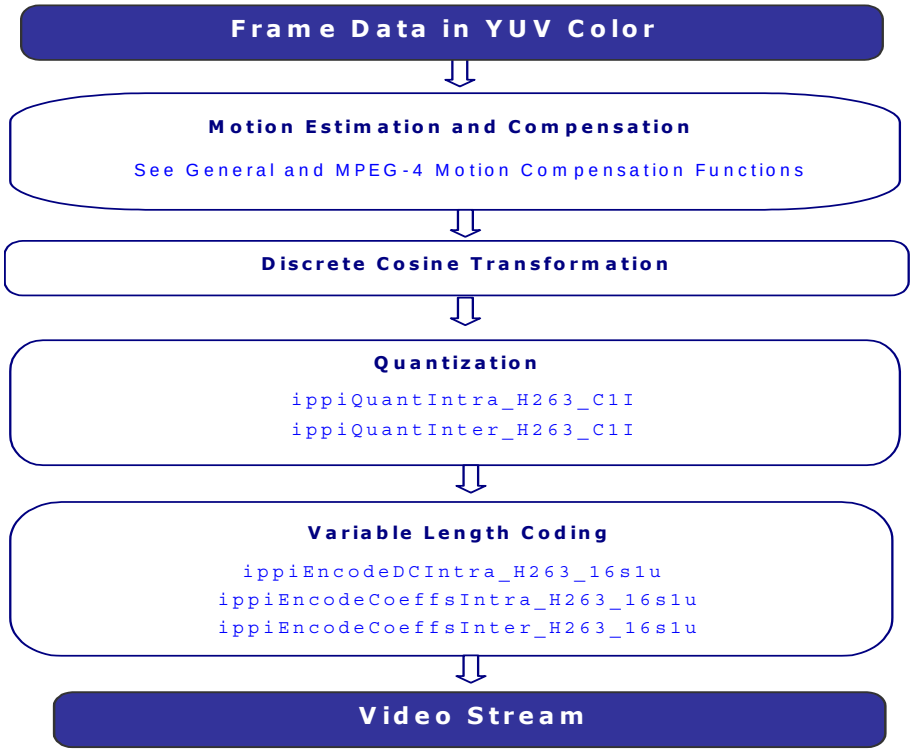
Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
ippStsVLCErr	Indicates an error condition if an illegal code is detected through the stream processing.
ippStsQPErr	Indicates an error condition if *QP* is out of the range [1, 31].

H.263 Encoder Functions

This section describes the main steps of H.263 ([ITUH263]) video encoding in accordance with the pipeline shown in Figure 16-50.

Figure 16-50 H.263 Encoding Pipeline



VLC Encoding

EncodeDCIntra_H263

Encodes and puts a quantized DC coefficient for an intra coded block into bitstream.

Syntax

```
IppStatus ippiEncodeDCIntra_H263_16s1u(Ipp16s qDC, Ipp8u** ppBitStream, int*
    pBitOffset);
```

Parameters

qDC	Quantized DC coefficient.
ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7.* *pBitOffset* is updated by the function.

Description

The function `ippiEncodeDCIntra_H263_16s1u` is declared in the `ippvc.h` header file. This function performs fixed length encoding of the DC coefficient for one intra coded block and puts the code into the bitstream. Intra DC encoding process is specified in [ITUH263], subclause 5.4.1.

This function is used in the H.263 and MPEG-4 encoders included into Intel IPP Samples. See introduction to H.263 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].

EncodeCoeffsIntra_H263

Encodes and puts quantized DCT coefficients for an intra coded block into bitstream.

Syntax

```
IppStatus ippiEncodeCoeffsIntra_H263_16s1u(Ipp16s* pQCoef, Ipp8u** ppBitStream,
    int* pBitOffset, int countNonZero, int advIntraFlag, int modQuantFlag, int
    scan);
```

Parameters

pQCoef	Pointer to the array of quantized DCT coefficients. *pQCoef*[0] is the DC coefficient.
ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7. * *pBitOffset* is updated by the function.
countNonZero	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
advIntraFlag	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
scan	Type of the scan to be performed on the coefficints before encoding, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan, IPPVC_SCAN_NONE, indicating that no scan is to be performed, that is, the input coefficients are already in the scan order. See the corresponding enumerator on p.16-3.

Description

The function ippiEncodeCoeffsIntra_H263_16s1u is declared in the ippvc.h header file. This function performs VLC encoding of the quantized AC coefficients in a scan order for one intra coded block and puts the codes into the bitstream. Intra AC VLC encoding

process is specified in [ITUH263], subclause 5.4.2, and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. When in Advanced Intra Coding mode, VLC Table I.2 from [ITUH263] Annex I is used for all intra DC and intra AC coefficients, otherwise Table 16 [ITUH263] is used to encode AC coefficients (starting from *pQCoef*[1]) only. In this case *countNonZero* is the number of non-zero AC coefficients.

This function is used in the H.263 and MPEG-4 encoders included into Intel IPP Samples. See introduction to H.263 section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsBitOffsetErr	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
ippStsOutOfRangeErr	Indicates an error condition if *countNonZero* is out of the range [1, 64].

EncodeCoeffsInter_H263

Encodes and puts quantized DCT coefficients for inter coded block into bitstream.

Syntax

```
IppStatus ippiEncodeCoeffsInter_H263_16s1u(Ipp16s* pQCoef, Ipp8u** ppBitStream,
    int* pBitOffset, int countNonZero, int modQuantFlag, int scan);
```

Parameters

pQCoef	Pointer to the array of quantized DCT coefficients. *pQCoef*[0] is the DC coefficient.
ppBitStream	Pointer to pointer to the current byte in the bitstream buffer.* *ppBitStream* is updated by the function.
pBitOffset	Pointer to the bit position in the byte pointed by* *ppBitStream*. Valid within the range 0 to 7. * *pBitOffset* is updated by the function.

countNonZero	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
scan	Type of the scan to be performed on the coefficints before encoding, takes one of the following values: `IPPVC_SCAN_ZIGZAG`, indicating the classical zigzag scan, `IPPVC_SCAN_NONE`, indicating that no scan is to be performed, that is, the input coefficients are already in the scan order. See the corresponding enumerator on p.16-3.

Description

The function `ippiEncodeCoeffsInter_H263_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the quantized DCT coefficients in a scan order for one inter coded block and puts the codes into the bitstream. Inter DCT VLC encoding process is specified in [ITUH263], subclause 5.4.2 (Table 16), and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use.

This function is used in the H.263 and MPEG-4 encoders included into Intel IPP Samples. See introduction to H.263 section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsBitOffsetErr`	Indicates an error condition if* *pBitOffset* is out of the range [0, 7].
`ippStsOutOfRangeErr`	Indicates an error condition if *countNonZero* is out of the range [1, 64].

Quantization

QuantIntra_H263

Performs quantization on an intra coded block.

Syntax

```
IppStatus ippiQuantIntra_H263_16s_C1I(Ipp16s* pSrcDst, int QP, int*
    pCountNonZero, int advIntraFlag, int modQuantFlag);
```

Parameters

pSrcDst	Pointer to the coefficient buffer of the block.
QP	Quantization parameter.
pCountNonZero	Pointer to the number of non-zero coefficients after quantization.
advIntraFlag	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

Description

The function `ippiQuantIntra_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs quantization on an intra coded block according to H.263 standard. The standard specifies dequantization process, while quantization decision levels are not defined. When not in Advanced Intra Coding mode, the intra DC coefficient is dequantized using uniformly placed reconstruction levels with a step size of 8, and the other DCT coefficients are reconstructed using equally spaced levels with a central dead-zone around zero and with a step size of 2*QP ([ITUH263], subclauses 4.2.4, 6.2). When in Advanced Intra Coding mode, all the block coefficients are dequantized using a reconstruction spacing without a dead-zone and with a step size of 2*QP ([ITUH263] Annex I, clause I.3). When not in Modified Quantization mode, the quantized intra DC coefficient (when not in Advanced Intra Coding mode) is clipped to the range [1, 254], and the other quantized coefficients (all coefficients, if in Advanced Intra Coding mode) are clipped to the range [-127, 127].

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See introduction to H.263 section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsQPErr	Indicates an error condition if *QP* is out of the range [1, 31].

QuantInter_H263

Performs quantization on an inter coded block.

Syntax

```
IppStatus ippiQuantInter_H263_16s_C1I(Ipp16s* pSrcDst, int QP, int*
    pCountNonZero, int modQuantFlag);
```

Parameters

pSrcDst	Pointer to the coefficient buffer of the block.
QP	Quantization parameter.
pCountNonZero	Pointer to the number of non-zero coefficients after quantization.
modQuantFlag	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

Description

The function ippiQuantInter_H263_16s_C1I is declared in the ippvc.h header file. This function performs quantization on an inter coded block according to H.263 standard. The standard specifies dequantization process, while quantization decision levels are not defined. The DCT coefficients are reconstructed using equally spaced levels with a central dead-zone around zero and with a step size of 2*QP ([ITUH263], subclauses 4.2.4, 6.2). When not in Modified Quantization mode, the quantized coefficients are clipped to the range [-127, 127].

This function is used in the H.261, H.263, and MPEG-4 encoders included into Intel IPP Samples. See introduction to H.263 section.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsQPErr	Indicates an error condition if *QP* is out of the range [1, 31].

Resampling

DownsampleFour16x16_H263

Performs factor-of-4 downsampling on a 16x16 block.

Syntax

```
IppStatus ippiDownsampleFour16x16_H263_16s_C1R(const Ipp16s* pSrc, int srcStep,
    Ipp16s* pDst, int dstStep);
```

Parameters

pSrc	Pointer to the origin of the source 16x16 block.
srcStep	Width in bytes of the source image plane, that is, distance in bytes between the starts of consecutive lines of the source block.
pDst	Pointer to the origin of the destination 8x8 block.
dstStep	Width in bytes of the destination image plane.

Description

The function ippiDownsampleFour16x16_H263_16s_C1R is declared in the ippvc.h header file. This function performs factor-of-4 downsampling of a 16x16 source block to an 8x8 destination block, which is used for block encoding in Reduced-Resolution Update mode specified in [ITUH263], Annex Q. The inverse factor-of-4 upsampling procedure, specified in [ITUH263], Annex Q, clause Q.6, is performed by the function UpsampleFour8x8_H263.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

H.264

This section describes functions for decoding of video data in accordance with JVT-G050 ([JVTG050]) and ITUH264 ([ITUH264]) standards.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm .

The following enumeration is used to indicate prediction modes of the Intra_4x4 prediction process for luma samples (8.3.1 of JVT-G050):

```
typedef enum {   IPP_4x4_VERT      = 0,
                 IPP_4x4_HOR       = 1,
                 IPP_4x4_DC        = 2,
                 IPP_4x4_DIAG_DL   = 3,
                 IPP_4x4_DIAG_DR   = 4,
                 IPP_4x4_VR        = 5,
                 IPP_4x4_HD        = 6,
                 IPP_4x4_VL        = 7,
                 IPP_4x4_HU        = 8
               } IppIntra4x4PredMode_H264;
```

Table 16-21 shows the correspondence between prediction modes and constants of `enum IppIntra4x4PredMode_H264`.

Table 16-21

Name of Constant	Prediction Mode	Chapter in JVT-G050
IPP_4x4_VERT	Intra_4x4_Vertical	8.3.1.2.1
IPP_4x4_HOR	Intra_4x4_Horizontal	8.3.1.2.2
IPP_4x4_DC	Intra_4x4_DC	8.3.1.2.3
IPP_4x4_DIAG_DL	Intra_4x4_Diagonal_Down_Left	8.3.1.2.4
IPP_4x4_DIAG_DR	Intra_4x4_Diagonal_Down_Right	8.3.1.2.5
IPP_4x4_VR	Intra_4x4_Vertical_Right	8.3.1.2.6
IPP_4x4_HD	Intra_4x4_Horizontal_Down	8.3.1.2.7
IPP_4x4_VL	Intra_4x4_Vertical_Left	8.3.1.2.8
IPP_4x4_HU	Intra_4x4_Horizontal_Up	8.3.1.2.9

The following enumeration is used to indicate prediction modes of the Intra_16x16 prediction process for luma samples (8.3.2 of JVT-G050):

```
typedef enum {   IPP_16X16_VERT     = 0,
                 IPP_16X16_HOR      = 1,
                 IPP_16X16_DC       = 2,
                 IPP_16X16_PLANE    = 3,
               } IppIntra16x16PredMode_H264;
```

Table 16-22 shows the correspondence between prediction modes and constants of `enum IppIntra16x16PredMode_H264`.

Table 16-22

Name of Constant	Prediction Mode	Chapter in JVT-G050
IPP_16X16_VERT	Intra_16x16_Vertical	8.3.2.1
IPP_16X16_HOR	Intra_16x16_Horizontal	8.3.2.2
IPP_16X16_DC	Intra_16x16_DC	8.3.2.3
IPP_16X16_PLANE	Intra_16x16_Plane	8.3.2.4

The following enumeration is used to indicate prediction modes of the intra prediction process for chroma samples (8.3.3 of JVT-G050):

```
typedef enum {   IPP_CHROMA_DC     = 0,
                 IPP_CHROMA_HOR    = 1,
                 IPP_CHROMA_VERT   = 2,
                 IPP_CHROMA_PLANE  = 3,
               } IppIntraChromaPredMode_H264;
```

Table 16-23 shows the correspondence between prediction modes and constants of `enum IppIntraChromaPredMode_H264`.

Table 16-23

Name of Constant	Prediction Mode	Chapter in JVT-G050
IPP_CHROMA_DC	Intra_Chroma_DC	8.3.3.1
IPP_CHROMA_HOR	Intra_Chroma_Horizontal	8.3.3.2
IPP_CHROMA_VERT	Intra_Chroma_Vertical	8.3.3.3
IPP_CHROMA_PLANE	Intra_Chroma_Plane	8.3.3.4

The following enumeration is used to indicate availability of the corresponding positions for prediction in the picture:

```
typedef enum { IPP_UPPER        = 1,
               IPP_LEFT         = 2,
                IPP_CENTER       = 4,
               IPP_RIGHT        = 8,
                IPP_LOWER        = 16,
                IPP_UPPER_LEFT   = 32,
                IPP_UPPER_RIGHT  = 64,
               IPP_LOWER_LEFT   = 128,
                IPP_LOWER_RIGHT  = 256,
              } IppLayoutFlag;
```

The following enumeration is used to indicate image type of the picture:

```
typedef enum _IPPVC_FRAME_FIELD_FLAG
{
  IPPVC_FRAME        = 0x0,
  IPPVC_TOP_FIELD    = 0X1,
  IPPVC_BOTTOM_FIELD = 0x2
 } IPPVC_FRAME_FIELD_FLAG;
```

Table 16-24 H.264 Video Decoder Functions

Function Short Name	Description
CAVLC Parsing	
DecodeCAVLCCoeffs_H264	Decodes any non-chroma DC coefficients CAVLC coded.
DecodeCAVLCChromaDcCoeffs_H264, DecodeCAVLCChroma422DcCoeffs_H264	Decode chroma DC coefficients CAVLC coded.
DecodeExpGolombOne_H264	Decodes one Exp-Golomb code accordance to 9.1 H.264 standard.
Inverse Quantization and Inverse Transform	
TransformDequantLumaDC_H264	Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.

Table 16-24 H.264 Video Decoder Functions (continued)

Function Short Name	Description
TransformDequantChromaDC_H264	Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.
TransformResidual4x4Inv_H264	Performs integer inverse transformation for a 4x4 block of residuals.
DequantTransformResidual_H264	Performs dequantization, integer inverse transformation, and shift for a 4x4 block of residuals.
DequantTransformResidualAndAdd_H264	Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.
TransformPrediction_H264	Performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks.
DequantTransformResidual_SISP_H264	Performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks.
TransformDequantChromaDC_SISP_H264	Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in P macroblocks in SP slices or SI macroblocks.
Intra Prediction	
PredictIntra_4x4_H264	Performs luma component prediction for intra 4x4 macroblock type.
PredictIntra_16x16_H264	Performs luma component prediction for intra 16x16 macroblock type.
PredictIntraChroma8x8_H264	Performs chroma component prediction for intra macroblock type.
Inter Prediction	
ExpandPlane_H264	Expands the plane.
InterpolateLuma_H264	Performs interpolation for motion estimation of the luma component using fractional part of motion vector.

Table 16-24 H.264 Video Decoder Functions (continued)

Function Short Name	Description
InterpolateLumaTop_H264	Performs interpolation for motion estimation of the luma component at the frame top boundary.
InterpolateLumaBottom_H264	Performs interpolation for motion estimation of the luma component at the frame bottom boundary.
InterpolateLumaBlock_H264	Performs interpolation for motion estimation of the luma component using entire motion vector.
InterpolateChroma_H264	Performs interpolation for motion estimation of the chroma component using fractional part of motion vector
InterpolateChromaTop_H264	Performs interpolation for motion estimation of the chroma component at the frame top boundary.
InterpolateChromaBottom_H264	Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.
InterpolateChromaBlock_H264	Performs interpolation for motion estimation of the chroma component using entire motion vector.
InterpolateBlock_H264, Bidir_H264	Calculate the average value for each source pair values in block.
WeightedAverage_H264	Averages two blocks with weights.
UniDirWeightBlock_H264, UnidirWeight_H264	Weigh source block.
BiDirWeightBlock_H264, BidirWeight_H264	Average two blocks with two weights and two offsets.
BiDirWeightBlockImplicit_H264, BidirWeightImplicit_H264	Average two blocks with weights if *ulog2wd* is equal to 5.
Macroblock Reconstruction	
ReconstructChromaInterMB_H264	Reconstructs inter chroma macroblock.
ReconstructChromaIntraHalvesMB_H264	Reconstructs two halves of intra chroma macroblock.
ReconstructChromaIntraMB_H264	Reconstructs intra chroma macroblock.
ReconstructChromaInter4x4MB_H264, ReconstructChromaInter4x4_H264High	Reconstruct 4X4 inter chroma macroblock for high profile.

Table 16-24 H.264 Video Decoder Functions (continued)

Function Short Name	Description
ReconstructChroma422Inter4x4_H264High	Reconstructs 4X4 inter chroma macroblock for 4:2:2 chroma mode.
ReconstructChromaIntraHalves4x4MB_H264, ReconstructChromaIntraHalf4x4_H264High	Reconstruct two independent halves of 4X4 intra chroma macroblock for high profile.
ReconstructChromaIntra4x4MB_H264, ReconstructChromaIntra4x4_H264High	Reconstruct 4X4 intra chroma macroblock for high profile.
ReconstructChroma422IntraHalf4x4_H264High	Reconstructs two independent halves of 4X4 intra chroma macroblock for 4:2:2 chroma mode.
ReconstructChroma422Intra4x4_H264High	Reconstructs 4X4 intra chroma macroblock for 4:2:2 chroma mode.
ReconstructLumaInterMB_H264	Reconstructs inter luma macroblock.
ReconstructLumaIntraHalfMB_H264	Reconstructs half of intra luma macroblock.
ReconstructLumaIntraMB_H264	Reconstructs intra luma macroblock.
ReconstructLumaInter4x4MB_H264, ReconstructLumaInter4x4_H264High	Reconstruct 4X4 inter luma macroblock for high profile.
ReconstructLumaIntraHalf4x4MB_H264, ReconstructLumaIntraHalf4x4_H264High	Reconstruct half of 4X4 intra luma macroblock for high profile.
ReconstructLumaIntra4x4MB_H264, ReconstructLumaIntra4x4_H264High	Reconstruct 4X4 intra luma macroblock for high profile.
ReconstructLumaInter8x8MB_H264, ReconstructLumaInter8x8_H264High	Reconstruct 8X8 inter luma macroblock for high profile.
ReconstructLumaIntraHalf8x8MB_H264, ReconstructLumaIntraHalf8x8_H264High	Reconstruct half of 8X8 intra luma macroblock for high profile.
ReconstructLumaIntra8x8MB_H264, ReconstructLumaIntra8x8_H264High	Reconstruct 8X8 intra luma macroblock for high profile.
ReconstructLumaIntra16x16MB_H264	Reconstructs intra 16X16 luma macroblock.
ReconstructLumaIntra_16x16MB_H264, ReconstructLumaIntra16x16_H264High	Reconstruct intra 16X16 luma macroblock for high profile.

Table 16-24 H.264 Video Decoder Functions (continued)

Function Short Name	Description
Deblocking Filtering	
FilterDeblockingLuma_VerEdge_H264	Performs deblocking filtering on the vertical edges of the luma 16x16 macroblock.
FilterDeblockingLuma_VerEdge_MBAFF_H264	Performs deblocking filtering on the external vertical edges of half of 16x16 luma macroblock.
FilterDeblockingLuma_HorEdge_H264	Performs deblocking filtering on the horizontal edges of the luma 16x16 macroblock.
FilterDeblockingChroma_VerEdge_H264, FilterDeblockingChroma422VerEdge_H264	Performs deblocking filtering on the vertical edges of the chroma 8x8 macroblock.
FilterDeblockingChroma_VerEdge_MBAFF_H264, FilterDeblockingChroma422VerEdgeMBAFF_H264	Performs deblocking filtering on the external vertical edges of half of 8x8 chroma macroblock.
FilterDeblockingChroma_HorEdge_H264, FilterDeblockingChroma422HorEdge_H264	Perform deblocking filtering on the chroma 8x8 macroblock.

Table 16-25 H.264 Video Encoder Functions

Function Short Name	Description
Edges Detection	
EdgesDetect16x16	Detects edges inside a 16X16 block.
Calculation of Inter Predicted Blocks	
InterpolateLuma_H264	Performs interpolation for motion estimation of the luma component.
InterpolateLumaTop_H264	Performs interpolation for motion estimation of the luma component at the frame top boundary.
InterpolateLumaBottom_H264	Performs interpolation for motion estimation of the luma component at the frame bottom boundary.

Table 16-25 H.264 Video Encoder Functions (continued)

Function Short Name	Description
InterpolateChroma_H264	Performs interpolation for motion estimation of the chroma component.
InterpolateChromaTop_H264	Performs interpolation for motion estimation of the chroma component at the frame top boundary.
InterpolateChromaBottom_H264	Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.
Calculation of Intra Predicted Blocks	
PredictIntra_4x4_H264	Performs intra prediction for a 4x4 luma component.
PredictIntra_16x16_H264	Performs intra prediction for a 16x16 luma component.
PredictIntraChroma8x8_H264	Performs intra prediction for a 8x8 chroma component.
Estimation of Inter and Intra Predicted Blocks	
SAD16x16	Evaluates sum of absolute difference between current and reference 16X16 blocks.
SAD16x8	Evaluates sum of absolute difference between current and reference 16X8 blocks.
SAD8x16	Evaluates sum of absolute difference between current and reference 8X16 blocks.
SAD8x8	Evaluates sum of absolute difference between current and reference 8X8 blocks.
SAD8x4	Evaluates sum of absolute difference between current and reference 8X4 blocks.
SAD4x8	Evaluates sum of absolute difference between current and reference 4X8 blocks.
SAD4x4	Evaluates sum of absolute difference between current and reference 4X4 blocks.

Table 16-25 H.264 Video Encoder Functions (continued)

Function Short Name	Description
SAD16x16Blocks8x8	Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.
SAD16x16Blocks4x4	Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.
SATD16x16	Evaluates sum of absolute transformed differences between current and reference 16X16 blocks using 4x4 transform.
SATD16x8	Evaluates sum of absolute transformed differences between current and reference 16X8 blocks using 4x4 transform.
SATD8x16	Evaluates sum of absolute transformed differences between current and reference 8X16 blocks using 4x4 transform.
SATD8x8	Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 4x4 transform.
SATD8x4	Evaluates sum of absolute transformed differences between current and reference 8X4 blocks using 4x4 transform.
SATD4x8	Evaluates sum of absolute transformed differences between current and reference 4X8 blocks using 4x4 transform.
SATD4x4	Evaluates sum of absolute transformed differences between current and reference 4X4 blocks using 4x4 transform.
SAT8x8D	Evaluates sum of absolute transformed differences between current and reference 8X8 blocks using 8x8 transform.

Table 16-25 H.264 Video Encoder Functions (continued)

Function Short Name	Description
Obtaining of Residual and DC Blocks	
GetDiff4x4	Calculates a residual 4x4 block in the cases of intra 4x4 luma block or inter luma block.
SumsDiff16x16Blocks4x4	Calculates a 4x4 DC luma block and 4x4 residual blocks in the case of an intra 16x16 luma block.
SumsDiff8x8Blocks4x4	Calculates a 2x2 DC chroma block and residual 4x4 chroma blocks.
Forward Transform and Quantization	
TransformQuantChromaDC_H264	Performs forward transform and quantization for 2x2 DC chroma blocks.
TransformQuantLumaDC_H264	Performs forward transform and quantization for 4x4 DC luma blocks.
TransformResidual4x4Fwd_H264	Performs forward transform for 4X4 residual blocks.
QuantizeResidual4x4Fwd_H264	Performs forward quantization for 4X4 residual blocks.
TransformQuantResidual_H264	Performs forward transform and quantization for 4x4 residual blocks.
TransformLuma8x8Fwd_H264	Performs forward 8x8 transform for 8x8 luma blocks without normalization.
QuantLuma8x8_H264	Performs quantization for 8x8 luma block coefficients including 8x8 transform normalization.
GenScaleLevel8x8_H264	Generates ScaleLevel matrices for forward and inverse quantization including normalization for 8x8 forward and inverse transform.
CAVLC Functions	
EncodeCoeffsCAVLC_H264	Calculates characteristics of 4X4 block for CAVLC encoding.
EncodeChromaDcCoeffsCAVLC_H264	Calculates characteristics of 2X2 chroma DC block for CAVLC encoding.
Inverse Quantization and Transform	

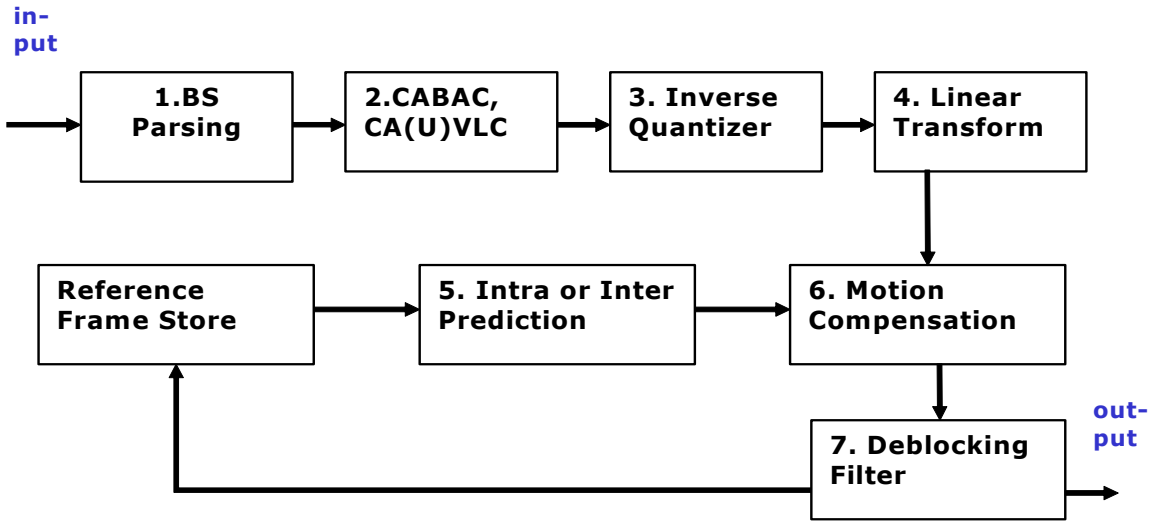
Table 16-25 H.264 Video Encoder Functions (continued)

Function Short Name	Description
TransformDequantLumaDC_H264	Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.
TransformDequantChromaDC_H264	Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.
DequantTransformResidualAndAdd_H264	Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.
QuantLuma8x8Inv_H264	Performs inverse quantization for 8x8 luma block coefficients including normalization of the following inverse 8x8 transform.
TransformLuma8x8InvAddPred_H264	Performs inverse 8x8 transform of 8x8 luma block coefficients with subsequent intra prediction or motion compensation.
Deblocking	
FilterDeblockingLuma_VerEdge_H264	Performs deblocking filtering on the vertical edges of the luma 16x16 macroblock.
FilterDeblockingLuma_VerEdge_MBAFF_H264	Performs deblocking filtering on the vertical edges of half of the luma 16x16 macroblock.
FilterDeblockingLuma_HorEdge_H264	Performs deblocking filtering on the horizontal edges of the luma 16x16 macroblock.
FilterDeblockingChroma_VerEdge_H264, FilterDeblockingChroma422VerEdge_H264	Performs deblocking filtering on the vertical edges of the chroma 8x8 macroblock.
FilterDeblockingChroma_VerEdge_MBAFF_H264, FilterDeblockingChroma422VerEdgeMBAFF_H264	Performs deblocking filtering on the vertical edges of half of the chroma 8x8 macroblock.
FilterDeblockingChroma_HorEdge_H264, FilterDeblockingChroma422HorEdge_H264	Performs deblocking filtering on the horizontal edges of the chroma 8x8 macroblock.

H.264 Decoder Functions

This section describes the functions for H.264 Decoder in accordance with JVT-G050 ([JVTG050]) standard.

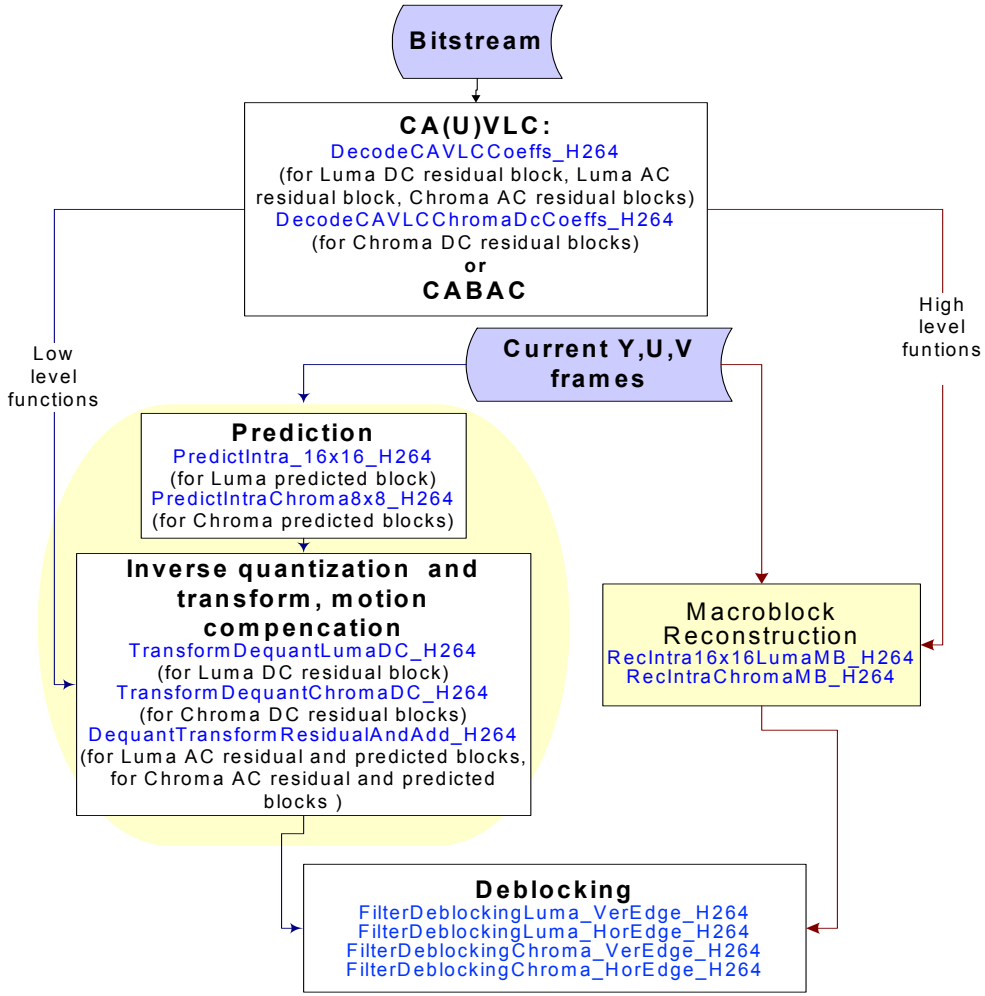
Figure 16-51 H.264 Decoder Structure



NOTE. This Motion Compensation process is performed by general motion compensation functions (See General Functions), which sum the predictor data and the residual and stores the result into the destination picture.

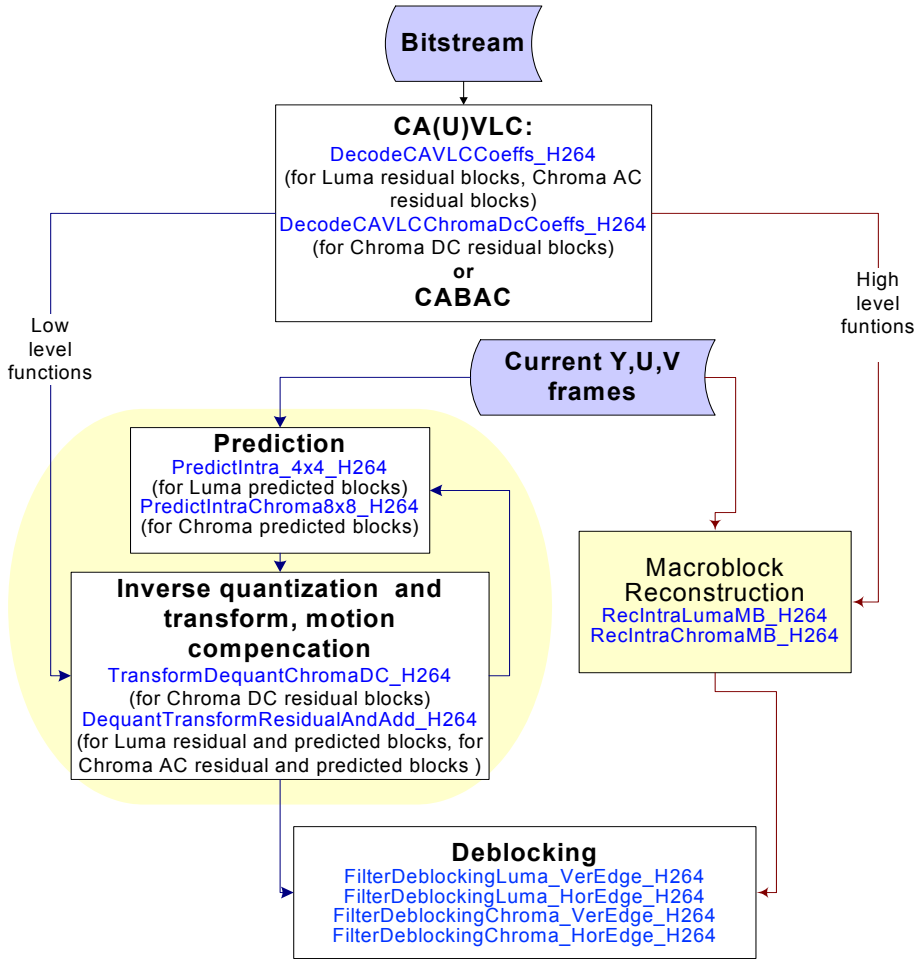
The following schemes show macroblock decoding with the help of video coding functions. The schemes for Intra16x16, intra 4x4, and P or B macroblock decoding show two paths of video coding functions using low and high level functions respectively. The blocks marked with yellow background color are interchangeable.

Figure 16-52 Intra 16x16 Macroblock Decoding



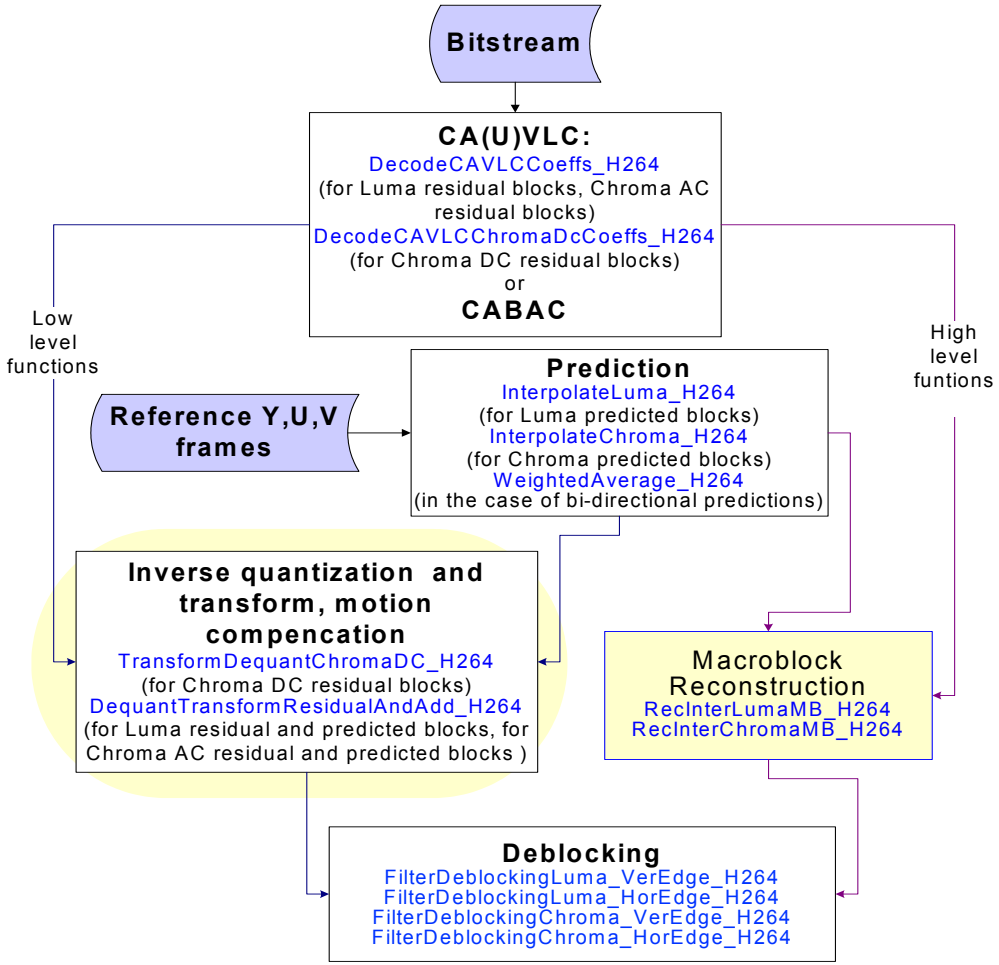
The following graph shows decoding process for I macroblock type in the case of Intra_4x4 prediction mode.

Figure 16-53 Intra 4x4 Macroblock Decoding



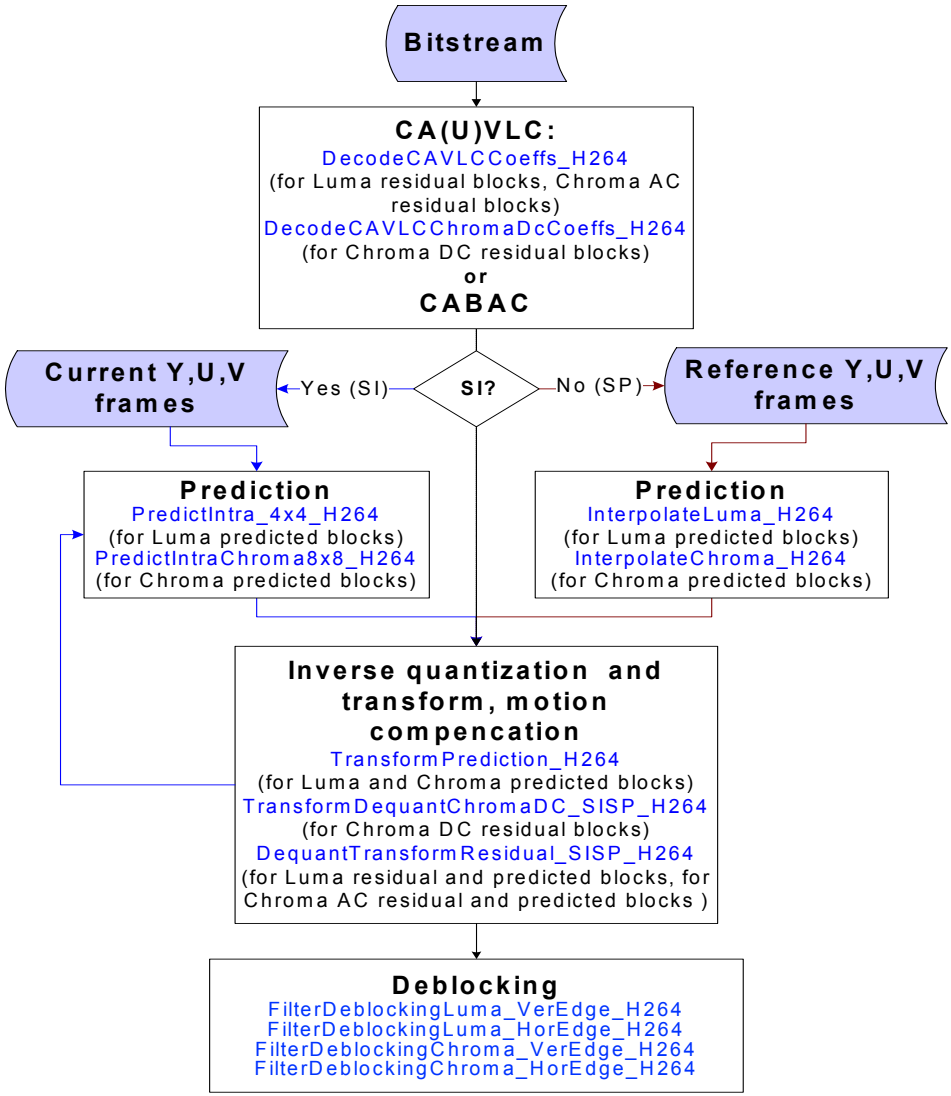
The following graph shows decoding process for P or B macroblock type.

Figure 16-54 P or B Macroblock Decoding



The following graph shows decoding process for SI or SP macroblock type. Note that SI is an intra macroblock and SP is a predicted macroblock but their decoding schemes are similar.

Figure 16-55 SI and SP Macroblocks Decoding



CAVLC Parsing

DecodeCAVLCCoeffs_H264

Decodes any non-chroma DC coefficients CAVLC coded.

Syntax

```
IppStatus ippiDecodeCAVLCCoeffs_H264_1u16s(Ipp32u** ppBitStream, Ipp32s*
    pBitOffset, Ipp16s* pNumCoeff, Ipp16s** ppDstCoeffs, const Ipp32u
    uVLCSelect, const Ipp16s uMaxNumCoeff, const Ipp32s** ppTblCoeffToken,
    Ipp32s** ppTblTotalZeros, Ipp32s** ppTblRunBefore, Ipp32s* pScanMatrix);
IppStatus ippiDecodeCAVLCCoeffs_H264_1u32s(Ipp32u** ppBitStream, Ipp32s*
    pBitOffset,Ipp16s*pNumCoeff,Ipp32s**ppDstCoeffs,Ipp32uuVLCSelect,
    Ipp16s uMaxNumCoeff, const Ipp32s** ppTblCoeffToken, const Ipp32s**
    ppTblTotalZeros, const Ipp32s** ppTblRunBefore, const Ipp32s* pScanMatrix);
```

Parameters

ppBitStream	Double pointer to the current position in the bitstream. The pointer is updated by the function.
pBitOffset	Pointer to offset between the bit that *ppBitStream* points to and the start of the code. The pointer is updated by the function.
pNumCoeff	Pointer to the output number of non-zero coefficients.
ppDstCoeffs	Double pointer to a 4x4 block of coefficients. These coefficients are calculated by the function. The function shifts pointer* *ppDstCoeffs* on 16.
uVLCSelect	Predictor on number of CoeffToken Table, which is designated in [JVTG050] as *nC*. It can be calculated in accordance with 9.2.1 of [JVTG050]
uMaxNumCoeff	Maximum coefficients in block (16 for intra 16x16, 15 for the rest).
ppTblCoeffToken	Double pointer to CoeffToken Tables.
ppTblTotalZeros	Double pointer to TotalZeros Tables.
ppTblRunBefore	Double pointer to RunBefore Tables.
pScanMatrix	Inverse scan matrix for coefficients in block.

Description

This function is declared in the `ippvc.h` header file. The functions `ippiDecodeCAVLCCoeffs_H264_1u16s` and `ippiDecodeCAVLCCoeffs_H264_1u32s` decode any non-chroma DC (chroma AC and Luma) coefficients CAVLC-coded in accordance with 9.2 of [JVTG050].

The table *pTblCoeffToken* is an array of size 4. Each element of this array is a pointer to a table that contains codes, number of trailing one transform coefficient and total number of non-zero transform coefficients in accordance with Table 9-5 of [JVTG050].

- *pTblCoeffToken*[0], where 0 ≤ *uVLCSelect* < 2;
- *pTblCoeffToken*[1], where 2 ≤ *uVLCSelect* < 4;
- *pTblCoeffToken*[2], where 4 ≤ *uVLCSelect* < 8;
- *pTblCoeffToken*[3], where *uVLCSelect* = -1 (used only for `DecodeCAVLCChromaDcCoeffs_H264`) ;
- *pTblCoeffToken*[4], where *uVLCSelect* = -2 (used only for `DecodeCAVLCChroma422DcCoeffs_H264`) ;
- If *uVLCSelect* > 8, this function uses its own table in accordance with table 9-5 of [JVTG050], *nC* > 8)

Use function `HuffmanRunLevelTableInitAlloc` for creation of tables *PTblCoeffToken*[i].

The table *ppTblTotalZeros* is an array of size 16. Each element of this array (except 0) is a pointer to a table that contains codes and values (`total_zeros` in [JVTG050]) in accordance with tables 9-7 and 9-8 of [JVTG050][JVTG050].

- *ppTblTotalZeros*[0] is not used.
- *ppTblTotalZeros*[i] contains codes and values that correspond to *TotalCoeff* (in [JVTG050]) = i, 0 < i < 16.

The table *ppTblRunBefore* is array of size 8. Each element of this array (except 0) is a pointer to a table that contains codes and values (`run_before` in [JVTG050]) in accordance with table 9-10 of [JVTG050].

- *ppTblRunBefore*[0] is not used.
- *ppTblRunBefore*[i] contains codes and values that correspond with *zerosLeft* (in [JVTG050]) =i, 0 < i < 7.
- *ppTblRunBefore*[7] contains codes and values that correspond with *zerosLeft* (in [JVTG050]) > 6.

Use function HuffmanTableInitAlloc for creation of tables *PpTblTotalZeros*[i], *PpTblRunBefore*[i].

WARNING. For proper operation of the function, remove emulation_prevention_three_byte described in 7.4.1 of [JVTG050] from the stream that undergoes decoding.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

DecodeCAVLCChromaDcCoeffs_H264, DecodeCAVLCChroma422DcCoeffs_H264

Decode chroma DC coefficients CAVLC coded.

Syntax

```
IppStatus ippiDecodeCAVLCChromaDcCoeffs_H264_1u16s(Ipp32u** ppBitStream,
    Ipp32s* pBitOffset, Ipp16s* pNumCoeff, Ipp16s** ppDstCoeffs, const Ipp32*
    pTblCoeffToken, const Ipp32s** ppTblTotalZerosCR, const Ipp32s**
    ppTblRunBefore);
IppStatus ippiDecodeCAVLCChromaDcCoeffs_H264_1u32s(Ipp32u** ppBitStream,
    Ipp32s* pBitOffset, Ipp16s* pNumCoeff, Ipp32s** ppDstCoeffs, const Ipp32s*
    pTblCoeffToken, const Ipp32s** ppTblTotalZerosCR, const Ipp32s**
    ppTblRunBefore);
IppStatus ippiDecodeCAVLCChroma422DcCoeffs_H264_1u16s(Ipp32u** ppBitStream,
    Ipp32s* pBitOffset, Ipp16s* pNumCoeff, Ipp16s** ppDstCoeffs, const Ipp32s*
    pTblCoeffToken, const Ipp32s** ppTblTotalZerosCR, const Ipp32s**
    ppTblRunBefore);
IppStatus ippiDecodeCAVLCChroma422DcCoeffs_H264_1u32s(Ipp32u** ppBitStream,
    Ipp32s* pBitOffset, Ipp16s* pNumCoeff, Ipp32s** ppDstCoeffs, const Ipp32s*
    pTblCoeffToken, const Ipp32s** ppTblTotalZerosCR, const Ipp32s**
    ppTblRunBefore);
```

Parameters

ppBitStream	Double pointer to the current position in the bitstream. The pointer is updated by the function.
pBitOffset	Pointer to offset between the bit that *ppBitStream* points to and the start of the code. The pointer is updated by the function.
pNumCoeff	Pointer to the output number of coefficients.
ppDstCoeffs	Double pointer to a 2x2 or 2x4 (for chroma 422) block of coefficients. These coefficients are calculated by the function. The function shifts pointer* *ppDstCoeffs* by 4 or 8 (for chroma 422).
pTblCoeffToken	Pointer to chroma DC CoeffToken Table.
ppTblTotalZerosCR	Double pointer to chroma DC TotalZeros Tables.
ppTblRunBefore	Double pointer to RunBefore Tables.

Description

These functions are declared in the `ippvc.h` header file. The functions `ippiDecodeCAVLCChromaDcCoeffs_H264_1u16s` and `ippiDecodeCAVLCChromaDcCoeffs_H264_1u32s` decode chroma DC coefficients and `ippiDecodeCAVLCChroma422DcCoeffs_H264_1u16s` and `ippiDecodeCAVLCChroma422DcCoeffs_H264_1u32s` decode Chroma422 (4:2:2 chroma mode) DC coefficients, CAVLC-coded in accordance with 9.2 of [JVTG050] in the case of *nC* = -1 and *nC* = -2 (in Chroma422 case).

pTblCoeffToken is equal to *ppTblCoeffToken*[3] or *ppTblCoeffToken*[4] for chroma422. *ppTblCoeffToken* and *ppTblRunBefore* tables are defined in the same way as in `DecodeCAVLCCoeffs_H264` function.

The table *ppTblTotalZerosCR* is an array of size 4 (size 8 for Chroma422). Each element of this array is a pointer to a table that contains codes and values (`total_zeros` in [JVTG050]) in accordance with tables 9-9 of [JVTG050].

- *ppTblTotalZerosCR*[0] is not used.
- *ppTblTotalZerosCR*[i] contains codes and values that correspond to *TotalCoeff* (in [JVTG050]) = i, 0 < i < 4 (0 < i < 8 for Chroma422).

WARNING. For proper operation of the function, remove `emulation_prevention_three_byte` described in 7.4.1 of [JVTG050] from the stream that undergoes decoding.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

DecodeExpGolombOne_H264

Decodes one Exp-Golomb code according to 9.1 H.264 standard.

Syntax

```
IppStatus ippiDecodeExpGolombOne_H264_1u16s(Ipp32u** ppBitStream, Ipp32s*
    pBitOffset, Ipp16s* pDst, Ipp8u isSigned);
```

Parameters

ppBitStream	Double pointer to the current position in the bitstream.
pBitOffset	Pointer to the offset between the bit pointed by *pBitStream* and the start of the code.
pDst	Pointer to the destination result.
isSigned	Flag that indicates if output value has to be decoded as signed.

Description

The function ippiDecodeExpGolombOne_H264_1u16s is declared in the ippvc.h header file. The function decodes one Exp-Golomb code in accordance with 9.1 H.264 standard.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one input pointer is NULL.

TransformDequantLumaDC_H264

Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.

Syntax

```
IppStatus ippiTransformDequantLumaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP);
```

Parameters

pSrcDst	Pointer to the initial coefficients and resultant DC (4x4 block).
QP	Quantization parameter.

Description

The function `ippiTransformDequantLumaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs integer inverse transformation and dequantization for 4x4 luma DC coefficients in accordance with 8.5.6 of [JVTG050].

The argument *QP* stands here for a luma quantization parameter, which is designated in JVT-G050 as QP_Y.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	Indicates an error condition if *QP* is less than 1 or greater than 51.

TransformDequantChromaDC_H264

Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.

Syntax

```
IppStatus ippiTransformDequantChromaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s
    QP);
```

Parameters

pSrcDst	Pointer to the initial coefficients and resultant chroma DC (2x2 block).
QP	Quantization parameter.

Description

The function `ippiTransformDequantChromaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in accordance with 8.5.7 of [JVTG050].

The argument *QP* stands here for a chroma quantization parameter, which is designated in JVT-G050 as QP_C.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	Indicates an error condition if *QP* is less than 1 or greater than 51.

TransformResidual4x4Inv_H264

Performs integer inverse transformation for a 4x4 block of residuals.

Syntax

```
IppStatus ippiTransformResidual4x4Inv_H264_16s_C1(const Ipp16s* pSrc, const
    Ipp16s* pDst);
IppStatus ippiTransformResidual4x4Inv_H264_32s_C1(const Ipp32s* pSrc, const
    Ipp32s* pDst);
```

Parameters

pSrc	Pointer to the initial residual coefficients 4x4 block (AC luma or AC chroma or luma) - source array of size 16.
pDst	Pointer to the resulting transformed 4x4 block - destination array of size 16.

Description

The functions `ippiTransformResidual4x4Inv_H264_16s_C1` and `ippiTransformResidual4x4Inv_H264_32s_C1` are declared in the `ippvc.h` file and perform inverse transformation for a residual 4x4 block described in 8.5.8 of [JVTG050].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

DequantTransformResidual_H264

Performs dequantization, integer inverse transformation, and shift for a 4x4 block of residuals.

Syntax

```
IppStatus ippiDequantTransformResidual_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s
    step, Ipp16s* pDC, Ipp32s AC, Ipp32s QP);
```

Parameters

pSrcDst	Pointer to the initial residual coefficients 4x4 block (AC luma or AC chroma or luma) and resultant transformed 4x4 block - source&destination array of size 16.
step	Distance in bytes between starts of the consecutive lines in the source/destination plane.
pDC	Pointer to the DC coefficient. If the case of luma block (not intra 16x16 macroblock type), *pDC* is set to NULL.
AC	Flag that is not equal to zero, if at least one AC coefficient exists, and is equal to zero otherwise. In the case of luma block, which is not intra 16x16 macroblock type, AC should be: • = 0, if only *pSrcDst*[0] is not equal to 0 and other components are equal to 0, • = 1, if any *pSrcDst*[*i*] is non-zero (*i* is within range [1, 15]).
QP	Quantization parameter for luma or for chroma.

Description

The function `ippiDequantTransformResidual_H264_16s_C1I` is declared in the `ippvc.h` file and performs scaling, inverse transformation, and shift for a residual 4x4 block described in 8.5.8 of [JVTG050].

For transformation of a chroma block this function is called after `TransformDequantChromaDC_H264`. In the case of 16x16 intra prediction mode, this function should be called after `TransformDequantLumaDC_H264` for transforming a luma block.

If *pDC* is not equal to NULL, the function, before starting the scaling and transformation process, places the DC coefficient specified by *pDC* in the top left corner of the source/destination array, which is designated as c_{00} in JVT-G050. Also during the scaling and transformation process the formula 8-266 ([JVTG050]) is used when calculating w_{00}.

If the pointer *pDC* is NULL, DC coefficient is dequantized like all the other coefficients. During the scaling and transformation process the formula 8-267 ([JVTG050]) is used when calculating w_{00}.

When calling the function, the argument *QP*, which is designated as *qP* in JVT-G050, should be calculated from the formulas 8-262 through 8-265 ([JVTG050]).

If *AC* is not equal to zero, formulas 8-278 through 8-277 ([JVTG050]) are applied to perform Inverse Transformations.

If *AC* = 0, all the coefficients are set equal to DC. The final shift is specified by formula 8-278.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	Indicates an error condition if *QP* is less than 1 or greater than 51.
ippStsStepErr	Indicates an error condition if *srcStep* value is less than 8.

DequantTransformResidualAndAdd_H264

Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.

Syntax

```
IppStatus ippiDequantTransformResidualAndAdd_H264_16s_C1I(const Ipp8u* pPred,
    Ipp16s*  pSrcDst, Ipp16s* pDC, Ipp8u* pDst, Ipp32s PredStep, Ipp32s DstStep,
    Ipp32s QP, Ipp32s AC);
```

Parameters

pPred	Pointer to the reference 4x4 block, which is used for intra prediction or motion compensation.
pSrcDst	If not equal to 0, pointer to the initial residual 4x4 block (AC luma or AC chroma or luma) and resultant transformed 4x4 block - array of size 16. If all coefficients of the residual AC block are equal to 0, *pSrcDst* should be 0 (for chroma block or luma block of 16x16 intra nacroblock type.
pDC	Pointer to the DC coefficient. If luma block is not intra 16x16 macroblock type, *pDC* is set to NULL.
pDst	Pointer to the destination 4x4 block.
PredStep	Distance in bytes between starts of the consecutive lines in the reference frame.
DstStep	Distance in bytes between starts of the consecutive lines in the destintion frame.
QP	Quantization parameter for luma and for chroma.
AC	Flag that is not equal to zero, if at least one AC coefficient exists, and is equal to zero otherwise. In the case of luma block, which is not intra 16x16 macroblock type, AC should be: • = 0, if only *pSrcDst*[0] is not equal to 0 and other components are equal to 0, • = 1, if any *pSrcDst*[*i*] is non-zero (*i* is within range [1, 15]).

Description

The function `ippiDequantTransformResidualAndAdd_H264_16s_C1I` is declared in the `ippvc.h` file and performs scaling, transformation, and shift for a residual 4x4 block described in 8.5.8 of [JVTG050] with subsequent intra prediction or motion compensation (the formula 8-245 of [JVTG050]).

For transformation of a chroma block this function should be called after `TransformDequantChromaDC_H264`. In the case of luma block (16x16 intra prediction mode), this function should be called after `TransformDequantLumaDC_H264` for transforming a luma block.

If *pDC* is not equal to NULL, the function, before starting the scaling and transformation process, places the DC coefficient specified by *pDC* in the top left corner of the initial coefficients block, which is designated as c_{00} in JVT-G050. Also during the scaling and transformation process the formula 8-266 ([JVTG050]) is used when calculating w_{00}.

If the pointer *pDC* is NULL, DC coefficient is dequantized like all the other coefficients. During the scaling and transformation process the formula 8-267 ([JVTG050]) is used when calculating w_{00}.

When calling the function, the argument *QP*, which is designated as *qP* in JVT-G050, should be calculated from the formulas 8-262 through 8-265 ([JVTG050]).

If *AC* is not equal to zero, formulas 8-278 through 8-277 ([JVTG050]) are applied to perform Inverse Transformations.

If *AC* = 0, all the coefficients are set equal to DC. The final shift is specified by formula 8-278.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	Indicates an error condition if *QP* is less than 1 or greater than 51.

TransformPrediction_H264

Performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks.

Syntax

```
IppStatus ippiTransformPrediction_H264_8u16s_C1(const Ipp8u* pSrc, Ipp32s step,
    Ipp16s* pDst);
```

Parameters

pSrc	Pointer to the array of inter prediction samples for the current macroblock (4x4 block).
step	Distance in bytes between starts of the consecutive lines in the source buffer.

pDst	Destination array of size 16.

Description

The function `ippiTransformPrediction_H264_8u16s_C1` is declared in the `ippvc.h` file. The function performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks (the formula 8-286 of [JVTG050]).

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

DequantTransformResidual_SISP_H264

Performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks.

Syntax

```
IppStatus ippiDequantTransformResidual_SISP_H264_16s_C1I(Ipp16s* pSrcDst, const
    Ipp16s* pPredictBlock, const Ipp16s* pDC, Ipp32s AC, Ipp32s qp, Ipp32s qs,
    Ipp32s Switch);
```

Parameters

pSrcDst	Pointer to array (size 16) of the prediction residual transform coefficient (luma or chroma) - source/destination array.
pPredictBlock	Pointer to array (size 16) of the of inter prediction samples for the current macroblock after inverse transform. They can be calculated using `TransformPrediction_H264` function.
pDC	Pointer to the DC coefficient. In the case of luma block type *pDC* must be 0. In the case of chroma block pDC is pointer to array (size 4) of ChromaDC after inverse transform. They can be calculated using `TransformDequantChromaDC_SISP_H264` function.

AC	Flag that is not equal to zero, if at least one AC coefficient exists, and is equal to zero otherwise.
qp	Quantization parameter *qp*, which is used in the case of non-switching pictures.
qs	Quantization parameter *qs*.
Switch	Flag that is not equal to zero, if it switches pictures, and is equal to zero otherwise.

Description

The function `ippiDequantTransformResidual_SISP_H264_16s_C1I` is declared in the `ippvc.h` file. The function performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks (8-6 of [JVTG050]).

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

TransformDequantChromaDC_SISP_H264

Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in P macroblocks in SP slices or SI macroblocks.

Syntax

```
IppStatus ippiTransformDequantChromaDC_SISP_H264_16s_C1I(Ipp16s* pSrcDst, const
    Ipp16s* pDCPredict, Ipp32s qp, Ipp32s qs, Ipp32s Switch);
```

Parameters

pSrcDst	Pointer to array (size 4) of the prediction residual transform chroma DC coefficient - source/destination array.

pDCPredict	Pointer to array (size 4) of the of inter prediction DC samples for the current macroblock after inverse transform. Inverse transform of the inter prediction samples can be calculated using TransformPrediction_H264 function for each 4x4 block.
qp	Quantization parameter (*QP.* in [JVTG050]), which is used in the case of non-switching pictures.
qs	Quantization parameter (*QS.* in [JVTG050]).
Switch	Flag that is not equal to zero, if it switches pictures, and is equal to zero otherwise.

Description

The function `ippiTransformDequantChromaDC_SISP_H264_16s_C1I` is declared in the `ippvc.h` file. The function performs integer inverse transformation and dequantization for 2x2 chroma coefficients in P macroblocks in SP slices or SI macroblocks (8.6.1.2, 8.6.2.2 of [JVTG050]).

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

Intra Prediction

PredictIntra_4x4_H264

Performs intra prediction for a 4x4 luma component.

Syntax

```
IppStatus ippiPredictIntra_4x4_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
    IppIntra4x4PredMode predMode, Ipp32s availability);
```

Parameters

pSrcDst	Pointer to the source and destination array.
srcdstStep	Distance in bytes between starts of the consecutive lines in the source and destination arrays. (Y-frame width in bytes).
predMode	Prediction mode of the `Intra_4x4` prediction process for luma samples.
availability	Flag that specifies availability of the samples used for prediction.

Description

The function `ippiPredictIntra_4x4_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [JVTG050] in a given prediction mode *predMode* (See enumeration `IppIntra4x4PredMode`).

The flag *availability* is computed as follows:

`B1*IPP_LEFT + B2*IPP_UPPER_LEFT + B3*IPP_UPPER + B4*IPP_UPPER_RIGHT,`

where the constants `IPP_LEFT`, `IPP_UPPER_LEFT`, `IPP_UPPER`, `IPP_UPPER_RIGHT` are from `IppLayoutFlag` and the variables `B1`, `B2`, `B3`, `B4` may take the values of

0 - if the block is unavailable for 4x4 prediction,

1 - if the block is available for 4x4 prediction.

Figure 16-56 Layout of Blocks Used for Prediction

2- IPP_UPPER_LEFT	3- IPP_UPPER	4- IPP_UPPER_RIGHT
1- IPP_LEFT	*SrcDst*	

Example 16-16 PredictIntra_4x4_H264 Usage Example

```
// declare block parameters structure
typedef struct H264Block4x4Info_t
{
    Ipp32s x;
    Ipp32s y;
    Ipp32s existNeighbour;
    Ipp32s absentNeighbour;
```

Example 16-16 PredictIntra_4x4_H264 Usage Example (continued)

```
} H264Block4x4Info_t;

// define parameters for all 16 blocks of a macroblock
static
H264Block4x4Info_t H264Block4x4Info[16] =
{
    { 0,  0,  0,  0},
    { 4,  0,  IPP_LEFT,  0},
    { 0,  4,  IPP_UPPER | IPP_UPPER_RIGHT,  0},
    { 4,  4,  IPP_UPPER | IPP_UPPER_LEFT,  IPP_UPPER_RIGHT},

    { 8,  0,  IPP_LEFT,  0},
    {12,  0,  IPP_LEFT,  0},
    { 8,  4,  IPP_UPPER | IPP_UPPER_LEFT | IPP_UPPER_RIGHT,  0},
    {12,  4,  IPP_UPPER | IPP_UPPER_LEFT,  IPP_UPPER_RIGHT},

    { 0,  8,  IPP_UPPER | IPP_UPPER_RIGHT,  0},
    { 4,  8,  IPP_UPPER | IPP_UPPER_LEFT | IPP_UPPER_RIGHT,  0},
    { 0, 12,  IPP_UPPER | IPP_UPPER_RIGHT,  0},
    { 4, 12,  IPP_UPPER | IPP_UPPER_LEFT,  IPP_UPPER_RIGHT},

    { 8,  8,  IPP_UPPER | IPP_UPPER_LEFT | IPP_UPPER_RIGHT,  0},
    {12,  8,  IPP_UPPER | IPP_UPPER_LEFT,  IPP_UPPER_RIGHT},
    { 8, 12,  IPP_UPPER | IPP_UPPER_LEFT | IPP_UPPER_RIGHT,  0},
    {12, 12,  IPP_UPPER | IPP_UPPER_LEFT,  IPP_UPPER_RIGHT}
};

void DoIntraPrediction4x4Granulation(void)
{
    // given variables
    Ipp32s mbX, mbY, mbWidth;
    Ipp8u* pMb;
    Ipp32s imageStep;
    // working variables
    Ipp32u luma4x4BlkIdx;
    Ipp32s edgeType;
    Ipp32s mbAvailability;

    //...

    // calculate neighbouring macroblock availability
    edgeType = 0;
    if (mbX)
    {
        if (left_block_belongs_the_same_slice)
            edgeType |= IPP_LEFT;
    }
    if (mbY)
    {
        if ((mbX - 1 < mbWidth) &&
            (top_right_block_belongs_the_same_slice))
```

Example 16-16 PredictIntra_4x4_H264 Usage Example (continued)

```
                mbAvailability |= IPP_UPPER_RIGHT;
            if (top_block_belongs_the_same_slice)
                mbAvailability |= IPP_UPPER;
            if ((mbX) &&
                (top_left_block_belongs_the_same_slice))
                mbAvailability |= IPP_UPPER_LEFT;
        }

        // main working cycle
        for (luma4x4BlkIdx = 0; luma4x4BlkIdx < 16; luma4x4BlkIdx += 1)
        {
            IppIntra4x4PredMode_H264 predMode;
            Ipp32s blockAvailability;
            Ipp8u* pBlock4x4;

            // get intra prediction mode
            predMode = (IppIntra4x4PredMode_H264)
Intra4x4PredMode[luma4x4BlkIdx];

            // get block parameters
            {
                H264Block4x4Info_t blockInfo =
H264Block4x4Info[luma4x4BlkIdx];

                // get block pointer
                pBlock4x4 = pMb + blockInfo.y*  imageStep + blockInfo.x;

                // get block's neighbours
                blockAvailability = mbAvailability &
~(blockInfo.absentNeighbour);
                blockAvailability |= blockInfo.existNeighbour;
            }

            // do intra prediction
            ippiPredictIntra_4x4_H264_8u_C1IR(pBlock4x4,
                                              imageStep,
                                              predMode,
                                              blockAvailability);

            // do block reconstruction
            // ...
        }

} // void DoIntraPrediction4x4Granulation(void)
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

ippStsStepErr	Indicates an error condition if *srcdstStep* value is less than 4.
ippStsOutOfRangeErr	Indicates an error condition if *predMode* value falls outside [0,8].
ippStsResFloor	Indicates a warning condition if *predMode* is not allowed for this block.

PredictIntra_16x16_H264

Performs intra prediction for a 16x16 luma component.

Syntax

```
IppStatus ippiPredictIntra_16x16_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s
    srcdstStep, IppIntra16x16PredMode predMode, Ipp32s availability);
```

Parameters

pSrcDst	Pointer to the destination array.
srcdstStep	Distance in bytes between starts of the consecutive lines in the destination array.
predMode	Prediction mode of the `Intra_16x16` prediction process for luma samples.
availability	Flag that specifies the availability of the macroblocks used for prediction.

Description

The function `ippiPredictIntra_16x16_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs intra prediction for a 16x16 luma component in accordance with 8.3.2 of [JVTG050] in a given prediction mode *predMode* (See enumeration `IppIntra16x16PredMode`).

The flag *availability* is computed as follows:

```
B1*IPP_LEFT + B2*IPP_UPPER_LEFT + B3*IPP_UPPER,
```

where the constants `IPP_LEFT`, `IPP_UPPER_LEFT`, `IPP_UPPER` are from `IppLayoutFlag` and the variables `B1`, `B2`, `B3` may take the values of

0 - if the macroblock is unavailable for 16x16 prediction,

1 - if the macroblock is available for 16x16 prediction.

Figure 16-57 Layout of Blocks Used for Prediction

2- IPP_UPPER_LEFT	3- IPP_UPPER
1- IPP_LEFT	*SrcDst*

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if *srcdstStep* value is less than 16.
ippStsOutOfRangeErr	Indicates an error condition if *predMode* value falls outside [0,3].
ippStsLPCCalcErr	Indicates an error condition if *predMode* is not allowed for this block.

PredictIntraChroma8x8_H264

Performs intra prediction for a 8x8 chroma component.

Syntax

```
IppStatus ippiPredictIntraChroma8x8_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s
    srcdstStep, IppIntraChroma8x8PredMode predMode, Ipp32s availability);
```

Parameters

pSrcDst	Pointer to the 8x8 chroma-block, which is inside U-frame or V-frame and which is under reconstruction (destination array).
srcdstStep	Distance in bytes between starts of the consecutive lines in the destination chroma-frame.
predMode	Prediction mode of the Intra_chroma prediction process for chroma samples.

availability	Flag that specifies the availability of the macroblocks used for prediction.

Description

The function `ippiPredictIntraChroma8x8_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs intra prediction for a 8x8 chroma component in accordance with 8.3.3 of [JVTG050] in a given prediction mode *predMode* (See enumeration `IppIntraChromaPredMode_H264`).

The flag *availability* is computed as follows:

`B1*IPP_LEFT + B2*IPP_UPPER_LEFT + B3*IPP_UPPER,`

where the constants `IPP_LEFT`, `IPP_UPPER_LEFT`, `IPP_UPPER` are from `IppLayoutFlag` and the variables `B1`, `B2`, `B3` may take the values of

0 - if the macroblock is unavailable for intra chroma prediction,

1 - if the macroblock is available for intra chroma prediction.

Figure 16-58 Layout of Blocks Used for Prediction

2- IPP_UPPER_LEFT	3- IPP_UPPER
1- IPP_LEFT	*SrcDst*

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	Indicates an error condition if *predMode* value falls outside [0,3].
`ippStsStepErr`	Indicates an error condition if *srcdstStep* value is less than 8.
`ippStsLPCCalcErr`	Indicates an error condition if *predMode* is not allowed for this block.

Inter Prediction

Note that the functions with _16u_ and _8u_ data types use the following structures respectively:

```
typedef struct _IppVCInterpolate_16u
{
    const Ipp16u* pSrc;
    Ipp32s      srcStep;
    Ipp16u*     pDst;
    Ipp32s      dstStep;
    Ipp32s      dx;
    Ipp32s      dy;
    IppiSize    roiSize;
    Ipp32s      bitDepth;
} IppVCInterpolate_16u;

typedef struct _IppVCInterpolate_8u
{
    const Ipp8u* pSrc;
    Ipp32s      srcStep;
    Ipp8u*      pDst;
    Ipp32s      dstStep;
    Ipp32s      dx;
    Ipp32s      dy;
    IppiSize    roiSize;
    Ipp32s      roundControl;
} IppVCInterpolate_8u;
```

where

pSrc	Pointer to the source.
srcStep	Distance in items between starts of the consecutive lines in the source image.
pDst	Pointer to the destination.
dstStep	Distance in items between starts of the consecutive lines in the destination image.

dx, dy	Fractional parts of the motion vector in 1/4 pel units for luma (0, 1, 2, or 3) and 1/8 pel units for chroma (0, 1, ..., 7)
roiSize	Flag that specifies the dimensions of the region of interest. See structure IppiSize.
roundControl	Frame level rounding control value as described in section 8.3.7 of [SMPTE 421M], should be equal to 0 or 1. (For VC1 only, reserved for H.264)
bitDepth	Number of bits of the plane sample with the range [8.. 14].

```
typedef struct _IppVCBidir_16u
{
    const Ipp16u*  pSrc1;
    Ipp32s   srcStep1;
    const Ipp16u*  pSrc2;
    Ipp32s   srcStep2;
    Ipp16u*  pDst;
    Ipp32s   dstStep;
    IppiSize roi;
    Ipp32s   bitDepth;
} IppVCBidir_16u;

typedef struct _IppVCBidir_8u
{
    const Ipp8u*  pSrc1;
    Ipp32s   srcStep1;
    const Ipp8u*  pSrc2;
    Ipp32s   srcStep2;
    Ipp8u*  pDst;
    Ipp32s   dstStep;
    IppiSize roiSize;
} IppVCBidir_8u;
```

where

pSrc1	Pointer to the first input block

srcStep1	Distance in items between starts of the consecutive lines in the first input block.
pSrc2	Pointer to the second input block.
srcStep2	Distance in items between starts of the consecutive lines in the second input block.
pDst	Pointer to the first output block.
dstStep	Distance in items between starts of the consecutive lines in the destination block.
roiSize	Flag that specifies the dimensions of the region of interest. See structure IppiSize.
bitDepth	Number of bits of the plane sample with range [8.. 14].

The InterpolateLumaBlock_H264 and InterpolateChromaBlock_H264 functions use the following structures:

```
typedef struct _IppVCInterpolateBlock_16u
{
    const Ipp16u* pSrc[2];/* pointers to reference image planes*/
    Ipp32s srcStep;       /* step of the reference image planes*/
    Ipp16u* pDst[2];      /* pointers to destination image planes*/
    Ipp32s dstStep;       /* step of the destination image planes*/
    IppiSize sizeFrame;   /* dimensions of the reference image planes*/
    IppiSize sizeBlock;   /* dimensions of the block to be interpolated*/
    IppiPoint pointBlockPos; /* current position of the block in the
                              image being interpolated*/
    IppiPoint pointVector;   /* relative difference between current
                              position and reference data to be used*/
    Ipp32s bitDepth          /* data capacity depth in range 8..14*/
} IppVCInterpolateBlock_16u;

typedef struct _IppVCInterpolate_8u
{
   const Ipp8u* pSrc[2];  /* pointers to reference image planes*/
    Ipp32s srcStep;       /* step of the reference image planes*/
```

```
    Ipp8u* pDst[2];       /* pointers to destination image planes*/
    Ipp32s dstStep;       /* step of the destination image planes*/
    IppiSize sizeFrame;   /* dimensions of the reference image planes*/
    IppiSize sizeBlock;   /* dimensions of the block to be interpolated*/
    IppiPoint pointBlockPos; /* current position of the block in the
                              image being interpolated*/
    IppiPoint pointVector;   /* relative difference between current
                              position and reference data to be used*/
} IppVCInterpolate_8u;
```

where

pSrc	Array of pointers to reference planes; `InterpolateLumaBlock_H264` uses only first pointer of the array, `InterpolateChromaBlock_H264` uses both pointers.
srcStep	Stride of the reference planes (the distance between the nearest image lines)
pDst	Array of pointers to destination buffers; `InterpolateLumaBlock_H264` uses only first pointer of the array, `InterpolateChromaBlock_H264` uses both pointers. The array should be big enough to hold the interpolated block.
dstStep	Stride of the destination buffer (the distance between the nearest image lines)
sizeFrame	Dimensions of the reference planes that are pointed to by the source pointers
sizeBlock	Dimentions of the block to interpolate
pointBlockPos	Current position of the block in the frame under reconstruction
pointVector	Motion vector
bitDepth	Bit capacity of data that is being processed.

ExpandPlane_H264

Expands plane.

Syntax

```
IppStatus ippiExpandPlane_H264_8u_C1R(Ipp8u* StartPtr, Ipp32u uFrameWidth,
    Ipp32u uFrameHeight, Ipp32u uPitch, Ipp32u uPels, IppvcFrameFieldFlag
    uFrameFieldFlag);
```

Parameters

StartPtr	Pointer to the frame source data.
uFrameWidth	Frame width in pixels.
uFrameHeight	Frame height in pixels.
uPitch	Frame width in bytes.
uPels	Number of pixels filled to the right/left and up/down in process of expansion.
uFrameFieldFlag	Flag that defines expansion method for various image types: frame, top field, bottom field. See the corresponding enumeration IPPVC_FRAME_FIELD_FLAG.

Description

The function ippiExpandPlane_H264_8u_C1R is declared in the ippvc.h file. This function expands the plane by adding pixels to the top and bottom rows and to the left and right columns of the frame.

Example 16-17 ExpandPlane_H264 Usage Example

```
{
    Ipp8u*  pPlane = pointer_on_frame;
    Ipp32s width, height, picth, padding;

    width = width_of_frame; // without padding
    height = height_of_frame; // without padding
    picth = picth_of_frame; // with padding

    padding = 32; // padding size.

    IppvcFrameFieldFlag ff_flag;
```

Example 16-17 ExpandPlane_H264 Usage Example (continued)

```
        if (!is_field_flag)
        {
            ff_flag = IPPVC_FRAME;
        }
        else
        {
            if(!is_bottom_field)
            {
                ff_flag = IPPVC_TOP_FIELD;
            }
            else
            {
                ff_flag = IPPVC_BOTTOM_FIELD;
            }
        }

        ippiExpandPlane_H264_8u_C1R(pPlane,
                                    width,
                                    height,
                                    picth,
                                    padding,
                                    ff_flag);

    }
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if *StartPtr* is NULL.

InterpolateLuma_H264

Performs interpolation for motion estimation of the luma component using fractional part of motion vector.

Syntax

```
IppStatus ippiInterpolateLuma_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep,
    Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, IppiSize roiSize);
IppStatus ippiInterpolateLuma_H264_16u_C1R(const IppVCInterpolate_16u*
    interpolateInfo);
```

Parameters

pSrc	Pointer to the source.
srcStep	Distance in items between starts of the consecutive lines in the source block.
pDst	Pointer to the destination.
dstStep	Distance in items between starts of the consecutive lines in the destination block.
dx, dy	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).
roiSize	Flag that specifies the dimensions of the region of interest (could be 16, 8 or 4 in each dimension). See structure IppiSize.
interpolateInfo	Pointer to the IppVCInterpolate_16u structure.

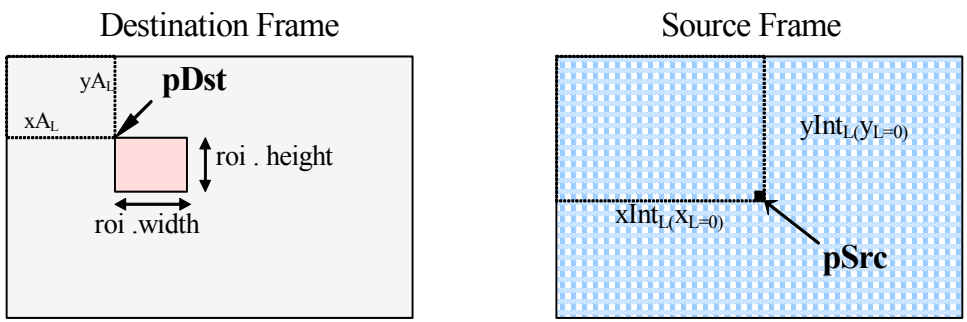
Description

The functions ippiInterpolateLuma_H264_8u_C1R and ippiInterpolateLuma_H264_16u_C1R are declared in the ippvc.h file. These functions perform interpolation (convolution with 6x6 kernel) for motion estimation of the luma component in accordance with 8.4.2.2.1 of [JVTG050].

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

The ippiInterpolateLuma_H264 functions do not provide any padding, therefore the whole data region indicated by the pointer should be real and increased by padding, if necessary. The functions can use up to 3 pixels up and left from the pointer and maximum 2 pixels down and right from the pointed data size.

Figure 16-59 Interpolation for Motion Estimation of Luma Component



NOTE. Make sure all samples that are used for interpolation are within the boundaries of the source frame. To solve the problem, enlarge the source frame by using boundary samples or use `InterpolateLumaTop_H264` or `InterpolateLumaBottom_H264` functions.

Example 16-18 Usage of `ippiInterpolateLuma_H264`

```
Ipp32s iHVector, iVVector;
IppiSize roi = {width_of_luma_block, height_of_luma_block};
Ipp32s iLumaXPos = x_position_of_current_luma_block;
Ipp32s iLumaYPos = y_position_of_current_luma_block;
Ipp8u* pRefYPlane = corresponding_luma_block_of_reference_luma_frame;
Ipp32s iPitch = reference_luma_frame_pitch;
Ipp8u* pDstY = destination_luma_block;
Ipp32s iDstPitch = destination_luma_frame_pitch;

// set motion vectors
iHVector = horizontal_motion_vector;
iVVector = vertical_motion_vector;

{
    Ipp32s iHFraction, iVFraction;
    Ipp32s width, height;

    width = width_of_frame;
    height = height_of_frame;

    // prepare the horizontal vector, extract an interpolation factor
    iHFraction = iHVector & 3;
    iHVector >>= 2;

    // saturate the horizontal vector
    {
        Ipp32s iLeft = ((iHFraction) ? (1) : (0));
        Ipp32s iRight;

        // set amount of additional pixels
        iRight = iLeft*  3;
        iLeft = iLeft*  2;

        if (((unsigned) (width - (iRight + iLeft) - roi.width)) <
            ((unsigned) (iHVector + iLumaXPos - iLeft)))
        {
            // saturate to the left bound
            if (- iRight - roi.width >
                iHVector + iLumaXPos)
            {
```

Example 16-18 Usage of `ippiInterpolateLuma_H264` (continued)

```
                iHVector = - iLumaXPos - roi.width - 3;
            }
            // saturate to the right bound
            else if (iLeft + width <
                     iHVector + iLumaXPos)
            {
                iHVector = width - iLumaXPos + 2;
            }
        }
    }

    // prepare the vertical vector, extract an interpolation factor
    iVFraction = iVVector & 3;
    iVVector >>= 2;

    // most usual prediction
    // NOTE: the height is decremented to avoid factor's influence
    {
        Ipp32s iTop = ((iVFraction) ? (1) : (0));
        Ipp32s iBottom;

        // set amount of additional pixels
        iBottom = iTop*  3;
        iTop = iTop*  2;

        if (((unsigned) (height - (iTop + iBottom) - roi.height)) >=
            ((unsigned) (iVVector + iLumaYPos - iTop)))
        {
            ippiInterpolateLuma_H264_8u_C1R(pRefYPlane + iVVector*  iPitch +
iHVector,
                                            iPitch,
                                            pDstY,
                                            iDstPitch,
                                            iHFraction,
                                            iVFraction,
                                            roi);
        }
        // prediction from the top of the frame
        else if (0 > iVVector + iLumaYPos - iTop)
        {
            // sometimes the vector points to nothing
           ippiInterpolateLumaTop_H264_8u_C1R(pRefYPlane + iVVector*  iPitch +
iHVector,
                                               iPitch,
                                               pDstY,
                                               iDstPitch,
                                               iHFraction,
                                               iVFraction,
                                               - iVVector - iLumaYPos,
                                               roi);
        }
```

Example 16-18 Usage of `ippiInterpolateLuma_H264` (continued)

```
        // prediction from the bottom of the frame
        else
        {
            // sometimes the vector points to nothing
           ippiInterpolateLumaBottom_H264_8u_C1R(pRefYPlane + iVVector*  iPitch
+ iHVector,
                                                    iPitch,
                                                    pDstY,
                                                    iDstPitch,
                                                    iHFraction,
                                                    iVFraction,
                                                iVVector + iLumaYPos + roi.height
- height,
                                                    roi);
        }
    }
}
```

These functions are used in the H.264 decoder and encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *`srcStep`* value is less than 16.
`ippStsBadArgErr`	Indicates an error condition if *`dx`* or *`dy`* values fall outside [0,3].
`ippStsSizeErr`	Indicates an error condition if *`roi`*`.width` or *`roi`*`.height` take values other than {16, 8, 4}.

InterpolateLumaTop_H264

Performs interpolation for motion estimation of the luma component at the frame top boundary.

Syntax

```
IppStatus ippiInterpolateLumaTop_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep,
    Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s outPixels,
    IppiSize roiSize);
```

```
IppStatus ippiInterpolateLumaTop_H264_16u_C1R(const IppVCInterpolate_16u*
    interpolateInfo, Ipp32s outPixels);
```

Parameters

pSrc	Pointer to the source.
srcStep	Distance in items between starts of the consecutive lines in the source image.
pDst	Pointer to the destination.
dstStep	Distance in items between starts of the consecutive lines in the destination image.
dx, dy	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).
outPixels	Number of pixels by which the data specified by *pSrc* reaches over the frame top boundary.
roiSize	Flag that specifies the dimensions of the region of interest (could be 16, 8 or 4 in each dimension). See structure IppiSize.
interpolateInfo	Pointer to the IppVCInterpolate_16u structure.

Description

The functions ippiInterpolateLumaTop_H264_8u_C1R and ippiInterpolateLumaTop_H264_16u_C1R are declared in the ippvc.h file. These functions perform interpolation (convolution with 6x6 kernel) for motion estimation of the luma component near the top boundary of the frame in accordance with 8.4.2.2.1 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

The functions are intended for processing data close to the frame top border and do not require any vertical padding. Horizontal padding is required. The function can use up to 3 pixels left from the pointer and maximum 2 pixels right from the pointed data size.

See Example 16-18 for a usage example.

Figure 16-60 shows the data specified by *pSrc* and the data actually used in interpixel interpolation at the picture top boundary with a positive *outPixels* value. Figure 16-61 shows the data with a negative *outPixels* value.

Figure 16-60 Luma Interpolation at Top Boundary with Positive *outPixels* Value

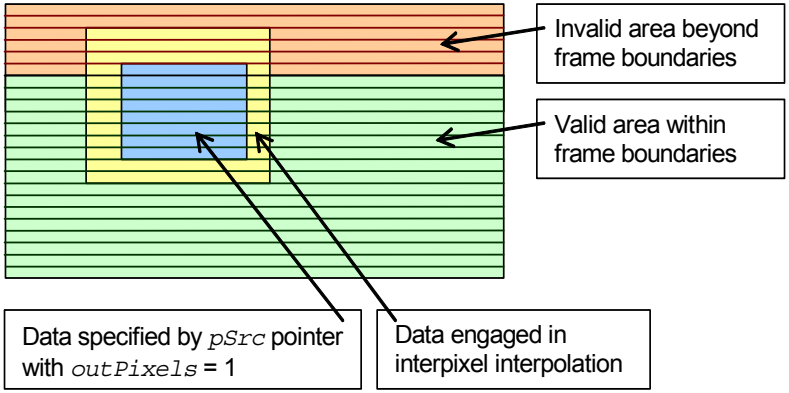
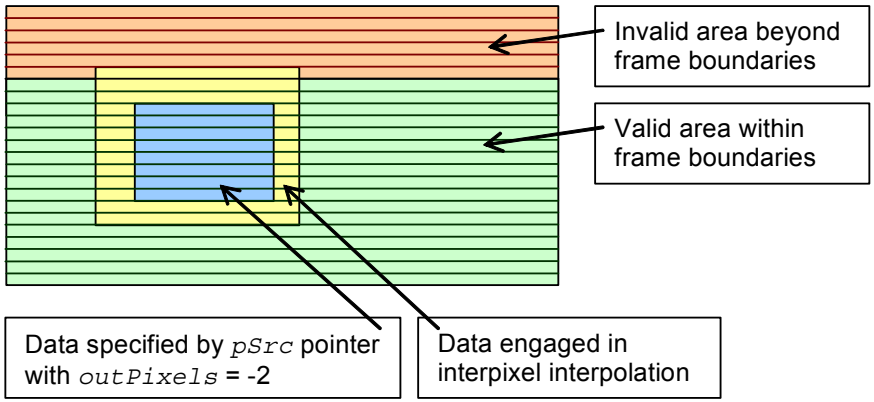


Figure 16-61 Luma Interpolation at Top Boundary with Negative *outPixels* Value



Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* value is less than 16.
ippStsBadArgErr	Indicates an error condition if *dx* or *dy* values fall outside [0,3].
ippStsSizeErr	Indicates an error condition if *roi*.width or *roi*.height take values other than {16, 8, 4}.

InterpolateLumaBottom_H264

Performs interpolation for motion estimation of the luma component at the frame bottom boundary.

Syntax

```
IppStatus ippiInterpolateLumaBottom_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s
    outPixels, IppiSize roiSize);
IppStatus ippiInterpolateLumaBottom_H264_16u_C1R(const IppVCInterpolate_16u*
    interpolateInfo, Ipp32s outPixels);
```

Parameters

pSrc	Pointer to the source.
srcStep	Distance in items between starts of the consecutive lines in the source block.
pDst	Pointer to the destination.
dstStep	Distance in items between starts of the consecutive lines in the destination block.
dx, dy	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).
outPixels	Number of pixels by which the data specified by *pSrc* reaches over the frame bottom boundary.
roiSize	Flag that specifies the dimensions of the region of interest (could be 16, 8 or 4 in each dimension). See structure IppiSize.
interpolateInfo	Pointer to the IppVCInterpolate_16u structure.

Description

The functions `ippiInterpolateLumaBottom_H264_8u_C1R` and `ippiInterpolateLumaBottom_H264_16u_C1R` are declared in the `ippvc.h` file. These functions perform interpolation (convolution with 6x6 kernel) for motion estimation of the luma component near the bottom boundary of the frame in accordance with 8.4.2.2.1 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

See Example 16-18 for a usage example.

Figure 16-62 shows the data specified by *pSrc* and the data actually used in interpixel interpolation at the picture top boundary with a positive *outPixels* value. Figure 16-63 shows the data with a negative *outPixels* value.

Figure 16-62 Luma Interpolation at Bottom Boundary with Positive *outPixels* Value

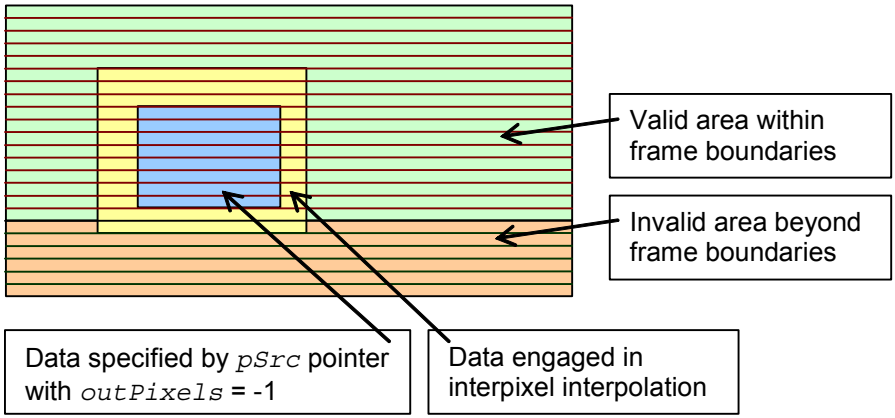
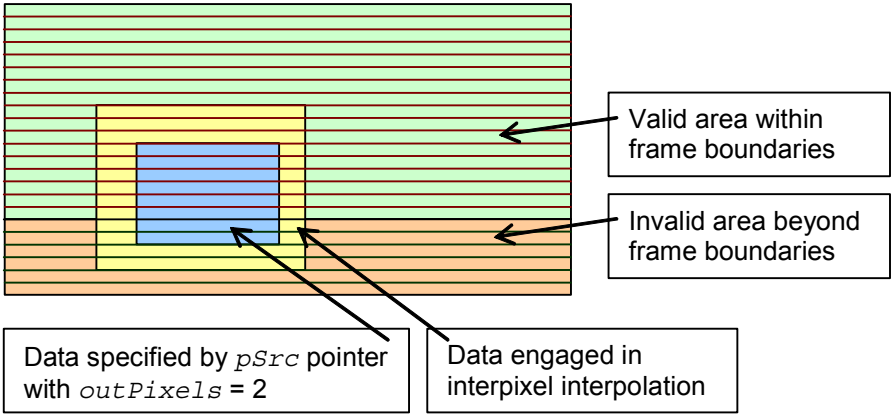


Figure 16-63 Luma Interpolation at Bottom Boundary with Negative *outPixels* Value



Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* value is less than 16.
ippStsBadArgErr	Indicates an error condition if *dx* or *dy* values fall outside [0,3].
ippStsSizeErr	Indicates an error condition if *roi*.width or *roi*.height take values other than {16, 8, 4}.

InterpolateLumaBlock_H264

Performs interpolation for motion estimation of the luma component using entire motion vector.

Syntax

```
IppStatus ippiInterpolateLumaBlock_H264_8u_P1R(const IppVCInterpolateBlock_8u*
    interpolateInfo);
IppStatus ippiInterpolateLumaBlock_H264_16u_P1R(const
    IppVCInterpolateBlock_16u* interpolateInfo);
```

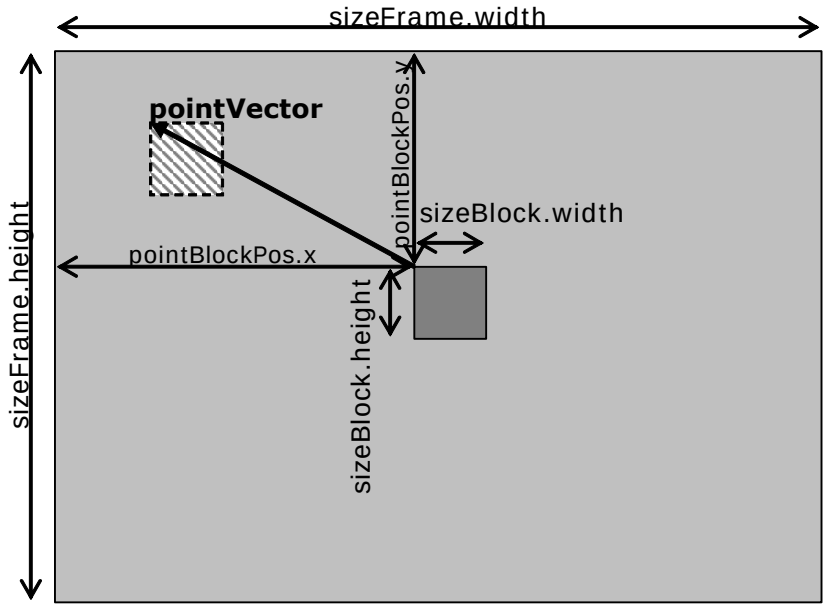
Parameters

interpolateInfo	Pointer to an instance of the structure holding interpolation parameters. See IppVCInterpolateBlock_8u and IppVCInterpolateBlock_16u structures.

Description

The functions ippiInterpolateLumaBlock_H264_8u_P1R and ippiInterpolateLumaBlock_H264_16u_P1R are declared in the ippvc.h file. These functions perform interpolation (convolution with 6x6 kernel) for motion estimation of the luma component in accordance with 8.4.2.2.1 of [JVTG050]. Unlike InterpolateLuma_H264, the functions use an entire motion vector when they calculate fractional and integer parts of the vectors. The functions also handle cases of overlapping, when the source block pointed to by the motion vector lies out of the source frame. Non-existing samples are cloned from the nearest existing samples.

Figure 16-64 Interpolation of Luminance Component Block for Motion Estimation



Example 16-19 `ippiInterpolateLumaBlock_H264_8u_P1R` **Usage Example**

```
{
  IppVCInterpolateBlock_8u interpolateInfo;

        interpolateInfo.pSrc[0] = reference_luma_frame;
        interpolateInfo.srcStep = reference_luma_frame_pitch;

        // create vectors & factors
        interpolateInfo.pointVector.x = horizontal_motion_vector;
        interpolateInfo.pointVector.y = vertical_motion_vector;

        // fill parameters
        interpolateInfo.pDst[0] = destination_luma_block;
        interpolateInfo.dstStep = destination_luma_frame_pitch;
        interpolateInfo.sizeFrame = {width_of_ luma_frame,
height_of_luma_frame};
        interpolateInfo.sizeBlock = { width_of_luma_block,
height_of_luma_block};
```

Example 16-19 `ippiInterpolateLumaBlock_H264_8u_P1R` **Usage Example** (continued)

```
        interpolateInfo.pointBlockPos.x = x_position_of_current_luma_block;
        interpolateInfo.pointBlockPos.y = y_position_of_current_luma_block;

        ippiInterpolateLumaBlock_H264_8u_P1R(&interpolateInfo);
  }
```

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`roi`*`.width` or *`roi`*`.height` take values other than {16, 8, 4}.

InterpolateChroma_H264

Performs interpolation for motion estimation of the chroma component using fractional part of motion vector.

Syntax

```
IppStatus ippiInterpolateChroma_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep,
    Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, IppiSize roiSize);
IppStatus ippiInterpolateChroma_H264_16u_C1R(const IppVCInterpolate_16u*
    interpolateInfo);
```

Parameters

`pSrc`	Pointer to the source.
`srcStep`	Distance in items between starts of the consecutive lines in the source image.
`pDst`	Pointer to the destination.
`dstStep`	Distance in items between starts of the consecutive lines in the destination image.
`dx, dy`	Fractional parts of the motion vector in 1/8 pel units (0, 1, ..., 7).

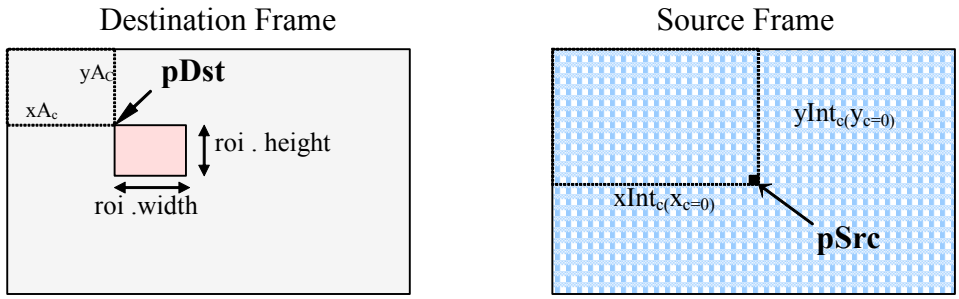
roiSize	Flag that specifies the dimensions of the region of interest (could be 16, 8, 4 or 2 in height dimension and 8, 4 or 2 in width dimension). See structure IppiSize.
interpolateInfo	Pointer to the IppVCInterpolate_16u structure.

Description

The functions ippiInterpolateChroma_H264_8u_C1R and ippiInterpolateChroma_H264_16u_C1R are declared in the ippvc.h file. These functions perform interpolation for motion estimation of the chroma component in accordance with 8.4.2.2.2 of [JVTG050].

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

Figure 16-65 Interpolation for Motion Estimation of Chroma Component



NOTE. Make sure all samples that are used for interpolation are within the boundaries of the source frame. To solve the problem, enlarge the source frame by using boundary samples or use InterpolateChromaTop_H264 or InterpolateChromaBottom_H264 functions.

Example 16-20Usage of ippiInterpolateChroma_H264

```
Ipp32s iHVector, iVVector;
IppiSize roi_chroma = {width_of_chroma_block, height_of_chroma_block};
```

Example 16-20Usage of `ippiInterpolateChroma_H264` (continued)

```
Ipp32s iChromaXPos = x_position_of_current_chroma_block;
Ipp32s iChromaYPos = y_position_of_current_chroma_block;
Ipp8u* pRefUPlane = corresponding_u_block_of_reference_u_frame;
Ipp8u* pRefVPlane = corresponding_v_block_of_reference_v_frame;
Ipp32s iPitch = reference_chroma_frame_pitch;
Ipp8u* pDstU = destination_u_block;
Ipp8u* pDstV = destination_v_block;
Ipp32s iDstPitch = destination_chroma_frame_pitch;

// create motion vectors
iHVector = horizontal_motion_vector;
iVVector = vertical_motion_vector;

{
    Ipp32s iHFraction, iVFraction;
    Ipp32s width, height;

    width = width_of_chroma_frame;
    height = height_of_chroma_frame;

    // prepare the horizontal vector, extract an interpolation fraction
    iHVector <<= ((Ipp32s) (3 <= color_format));
    iHFraction = iHVector & 7;
    iHVector >>= 3;

    // saturate the horizontal vector
    if (((unsigned) (width - 1 - roi_chroma.width)) <
        ((unsigned) (iHVector + iChromaXPos)))
    {
        // saturate to the left bound
        if (- 1 - roi_chroma.width >
            iHVector + iChromaXPos)
        {
            iHVector = - iChromaXPos - roi_chroma.width - 1;
        }
        // saturate to the right bound
        else if (width <
                iHVector + iChromaXPos)
        {
            iHVector = width - iChromaXPos;
        }
    }

    // prepare the vertical vector, extract an interpolation factor
    iVVector <<= ((Ipp32s) (2 <= color_format));
    iVFraction = iVVector & 7;
    iVVector >>= 3;

    // most usual prediction
    // NOTE: we decrement the height to avoid factor's influence
    if (((unsigned) (height - 1 - roi_chroma.height)) >=
```

Example 16-20Usage of `ippiInterpolateChroma_H264` (continued)

```
        ((unsigned) (iVVector + iChromaYPos)))
    {
        ippiInterpolateChroma_H264_8u_C1R(pRefVPlane + iVVector*  iPitch +
iHVector,
                                          iPitch,
                                          pDstV,
                                          iDstPitch,
                                          iHFraction,
                                          iVFraction,
                                          roi_chroma);

        ippiInterpolateChroma_H264_8u_C1R(pRefUPlane + iVVector*  iPitch +
iHVector,
                                          iPitch,
                                          pDstU,
                                          iDstPitch,
                                          iHFraction,
                                          iVFraction,
                                          roi_chroma);
    }
    // prediction from the top of the frame
    else if (0 > iVVector)
    {
        // sometimes the vector points nothing
        ippiInterpolateChromaTop_H264_8u_C1R(pRefVPlane + iVVector*  iPitch +
iHVector,
                                             iPitch,
                                             pDstV,
                                             iDstPitch,
                                             iHFraction,
                                             iVFraction,
                                             - iVVector - iChromaYPos,
                                             roi_chroma);

        ippiInterpolateChromaTop_H264_8u_C1R(pRefUPlane + iVVector*  iPitch +
iHVector,
                                             iPitch,
                                             pDstU,
                                             iDstPitch,
                                             iHFraction,
                                             iVFraction,
                                             - iVVector - iChromaYPos,
                                             roi_chroma);

    }
    // prediction from the bottom of the frame
    else
    {
        // sometimes the vector points nothing
```

Example 16-20Usage of `ippiInterpolateChroma_H264` (continued)

```
        ippiInterpolateChromaBottom_H264_8u_C1R(pRefVPlane + iVVector*  iPitch +
iHVector,
                                                iPitch,
                                                pDstV,
                                                iDstPitch,
                                                iHFraction,
                                                iVFraction,
                                                iVVector + roi_chroma.height +
                                                    iChromaYPos - height,
                                                roi_chroma);

        ippiInterpolateChromaBottom_H264_8u_C1R(pRefUPlane + iVVector*  iPitch +
iHVector,
                                                iPitch,
                                                pDstU,
                                                iDstPitch,
                                                iHFraction,
                                                iVFraction,
                                                iVVector + roi_chroma.height +
                                                    iChromaYPos - height,
                                                roi_chroma);
    }
}
```

These functions are used in the H.264 decoder and encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *`srcStep`* value is less than 8.
`ippStsBadArgErr`	Indicates an error condition if *`dx`* or *`dy`* values fall outside [0,7].
`ippStsSizeErr`	Indicates an error condition if *`roi`*`.width` or *`roi`*`.height` take values other than {8, 4, 2}.

InterpolateChromaTop_H264

Performs interpolation for motion estimation of the chroma component at the frame top boundary.

Syntax

```
IppStatus ippiInterpolateChromaTop_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s
    outPixels, IppiSize roiSize);
IppStatus ippiInterpolateChromaTop_H264_16u_C1R(const IppVCInterpolate_16u*
    interpolateInfo, Ipp32s outPixels);
```

Parameters

pSrc	Pointer to the source.
srcStep	Distance in items between starts of the consecutive lines in the source block.
pDst	Pointer to the destination.
dstStep	Distance in items between starts of the consecutive lines in the destination block.
dx, dy	Fractional parts of the motion vector in 1/8 pel units (0, 1, ..., 7).
outPixels	Number of pixels by which the data specified by *pSrc* reaches over the frame top boundary.
roiSize	Flag that specifies the dimensions of the region of interest (could be 16, 8, 4 or 2 in height dimension and 8, 4 or 2 in width dimension) See structure IppiSize.
interpolateInfo	Pointer to the IppVCInterpolate_16u structure.

Description

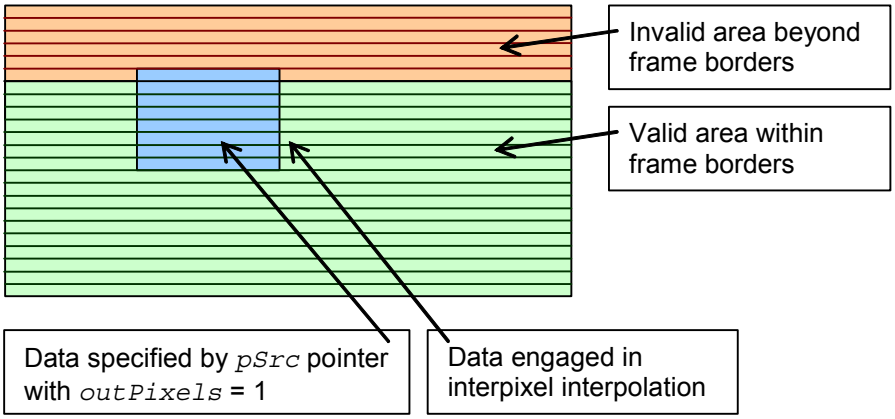
The functions ippiInterpolateChromaTop_H264_8u_C1R and ippiInterpolateChromaTop_H264_16u_C1R are declared in the ippvc.h file. These functions perform interpolation for motion estimation of the chroma component near the top boundary of the frame in accordance with 8.4.2.2.2 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

See Example 16-20 for a usage example.

Figure 16-66 shows the data specified by *pSrc* in interpixel interpolation at the picture top boundary.

Figure 16-66 Chroma Interpolation at Top Boundary



Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* value is less than 16.
ippStsBadArgErr	Indicates an error condition if *dx* or *dy* values fall outside [0,3].
ippStsSizeErr	Indicates an error condition if *roi*.width or *roi*.height take values other than {16, 8, 4}.

InterpolateChromaBottom_H264

Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.

Syntax

```
IppStatus ippiInterpolateChromaBottom_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s
    outPixels, IppiSize roiSize);
IppStatus ippiInterpolateChromaBottom_H264_16u_C1R(const IppVCInterpolate_16u*
    interpolateInfo, Ipp32s outPixels);
```

Parameters

pSrc	Pointer to the source.
srcStep	Distance in items between starts of the consecutive lines in the source block.
pDst	Pointer to the destination.
dstStep	Distance in items between starts of the consecutive lines in the destination block.
dx, dy	Fractional parts of the motion vector in 1/8 pel units (0, 1, ..., 7).
outPixels	Number of pixels by which the data specified by *pSrc* reaches over the frame bottom boundary.
roiSize	Flag that specifies the dimensions of the region of interest (could be 16, 8, 4 or 2 in height dimension and 8, 4 or 2 in width dimension). See structure IppiSize.
interpolateInfo	Pointer to the IppVCInterpolate_16u structure.

Description

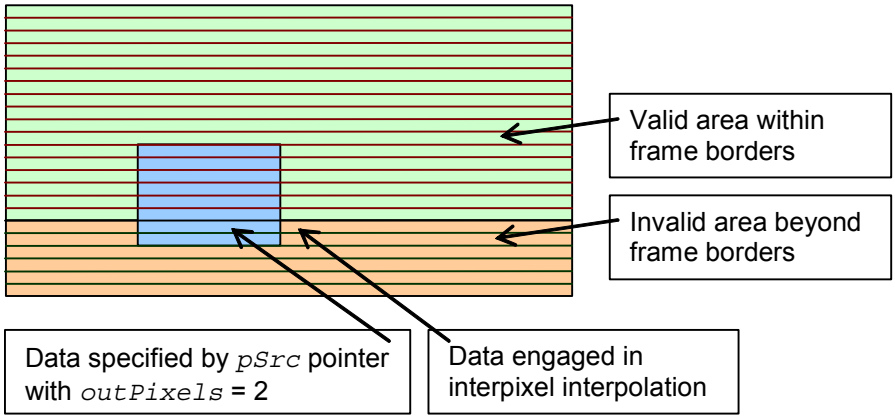
The functions ippiInterpolateChromaBottom_H264_8u_C1R and ippiInterpolateChromaBottom_H264_16u_C1R are declared in the ippvc.h file. These functions perform interpolation for motion estimation of the chroma component near the bottom boundary of the frame in accordance with 8.4.2.2.2 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

The functions use only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

See Example 16-20 for a usage example.

Figure 16-67 shows the data specified by *pSrc* in interpixel interpolation at the picture top boundary.

Figure 16-67 Chroma Interpolation at Bottom Boundary



Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* value is less than 16.
ippStsBadArgErr	Indicates an error condition if *dx* or *dy* values fall outside [0,3].
ippStsSizeErr	Indicates an error condition if *roi*.width or *roi*.height take values other than {16, 8, 4}.

InterpolateChromaBlock_H264

Performs interpolation for motion estimation of the chroma component using entire motion vector.

Syntax

```
IppStatus ippiInterpolateChromaBlock_H264_8u_P2R(const
    IppVCInterpolateBlock_8u* interpolateInfo);
IppStatus ippiInterpolateChromaBlock_H264_16u_P2R(const
    IppVCInterpolateBlock_16u* interpolateInfo);
```

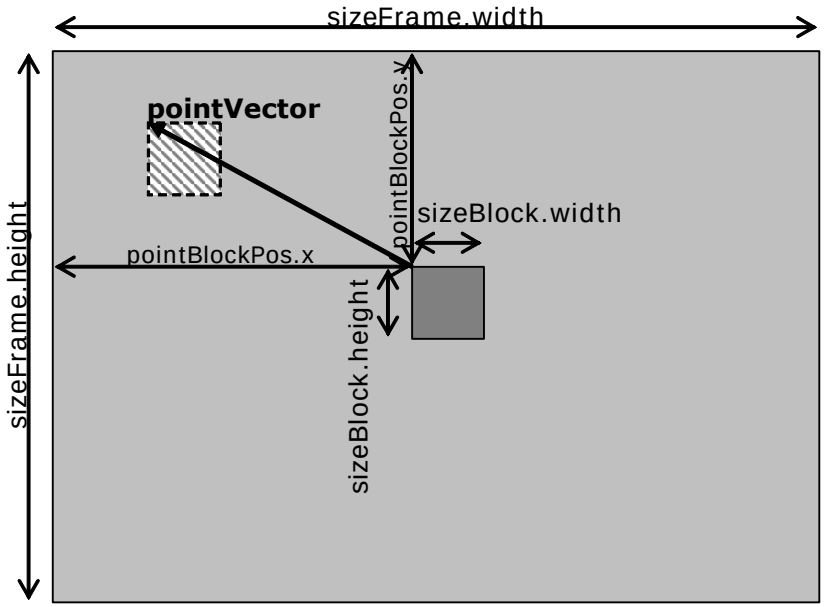
Parameters

interpolateInfo	Pointer to an instance of the structure holding interpolation parameters. See IppVCInterpolateBlock_8u and IppVCInterpolateBlock_16u structures.

Description

The functions ippiInterpolateChromaBlock_H264_8u_P2R and ippiInterpolateChromaBlock_H264_16u_P2R are declared in the ippvc.h file. These functions perform interpolation (convolution with 2x2 kernel) for motion estimation of the chroma component in accordance with 8.4.2.2.2 of [JVTG050]. Unlike InterpolateChroma_H264, the functions use an entire motion vector when they calculate fractional and integer parts of the vectors. The functions also handle cases of overlapping, when the source block pointed to by the motion vector lies out of the source frame. Non-existing samples are cloned from the nearest existing samples.

Figure 16-68 Interpolation of Chrominance Component Block for Motion Estimation



Example 16-21 `ippiInterpolateChromaBlock_H264_8u_P2R` **Usage Example**

```
{
        IppVCInterpolateBlock_8u interpolateInfo;

        // get reference frame & pitch
        interpolateInfo.pSrc[0] = reference_chroma_u_frame;
        interpolateInfo.pSrc[1] = reference_chroma_v_frame;
        interpolateInfo.srcStep = reference_chroma_frame_pitch;

        // save vector
        interpolateInfo.pointVector.x = horizontal_motion_vector;
        interpolateInfo.pointVector.y = vertical_motion_vector;

        // scale motion vector for chroma
        interpolateInfo.pointVector.x <<= ((Ipp32s) (3 <= color_format));
        interpolateInfo.pointVector.y <<= ((Ipp32s) (2 <= color_format));

        // fill parameters
        interpolateInfo.pDst[0] = destination_chroma_u_block;
```

Example 16-21 `ippiInterpolateChromaBlock_H264_8u_P2R` **Usage Example** (continued)

```
        interpolateInfo.pDst[1] = destination_chroma_v_block;
        interpolateInfo.dstStep = destination_chroma_frame_pitch;
        interpolateInfo.sizeFrame = {width_of_chroma_frame,
height_of_chroma_frame};
        interpolateInfo.sizeBlock = {width_of_chroma_block,
height_of_chroma_block};
        interpolateInfo.pointBlockPos.x = x_position_of_current_chroma_block;
        interpolateInfo.pointBlockPos.y = y_position_of_current_chroma_block;

        ippiInterpolateChromaBlock_H264_8u_P2R(&interpolateInfo);
    }
```

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *roi*`.width` or *roi*`.height` take values other than {16, 8, 4}.

InterpolateBlock_H264, Bidir_H264

Calculate average value for each source pair values in block.

Syntax

```
IppStatus ippiInterpolateBlock_H264_8u_P2P1R(const Ipp8u* pSrc1, const Ipp8u*
    pSrc2, Ipp8u* pDst, Ipp32u width, Ipp32u height, Ipp32u pitch);
IppStatus ippiInterpolateBlock_H264_8u_P3P1R(const Ipp8u* pSrc1, const Ipp8u*
    pSrc2, Ipp8u* pDst, Ipp32u width, Ipp32u height, Ipp32s srcStep1, Ipp32s
    srcStep2, Ipp32s dstStep);
IppStatus ippiBidir_H264_16u_P2P1R(IppVCBidir_16u*  bidirInfo)
```

Parameters

pSrc1	Pointer to the first input block.
pSrc2	Pointer to the second input block.
pDst	Pointer to the first output block.

width	Number of values in the block line.
height	Number of lines in the block.
srcStep1	Distance in items between starts of the consecutive lines in the first input block.
srcStep2	Distance in items between starts of the consecutive lines in the second input block.
dstStep	Distance in items between starts of the consecutive lines in the first output block.
pitch	Memory pitch.
bidirInfo	Pointer to the `IppVCBidir_16u` structure.

Description

The functions `ippiInterpolateBlock_H264_8u_P2P1R`, `ippiInterpolateBlock_H264_8u_P3P1R`, and `ippiBidir_H264_16u_P2P1R` are declared in the `ippvc.h` file. These functions calculate the average value for each corresponding source pair values in a block with height lines and width values per line.

$$pDst[i] = \frac{pSrc1[i] + pSrc2[i] + 1}{2}.$$

These functions are used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

WeightedAverage_H264

Averages two blocks with weights.

Syntax

```
IppStatus ippiWeightedAverage_H264_8u_C1IR(const Ipp8u* pSrc1, Ipp8u*
    pSrc2Dst, Ipp32s srcDstStep, Ipp32s weight1, Ipp32s weight2, Ipp32s shift,
    Ipp32s offset, IppiSize roiSize);
```

Parameters

pSrc1	Pointer to the first source (output of preceding functions).
pSrc2Dst	Pointer to the second source and result.
srcDstStep	Distance in bytes between starts of the consecutive lines in the source and result images.
weight1, weight2	Weights.
shift	Shift.
offset	Offset.
roiSize	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).

Description

The function `ippiWeightedAverage_H264_8u_C1IR` is declared in the `ippvc.h` file. This function averages two blocks with weights (for weighted bi-directional predictions) as specified in 8.4.2.3.2 of [JVTG050] when `predFlagL0` and `predFlagL1` are equal to 1.

The function uses the following formula:

$$Source2[x, y] =$$

$$\mathrm{Clip1}((weight1 * Source1[x, y] + weight2 * Source2[x, y] + 1 \ll (shift - 1)) \gg shift + offset)$$

where $x \in [0, roi.\mathrm{width} - 1]$, $y \in [0, roi.\mathrm{height} - 1]$,

$$\mathrm{Clip1}(z) = \begin{cases} 0 & ; \ z < 0 \\ 255 & ; \ z > 255 \\ z & ; \ otherwise \end{cases}$$

The above formula corresponds to formula 8-220 of JVT-G050. See Table 16-26 for matching the function arguments and the standard variables.

Table 16-26 `WeightedAverage_H264` Arguments vs JTV-G050 Variables

ippiWeightedAverage	8.4.2.3.2 JVT-G050
weight1	w_0
weight2	w_1
shift	*logWD*+1
offset	$(o_0 + o_1 + 1) \gg 1$
roi.width	*partWidth*
roi.height	*partHeight*

Values of w_0, w_1, *logWD*, o_0, o_1 are calculated from formulas 8-221 through 8-244.

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is NULL.
`ippStsStepErr`	Indicates an error condition if *srcDstStep* value is less than *roi*.width.

UniDirWeightBlock_H264, UnidirWeight_H264

Weigh source block.

Syntax

```
IppStatus ippiUniDirWeightBlock_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32u srcDstStep,
    Ipp32u ulog2wd, Ipp32s iWeight, Ipp32s iOffset, IppiSize roi);
IppStatus ippiUnidirWeight_H264_16u_IP2P1R(Ipp16u* pSrcDst, Ipp32u srcDstStep,
    Ipp32u ulog2wd, Ipp32s iWeight, Ipp32s iOffset, IppiSize roi, Ipp32s
    bitDepth);
```

Parameters

pSrcDst	Pointer to the source and the result.
srcDstStep	Distance in items between starts of the consecutive lines in the source and result images.
ulog2wd	`log2` weight denominator.
iWeight	Weight.
iOffset	Offset.
roi	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).
bitDepth	Number of bits of the plane sample with range [8.. 14].

Description

The functions `ippiUniDirWeightBlock_H264_8u_C1IR` and `ippiUnidirWeight_H264_16u_IP2P1R` are declared in the `ippvc.h` file. These functions perform weighted prediction as specified in 8.4.2.3.2 of [ITUH264] when `predFlagL0` is equal to 1 and `predFlagL1` are equal to 0 or when `predFlagL0` is equal to 0 and `predFlagL1` are equal to 1.

The functions use the following formula:

`if(logWD ≥ 1)`

$$pSrcDst[x,y] = \mathrm{Clip1c}(((pSrcDst[x,y] \cdot iWeight + 2^{\mathrm{ulog2}wd-1}) \gg \mathrm{ulog2}wd) + iOffset)$$

`else`

$$pSrcDst[x,y] = \mathrm{Clip1c}((pSrcDst[x,y] \cdot iWeight) + iOffset)$$

where $x \in [0, roi.\mathrm{width} - 1]$, $y \in [0, roi.\mathrm{height} - 1]$,

$$\mathrm{Clip1}(z) = \begin{cases} 0 & ; \ z < 0 \\ 255 & ; \ z > 255 \\ z & ; \ otherwise \end{cases}$$

The above formula corresponds to formula 8-270 and 8-271 of ITUH264.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

ippStsStepErr	Indicates an error condition if *srcDstStep* value is less than *roi*.width.
ippStsSizeErr	Indicates an error condition if *roi*.width or *roi*.height value is not equal to 2 or 4 or 8 or 16.

BiDirWeightBlock_H264, BidirWeight_H264

Average two blocks with two weights and two offsets.

Syntax

```
IppStatus ippiBiDirWeightBlock_H264_8u_P2P1R(const Ipp8u* pSrc1, const Ipp8u*
    pSrc2, Ipp8u* pDst, Ipp32u srcStep, Ipp32u dstStep, Ipp32u ulog2wd, Ipp32s
    iWeight1, Ipp32s iOffset1, Ipp32s iWeight2, Ipp32s iOffset2, IppiSize roi);
IppStatus ippiBidirWeight_H264_16u_P2P1R(const IppVCBidir_16u*  bidirInfo,
    Ipp32u ulog2wd, Ipp32s iWeight1, Ipp32s iOffset1, Ipp32s iWeight2, Ipp32s
    iOffset2);
```

Parameters

pSrc1	Pointer to the first source.
pSrc2	Pointer to the second source.
pDst	Pointer to the result.
srcStep	Distance in items between starts of the consecutive lines in the source images.
dstStep	Distance in items between starts of the consecutive lines in the result image.
ulog2wd	log2 weight denominator.
iWeight1, *iWeight2*	Weights.
iOffset1, *iOffset2*	Offsets.
roi	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).
bidirInfo	Pointer to the IppVCBidir_16u structure.

Description

The functions `ippiBiDirWeightBlock_H264_8u_P2P1R` and `ippiBidirWeight_H264_16u_P2P1R` are declared in the `ippvc.h` file. These functions perform weighted prediction as specified in 8.4.2.3.2 of [ITUH264] when both `predFlagL0` and `predFlagL1` are equal to 1.

The functions use the following formula:

$$pDst[x,y] = \mathrm{Clip1c}(((pSrc1[x,y] \cdot iWeight1 + pSrc2[x,y] \cdot iWeight2 + 2^{\mathrm{ulog2}wd-1}) \gg (\mathrm{ulog2}wd+1) +$$

$$+((iOffset1 + iOffset2 + 1) \gg 2)$$

where $x \in [0, roi.\mathrm{width} - 1]$, $y \in [0, roi.\mathrm{height} - 1]$,

$$\mathrm{Clip1}(z) = \begin{cases} 0 & ; \; z < 0 \\ 255 & ; \; z > 255 \\ z & ; \; otherwise \end{cases}$$

The above formula corresponds to formula 8-272 of ITUH264.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if *`srcStep`* value or *`dstStep`* value is less than *`roi`*`.width`.
`ippStsSizeErr`	Indicates an error condition if *`roi`*`.width` or *`roi`*`.height` value is not equal to 2 or 4 or 8 or 16.

BiDirWeightBlockImplicit_H264, BidirWeightImplicit_H264

Average two blocks with weights if `ulog2wd` *is equal to 5.*

Syntax

```
IppStatus ippiBiDirWeightBlockImplicit_H264_8u_P2P1R(const Ipp8u* pSrc1, const
    Ipp8u* pSrc2, Ipp8u* pDst, Ipp32u srcStep, Ipp32u dstStep, Ipp32s iWeight1,
    Ipp32s iWeight2, IppiSize roi);
IppStatus ippiBidirWeightImplicit_H264_16u_P2P1R(const IppVCBidir_16u*
    bidirInfo, Ipp32s iWeight1, Ipp32s iWeight2);
```

Parameters

pSrc1	Pointer to the first source.
pSrc2	Pointer to the second source.
pDst	Pointer to the result.
srcStep	Distance in items between starts of the consecutive lines in the source images.
dstStep	Distance in items between starts of the consecutive lines in the result image.
iWeight1, *iWeight2*	Weights.
roi	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).
bidirInfo	Pointer to the `IppVCBidir_16u` structure.

Description

The functions `ippiBiDirWeightBlockImplicit_H264_8u_P2P1R` and `ippiBidirWeightImplicit_H264_16u_P2P1R` ares declared in the `ippvc.h` file. These functions perform implicit mode weighted prediction as specified in 8.4.2.3.2 of [ITUH264].

These functions use formula 8-272 of [ITUH264], but according to 8-273, 8-274, and 8-275 of ITUH264 the `log2` weight denominator *ulog2wd* is equal to 5, and both *iOffset1*, *iOffset2* offsets are equal to zero. So the functions use the following formula:

$$pDst[x,y] = \mathrm{Clip1c}((pSrc1[x,y] \cdot iWeight1 + pSrc2[x,y] \cdot iWeight2 + 32) \gg 6)$$

where $x \in [0, roi.\mathrm{width} - 1]$, $y \in [0, roi.\mathrm{height} - 1]$,

$$\mathrm{Clip1}(z) = \begin{cases} 0 & ; \; z < 0 \\ 255 & ; \; z > 255 \\ z & ; \; otherwise \end{cases}$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if *srcStep* value or *dstStep* value is less than *roi*.width.
ippStsSizeErr	Indicates an error condition if *roi*.width or *roi*.height value is not equal to 2 or 4 or 8 or 16.

Macroblock Reconstruction

Note that the functions with _IP2R descriptor use the following structures:

```
typedef struct _IppiMBReconstructHigh_32s16u
{
    Ipp32s** ppSrcDstCoeff;
    Ipp16u* pSrcDstPlane;
    Ipp32s srcDstStep;
    Ipp32u cbp;
    Ipp32s qp;
    Ipp16s* pQuantTable;
    Ipp32s bypassFlag;
    Ipp32s bitDepth;
} IppiReconstructHighMB_32s16u;

typedef struct _IppiReconstructHighMB_16s8u
{
```

```
    Ipp16s** ppSrcDstCoeff;
    Ipp8u* pSrcDstPlane;
    Ipp32s srcDstStep;
    Ipp32u cbp;
    Ipp32s qp;
    Ipp16s* pQuantTable;
    Ipp32s bypassFlag;
} IppiReconstructHighMB_16s8u;
```

with parameters:

ppSrcDstCoeff	Pointer to the residual coefficients; pointer is updated by the function.
pSrcDstPlane	Pointer to the macroblock that is reconstructed in the current plane.
srcDstStep	Distance in items between starts of the consecutive lines in the source/destination images.
cbp	Coded block pattern. If `cbp & IPPVC_CBP_DC` is not equal to 0, DC block is not zero-filled and it exists in *ppSrcDstCoeff*. If `cbp & (1<<(IPPVC_CBP_1ST_AC_BITPOS+i))` is not equal to 0 (0 < `i` < *N*), `i`-th AC block is not zero-filled and it exists in *ppSrcDstCoeff*. The *N* value depends on a function. For Luma, *N* is 16, for chroma – N is 4, and for Chroma422 – N is 8. Sizes of DC/AC blocks depend on a function.
qp	Quantizer with range depending on the function (Luma/Chroma). For `_32s16u_IP2R` functions, the range depends on *bitDepth*. It should be [0 .. 39 + (6*((`bitDepth`) - 8))] for chroma and [0 .. 51 + (6*((`bitDepth`) - 8))] for luma. For `_16s8u_IP2R` functions, the range equals [0..39] for chroma and [0..51] for luma.
pQuantTable	Quantization table. Pointer to the quantization table for plane (*LevelScale*(*qP%6*, *i*, *j*) in [ITUH264])
bypassFlag	Flag enabling lossless coding. Lossless coding is enabled only if *bypassFlag* is 1 and *qp* is 0. (See `qpprime_y_zero_transform_bypass_flag` in [ITUH264]
bitDepth	Number of bits of the plane sample with range [8.. 14].

ReconstructChromaInterMB_H264

Reconstructs inter chroma macroblock.

Syntax

```
IppStatus ippiReconstructChromaInterMB_H264_16s8u_P2R(Ipp16s** ppSrcCoeff,
    Ipp8u* pSrcDstUPlane, Ipp8u* pSrcDstVPlane, const Ipp32u srcDstStep, const
    Ipp32u cbp4x4, const Ipp32u ChromaQP);
```

Parameters

ppSrcCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC *U*-block, 2x2 DC *V*-block, 4x4 AC *U*-blocks, 4x4 AC *V*-blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to macroblock that is reconstructed in current *U* plane. This macroblock must contain inter prediction samples.
pSrcDstVPlane	Pointer to macroblock that is reconstructed in current *V* plane. This macroblock must contain inter prediction samples.
srcDstStep	Plane step.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1))` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+4))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcCoeff*.
ChromaQP	chroma quantizer (QP_C in [JVTG050]). It must be within the range [0;39].

Description

The function `ippiReconstructChromaInterMB_H264_16s8u_P2R` is declared in the `ippvc.h` file. This function reconstructs inter chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock):

1. Performs integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [JVTG050] in the same way as `TransformDequantChromaDC_H264` function does.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7 of [JVTG050].
3. Performs adding of 8x8 inter prediction blocks and 8x8 residual blocks in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*ChromaQP* is less than 0 or greater than 39.

ReconstructChromaIntraHalvesMB_H264

Reconstructs two halves of intra chroma macroblock.

Syntax

```
IppStatus ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R(Ipp16s**
    ppSrcCoeff, Ipp8u* pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32u srcDstUVStep,
    IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u cbp4x4, Ipp32u ChromaQP,
    Ipp8u edgeTypeTop, Ipp8u edgeTypeBottom);
```

Parameters

ppSrcCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC *U*-block, 2x2 DC *V*-block, 4x4 AC *U*-blocks, 4x4 AC *V*-blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to current *U* plane that is reconstructed.
pSrcDstVPlane	Pointer to current *V* plane that is reconstructed.
srcDstUVStep	Plane step.
intraChromaMode	Prediction mode of the Intra_chroma prediction process for chroma samples. It is defined in the same way as in `PredictIntraChroma8x8_H264` function.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1))` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+4))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcCoeff*.
ChromaQP	Chroma quantizer (QP_C in [JVTG050]). It must be within the range [0;39].
edgeTypeTop, *edgeTypeBottom*	Flags that specify the availability of the macroblocks used for prediction. Upper and lower halves of the macroblock may have different prediction vectors, so use two flags: one for the upper half of the macroblock, the other - for the lower half.

Description

The function `ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R` is declared in the `ippvc.h` file. This function reconstructs intra chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock), divided into upper and lower halves, 16x8 each, differing only in prediction, that is, having independent prediction vectors. Otherwise their performance is the same:

1. Performs integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [JVTG050] in the same way as `TransformDequantChromaDC_H264` function does.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7 of [JVTG050].
3. Performs intra prediction for an 8x8 *U*-component and for an 8x8 *V*-component in accordance with 8.3.2 of [JVTG050] in the same way as `PredictIntraChroma8x8_H264` function does.
4. Performs adding of 8x8 intra prediction blocks and 8x8 residual blocks in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*ChromaQP* is less than 0 or greater than 39.

ReconstructChromaIntraMB_H264

Reconstructs intra chroma macroblock.

Syntax

```
IppStatus ippiReconstructChromaIntraMB_H264_16s8u_P2R(Ipp16s** ppSrcCoeff,
    Ipp8u* pSrcDstUPlane, Ipp8u* pSrcDstVPlane, const Ipp32u srcDstUVStep, const
    IppIntraChromaPredMode_H264 intraChromaMode, const Ipp32u cbp4x4, const
    Ipp32u ChromaQP, const Ipp8u edgeType);
```

Parameters

ppSrcCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC *U*-block, 2x2 DC *V*-block, 4x4 AC *U*-blocks, 4x4 AC *V*-blocks if

	the block is not zero-filled) in the same order as is shown in Figure 8-7 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to current *U* plane that is reconstructed.
pSrcDstVPlane	Pointer to current *V* plane that is reconstructed.
srcDstUVStep	Plane step.
intraChromaMode	Prediction mode of the Intra_chroma prediction process for chroma samples. It is defined in the same way as in `PredictIntraChroma8x8_H264` function.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1))` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i))` is not equal to 0, ($0 \le i < 4$), *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+4))` is not equal to 0, ($0 \le i < 4$), *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcCoeff*.
ChromaQP	Chroma quantizer (QP_C in [JVTG050]). It must be within the range [0;39].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero.

Description

The function `ippiReconstructChromaIntraMB_H264_16s8u_P2R` is declared in the `ippvc.h` file. This function reconstructs intra chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock):

1. Performs integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [JVTG050] in the same way as `TransformDequantChromaDC_H264` function does.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7 of [JVTG050].
3. Performs intra prediction for an 8x8 *U*-component and for an 8x8 *V*-component in accordance with 8.3.2 of [JVTG050] in the same way as `PredictIntraChroma8x8_H264` function does.
4. Performs adding of 8x8 intra prediction blocks and 8x8 residual blocks in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*ChromaQP* is less than 0 or greater than 39.

ReconstructChromaInter4x4MB_H264, ReconstructChromaInter4x4_H264High

Reconstruct 4X4 inter chroma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructChromaInter4x4MB_H264_16s8u_P2R(Ipp16s**
    ppSrcDstCoeff, Ipp8u* pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32u
    srcDstUVStep, Ipp32u cbp4x4, Ipp32s ChromaQPU, Ipp32u ChromaQPV, const
    Ipp16s* pQuantTableU, const Ipp16s* pQuantTableV, Ipp8u bypassFlag);
IppStatus ippiReconstructChromaInter4x4_H264High_32s16u_IP2R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo[2]);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC *U*-block, 2x2 DC *V*-block, 4x4 AC *U*-blocks, 4x4 AC *V*-blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to macroblock that is reconstructed in current *U* plane. This macroblock must contain inter prediction samples.
pSrcDstVPlane	Pointer to macroblock that is reconstructed in current *V* plane. This macroblock must contain inter prediction samples.
srcDstUVStep	Plane step.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1))` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+4))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*.
ChromaQPU	Chroma quantizer for *U* plane (QP_C in [ITUH264]). It must be within the range [0;39].
ChromaQPV	Chroma quantizer for *V* plane (QP_C in [ITUH264]). It must be within the range [0;39].
pQuantTableU	Pointer to the quantization table for *U* plane (*LevelScale(qP%6, i, j)* in [ITUH264]).
pQuantTableV	Pointer to the quantization table for *V* plane (*LevelScale(qP%6, i, j)* in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo[2]	Pointer to the `IppiReconstructHighMB_32s16u` structure. The first pointer *pReconstructInfo*[0] points to the structure with parameters for restructuring chroma macroblock of plane *U*, and the second pointer *pReconstructInfo*[1] points to the structure with parameters for restructuring Chroms macroblock of plane *V*.

Description

The function `ippiReconstructChromaInter4x4MB_H264_16s8u_P2R` and `ippiReconstructChromaInter4x4_H264High_32s16u_IP2R` are declared in the `ippvc.h` file. These functions reconstruct 4x4 inter chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock) for high profile:

1. Perform integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.9 of [ITUH264] according to 8-323 when `chroma_format_idc` is equal to 1.
2. Perform scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7(a) of [ITUH264] when `chroma_format_idc` is equal to 1.
3. Perform adding of 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-306 of [ITUH264].

See Example below for the usage of `ippiReconstructChromaInter4x4MB_H264_16s8u_P2R`.

Example 16-22Usage of `ippiReconstructChromaInter4x4MB_H264`

```
{

    #define            D_CBP_CHROMA_DC 0x00001
    #define            D_CBP_CHROMA_AC 0x1fffe
    #define            D_CBP_CHROMA_AC_420 0x0001e
    #define            D_CBP_1ST_CHROMA_DC_BITPOS 17
    #define            D_CBP_1ST_CHROMA_AC_BITPOS 19

    Ipp8u   planeU[8*8] =
  {  0x68, 0xaf, 0x16, 0x9a, 0x38, 0x8f, 0x7f, 0x6a,
     0x7e, 0x08, 0x01, 0x8c, 0x76, 0x87, 0xa4, 0xc6,
     0x9d, 0x47, 0x0f, 0xa1, 0x90, 0x0b, 0x36, 0x47,
     0x87, 0x28, 0x0f, 0xb3, 0x24, 0xa4, 0x5c, 0x81,
     0x4b, 0x96, 0x33, 0xa3, 0x9f, 0x25, 0x5c, 0x44,
     0x61, 0x92, 0x36, 0x9c, 0x07, 0x05, 0x82, 0x9b,
     0xc7, 0xb7, 0x99, 0x48, 0x33, 0x9a, 0x75, 0x03,
     0xb6, 0x53, 0x9d, 0x03, 0x9c, 0x3a, 0x0b, 0x59 };

    Ipp8u   planeV[8*8] = {
     0x94, 0x22, 0x8a, 0xbc, 0xbb, 0x23, 0x60, 0xbb,
     0xba, 0x73, 0x65, 0x14, 0x9a, 0xc6, 0x34, 0xae,
     0xaf, 0x16, 0x22, 0x9a, 0x38, 0x6c, 0x54, 0x8f,
     0x08, 0x01, 0xb9, 0x8c, 0x76, 0x7b, 0x16, 0x87,
     0x47, 0x0f, 0x9f, 0xa1, 0x90, 0x5f, 0x1d, 0x0b,
     0x28, 0x0f, 0x4a, 0xb3, 0x24, 0x83, 0x14, 0xa4,
     0x96, 0x33, 0x65, 0xa3, 0x9f, 0x3b, 0x08, 0x25,
     0x59, 0x2c, 0xb6, 0x30, 0x24, 0x7b, 0x47, 0x6b };
```

Example 16-22Usage of `ippiReconstructChromaInter4x4MB_H264` (continued)

```
    Ipp32s  stepUV = 8;
    Ipp16s  quantTableU[16] = {22,12,34,50,34,33,46,21,18,19,34,28,28,31,15,3};
    Ipp16s  quantTableV[16] = {32,50,34, 2,46,21,18, 9,34,28,28,31,15,3,12,44};

    Ipp16s  coef[4+16*4+4+16*4] = {
  0,  0,  0,  0,
  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1, -1,  0,
  1,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,
  0,  0,  1,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0, -1,  0,  1,
  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 -3, -3, -3, -3,
 -1, -1, -1, -1,  4,  4,  4,  4, -6, -6, -6, -6, -6, -6, -6, -6,
 -6, -6, -6, -6, -6, -6, -6, -6,-64,-64,-64,-64,  7,  1, -4, -3,
  7,  1, -4, -3,  2, -2, -2,  2, -1,  5, 10, 11, 11, 15, 15,  9,
 -9,-10,-13, 12, -7, -7, -8, 12, -3, -3,  3, 13, -1, -1,  8, 14  };

    Ipp32u  cbpU = 0x00000014, cbpV = 0x00000014;
    Ipp32s  QPU = 17, QPV = 31;
    Ipp8u   bypass_flag = 0;
    IppStatus result;
    Ipp16s* pCoef;
    Ipp32u cbp4x4 = (((cbpU & D_CBP_CHROMA_DC) | ((cbpV & D_CBP_CHROMA_DC) <<
1)) << D_CBP_1ST_CHROMA_DC_BITPOS) | ((cbpU & D_CBP_CHROMA_AC_420) <<
(D_CBP_1ST_CHROMA_AC_BITPOS - 1)) | ((cbpV & D_CBP_CHROMA_AC_420) <<
(D_CBP_1ST_CHROMA_AC_BITPOS + 4 - 1));

    pCoef = coef;

    result = ippiReconstructChromaInter4x4MB_H264_16s8u_P2R(
                                            &pCoef,
                                            planeU, planeV,
                                            stepUV,
                                            cbp4x4,
                                            QPU,QPV,
                                            quantTableU,quantTableV,
                                            bypass_flag);
  }
```

These functions are used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*ChromaQPU* or *ChromaQPV* is less than 0 or greater than 39.

ReconstructChroma422Inter4x4_H264High

Reconstructs 4X4 inter chroma macroblock for 4:2:2 chroma mode.

Syntax

```
IppStatus ippiReconstructChroma422Inter4x4_H264High_32s16u_IP2R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo[2], Ipp32u levelScaleDCU,
    Ipp32u levelScaleDCV);
IppStatus ippiReconstructChroma422Inter4x4_H264High_16s8u_IP2R(const
    IppiReconstructHighMB_16s8u*  pReconstructInfo[2], Ipp32u levelScaleDCU,
    Ipp32u levelScaleDCV);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC *U*-block, 2x2 DC *V*-block, 4x4 AC *U*-blocks, 4x4 AC *V*-blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7(b) of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to macroblock that is reconstructed in current *U* plane. This macroblock must contain inter prediction samples.
pSrcDstVPlane	Pointer to macroblock that is reconstructed in current *V* plane. This macroblock must contain inter prediction samples.
srcDstUVStep	Plane step.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1))` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i))` is not equal to 0, ($0 \le i < 4$), i-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+4))` is not equal to 0, ($0 \le i < 4$), i-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*.

ChromaQPU	Chroma quantizer for *U* plane (QP_C in [ITUH264]). It must be within the range [0;39].
ChromaQPV	Chroma quantizer for *V* plane (QP_C in [ITUH264]). It must be within the range [0;39].
levelScaleDCU	Chroma quantizer for *U* plane (*LevelScale*($QP_{C,DC}$%6, 0, 0) in [ITUH264]) for chroma DC coefficients.
levelScaleDCV	Chroma quantizer for *V* plane (*LevelScale*($QP_{C,DC}$%6, 0, 0) in [ITUH264]) for chroma DC coefficients.
pQuantTableU	Pointer to the quantization table for *U* plane (*LevelScale(qP%6, i, j*) in [ITUH264]).
pQuantTableV	Pointer to the quantization table for *V* plane (*LevelScale(qP%6, i, j*) in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo[2]	Pointer to the IppiReconstructHighMB_32s16u and IppiReconstructHighMB_16s8u structures. The first pointer *pReconstructInfo*[0] points to the structure with parameters for restructuring chroma macroblock of plane *U*, and the second pointer *pReconstructInfo*[1] points to the structure with parameters for restructuring Chroms macroblock of plane *V*.

Description

The functions ippiReconstructChroma422Inter4x4_H264High_32s16u_IP2R and ippiReconstructChroma422Inter4x4_H264High_16s8u_IP2R are declared in the ippvc.h file. These functions reconstruct eight 4x4 inter chroma macroblocks (16x8 *U* macroblock and 16x8 *V* macroblock) for chroma format 4:2:2 (chroma_format_idc is equal to 2):

1. Perform integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [ITUH264].
2. Perform scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7(b) of [ITUH264].
3. Perform adding of 16x8 inter prediction blocks and 16x8 residual blocks in accordance with 8-306 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	*ChromaQPU* or *ChromaQPV* is less than 0 or greater than 39.

ReconstructChromaIntraHalves4x4MB_H264, ReconstructChromaIntraHalf4x4_H264High

Reconstruct two independent halves of 4X4 intra chroma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructChromaIntraHalves4x4MB_H264_16s8u_P2R(Ipp16s**
    ppSrcDstCoeff, Ipp8u* pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32u
    srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u cbp4x4,
    Ipp32s ChromaQPU, Ipp32u ChromaQPV, Ipp8u edgeTypeTop, Ipp8u edgeTypeBottom,
    const Ipp16s* pQuantTableU, const Ipp16s* pQuantTableV, Ipp8u bypassFlag);
IppStatus ippiReconstructChromaIntraHalf4x4_H264High_32s16u_IP2R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo[2],
    IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u edgeTypeTop, Ipp32u
    edgeTypeBottom);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC *U*-block, 2x2 DC *V*-block, 4x4 AC *U*-blocks, 4x4 AC *V*-blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to current *U* plane that is reconstructed.
pSrcDstVPlane	Pointer to current *V* plane that is reconstructed.
srcDstUVStep	Plane step.
intraChromaMode	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in PredictIntraChroma8x8_H264 function.

cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1))` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+4))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*.
ChromaQPU	Chroma quantizer for *U* plane (QP_C in [ITUH264]). It must be within the range [0;39].
ChromaQPV	Chroma quantizer for *V* plane (QP_C in [ITUH264]). It must be within the range [0;39].
edgeTypeTop, *edgeTypeBottom*	Flags that specify the availability of the macroblocks used for prediction. Upper and lower halves of the macroblock may have different prediction vectors, so use two flags: one for the upper half of the macroblock, the other - for the lower half.
pQuantTableU	Pointer to the quantization table for *U* plane (*LevelScale(qP%6, i, j*) in [ITUH264]).
pQuantTableV	Pointer to the quantization table for *V* plane (*LevelScale(qP%6, i, j*) in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo[2]	Pointer to the `IppiReconstructHighMB_32s16u` structure. The first pointer *pReconstructInfo*[0] points to the structure with parameters for restructuring chroma macroblock of plane *U*, and the second pointer *pReconstructInfo*[1] points to the structure with parameters for restructuring Chroms macroblock of plane *V*.

Description

The functions `ippiReconstructChromaIntraHalves4x4MB_H264_16s8u_P2R` and `ippiReconstructChromaIntraHalf4x4_H264High_32s16u_IP2R` are declared in the `ippvc.h` file. These functions reconstruct 4x4 intra chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock) for high profile. The macroblock is divided into upper and lower halves, 16x8 each. The halves differ only in prediction, that is, have independent prediction vectors. Otherwise their performance is the same:

1. Perform integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.9 of [ITUH264] according to 8-323 when `chroma_format_idc` is equal to 1.
2. Perform scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as shown in Figure 8-7(a) of [ITUH264] when `chroma_format_idc` is equal to 1.
3. Perform adding of 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-306 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*ChromaQPU* or *ChromaQPV* is less than 0 or greater than 39.

ReconstructChromaIntra4x4MB_H264, ReconstructChromaIntra4x4_H264High

Reconstruct 4X4 intra chroma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructChromaIntra4x4MB_H264_16s8u_P2R(Ipp16s**
    ppSrcDstCoeff, Ipp8u* pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32u
    srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u cbp4x4,
    Ipp32s ChromaQPU, Ipp32u ChromaQPV, Ipp8u edgeType, const Ipp16s*
    pQuantTableU, const Ipp16s* pQuantTableV, Ipp8u bypassFlag);
IppStatus ippiReconstructChromaIntra4x4_H264High_32s16u_IP2R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo[2],
    IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u edgeType);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC *U*-block, 2x2 DC *V*-block, 4x4 AC *U*-blocks, 4x4 AC *V*-blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to macroblock that is reconstructed in current *U* plane.
pSrcDstVPlane	Pointer to macroblock that is reconstructed in current *V* plane.
srcDstUVStep	Plane step.
intraChromaMode	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in `PredictIntraChroma8x8_H264` function.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1))` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+4))` is not equal to 0, $(0 \le i < 4)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*.
ChromaQPU	Chroma quantizer for *U* plane (QP_C in [ITUH264]). It must be within the range [0;39].
ChromaQPV	Chroma quantizer for *V* plane (QP_C in [ITUH264]). It must be within the range [0;39].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero.
pQuantTableU	Pointer to the quantization table for *U* plane (*LevelScale(qP%6, i, j)* in [ITUH264]).

pQuantTableV	Pointer to the quantization table for *V* plane (*LevelScale(qP%6, i, j*) in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo[2]	Pointer to the `IppiReconstructHighMB_32s16u` structure. The first pointer *pReconstructInfo*[0] points to the structure with parameters for restructuring chroma macroblock of plane *U*, and the second pointer *pReconstructInfo*[1] points to the structure with parameters for restructuring Chroms macroblock of plane *V*.

Description

The functions `ippiReconstructChromaIntra4x4MB_H264_16s8u_P2R` and `ippiReconstructChromaIntra4x4_H264High_32s16u_IP2R` are declared in the `ippvc.h` file. These functions reconstruct 4x4 intra chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock) for high profile:

1. Perform integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.9 of [ITUH264] according to 8-323 when `chroma_format_idc` is equal to 1.
2. Perform scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as shown in Figure 8-7(a) of [ITUH264] when `chroma_format_idc` is equal to 1.
3. Perform adding of 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-306 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*ChromaQPU* or *ChromaQPV* is less than 0 or greater than 39.

ReconstructChroma422IntraHalf4x4_H264High

Reconstructs two independent halves of 4X4 intra chroma macroblock for 4:2:2 chroma mode.

Syntax

```
IppStatus ippiReconstructChroma422IntraHalf4x4_H264High_32s16u_IP2R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo[2],
    IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u edgeTypeTop, Ipp32u
    edgeTypeBottom, Ipp32u levelScaleDCU, Ipp32u levelScaleDCV);
IppStatus ippiReconstructChroma422IntraHalf4x4_H264High_16s8u_IP2R(const
    IppiReconstructHighMB_16s8u*  pReconstructInfo[2],
    IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u edgeTypeTop, Ipp32u
    edgeTypeBottom, Ipp32u levelScaleDCU, Ipp32u levelScaleDCV);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC *U*-block, 2x2 DC *V*-block, 4x4 AC *U*-blocks, 4x4 AC *V*-blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7(b) of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to the current *U* plane that is reconstructed.
pSrcDstVPlane	Pointer to the current *V* plane that is reconstructed.
srcDstUVStep	Plane step.
intraChromaMode	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in PredictIntraChroma8x8_H264 function.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1))` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i))` is not equal to 0, $(0 \le i < 8)$, *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*.

	If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+4))` is not equal to 0, ($0 \le i < 8$), *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*.
ChromaQPU	Chroma quantizer for *U* plane (QP_C in [ITUH264]). It must be within the range [0;39].
ChromaQPV	Chroma quantizer for *V* plane (QP_C in [ITUH264]). It must be within the range [0;39].
levelScaleDCU	Chroma quantizer for *U* plane ($LevelScale(QP_{C,DC}\%6, 0, 0)$ in [ITUH264]) for chroma DC coefficients.
levelScaleDCV	Chroma quantizer for *V* plane ($LevelScale(QP_{C,DC}\%6, 0, 0)$ in [ITUH264]) for chroma DC coefficients.
edgeTypeTop, *edgeTypeBottom*	Flags that specify the availability of the macroblocks used for prediction. Upper and lower halves of the macroblock may have different prediction vectors, so use two flags: one for the upper half of the macroblock, the other - for the lower half.
pQuantTableU	Pointer to the quantization table for *U* plane (*LevelScale(qP%6, i, j)* in [ITUH264]).
pQuantTableV	Pointer to the quantization table for *V* plane (*LevelScale(qP%6, i, j)* in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo[2]	Pointer to the `IppiReconstructHighMB_32s16u` and `IppiReconstructHighMB_16s8u` structures. The first pointer *pReconstructInfo*[0] points to the structure with parameters for restructuring chroma macroblock of plane *U*, and the second pointer *pReconstructInfo*[1] points to the structure with parameters for restructuring Chroms macroblock of plane *V*.

Description

The functions `ippiReconstructChroma422IntraHalf4x4_H264High_32s16u_IP2R` and `ippiReconstructChroma422IntraHalf4x4_H264High_16s8u_IP2R` are declared in the `ippvc.h` file. These functions reconstruct 4x4 intra chroma macroblocks (16x8 *U* macroblock and 16x8 *V* macroblock) divided into upper and lower 8x8 halves for chroma format 4:2:2 (`chroma_format_idc` is equal to 2). The halves differ only in prediction, that is, have independent prediction vectors. Otherwise their performance is the same:

1. Perform integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [ITUH264] in the same way as `TransformDequantChromaDC_H264` function does.
2. Perform scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [ITUH264] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7(b) of [ITUH264].
3. Perform adding of 16x8 intra prediction blocks and 16x8 residual blocks in accordance with 8-306 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*ChromaQPU* or *ChromaQPV* is less than 0 or greater than 39.

ReconstructChroma422Intra4x4_H264High

Reconstructs 4X4 intra chroma macroblock for 4:2:2 chroma mode.

Syntax

```
IppStatus ippiReconstructChroma422Intra4x4_H264High_32s16u_IP2R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo[2],
    IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u edgeType, Ipp32u
    levelScaleDCU, Ipp32u levelScaleDCV);
IppStatus ippiReconstructChroma422Intra4x4_H264High_16s8u_IP2R(const
    IppiReconstructHighMB_16s8u*  pReconstructInfo[2],
    IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u edgeType, Ipp32u
    levelScaleDCU, Ipp32u levelScaleDCV);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC *U*-block, 2x2 DC *V*-block, 4x4 AC *U*-blocks, 4x4 AC *V*-blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7(b) of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to macroblock that is reconstructed in current *U* plane. This macroblock must contain intra prediction samples.
pSrcDstVPlane	Pointer to macroblock that is reconstructed in current *V* plane. This macroblock must contain intra prediction samples.
srcDstUVStep	Plane step.
intraChromaMode	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in `PredictIntraChroma8x8_H264` function.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1))` is not equal to 0, 2x2 DC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i))` is not equal to 0, ($0 \le i < 8$), *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+8))` is not equal to 0, ($0 \le i < 8$), *i*-th 4x4 AC *U*-block is not zero-filled and exists in *ppSrcDstCoeff*.
ChromaQPU	Chroma quantizer for *U* plane (QP_C in [ITUH264]). It must be within the range [0;39].
ChromaQPV	Chroma quantizer for *V* plane (QP_C in [ITUH264]). It must be within the range [0;39].
levelScaleDCU	Chroma quantizer for *U* plane ($LevelScale(QP_{C,DC}\%6, 0, 0)$ in [ITUH264]) for chroma DC coefficients.
levelScaleDCV	Chroma quantizer for *V* plane ($LevelScale(QP_{C,DC}\%6, 0, 0)$ in [ITUH264]) for chroma DC coefficients.
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for intra prediction,

	`edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero.
pQuantTableU	Pointer to the quantization table for *U* plane (*LevelScale(qP%6, i, j)* in [ITUH264]).
pQuantTableV	Pointer to the quantization table for *V* plane (*LevelScale(qP%6, i, j)* in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo[2]	Pointer to the `IppiReconstructHighMB_32s16u` and `IppiReconstructHighMB_16s8u` structures. The first pointer *pReconstructInfo*[0] points to the structure with parameters for restructuring chroma macroblock of plane *U*, and the second pointer *pReconstructInfo*[1] points to the structure with parameters for restructuring Chroms macroblock of plane *V*.

Description

The functions `ippiReconstructChroma422Intra4x4_H264High_32s16u_IP2R` and `ippiReconstructChroma422Intra4x4_H264High_16s8u_IP2R` are declared in the `ippvc.h` file. These functions reconstruct eight 4x4 intra chroma macroblocks (16x8 *U* macroblock and 16x8 *V* macroblock) for chroma format 4:2:2 (`chroma_format_idc` is equal to 2):

1. Perform integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [ITUH264] in the same way as `TransformDequantChromaDC_H264` function does.
2. Perform scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [ITUH264] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7(b) of [ITUH264].
3. Perform adding of 16x8 intra prediction blocks and 16x8 residual blocks in accordance with 8-306 of [ITUH264].

These functions ares used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*ChromaQPU* or *ChromaQPV* is less than 0 or greater than 39.

ReconstructLumaInterMB_H264

Reconstructs inter luma macroblock.

Syntax

```
IppStatus ippiReconstructLumaInterMB_H264_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
    pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u cbp4x4, Ipp32s QP);
```

Parameters

ppSrcCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in current *Y*-plane. This macroblock must contain inter prediction samples.
srcDstStep	Plane step.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(1+i))` is not equal to 0 ($0 \le i < 16$), i-th 4x4 AC luma block is not zero-filled and it exists in *ppSrcCoeff*.
QP	Quantization parameter (QP_Y in [JVTG050]). It must be within the range [0;51].

Description

The function `ippiReconstructLumaInterMB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs inter luma macroblock:

1. Performs scaling, integer inverse transformation, and shift for each 4x4 block in the same order as is shown in Figure 6-6 of [JVTG050] in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does.
2. Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	*QP* is less than 0 or greater than 51.

ReconstructLumaIntraHalfMB_H264

Reconstructs half of intra luma macroblock.

Syntax

```
IppStatus ippiReconstructLumaIntraHalfMB_H264_16s8u_C1R(Ipp16s** ppSrcCoeff,
    Ipp8u* pSrcDstYPlane, Ipp32s srcDstYStep, IppIntra4x4PredMode_H264*
    pMBIntraTypes, Ipp32u cbp4x2, Ipp32u QP, Ipp8u edgeType);
```

Parameters

ppSrcCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in *Y*-plane.
srcDstYStep	*Y*-Plane step.
pMBIntraTypes	Array of Intra_4x4 luma prediction modes for 8 subblocks. *pMBIntraTypes*[*i*] is defined in the same way as in PredictIntra_4x4_H264 function ($0 \le i < 8$).
cbp4x2	Coded block pattern. If cbp4x2 & (1<<(1+*i*))is not equal to 0 ($0 \le i < 8$), *i*-th 4x4 AC luma block is not zero-filled and it exists in *ppSrcCoeff*.
QP	Quantization parameter (QP_Y in [JVTG050]). It must be within the range [0;51].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 intra prediction, edgeType&IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 4x4 intra prediction, edgeType&IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 4x4 intra

prediction, edgeType&IPPVC_TOP_LEFT_EDGE must be non-zero. If the upper-right macroblock is not available for 4x4 intra prediction, edgeType&IPPVC_TOP_RIGHT_EDGE must be non-zero.

Description

The function ippiReconstructLumaIntraHalfMB_H264_16s8u_C1R is declared in the ippvc.h file. This function reconstructs a half of intra luma macroblock, that is, eight 4x4 subblocks. The process for the eight 4x4 blocks (from 0 to 7 or from 8 to 15) in the same order as is shown in Figure 6-6 of [JVTG050] is described below:

1. Performs scaling, integer inverse transformation, and shift for a 4x4 block in accordance with 8.5.8 of [JVTG050] in the same way as DequantTransformResidual_H264 function does.
2. Performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [JVTG050] in the same way as PredictIntra_4x4_H264 function does.
3. Performs adding of 16x8 prediction block and 16x8 residual block in accordance with 8-247 of [JVTG050].

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	*QP* is less than 0 or greater than 51.

ReconstructLumaIntraMB_H264

Reconstructs intra luma macroblock.

Syntax

```
IppStatus ippiReconstructLumaIntraMB_H264_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
    pSrcDstYPlane, Ipp32s srcDstYStep, const IppIntra4x4PredMode_H264*
    pMBIntraTypes, const Ipp32u cbp4x4, const Ipp32u QP, const Ipp8u edgeType);
```

Parameters

ppSrcCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in *Y*-plane.
srcDstYStep	*Y*-Plane step.
pMBIntraTypes	Array of `Intra_4x4` luma prediction modes for each subblock. *pMBIntraTypes*[*i*] is defined in the same way as in `PredictIntra_4x4_H264` function ($0 \le i < 16$).
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(1+i))` is not equal to 0 ($0 \le i < 16$), *i*-th 4x4 AC luma block is not zero-filled and it exists in *ppSrcCoeff*.
QP	Quantization parameter (QP_Y in [JVTG050]). It must be within the range [0;51].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero. If the upper-right macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_TOP_RIGHT_EDGE` must be non-zero.

Description

The function `ippiReconstructLumaIntraMB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs intra luma macroblock. The process for each 4x4 block in the same order as is shown in Figure 6-6 of [JVTG050] is described below:

1. Performs scaling, integer inverse transformation, and shift for a 4x4 block in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does.
2. Performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [JVTG050] in the same way as `PredictIntra_4x4_H264` function does.
3. Performs adding of 4x4 prediction block and 4x4 residual block in accordance with 8-247 of [JVTG050].

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	*QP* is less than 0 or greater than 51.

ReconstructLumaInter4x4MB_H264, ReconstructLumaInter4x4_H264High

Reconstruct 4X4 inter luma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructLumaInter4x4MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff,
    Ipp8u* pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u cbp4x4, Ipp32s QP, const
    Ipp16s* pQuantTable, Ipp8u bypassFlag);
IppStatus ippiReconstructLumaInter4x4_H264High_32s16u_IP1R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in current *Y*-plane. This macroblock must contain inter prediction samples.
srcDstStep	Plane step.
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(1+i))` is not equal to 0 ($0 \le i < 16$), i-th 4x4 AC luma block is not zero-filled and it exists in *ppSrcDstCoeff*.
QP	Quantization parameter (QP_Y in [ITUH264]). It must be within the range [0;51].
pQuantTable	Pointer to the quantization table (*LevelScale(qP%6, i, j)* in [ITUH264]).
bypassFlag	Flag enabling lossless coding.

pReconstructInfo Pointer to the `IppiReconstructHighMB_32s16u` structure.

Description

The functions `ippiReconstructLumaInter4x4MB_H264_16s8u_C1R` and `ippiReconstructLumaInter4x4_H264High_32s16u_IP1R` are declared in the `ippvc.h` file. These functions reconstruct a 4x4 inter luma macroblock for high profile:

1. Perform scaling, integer inverse transformation, and shift for each 4x4 block in the same order as is shown in Figure 6-10 of [ITUH264] in accordance with 8.5.10 of [ITUH264].
2. Perform adding of sixteen 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-295 of [ITUH264].

See Example 16-23 below for the usage example of `ippiReconstructLumaInter4x4MB_H264_16s8u_C1R`.

Example 16-23 Usage of `ippiReconstructLumaInter4x4MB_H264`

```
{
    Ipp8u   planeY[16*16] =
  { 0x9c, 0x92, 0x7f, 0x8b, 0x94, 0x22, 0x8a, 0xbc, 0xbb, 0x23, 0x60, 0xbb, 0x9a, 0xa9, 0x54, 0x48,
    0x13, 0xa3, 0x29, 0x8a, 0xba, 0x73, 0x65, 0x14, 0x9a, 0xc6, 0x34, 0xae, 0x2e, 0x5e, 0xbb, 0x38,
    0x70, 0x48, 0x5e, 0x68, 0xaf, 0x16, 0x22, 0x9a, 0x38, 0x6c, 0x54, 0x8f, 0x3e, 0x7f, 0x6a, 0x69,
    0x3e, 0x5e, 0x11, 0x7e, 0x08, 0x01, 0xb9, 0x8c, 0x76, 0x7b, 0x16, 0x87, 0xa5, 0xa4, 0xc6, 0x78,
    0xc3, 0x0a, 0x8a, 0x9d, 0x47, 0x0f, 0x9f, 0xa1, 0x90, 0x5f, 0x1d, 0x0b, 0x79, 0x36, 0x47, 0x79,
    0x5a, 0x45, 0x28, 0x87, 0x28, 0x0f, 0x4a, 0xb3, 0x24, 0x83, 0x14, 0xa4, 0x56, 0x5c, 0x81, 0x0b,
    0x29, 0x61, 0xc1, 0x4b, 0x96, 0x33, 0x65, 0xa3, 0x9f, 0x3b, 0x08, 0x25, 0xa9, 0x5c, 0x44, 0x97,
    0x50, 0x2b, 0x77, 0x64, 0x59, 0x2c, 0xb6, 0x30, 0x24, 0x7b, 0x47, 0x6b, 0xaa, 0x85, 0xa3, 0xa2,
    0x58, 0x66, 0x15, 0x31, 0x78, 0x7e, 0x3b, 0x20, 0x72, 0x25, 0xb3, 0x9a, 0x82, 0x13, 0x48, 0x53,
    0x61, 0x4f, 0x45, 0x0c, 0x69, 0xa2, 0xa1, 0x01, 0x62, 0x0f, 0x0a, 0x94, 0x17, 0x4b, 0x77, 0x41,
    0xa4, 0x59, 0x16, 0x17, 0x8e, 0x32, 0x61, 0x94, 0xbe, 0x1d, 0x22, 0x05, 0x11, 0x16, 0xb0, 0x4f,
    0x31, 0x97, 0x3e, 0x61, 0x92, 0x36, 0x13, 0x9c, 0x07, 0x27, 0x1a, 0x05, 0x5d, 0x82, 0x9b, 0x4e,
    0x94, 0x6f, 0x57, 0xc7, 0xb7, 0x99, 0x1e, 0x48, 0x33, 0x3f, 0x15, 0x9a, 0x30, 0x75, 0x03, 0x69,
    0x09, 0x86, 0x86, 0xb6, 0x53, 0x9d, 0x9b, 0x03, 0x9c, 0x80, 0xbf, 0x3a, 0x81, 0x0b, 0x59, 0xb5,
    0x6f, 0x81, 0x65, 0x29, 0xb7, 0x93, 0xb4, 0x2a, 0xbf, 0x05, 0x11, 0x53, 0x49, 0x92, 0x97, 0x3c,
    0x0d, 0xac, 0x8f, 0x5d, 0x88, 0x25, 0x99, 0x9b, 0x5e, 0x53, 0x5c, 0x07, 0x0f, 0x0c, 0x41, 0xc2
};

    Ipp32s  stepY = 16;
    Ipp16s  quantTable[16] = {22,12,34,50,34,33,46,21,18,19,34,28,28,31,15,3};
    Ipp16s  coef[16*16] = {
      12,-4, 0,  0, 0,-7, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0,
      0, 4, -5,  1, 0,-4, 3, 0, 0, 2, 0, 0, 0, 0, 0, 0,
      0, 0, 10, -2,-1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 7, -13, 2, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
      0, 0, -19, 6, 0, 0, 8,-4, 0, 0, 0, 1, 0, 0, 0, 0,
      0, 0, -1, -1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 25, -3,-2, 0, 0, 2,-1, 0,-2, 1, 0, 0, 0, 0,
      0, 0, -3,  0, 1,-2,-1, 0, 0, 0,-1, 0, 0, 0, 0, 0,
      0, 0, -1, -1, 1, 1, 2, 1,-2,-1, 0, 1, 0, 1, 0, 0,
      0, 0,  0,  0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1,-1,
```

Example 16-23 Usage of `ippiReconstructLumaInter4x4MB_H264` (continued)

```
    0, 0,  0,  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1, 1, 0,
    0, 0,  0,  0,-1, 0, 0, 0, 0, 0, 0, 0, 0,-1, 1, 0,
    0, 0,  0,  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0,  0,  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0,  0,  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0,  0,  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0  };

  Ipp32u  cbp4x4 = 0x0001fe00;
  Ipp32s  QP = 22;
  Ipp8u   bypass_flag = 0;
  IppStatus result;
  Ipp16s* pCoef;

  pCoef = coef;

  result = ippiReconstructLumaInter4x4MB_H264_16s8u_C1R(&pCoef,
                                                        planeY,
                                                        stepY,
                                                        cbp4x4,
                                                        QP,
                                                        quantTable,
                                                        bypass_flag);
}
```

These functions are used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*QP* is less than 0 or greater than 51.

ReconstructLumaIntraHalf4x4MB_H264, ReconstructLumaIntraHalf4x4_H264High

Reconstruct half of 4X4 intra luma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructLumaIntraHalf4x4MB_H264_16s8u_C1R(Ipp16s**
    ppSrcDstCoeff, Ipp8u* pSrcDstYPlane, Ipp32s srcDstYStep,,
    IppIntra4x4PredMode_H264* pMBIntraTypes, Ipp32u cbp4x2, Ipp32s QP, Ipp8u
    edgeType, const Ipp16s* pQuantTable, Ipp8u bypassFlag);
IppStatus ippiReconstructLumaIntraHalf4x4_H264High_32s16u_IP1R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo, const
    IppIntra4x4PredMode_H264* pMBIntraTypes, Ipp32u edgeType);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in *Y*-plane.
srcDstYStep	*Y*-plane step.
pMBIntraTypes	Array of `Intra_4x4` luma prediction modes for eight subblocks. *pMBIntraTypes*[*i*] is defined in the same way as in `PredictIntra_4x4_H264` function ($0 \le i < 8$).
cbp4x2	Coded block pattern. If `cbp4x2 & (1<<(1+i))` is not equal to 0 ($0 \le i < 8$), *i*-th 4x4 AC luma block is not zero-filled and it exists in *ppSrcDstCoeff*.
QP	Quantization parameter (QP_Y in [ITUH264]). It must be within the range [0;51].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 4x4 intra

	prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero. If the upper-right macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_TOP_RIGHT_EDGE` must be non-zero.
pQuantTable	Pointer to the quantization table (*LevelScale(qP%6, i, j)* in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo	Pointer to the `IppiReconstructHighMB_32s16u` structure.

Description

The functions `ippiReconstructLumaIntraHalf4x4MB_H264_16s8u_C1R` and `ippiReconstructLumaIntraHalf4x4_H264High_32s16u_IP1R` are declared in the `ippvc.h` file. These functions reconstruct half of an intra luma macroblock for high profile. The process for eight 4x4 blocks (from 0 to 7 or from 8 to 15) in the same order as is shown in Figure 6-10 of [ITUH264] is described below:

1. Performs scaling, integer inverse transformation, and shift for 4x4 blocks in accordance with 8.5.10 of [ITUH264].
2. Performs intra prediction for 4x4 luma components in accordance with 8.3.1.2 of [ITUH264].
3. Performs adding of a 16x8 prediction block (eight 4x4 blocks) and a 16x8 residual block in accordance with 8-295 of [ITUH264].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*QP* is less than 0 or greater than 51.

ReconstructLumaIntra4x4MB_H264, ReconstructLumaIntra4x4_H264High

Reconstruct 4X4 intra luma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructLumaIntra4x4MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff,
    Ipp8u* pSrcDstYPlane, Ipp32s srcDstYStep,, IppIntra4x4PredMode_H264*
    pMBIntraTypes, Ipp32u cbp4x4, Ipp32s QP, Ipp8u edgeType, const Ipp16s*
    pQuantTable, Ipp8u bypassFlag);
IppStatus ippiReconstructLumaIntra4x4_H264High_32s16u_IP1R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo, const
    IppIntra4x4PredMode_H264* pMBIntraTypes, Ipp32s edgeType);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in current *Y*-plane.
srcDstYStep	*Y*-plane step.
pMBIntraTypes	Array of `Intra_4x4` luma prediction modes for each subblock. *pMBIntraTypes*[*i*] is defined in the same way as in `PredictIntra_4x4_H264` function ($0 \le i < 16$).
cbp4x4	Coded block pattern. If `cbp4x4 & (1<<(1+i))` is not equal to 0 ($0 \le i < 16$), *i*-th 4x4 AC luma block is not zero-filled and it exists in *ppSrcDstCoeff*.
QP	Quantization parameter (QP_Y in [ITUH264]). It must be within the range [0;51].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 4x4 intra

	prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero. If the upper-right macroblock is not available for 4x4 intra prediction, `edgeType&IPPVC_TOP_RIGHT_EDGE` must be non-zero.
pQuantTable	Pointer to the quantization table (*LevelScale(qP%6, i, j)* in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo	Pointer to the `IppiReconstructHighMB_32s16u` structure.

Description

The functions `ippiReconstructLumaIntra4x4MB_H264_16s8u_C1R` and `ippiReconstructLumaIntra4x4_H264High_32s16u_IP1R` are declared in the `ippvc.h` file. These functions reconstruct inter luma macroblock for high profile. The process for each 4x4 block in the same order as is shown in Figure 6-10 of [ITUH264] is described below:

1. Perform scaling, integer inverse transformation, and shift for each 4x4 block in accordance with 8.5.10 of [ITUH264].
2. Perform intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [ITUH264].
3. Perform adding of sixteen 4x4 prediction blocks and sixteen 4x4 residual blocks in accordance with 8-295 of [ITUH264].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*QP* is less than 0 or greater than 51.

ReconstructLumaInter8x8MB_H264, ReconstructLumaInter8x8_H264High

Reconstructs 8X8 inter luma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructLumaInter8x8MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff,
    Ipp8u* pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u cbp8x8, Ipp32s QP, const
    Ipp16s* pQuantTable, Ipp8u bypassFlag);
IppStatus ippiReconstructLumaInter8x8_H264High_32s16u_IP1R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (8x8 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-11 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in current *Y*-plane. This macroblock must contain inter prediction samples.
srcDstStep	Plane step.
cbp8x8	Coded block pattern. If `cbp8x8 & (1<<(1+i))` is not equal to 0 ($0 \le i < 4$), i-th 8x8 AC luma block is not zero-filled and it exists in *ppSrcDstCoeff*.
QP	Quantization parameter (QP_Y in [ITUH264]). It must be within the range [0;51].
pQuantTable	Pointer to the quantization table (*LevelScale(qP%6, i, j)* in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo	Pointer to the `IppiReconstructHighMB_32s16u` structure.

Description

The function `ippiReconstructLumaInter8x8MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs 8x8 inter luma macroblock for high profile:

1. Performs scaling, integer inverse transformation, and shift for each 8x8 block in the same order as is shown in Figure 6-11 of [ITUH264] in accordance with 8.5.11 of [ITUH264].
2. Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-298 of [ITUH264].

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	*QP* is less than 0 or greater than 51.

ReconstructLumaIntraHalf8x8MB_H264, ReconstructLumaIntraHalf8x8_H264High

Reconstruct half of 8X8 intra luma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructLumaIntraHalf8x8MB_H264_16s8u_C1R(Ipp16s**
    ppSrcDstCoeff, Ipp8u* pSrcDstYPlane, Ipp32s srcDstYStep,
    IppIntra8x8PredMode_H264* pMBOIntraTypes, Ipp32u cbp8x2, Ipp32s QP, Ipp8u
    edgeType, const Ipp16s* pQuantTable, Ipp8u bypassFlag);
IppStatus ippiReconstructLumaIntraHalf8x8_H264High_32s16u_IP1R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo, IppIntra8x8PredMode_H264*
    pMBIntraTypes, Ipp32u edgeType);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (8x8 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-11 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in *Y*-plane.
srcDstStep	*Y*-plane step.

pMBIntraTypes	Array of `Intra_8x8` luma prediction modes for each subblock. *pMBIntraTypes*[*i*] is defined for subblocks ($0 \le i < 2$) as described in 8.3.2.1 of [ITUH264].
cbp8x2	Coded block pattern. If `cbp8x2 & (1<<(1+i))` is not equal to 0 ($0 \le i < 2$), *i*-th 8x8 AC luma block is not zero-filled and it exists in *ppSrcDstCoeff*.
QP	Quantization parameter (QP_Y in [ITUH264]). It must be within the range [0;51].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero. If the upper-right macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_RIGHT_EDGE` must be non-zero.
pQuantTable	Pointer to the quantization table (*LevelScale(qP%6, i, j)* in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo	Pointer to the `IppiReconstructHighMB_32s16u` structure.

Description

The functions `ippiReconstructLumaIntraHalf8x8MB_H264_16s8u_C1R` and `ippiReconstructLumaIntraHalf8x8_H264High_32s16u_IP1R` are declared in the `ippvc.h` file. These functions reconstruct 8x8 intra luma macroblock for high profile. The process for each 8x8 block in the same order as shown in Figure 6-11 of [ITUH264] is described below:

1. Performs scaling, integer inverse transformation, and shift for a 8x8 block in accordance with 8.5.11 of [ITUH264].
2. Performs intra prediction for a 8x8 luma component in accordance with 8.3.2 of [ITUH264].
3. Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-298 of [ITUH264].

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	*QP* is less than 0 or greater than 51.

ReconstructLumaIntra8x8MB_H264, ReconstructLumaIntra8x8_H264High

Reconstruct 8X8 intra luma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructLumaIntra8x8MB_H264_16s8u_C1R(Ipp16s** ppSrcDstCoeff,
    Ipp8u* pSrcDstYPlane, Ipp32s srcDstYStep, IppIntra8x8PredMode_H264*
    pMBOIntraTypes, Ipp32u cbp8x8, Ipp32s QP, Ipp8u edgeType, const Ipp16s*
    pQuantTable, Ipp8u bypassFlag);
IppStatus ippiReconstructLumaIntra8x8_H264High_32s16u_IP1R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo, IppIntra8x8PredMode_H264*
    pMBIntraTypes, Ipp32u edgeType);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (8x8 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-11 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in current *Y*-plane.
srcDstStep	*Y*-plane step.
pMBIntraTypes	Array of Intra_8x8 luma prediction modes for each subblock. *pMBIntraTypes*[*i*] is defined for subblocks ($0 \le i < 4$) as described in 8.3.2.1 of [ITUH264].
cbp8x8	Coded block pattern. If cbp8x8 & (1<<(1+i)) is not equal to 0 ($0 \le i < 4$), *i*-th 8x8 AC luma block is not zero-filled and it exists in *ppSrcDstCoeff*.
QP	Quantization parameter (QP_Y in [ITUH264]). It must be within the range [0;51].

edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero. If the upper-right macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_RIGHT_EDGE` must be non-zero.
pQuantTable	Pointer to the quantization table (*LevelScale(qP%6, i, j*) in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo	Pointer to the `IppiReconstructHighMB_32s16u` structure.

Description

The functions `ippiReconstructLumaIntra8x8MB_H264_16s8u_C1R` and `ippiReconstructLumaIntra8x8_H264High_32s16u_IP1R` are declared in the `ippvc.h` file. These functions reconstruct 8x8 intra luma macroblock for high profile. The process for each 8x8 block in the same order as shown in Figure 6-11 of [ITUH264] is described below:

1. Performs scaling, integer inverse transformation, and shift for a 8x8 block in accordance with 8.5.11 of [ITUH264].
2. Performs intra prediction for a 8x8 luma component in accordance with 8.3.2 of [ITUH264].
3. Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-298 of [ITUH264].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*QP* is less than 0 or greater than 51.

ReconstructLumaIntra16x16MB_H264

Reconstructs 16X16 intra luma macroblock.

Syntax

```
IppStatus ippiReconstructLumaIntra16x16MB_H264_16s8u_C1R(Ipp16s** ppSrcCoeff,
    Ipp8u* pSrcDstYPlane, Ipp32u srcDstYStep, const IppIntra16x16PredMode_H264
    intraLumaMode, const Ipp32u cbp4x4, const Ipp32u QP, const Ipp8u edgeType);
```

Parameters

ppSrcCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 DC luma blocks, 4x4 AC luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in *Y*-plane.
srcDstYStep	*Y*-Plane step.
intraLumaMode	Prediction mode of the `Intra_16x16` prediction process for luma samples. It is defined in the same way as in `PredictIntra_16x16_H264` function.
cbp4x4	Coded block pattern. If `cbp4x4 & IPPVC_CBP_LUMA_DC` is not equal to 0, 4x4 DC luma block is not zero-filled and it exists in *ppSrcCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_LUMA_AC_BITPOS+i))` is not equal to 0 ($0 \le i < 16$), *i*-th 4x4 AC luma block is not zero-filled and it exists in *ppSrcCoeff*.
QP	Quantization parameter (QP_Y in [JVTG050]). It must be within the range [0;51].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero.

Description

The function `ippiReconstructLumaIntra16x16MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs 16x16 intra luma macroblock:

1. Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients in accordance with 8.5.6 of [JVTG050] in the same way as `TransformDequantLumaDC_H264` function does.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC blocks in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 AC blocks in the same order as Figure 6-6 of [JVTG050] shows.
3. Performs intra prediction for a 16x16 luma component in accordance with 8.3.2 of [JVTG050] in the same way as `PredictIntra_16x16_H264` function does.
4. Performs adding of 16x16 prediction blocks and 16x16 residual blocks in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*QP* is less than 0 or greater than 51.

ReconstructLumaIntra_16x16MB_H264, ReconstructLumaIntra16x16_H264High

Reconstruct 16X16 intra luma macroblock for high profile.

Syntax

```
IppStatus ippiReconstructLumaIntra_16x16MB_H264_16s8u_C1R(Ipp16s**
    ppSrcDstCoeff, Ipp8u* pSrcDstYPlane, Ipp32u srcDstYStep,
    IppIntra16x16PredMode_H264 intraLumaMode, Ipp32u cbp4x4, Ipp32u QP, Ipp8u
    edgeType, const Ipp16s* pQuantTable, Ipp8u bypassFlag);
```

```
IppStatus ippiReconstructLumaIntra16x16_H264High_32s16u_IP1R(const
    IppiReconstructHighMB_32s16u*  pReconstructInfo, IppIntra16x16PredMode_H264
    intraLumaMode, Ipp32u edgeType);
```

Parameters

ppSrcDstCoeff	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in *Y*-plane.
srcDstYStep	*Y*-Plane step.
intraLumaMode	Prediction mode of the `Intra_16x16` prediction process for luma samples. It is defined in the same way as in `PredictIntra_16x16_H264` function.
cbp4x4	Coded block pattern. If `cbp4x4 & IPPVC_CBP_LUMA_DC` is not equal to 0, 4x4 DC luma block is not zero-filled and it exists in *ppSrcDstCoeff*. If `cbp4x4 & (1<<(IPPVC_CBP_1ST_LUMA_AC_BITPOS+i))` is not equal to 0 ($0 \le i < 16$), *i*-th 4x4 AC luma block is not zero-filled and it exists in *ppSrcDstCoeff*.
QP	Quantization parameter (QP_Y in [ITUH264]). It must be within the range [0;51].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 16x16 intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero.
pQuantTable	Pointer to the quantization table (*LevelScale(qP%6, i, j)* in [ITUH264]).
bypassFlag	Flag enabling lossless coding.
pReconstructInfo	Pointer to the `IppiReconstructHighMB_32s16u` structure.

Description

The functions `ippiReconstructLumaIntra_16x16MB_H264_16s8u_C1R` and `ippiReconstructLumaIntra16x16_H264High_32s16u_IP1R` are declared in the `ippvc.h` file. These functions reconstruct 16x16 intra luma macroblock for high profile:

1. Perform integer inverse transformation and dequantization for 4x4 luma DC coefficients in accordance with 8.5.8 of [ITUH264].
2. Perform scaling, integer inverse transformation, and shift for 4x4 AC blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 AC blocks in the same order as Figure 6-10 of [ITUH264] shows.
3. Perform intra prediction for a 16x16 luma component in accordance with 8.3.3 of [ITUH264] in the same way as `PredictIntra_16x16_H264` function does.
4. Perform adding of 16x16 prediction blocks and 16x16 residual blocks in accordance with 8-297 of [ITUH264].

These functions are used in the H.264 decoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*QP* is less than 0 or greater than 51.

Deblocking Filtering

Note that the functions with `_IP1R` descriptor use the following structures:

```
typedef struct _IppiFilterDeblock_16u
{
  Ipp16u*   pSrcDstPlane;
  Ipp32s    srcDstStep;
  Ipp16u*   pAlpha;
  Ipp16u*   pBeta;
  Ipp16u*   pThresholds;
  Ipp8u*    pBs;
  Ipp32s    bitDepth;
} IppiFilterDeblock_16u;

typedef struct _IppiFilterDeblock_8u
{
  Ipp8u*   pSrcDstPlane;
```

```
  Ipp32s     srcDstStep;
  Ipp8u*     pAlpha;
  Ipp8u*     pBeta;
  Ipp8u*     pThresholds;
  Ipp8u*     pBs;
} IppiFilterDeblock_8u;
```

with parameters:

pSrcDstPlane	Pointer to the upper left pixel of the macroblock and resultant samples.
srcDstStep	Distance in items between starts of the consecutive lines in the source/destination plane.
pAlpha	Alpha Thresholds.
pBeta	Beta Thresholds.
pThresholds	Thresholds (T_{c0}).
pBs	BS parameters.
bitDepth	Number of bits of the plane sample with range [8.. 14].

Number of elements in the arrays the `pAlpha`, `pBeta`, *pThresholds*, and *pBs* pointers point to may vary from function to function.

FilterDeblockingLuma_VerEdge_H264

Performs deblocking filtering on the vertical edges of the 16x16 luma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR(Ipp8u* pSrcDst,
    Ipp32s srcdstStep, const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u*
    pThresholds, const Ipp8u* pBs);
IppStatus ippiFilterDeblockingLumaVerEdge_H264_16u_C1IR(const
    IppiFilterDeblock_16u*  pDeblockInfo);
```

Parameters

pSrcDst	Pointer to the initial and resultant coefficients.

srcdstStep	Distance in items between starts of the consecutive lines in the array.
pAlpha	Array of size 2 of Alpha Thresholds (values for external and internal vertical edge).
pBeta	Array of size 2 of Beta Thresholds (values for external and internal vertical edge).
pThresholds	Array of size 16 of Thresholds (T_{c0}) (values for the left edge of each 4x4 block).
pBs	Array of size 16 of BS parameters (values for the left edge of each 4x4 block).
pDeblockInfo	Pointer to the `IppiFilterDeblock_16u` structure.

Description

The functions `ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR` and `ippiFilterDeblockingLumaVerEdge_H264_16u_C1IR` are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the vertical edges of the 16x16 luma macroblock in accordance with 8.7.2 of [JVTG050].

The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0] are values for the external vertical edge, and *pAlpha*[1], *pBeta*[1] are values for the internal vertical edge. See Figure 16-69 for the arrangement of *pThresholds* and *pBs* array elements.

Values of the arrays are calculated as follows:

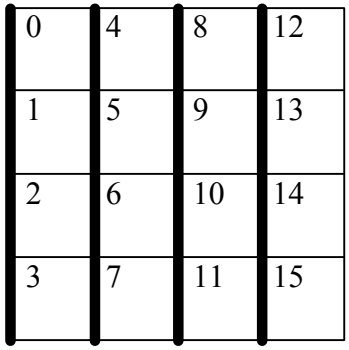
— *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.

— *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].

— *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].

— *pThresholds*[i] values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

Figure 16-69 Arrangement of *pThresholds* Array Elements into a macroblock



This function is used in the H.264 decoder and encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

FilterDeblockingLuma_VerEdge_MBAFF_H264

Performs deblocking filtering on the external vertical edges of half of 16X16 luma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingLuma_VerEdge_MBAFF_H264_8u_C1IR(Ipp8u* pSrcDst,
    Ipp32s srcdstStep, Ipp32u nAlpha, Ipp32u nBeta, const Ipp8u* pThresholds,
    const Ipp8u* pBs);
IppStatus ippiFilterDeblockingLumaVerEdgeMBAFF_H264_16u_C1IR(const
    IppiFilterDeblock_16u* pDeblockInfo);
```

Parameters

pSrcDst	Pointer to the initial and resultant coefficients.
srcdstStep	Distance in items between starts of the consecutive lines in the array.
pAlpha	Alpha Threshold.
pBeta	Beta Threshold.
pThresholds	Array of size 8 of Thresholds (T_{c0}) (two values for the left edge of each 4x4 block).
pBs	Array of size 8 of BS parameters (two values for the left edge of each 4x4 block).
pDeblockInfo	Pointer to the `IppiFilterDeblock_16u` structure.

Description

The functions `ippiFilterDeblockingLuma_VerEdge_MBAFF_H264_8u_C1IR` and `ippiFilterDeblockingLumaVerEdgeMBAFF_H264_16u_C1IR` are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the vertical edges of the 16x16 luma macroblock in accordance with 8.7.2 of [JVTG050].

In process of filling *pThresholds* and *pBs* parameters for deblocking of each 4x4 block, a number of parameters of the neighboring blocks are taken into account. In MBAFF mode, the edge blocks may border on two others when the left and the current macroblocks are coded differently: one as a field, the other as a frame, and vice versa. As a result, two values of *pThresholds* and *pBs* parameters are taken for each block. For higher flexibility, the function processes only half of the macroblock to fit both types of the macroblocks, that is, coded as a field and coded as a frame.

The functions use arrays *nAlpha*, *nBeta*, *pBs*, *pThresholds* as input arguments. See Figure 16-69 for the arrangement of *pThresholds* and *pBs* array elements.

Values of the arrays are calculated as follows:

— *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.

— *nAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].

— *nBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].

— *pThresholds*[i] values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

FilterDeblockingLuma_HorEdge_H264

Performs deblocking filtering on the horizontal edges of the 16X16 luma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingLuma_HorEdge_H264_8u_C1IR(Ipp8u* pSrcDst,
    Ipp32s srcdstStep, const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u*
    pThresholds, const Ipp8u* pBs);
IppStatus ippiFilterDeblockingLumaHorEdge_H264_16u_C1IR(const
    IppiFilterDeblock_16u*  pDeblockInfo);
```

Parameters

pSrcDst	Pointer to the initial and resultant coefficients.
srcdstStep	Distance in items between starts of the consecutive lines in the array.
pAlpha	Array of size 2 of Alpha Thresholds (values for external and internal horizontal edge).
pBeta	Array of size 2 of Beta Thresholds (values for external and internal horizontal edge).
pThresholds	Array of size 16 of Thresholds (T_{c0}) (values for upper edge of each 4x4 block).
pBs	Array of size 16 of BS parameters (values for upper edge of each 4x4 block).
pDeblockInfo	Pointer to the IppiFilterDeblock_16u structure.

Description

The functions ippiFilterDeblockingLuma_HorEdge_H264_8u_C1IR and ippiFilterDeblockingLumaHorEdge_H264_16u_C1IR are declared in the ippvc.h file. These functions perform Deblocking Filtering on the horizontal edges of the 16x16 luma macroblock in accordance with 8.7.2 of [JVTG050].

The functions use arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0] are values for the external horizontal edge, and *pAlpha*[1], *pBeta*[1] are values for the internal horizontal edge. See Figure 16-70 for the arrangement of *pThresholds* and *pBs* array elements.

Values of the arrays are calculated as follows:

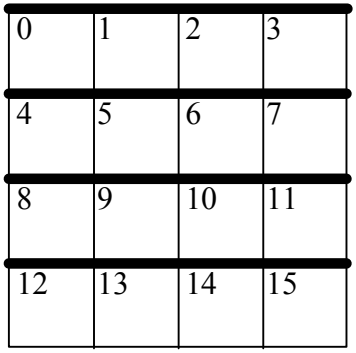
— *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.

— *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].

— *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].

— *pThresholds*[i] values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

Figure 16-70 Arrangement of *pThresholds* Array Elements into a Macroblock



This function is used in the H.264 decoder and encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

FilterDeblockingChroma_VerEdge_H264, FilterDeblockingChroma422VerEdge_H264

Performs deblocking filtering on the vertical edges of the 8X8 chroma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR(Ipp8u* pSrcDst,
    Ipp32s srcdstStep, const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u*
    pThresholds, const Ipp8u* pBs);
IppStatus ippiFilterDeblockingChromaVerEdge_H264_16u_C1IR(const
    IppiFilterDeblock_16u*  pDeblockInfo);
IppStatus ippiFilterDeblockingChroma422VerEdge_H264_8u_C1IR(const
    IppiFilterDeblock_8u*  pDeblockInfo);
IppStatus ippiFilterDeblockingChroma422VerEdge_H264_16u_C1IR(const
    IppiFilterDeblock_16u*  pDeblockInfo);
```

Parameters

pSrcDst	Pointer to the initial and resultant coefficients.
srcdstStep	Distance in items between starts of the consecutive lines in the array.
pAlpha	Array of size 2 of Alpha Thresholds (values for external and internal vertical edge).
pBeta	Array of size 2 of Beta Thresholds (values for external and internal vertical edge).
pThresholds	Array of size 8 of Thresholds (T_{c0}) (values for the left edge of each 2x2 block).
pBs	Array of size 16 of BS parameters (values for the left edge of each 2x2 block).
pDeblockInfo	Pointer to the `IppiFilterDeblock_16u` structure.

Description

The functions `ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR`, `ippiFilterDeblockingChromaVerEdge_H264_16u_C1IR`, `ippiFilterDeblockingChroma422VerEdge_H264_8u_C1IR`, and

`ippiFilterDeblockingChroma422VerEdge_H264_16u_C1IR` are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the vertical edge of the 8x8 chroma macroblock in accordance with 8.7.2 of [JVTG050].

The functions use arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments. *pAlpha*, *pBeta*, and *pBs* are the same arrays as in `FilterDeblockingLuma_VerEdge_H264` function. *pAlpha*[0], *pBeta*[0] are values for the external vertical edge, and *pAlpha*[1], *pBeta*[1] are values for the internal vertical edge. See Figure 16-71 for the arrangement of *pBs* and *pThresholds* array elements into an 8x8 chroma block of 4:2:0 format and Figure 16-72 for their arrangement into an 8x16 block of 4:2:2 format.

Figure 16-71 Arrangement of *pBs* and *pThresholds* Array Elements for 4:2:0 format

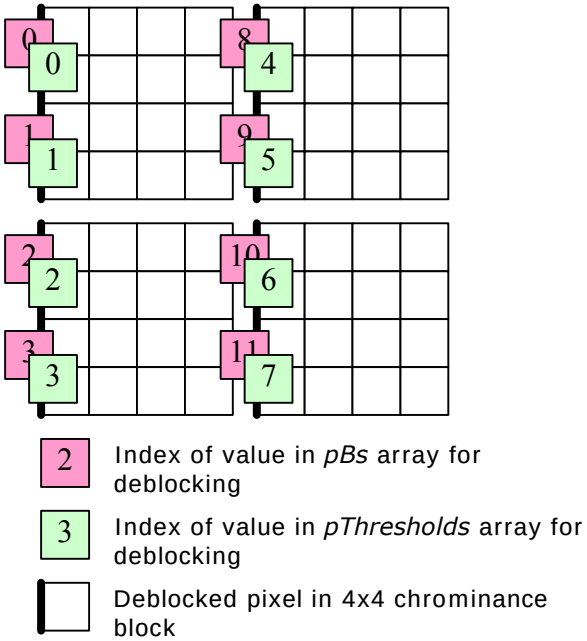
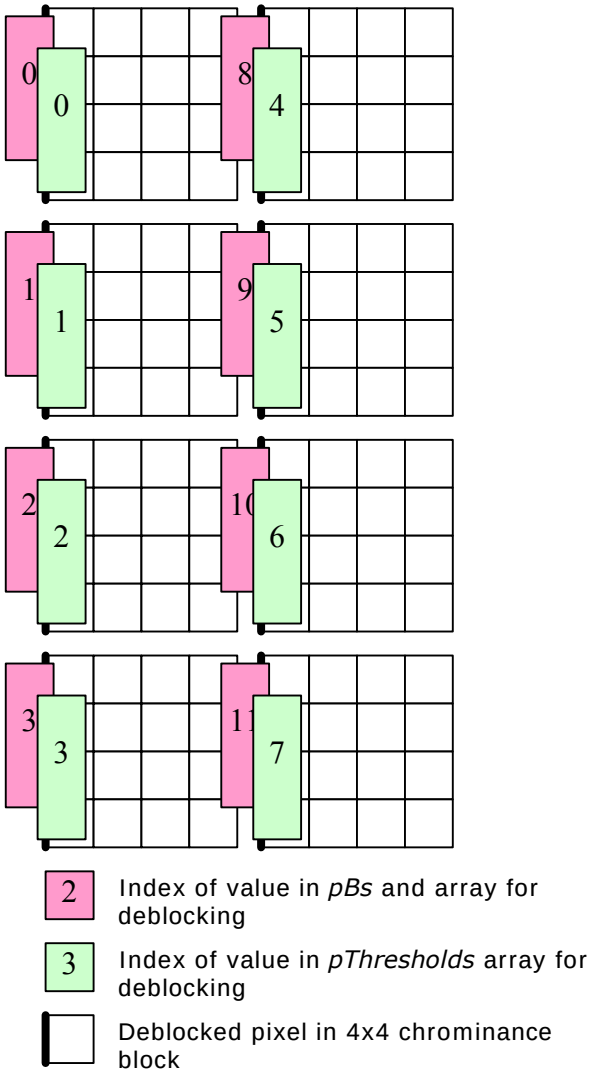


Figure 16-72 Arrangement of *pBs* and *pThresholds* Array Elements for 4:2:2 format



As two vertical edges are filtered for the chroma component, the function uses *pBs* array elements with indices [0,3] for the external edge and [8,11] for the internal edge.

Values of the arrays are calculated as follows:

— *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.

— *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].

— *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].

— *pThresholds*[i] values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

Example 16-24Usage of ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR

```
{
    Ipp8u    pSrcDst[8*8] =
    {
        0x78, 0x77, 0x76, 0x74, 0x73, 0x73, 0x74, 0x75,
        0x78, 0x77, 0x76, 0x74, 0x73, 0x73, 0x75, 0x79,
        0x76, 0x76, 0x76, 0x75, 0x75, 0x75, 0x7c, 0x7e,
        0x75, 0x75, 0x75, 0x76, 0x76, 0x76, 0x7c, 0x7e,
        0x78, 0x79, 0x7a, 0x7c, 0x7d, 0x7e, 0x81, 0x82,
        0x78, 0x79, 0x7a, 0x7c, 0x7d, 0x7e, 0x81, 0x82,
        0x7a, 0x7b, 0x7c, 0x7e, 0x7e, 0x7f, 0x81, 0x82,
        0x7d, 0x7e, 0x7f, 0x81, 0x81, 0x82, 0x82, 0x82
    };
    Ipp8u   pAlpha[2]       = {25,25};
    Ipp8u   pBeta[2]        = {8,8};
    Ipp8u   pThreshold[16]  =
{255,255,255,255,255,255,1,1,204,204,204,204,204,204,204,204};
    Ipp8u   pBs[16]         = {0,0,0,0,0,0,0,2,0,0,1,2,0,0,0,0};

    ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR (pSrcDst,
                                                     8,
                                                     pAlpha,
                                                     pBeta,
                                                     pThreshold,
                                                     pBs);
}
```

These functions are used in the H.264 decoder and encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

FilterDeblockingChroma_VerEdge_MBAFF_H264, FilterDeblockingChroma422VerEdgeMBAFF_H264

Performs deblocking filtering on the vertical edges of half of 8X8 chroma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR( Ipp8u*
    pSrcDst, Ipp32s srcdstStep, Ipp32u nAlpha, Ipp32u nBeta, const Ipp8u*
    pThresholds, const Ipp8u* pBs);
IppStatus ippiFilterDeblockingChromaVerEdgeMBAFF_H264_16u_C1IR(const
    IppiFilterDeblock_16u*  pDeblockInfo);
IppStatus ippiFilterDeblockingChroma422VerEdgeMBAFF_H264_8u_C1IR(const
    IppiFilterDeblock_8u*  pDeblockInfo);
IppStatus ippiFilterDeblockingChroma422VerEdgeMBAFF_H264_16u_C1IR(const
    IppiFilterDeblock_16u*  pDeblockInfo);
```

Parameters

pSrcDst	Pointer to the initial and resultant coefficients.
srcdstStep	Distance in items between starts of the consecutive lines in the array.
pAlpha	Alpha Threshold.
pBeta	Beta Threshold.
pThresholds	Array of size 4 of Thresholds (T_{c0}) (two values for the left edge of each 2x2 block).
pBs	Array of size 4 of BS parameters (two values for the left edge of each 2x2 block).
pDeblockInfo	Pointer to the IppiFilterDeblock_16u structure.

Description

The functions ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR, ippiFilterDeblockingChromaVerEdgeMBAFF_H264_16u_C1IR, ippiFilterDeblockingChroma422VerEdgeMBAFF_H264_8u_C1IR, and

`ippiFilterDeblockingChroma422VerEdgeMBAFF_H264_16u_C1IR` are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the vertical edge of the 8x8 chroma macroblock in accordance with 8.7.2 of [JVTG050].

In process of filling *pThresholds* and *pBs* parameters for deblocking of each 2x2 block, a number of parameters of the neighboring blocks are taken into account. In MBAFF mode, the edge blocks may border on two others when the left and the current macroblocks are coded differently: one as a field, the other as a frame, and vice versa. As a result, two values of *pThresholds* and *pBs* parameters are taken for each block. For higher flexibility, the functions process only half of the macroblock to fit both types of the macroblocks, that is, coded as a field and coded as a frame.

The functions use arrays *nAlpha*, *nBeta*, *pBs*, *pThresholds* as input arguments. *nAlpha*, *nBeta*, and *pBs* are the same arrays as in `FilterDeblockingLuma_VerEdge_H264` function. See Figure 16-71 for the arrangement of *pBs* and *pThresholds* array elements into an 8x8 chroma block of 4:2:0 format and Figure 16-72 for their arrangement into an 8x16 block of 4:2:2 format.

Values of the arrays are calculated as follows:

— *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.

— *nAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].

— *nBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].

— *pThresholds*[i] values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is NULL.

FilterDeblockingChroma_HorEdge_H264, FilterDeblockingChroma422HorEdge_H264

Performs deblocking filtering on the horizontal edges of the 8X8 chroma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR(Ipp8u* pSrcDst,
    Ipp32s srcdstStep, const Ipp8u* pAlpha, const Ipp8u* pBeta, const Ipp8u*
    pThresholds, const Ipp8u* pBs);
IppStatus ippiFilterDeblockingChromaHorEdge_H264_16u_C1IR(const
    IppiFilterDeblock_16u*  pDeblockInfo);
IppStatus ippiFilterDeblockingChroma422HorEdge_H264_8u_C1IR(const
    IppiFilterDeblock_8u*  pDeblockInfo);
IppStatus ippiFilterDeblockingChroma422HorEdge_H264_16u_C1IR(const
    IppiFilterDeblock_16u*  pDeblockInfo);
```

Parameters

pSrcDst	Pointer to the initial and resultant coefficients.
srcdstStep	Distance in items between starts of the consecutive lines in the array.
pAlpha	Array of size 2 of Alpha Thresholds (values for external and internal horizontal edge).
pBeta	Array of size 2 of Beta Thresholds (values for external and internal horizontal edge).
pThresholds	Array of size 8 of Thresholds (T_{c0}) (values for upper edge of each 2x2 block).
pBs	Array of size 16 of BS parameters (values for upper edge of each 2x2 block).
pDeblockInfo	Pointer to the `IppiFilterDeblock_16u` structure.

Description

The functions `ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR`, `ippiFilterDeblockingChromaHorEdge_H264_16u_C1IR`, `ippiFilterDeblockingChroma422HorEdge_H264_8u_C1IR`, and

`ippiFilterDeblockingChroma422HorEdge_H264_16u_C1IR` are declared in the `ippvc.h` file. These functions perform Deblocking Filtering on the horizontal edge of the 8x8 chroma macroblock in accordance with 8.7.2 of [JVTG050].

The functions use arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments. *pAlpha*, *pBeta*, and *pBs* are the same arrays as in `FilterDeblockingLuma_HorEdge_H264` function. *pAlpha*[0], *pBeta*[0] are values for the external horizontal edge, and *pAlpha*[1], *pBeta*[1] are values for the internal horizontal edge. See Figure 16-73 for the arrangement of *pBs* and *pThresholds* array elements into an 8x8 chroma block of 4:2:0 format and Figure 16-74 for their arrangement into an 8x16 block of 4:2:2 format.

Figure 16-73 Arrangement of *pBs* and *pThresholds* Array Elements for 4:2:0 format

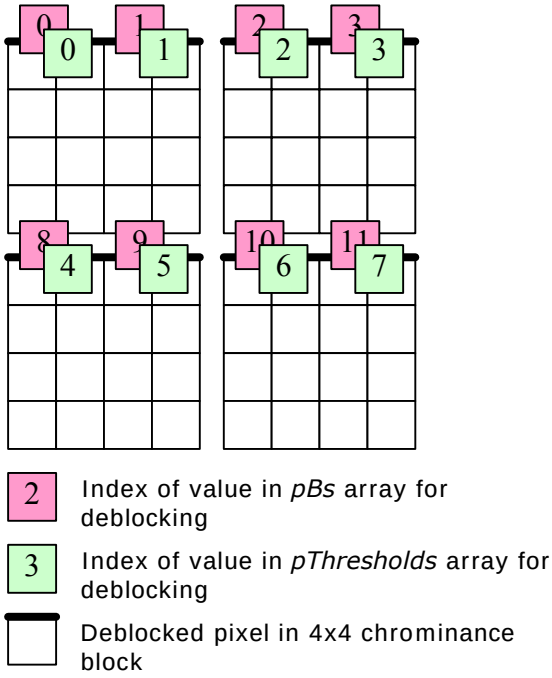
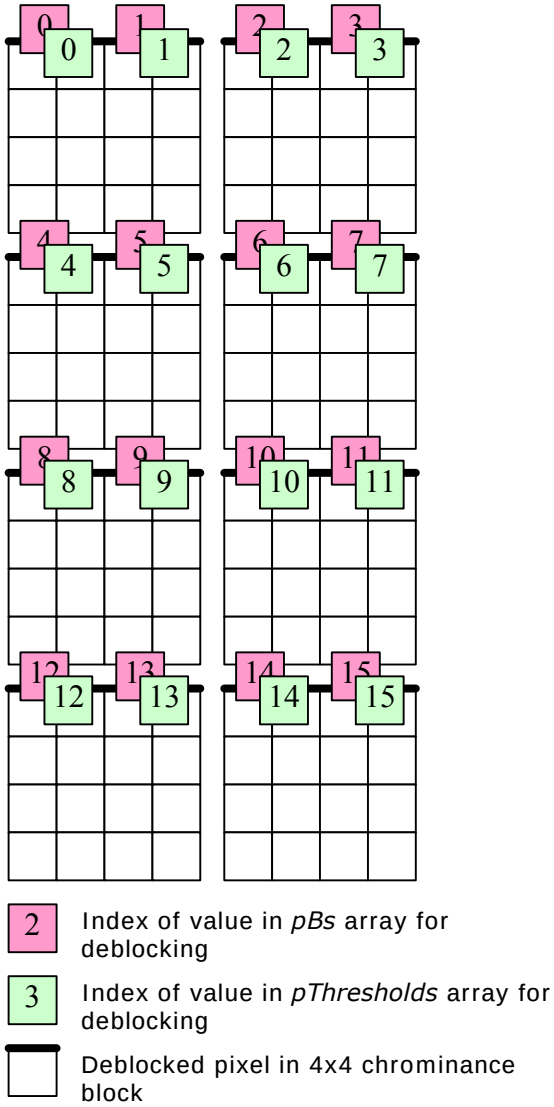


Figure 16-74 Arrangement of *pBs* and *pThresholds* Array Elements for 4:2:2 format



As two horizontal edges are filtered for the chroma component, the function uses *pBs* array elements with indices [0,3] for the external edge and [8,11] for the internal edge.

Values of the arrays are calculated as follows:

— *pBs* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.

— *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].

— *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].

pThresholds[i] values are calculated from the formulas 8-326, 8-327, values of *pBs* array and Table 8-15 of [JVTG050].

Example 16-25Usage of ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR

```
{
    Ipp8u    pSrcDst[8*8] =
    {
        0x78, 0x77, 0x76, 0x74, 0x73, 0x73, 0x74, 0x75,
        0x78, 0x77, 0x76, 0x74, 0x73, 0x73, 0x75, 0x79,
        0x76, 0x76, 0x76, 0x75, 0x75, 0x75, 0x7c, 0x7e,
        0x75, 0x75, 0x75, 0x76, 0x76, 0x76, 0x7c, 0x7e,
        0x78, 0x79, 0x7a, 0x7c, 0x7d, 0x7e, 0x81, 0x82,
        0x78, 0x79, 0x7a, 0x7c, 0x7d, 0x7e, 0x81, 0x82,
        0x7a, 0x7b, 0x7c, 0x7e, 0x7e, 0x7f, 0x81, 0x82,
        0x7d, 0x7e, 0x7f, 0x81, 0x81, 0x82, 0x82, 0x82
    };
    Ipp8u   pAlpha[2]       = {25,25};
    Ipp8u   pBeta[2]        = {8,8};
    Ipp8u   pThreshold[16]  =
{255,255,255,255,255,255,1,1,204,204,204,204,204,204,204,204};
    Ipp8u   pBs[16]         = {0,0,0,0,0,0,0,2,0,0,1,2,0,0,0,0};

    ippStaticInitCpu(ippCpuSSE2);

    ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR(pSrcDst,
                                                    8,
                                                    pAlpha,
                                                    pBeta,
                                                    pThreshold,
                                                    pBs);
}
```

These functions are used in the H.264 decoder and encoder included into Intel IPP Samples. See introduction to this section.

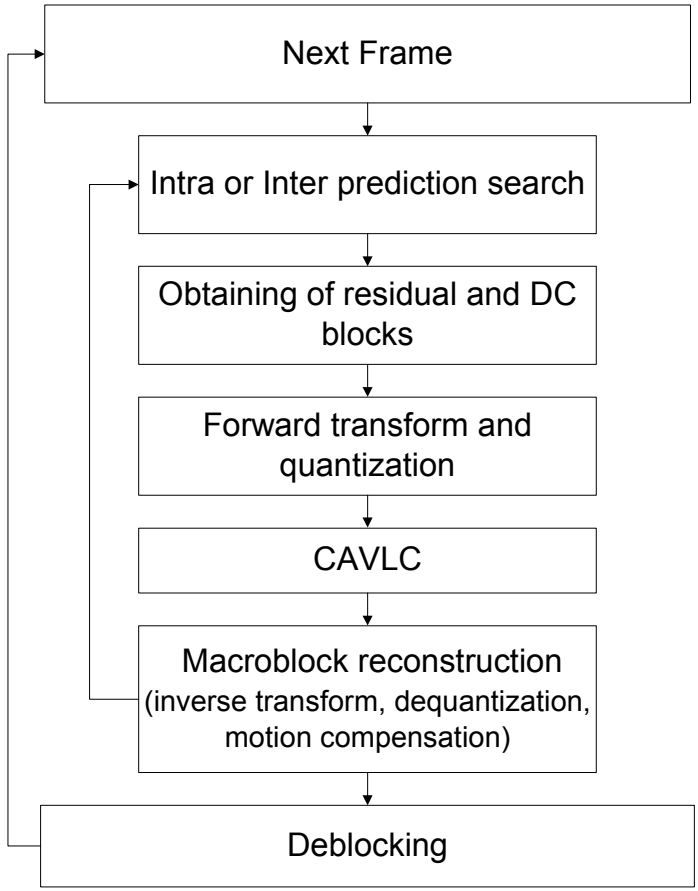
Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

H.264 Encoder Functions

This section describes the functions for H.264 Encoder in accordance with JVT-G050 ([JVTG050]) standard.

Figure 16-75 H.264 Encoder Structure



Edges detection function (`EdgesDetect16x16`) is taken from General Functions category. This function helps to choose a mode of intra macroblock coding: `INTRA 16X16` or `INTRA 4X4`. If edges are detected inside the macroblock, both modes should be estimated. Otherwise, `INTRA 16X16` mode should be selected.

Other general functions used by H.264 Encoder for predicted blocks estimation and for obtaining residual and DC blocks are SAD Functions, Sum of Differences Evaluation functions, and GetDiff4x4 function.

H.264 Encoder uses H.264 Decoder functions for calculation of inter and intra predicted blocks, Macroblock Reconstruction, and Deblocking Filtering. Forward transform and quantization, and CAVLC coding are performed by Encoder proper functions, which are described below.

Forward Transform and Quantization

TransformQuantChromaDC_H264

Performs forward transform and quantization for 2X2 DC chroma blocks.

Syntax

```
IppStatus ippiTransformQuantChromaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp16s*
    pTBlock, Ipp32s QPChroma, Ipp8s* NumLevels, Ipp8u Intra, Ipp8u
    NeedTransform);
```

Parameters

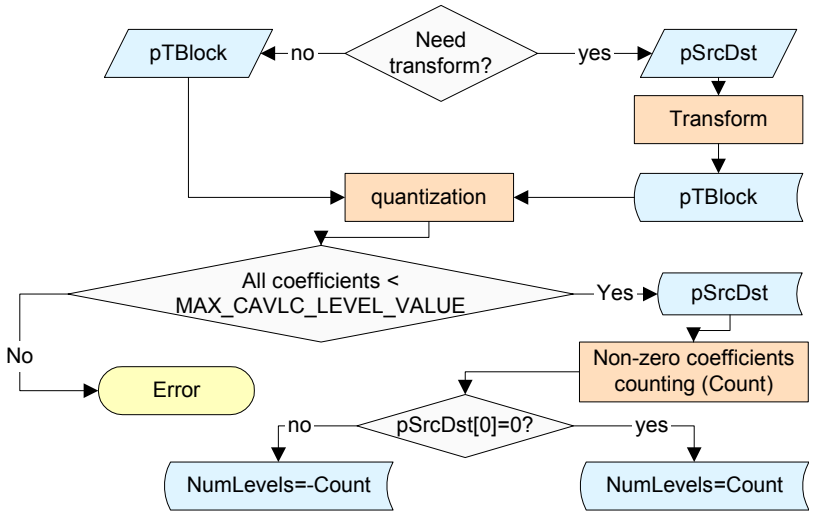
pSrcDst	Pointer to a 2x2 chroma DC block - source and destination array of size 4.
pTBlock	Pointer to a 2x2 transformed chroma DC block - source or destination array of size 4.
QPChroma	Quantization parameter for chroma, in range [0, 39].
NumLevels	Pointer to a value taht contains: • a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero), • a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero). This value is calculated by the function.
Intra	Flag equal to 1 in the case of intra macroblock, 0 otherwise.

NeedTransform	Flag; if equal to 1, the function is supposed to do DC coeffs transformation; if equal to 0, the DC coeffs passed to the function are already transformed.

Description

The function `ippiTransformQuantChromaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform, if necessary, and quantization for a 2x2 DC chroma block as shown in Figure 16-76:

Figure 16-76 Forward Transform and Quantization for 2x2 Block



If *NeedTransform* is equal to 0, transformed DC block coefficients (*pTBlock*) are used for quantization.

If *NeedTransform* is equal to 1, at first a 2x2 chroma DC block *pSrcDst* is transformed and is saved in *pTBlock*. Then quatization process is performed on the transformed chroma DC block.

If all coefficients after quantization are not greater than `MAX_CAVLC_LEVEL_VALUE`, they are saved in *pSrcDst*. Otherwise, the function returns error.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	Indicates an error if *QPChroma* > 39 or *QPChroma* < 0.
ippStsScaleRangeErr	Indicates an error condition if any coefficient after quantization is greater than MAX_CAVLC_LEVEL_VALUE.

TransformQuantLumaDC_H264

Performs forward transform and quantization for 4X4 DC luma blocks.

Syntax

```
IppStatus ippiTransformQuantLumaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp16s*
    pTBlock, Ipp32s QP, Ipp8s* NumLevels, Ipp8u NeedTransform, const Ipp16s*
    pScanMatrix, Ipp8u* LastCoeff);
```

Parameters

pSrcDst	Pointer to a 4x4 luma DC block - source and destination array of size 16.
pTBlock	Pointer to a 4x4 transformed luma DC block - source or destination array of size 16.
QP	Quantization parameter for luma, in range [0, 51].
NumLevels	Pointer to a value that contains: • a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero), • a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero). This value is calculated by the function.
NeedTransform	Flag; if equal to 1, the function is supposed to do DC coeffs transformation; if equal to 0, the DC coeffs passed to the function are already transformed.
pScanMatrix	Scan matrix for coefficients in block (array of size 16).

LastCoeff	Position of the last (in order of *pScanMatrix*) non-zero coefficient in block after quantization. This value is calculated by the function.

Description

The function ippiTransformQuantLumaDC_H264_16s_C1I is declared in the ippvc.h file. This function performs forward transform, if necessary, and quantization for a 4x4 DC luma block as shown in Figure 16-76:

If *NeedTransform* is equal to 0, transformed DC block coefficients (*pTBlock*) are used for quantization.

If *NeedTransform* is equal to 1, at first a 4x4 luma DC block *pSrcDst* is transformed and is saved in *pTBlock*. Then quatization process is performed on the transformed luma DC block.

If all coefficients after quantization are not greater than MAX_CAVLC_LEVEL_VALUE, they are saved in *pSrcDst*. Otherwise, the function returns error.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	Indicates an error if *QP* > 51 or *QP* < 0.
ippStsScaleRangeErr	Indicates an error condition if any coefficient after quantization is greater than MAX_CAVLC_LEVEL_VALUE.

TransformResidual4x4Fwd_H264

Performs forward transform for 4X4 residual blocks.

Syntax

```
IppStatus ippiTransformResidual4x4Fwd_H264_16s_C1(const Ipp16s* pSrc, Ipp16s*
    pDst);
IppStatus ippiTransformResidual4x4Fwd_H264_32s_C1(const Ipp32s* pSrc, Ipp32s*
    pDst);
```

Parameters

pSrc	Pointer to a 4x4 residual block - source array of size 16.
pDst	Pointer to a 4x4 residual block - destination array of size 16.

Description

The function `ippiTransformResidual4x4Fwd_H264_16s_C1` and `ippiTransformResidual4x4Fwd_H264_32s_C1` are declared in the `ippvc.h` file. This function performs forward transform for a 4x4 residual block. A 4x4 residual block (*pSrc*) is transformed and saved in *pDst*.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

QuantizeResidual4x4Fwd_H264

Performs forward quantization for 4X4 residual blocks.

Syntax

```
IppStatus ippiQuantizeResidual4x4Fwd_H264_16s_C1(const Ipp16s* pSrc, Ipp16s*
    pDst, Ipp32u* pNumNonZeros, Ipp32u* pLastNonZero, const Ipp16s* pQuantTable,
    const Ipp16s* pScanMatrix, Ipp32s QP, Ipp32s roundMode);
IppStatus ippiQuantizeResidual4x4Fwd_H264_16s32s_C1(const Ipp16s* pSrc, Ipp16s*
    pDst, Ipp32u* pNumNonZeros, Ipp32u* pLastNonZero, const Ipp32s* pQuantTable,
    const Ipp16s* pScanMatrix, Ipp32s QP, Ipp32s roundMode);
IppStatus ippiQuantizeResidual4x4Fwd_H264_32s_C1(const Ipp32s* pSrc, Ipp32s*
    pDst, Ipp32u* pNumNonZeros, Ipp32u* pLastNonZero, const Ipp32s* pQuantTable,
    const Ipp16s* pScanMatrix, Ipp32s QP, Ipp32s roundMode);
```

Parameters

pSrc	Pointer to a 4x4 residual block - source array of size 16.

pDst	Pointer to a 4x4 residual block - destination array of size 16.
pNumNonZeros	Pointer to a value that contains the number of non-zero elements in the block after quantization. This value is calculated by the function.
pLastNonZero	Position of the last (in order of *pScanMatrix*) non-zero coefficient in the block after quantization. This value is calculated by the function.
pQuantTable	Pointer to a user's specific quantization table.
pScanMatrix	Scan matrix for coefficients in block (array of size 16).
QP	Quantization parameter for luma or chroma, in range [0,51] or [0,39].
roundMode	Rounding mode. 0 means inter frame is processed, otherwise - intra frame is processed.

Description

The functions `ippiQuantizeResidual4x4Fwd_H264_16s_C1`, `ippiQuantizeResidual4x4Fwd_H264_16s32s_C1`, and `ippiQuantizeResidual4x4Fwd_H264_32s_C1` are declared in the `ippvc.h` file. These functions perform forward quantization for a 4x4 residual block. A 4x4 residual block (*pSrc*) is quantized. All values are rounded towards nearest (intra frame) or towards zero (inter frame). Then the number of non-zero elements and index of the last non-zero element are calculated and stored.

If *pQuantTable* is `NULL`, the default quantization table is used.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	Indicates an error if *QP* > 51 or *QP* < 0.

TransformQuantResidual_H264

Performs forward transform and quantization for 4X4 residual blocks.

Syntax

```
IppStatus ippiTransformQuantResidual_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP,
    Ipp8s* NumLevels, Ipp8u Intra, const Ipp16s* pScanMatrix, Ipp8u* LastCoeff);
```

Parameters

pSrcDst	Pointer to a 4x4 residual block - source and destination array of size 16.
QP	Quantization parameter for luma or for chroma, in range [0, 51] or [0, 39].
NumLevels	Pointer to a value that contains: • a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero), • a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero). This value is calculated by the function.
Intra	Flag equal to 1 in the case of intra slice, 0 otherwise.
pScanMatrix	Scan matrix for coefficients in block (array of size 16).
LastCoeff	Position of the last (in order of *pScanMatrix*) non-zero coefficient in block after quantization. This value is calculated by the function.

Description

The function `ippiTransformQuantResidual_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform and quantization for a 4x4 residual block.

A 4x4 residual block (*pSrcDst*) is transformed. Then quatization process is performed on the transformed residual block.

If all coefficients after quantization are not greater than `MAX_CAVLC_LEVEL_VALUE`, they are saved in *pSrcDst*. Otherwise, the function returns error.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	Indicates an error if *QP* > 51 or *QP* < 0.
ippStsScaleRangeErr	Indicates an error condition if any coefficient after quantization is greater than MAX_CAVLC_LEVEL_VALUE.

TransformLuma8x8Fwd_H264

Performs forward 8X8 transform for 8X8 luma blocks without normalization.

Syntax

```
IppStatus ippiTransformLuma8x8Fwd_H264_16s_C1I(Ipp16s* pSrcDst);
```

Parameters

pSrcDst	Pointer to a 8x8 luma block - source and destination array of size 64.

Description

The function ippiTransformLuma8x8Fwd_H264_16s_C1I is declared in the ippvc.h file. This function performs forward transform for a 8x8 luma block without final normalization (see normAdjust8x8 matrix described in [ITUH264]:

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if the specified pointer is NULL.

QuantLuma8x8_H264

Performs quantization for 8X8 luma block coefficients including 8X8 transform normalization.

Syntax

```
IppStatus ippiQuantLuma8x8_H264_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst, int
    Qp6, int Intra, const Ipp16s* pScanMatrix, const Ipp16s* pScaleLavels, int*
    pNumLevels, int* pLastCoeff);
```

Parameters

pSrc	Pointer to source luma block coefficients - array of size 64.
pDst	Pointer to the destination quantized block - array of size 64.
QP6	Quantization parameter divided by 6.
Intra	Flag that is equal to 1 if the slice is intra and 0 otherwise.
pScanMatrix	Pointer to a scan matrix for the coefficients in the block (array of size 64).
pScaleLevels	Pointer to a scale level matrix.
pNumLevels	Pointer to a value that contains: • a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero), • a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero).
pLastCoeff	Position of the last (in order of *pScanMatrix*) non-zero coefficient in block after quantization. This value is calculated by the function.

Description

The function `ippiQuantLuma8x8_H264_16s_C1` is declared in the `ippvc.h` file. This function performs quantization of the coefficients of a luma block after 8x8 transform including transform normalization (see normAdjust8x8 matrix described in [ITUH264]).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

ippStsQPErr	Indicates an error if *Qp6* > 8 or *Qp6* < 0.

GenScaleLevel8x8_H264

Generates ScaleLevel matrices for forward and inverse quantization including normalization for 8X8 forward and inverse transform.

Syntax

```
IppStatus ippiGenScaleLevel8x8_H264_8u16s_D2(const Ipp8u* pSrcInvScaleMatrix,
    int SrcStep, Ipp16s* pDstInvScaleMatrix, Ipp16s* pDstScaleMatrix, int
    QpRem);
```

Parameters

pSrcInvScaleMatrix	Pointer to the source inverse scaling matrix for 8x8 transform.
SrcStep	Step of *pSrcInvScaleMatrix*, in bytes.
pDstInvScaleMatrix	Pointer to the destination inverse scaling matrix - array of size 64.
pDstScaleMatrix	Pointer to the destination forward scaling matrix - array of size 64.
QpRem	Reminder from integer division of the quantization parameter by 6.

Description

The function ippiGenScaleLevel8x8_H264_8u16s_D2 is declared in the ippvc.h file. This function generates scaling matrices for forward and inverse quantization, taking into account normalization for forward and inverse 8x8 transform accordingly as defined.

NOTE. According to [ITUH264] standard an original inverse scaling matrix *pSrcIncScaleMatrix* cannot contain zeroes.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsQPErr	Indicates an error if *QpRem* > 5 or *QpRem* < 0.

CAVLC Functions

These functions calculate block characteristics for CAVLC encoding:

- number of `trailing_ones` transform coefficient levels
- code that describes signs of `trailing_ones`
- number of non-zero coefficients in block
- number of zero coefficients in block
- array of levels
- array of runs.

`EncodeChromaDcCoeffsCAVLC_H264` is used for a 2x2 DC chroma block. For other block types `EncodeCoeffsCAVLC_H264` is used.

EncodeCoeffsCAVLC_H264

Calculates characteristics of 4X4 block for CAVLC encoding.

Syntax

```
IppStatus ippiEncodeCoeffsCAVLC_H264_16s(const Ipp16s* pSrc, Ipp8u AC, const
    Ipp32u* pScanMatrix, Ipp8u Count, Ipp8u* Traling_One, Ipp8u*
    Traling_One_Signs, Ipp8u* NumOutCoeffs, Ipp8u* TotalZeros, Ipp16s* pLevels,
    Ipp8u* pRuns);
```

Parameters

pSrc	Pointer to 4x4 block - array of size 16.
AC	Flag that is equal to zero in the cases when zero coefficient should be encoded, and is equal to one otherwise.
pScanMatrix	Scan matrix for the coefficients in block (array of size 16).
Count	Position of the last non-zero block coefficient in the order of scan matrix. It should be in range [*AC*; 15].
Traling_One	The number of “trailing ones” transform coefficient levels in a range [0;3]. This value is calculated by the function.

Traling_One_Signs	Code that describes signs of “trailing ones”. (*Trailing_One –1 - i*)-bit in this code corresponds to a sign of *i*-“trailing one” in the current block. In this code 1 indicates negative value, 0 – positive value. This value is calculated by the function.
NumOutCoeffs	The number of non-zero coefficients in block (including “trailing ones”). This value is calculated by the function.
TotalZeros	The number of zero coefficients in block (except “trailing zeros”). This value is calculated by the function.
pLevels	Pointer to an array of size 16 that contains non-zero quantized coefficients of the current block (except “trailing ones”) in reverse scan matrix order. Elements of this array are calculated by the function.
pRuns	Pointer to an array of size 16 that contains runs before non-zero quantized coefficients (including “trailing ones”) of the current block in reverse scan matrix order (except run before the first non-zero coefficient in block, which can be calculated using *TotalZeros*). Elements of this array are calculated by the function.

Description

The function `ippiEncodeCoeffsCAVLC_H264_16s` is declared in the `ippvc.h` file. This function calculates some characteristics (*Trailing_One*, *Trailing_One_Signs*, *NumOutCoeffs*, *TotalZeros*, *pLevels*, *pRuns*) of a 4x4 block for CAVLC encoding. See Figure 16-77 for an example of the function operation:

Figure 16-77

Intra 16x16 AC Block

0	0	-3	6
0	0	5	0
-1	0	-1	1
0	1	0	0

Scan Matrix

0	1	4	8
5	2	3	6
9	12	13	10
7	11	14	15

Count=13.

Figure 16-78 Arrangement of Block Coefficients After Scanning

coef	0	0	**5**	**-3**	0	0	0	**6**	**-1**	**1**	**1**	0	**-1**	0	0
pos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Trailing_One = 3 (-1, 1, 1)
Trailing_One_Signs = 00000100
NumOutCoeffs = 7
TotalZeros = 6

Figure 16-79 Array of Runs Before Non-Zero Coefficients (Except First) in Reverse Order

pRuns	1	0	0	0	3	0
coefficient	**-1**	**1**	**1**	**-1**	**6**	**-3**

Figure 16-80 Array of Non-Zero Coefficients (Except "Trailing Ones") in Reverse Order

pLevel	**-1**	**6**	**-3**	**5**

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	Indicates an error when *Count* is out of range [*AC*; 15].

EncodeChromaDcCoeffsCAVLC_H264

Calculates characteristics of 2X2 chroma DC block for CAVLC encoding.

Syntax

```
IppStatus ippiEncodeChromaDcCoeffsCAVLC_H264_16s(const Ipp16s* pSrc, Ipp8u*
    Traling_One, Ipp8u* Traling_One_Signs, Ipp8u* NumOutCoeffs, Ipp8u*
    TotalZeros, Ipp16s* pLevels, Ipp8u* pRuns);
```

Parameters

pSrc	Pointer to 2x2 block - array of size 4.
Traling_One	The number of “trailing ones” transform coefficient levels in a range [0;3]. This value is calculated by the function.
Traling_One_Signs	Code that describes signs of “trailing ones”. (*Trailing_One –1 - i*)-bit in this code corresponds to a sign of *i*-“trailing one” in the current block. In this code 1 indicates negative value, 0 – positive value. This value is calculated by the function.
NumOutCoeffs	The number of non-zero coefficients in block (including “trailing ones”). This value is calculated by the function.
TotalZeros	The number of zero coefficients in block (except “trailing zeros”). This value is calculated by the function.
pLevels	Pointer to an array of size 4 that contains non-zero quantized coefficients of the current block (except “trailing ones”) in reverse scan matrix order. Elements of this array are calculated by the function.
pRuns	Pointer to an array of size 4 that contains runs before non-zero quantized coefficients (including “trailing ones”) of the current block in reverse scan matrix order (except run before the first

non-zero coefficient in block, which can be calculated using *TotalZeros*). Elements of this array are calculated by the function.

Description

The function `ippiEncodeChromaDcCoeffsCAVLC_H264_16s` is declared in the `ippvc.h` file. This function calculates some characteristics (*Trailing_One*, *Trailing_One_Signs*, *NumOutCoeffs*, *TotalZeros*, *pLevels*, *pRuns*) of a 2x2 chroma DC block for CAVLC encoding.

This function is used in the H.264 encoder included into Intel IPP Samples. See introduction to this section.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

Inverse Quantization and Transform

QuantLuma8x8Inv_H264

Performs inverse quantization for 8X8 luma block coefficients including normalization of the subsequent inverse 8X8 transform.

Syntax

```
IppStatus ippiQuantLuma8x8Inv_H264_16s_C1I(Ipp16s* pSrcDst, int Qp6, const
    Ipp16s* pInvLevelScale);
```

Parameters

pSrcDst	Pointer to luma block coefficients - source and destination array of size 64.
Qp6	Quantization parameter divided by 6.
pInvLevelScale	Pointer to an inverse scale level matrix.

Description

The function `ippiQuantLuma8x8Inv_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs inverse quantization and normalization for 8x8 inverse transform of luma block coefficients according to 8.5.13 of [ITUH264].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsQPErr`	Indicates an error condition if *Qp6* is less than 0 or greater than 8.

TransformLuma8x8InvAddPred_H264

Performs inverse 8X8 transform of 8X8 luma block coefficients with subsequent intra prediction or motion compensation.

Syntax

```
IppStatus ippiTransformLuma8x8InvAddPred_H264_16s8u_C1R(const Ipp8u* pPred, int
    PredStep, Ipp16s* pSrc, Ipp8u* pDst, int DstStep);
```

Parameters

pPred	Pointer to a reference 8x8 block, which is used either for intra prediction or motion compensation.
PredStep	Distance between starts of the consecutive lines in the reference frame.
pSrc	Pointer to 8x8 luma block coefficients – source and buffer array of size 64.
pDst	Pointer to the destination 8x8 block.
DstStep	Distance between starts of the consecutive lines in the destination frame.

Description

The function `ippiTransformLuma8x8InvAddPred_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function performs inverse transform of a 8x8 luma block described in 8.5.13 of [ITUH264], with subsequent intra prediction or motion compensation. Input coefficients are assumed to be inverse quantized and normalized. The *pSrc* array is used as a buffer during computations.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

AVS

This section describes video coding functions for encoding, decoding, processing and presenting of digital audio-video in accordance with AVS ([AVS]) standard. These functions apply to high and standard resolution of digital broadcast, laser digital media storage, broadband network streaming and multimedia communication etc.

The following enumeration is used to indicate prediction modes of the Intra_8x8 prediction process for luma samples (9.8.2 of [AVS]):

```
typedef enum {   IPP_8x8_VERT     = 0,
                 IPP_8x8_HOR      = 1,
                 IPP_8x8_DC       = 2,
                 IPP_8x8_DIAG_DL  = 3,
                 IPP_8x8_DIAG_DR  = 4,
              } IppIntra8x8PredMode_AVS;
```

Table 16-27 shows the correspondence between prediction modes and constants of `enum IppIntra8x8PredMode_AVS`.

Table 16-27

Name of Constant	Prediction Mode	Chapter in AVS
IPP_8x8_VERT	Intra_8x8_Vertical	9.8
IPP_8x8_HOR	Intra_8x8_Horizontal	9.8
IPP_8x8_DC	Intra_8x8_DC	9.8
IPP_8x8_DIAG_DL	Intra_8x8_Diagonal_Down_Left	9.8

Table 16-27

Name of Constant	Prediction Mode	Chapter in AVS
IPP_8x8_DIAG_DR	Intra_8x8_Diagonal_Down_Right	9.8

The following enumeration is used to indicate prediction modes of the intra prediction process for chroma samples (9.8.3 of AVS):

```
typedef enum {   IPP_CHROMA_DC     = 0,
                 IPP_CHROMA_HOR    = 1,
                 IPP_CHROMA_VERT   = 2,
                 IPP_CHROMA_PLANE  = 3,
               } IppIntraChromaPredMode_AVS;
```

Table 16-28 shows the correspondence between prediction modes and constants of `enum IppIntraChromaPredMode_AVS`.

Table 16-28

Name of Constant	Prediction Mode	Chapter in AVS
IPP_CHROMA_DC	Intra_Chroma_DC	9.8
IPP_CHROMA_HOR	Intra_Chroma_Horizontal	9.8
IPP_CHROMA_VERT	Intra_Chroma_Vertical	9.8
IPP_CHROMA_PLANE	Intra_Chroma_Plane	9.8

Table 16-29 AVS Decoder Functions

Function Short Name	Description
Entropy Decoding	
DecodeLumaBlockIntra_AVS	Decodes luminance coefficients of an intra block.
DecodeLumaBlockInter_AVS	Decodes luminance coefficients of an inter block.
DecodeChromaBlock_AVS	Decodes chrominance coefficients of a block.
Inter Prediction	
InterpolateLumaBlock_AVS	Performs interpolation for motion estimation of the luma component.
WeightPrediction_AVS	Applies weighting process to a predicted block.
Macroblock Reconstruction	
ReconstructLumaIntra_AVS	Reconstructs intra luma macroblock.

Table 16-29 AVS Decoder Functions (continued)

Function Short Name	Description
ReconstructLumaInter_AVS	Reconstructs inter luma macroblock.
ReconstructChromaIntra_AVS	Reconstructs intra chroma macroblock.
ReconstructChromaInter_AVS	Reconstructs inter chroma macroblock.
Deblocking Filtering	
FilterDeblockingLuma_VerEdge_AVS	Performs deblocking filtering on the vertical edges of the luma 16x16 macroblock.
FilterDeblockingLuma_HorEdge_AVS	Performs deblocking filtering on the horizontal edges of the luma 16x16 macroblock.
FilterDeblockingChroma_VerEdge_AVS	Performs deblocking filtering on the vertical edges of the chroma 8x8 macroblock.
FilterDeblockingChroma_HorEdge_AVS	Performs deblocking filtering on the horizontal edges of the chroma 8x8 macroblock.

Table 16-30 AVS Encoder Functions

Function Short Name	Description
Forward Transform and Quantization	
TransformQuant8x8Fwd_AVS	Performs forward transform and quantization for 8X8 residual blocks.
Macroblock Disassembling	
DisassembleLumaIntra_AVS	Disassembles intra luma macroblocks.
DisassembleChroma420Intra_AVS	Disassembles intra chroma macroblocks.

AVS Decoder Functions

This subsection describes functions for decoding of digital audio-video in accordance with AVS ([AVS]) standard.

Figure 16-81 AVS Intra Macroblock Decoding Process

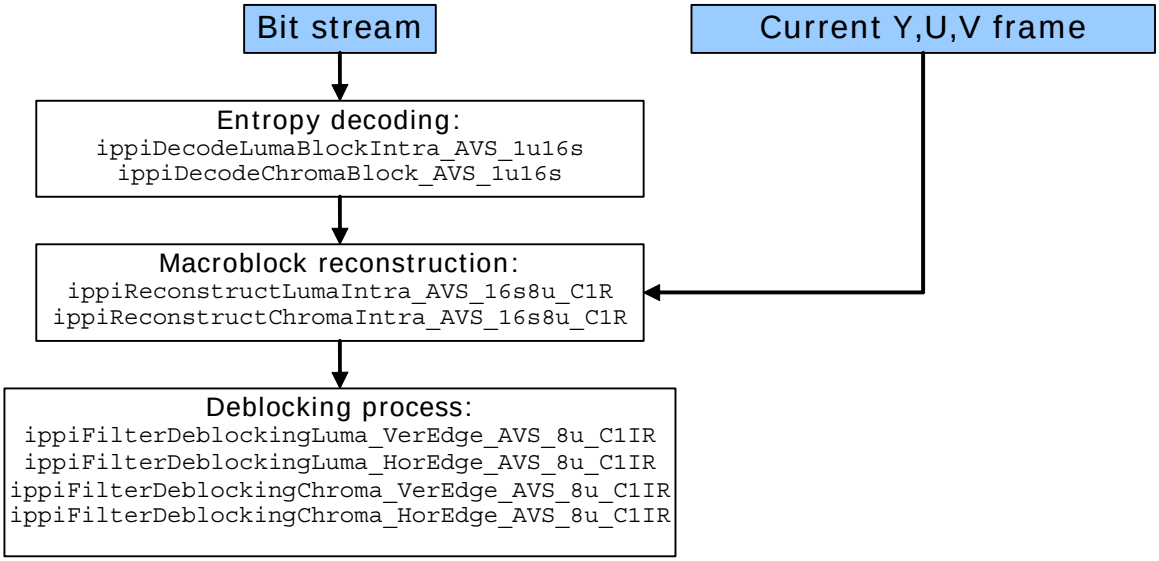
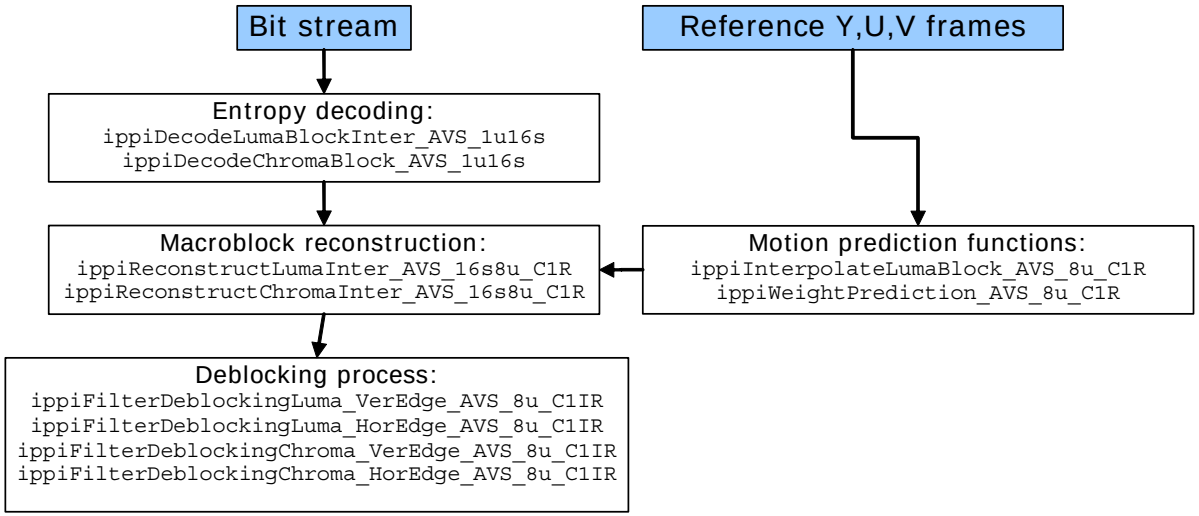


Figure 16-82 AVS Inter Macroblock Decoding Process



Entropy Decoding

DecodeLumaBlockIntra_AVS

Decodes luminance coefficients of an intra block.

Syntax

```
IppStatus ippiDecodeLumaBlockIntra_AVS_1u16s(Ipp32u** ppBitStream, Ipp32s*
    pBitOffset, Ipp32s* pNumCoeff, Ipp16s* pDstCoeffs, Ipp32u scanType);
```

Parameters

ppBitStream	Double pointer to the current position in the bitstream. The pointer is updated by the function.

pBitOffset	Pointer to offset between the bit that *ppBitStream* points to and the start of the code. The pointer is updated by the function.
pNumCoeff	Pointer to the output number of non-zero coefficients.
pDstCoeffs	Pointer to an 8x8 block of coefficients. These coefficients are calculated by the function.
scanType	Type of inverse scan method. 0 means the type described in Figure 9-5(a) of [AVS]. 1 means the type described in Figure 9-5(b) of [AVS].

Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeLumaBlockIntra_AVS_1u16s` decodes luminance coefficients of intra blocks in accordance with 9.5.1 of [AVS].

After decoding all coefficients of a block, the function performs inverse scanning, using the given type of scanning. The function returns the last coefficient index in classical scan order through the *pNumCoeff* variable as shown in Figure 9-5(a) of [AVS].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

DecodeLumaBlockInter_AVS

Decodes luminance coefficients of an inter block.

Syntax

```
IppStatus ippiDecodeLumaBlockInter_AVS_1u16s(Ipp32u** ppBitStream, Ipp32s*
    pBitOffset, Ipp32s* pNumCoeff, Ipp16s* pDstCoeffs, Ipp32u scanType);
```

Parameters

ppBitStream	Double pointer to the current position in the bitstream. The pointer is updated by the function.
pBitOffset	Pointer to offset between the bit that *ppBitStream* points to and the start of the code. The pointer is updated by the function.
pNumCoeff	Pointer to the output number of non-zero coefficients.

pDstCoeffs	Pointer to an 8x8 block of coefficients. These coefficients are calculated by the function.
scanType	Type of inverse scan method. 0 means the type described in Figure 9-5(a) of [AVS]. 1 means the type described in Figure 9-5(b) of [AVS].

Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeLumaBlockInter_AVS_1u16s` decodes luminance coefficients of inter blocks in accordance with 9.5.1 of [AVS].

After decoding all coefficients of a block, the function performs inverse scanning, using the given type of scanning. The function returns the last coefficient index in classical scan order through the *pNumCoeff* variable as shown in Figure 9-5(a) of [AVS].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

DecodeChromaBlock_AVS

Decodes chrominance coefficients of a block.

Syntax

```
IppStatus ippiDecodeChromaBlock_AVS_1u16s(Ipp32u** ppBitStream, Ipp32s*
    pBitOffset, Ipp32s* pNumCoeff, Ipp16s* pDstCoeffs, Ipp32u scanType);
```

Parameters

ppBitStream	Double pointer to the current position in the bitstream. The pointer is updated by the function.
pBitOffset	Pointer to offset between the bit that *ppBitStream* points to and the start of the code. The pointer is updated by the function.
pNumCoeff	Pointer to the output number of non-zero coefficients.
pDstCoeffs	Pointer to an 8x8 block of coefficients. These coefficients are calculated by the function.

scanType	Type of inverse scan method. 0 means the type described in Figure 9-5(a) of [AVS]. 1 means the type described in Figure 9-5(b) of [AVS].

Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeChromaBlock_AVS_1u16s` decodes chrominance coefficients of blocks in accordance with 9.5.1 of [AVS].

After decoding all coefficients of a block, the function performs inverse scanning, using the given type of scanning. The function returns the last coefficient index in classical scan order through the *pNumCoeff* variable as shown in Figure 9-5(a) of [AVS].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one input pointer is `NULL`.

Inter Prediction

InterpolateLumaBlock_AVS

Performs interpolation for motion estimation of the luma component.

Syntax

```
IppStatus ippiInterpolateLumaBlock_AVS_8u_P1R(const IppVCInterpolateBlock_8u*
    interpolateInfo);
```

Parameters

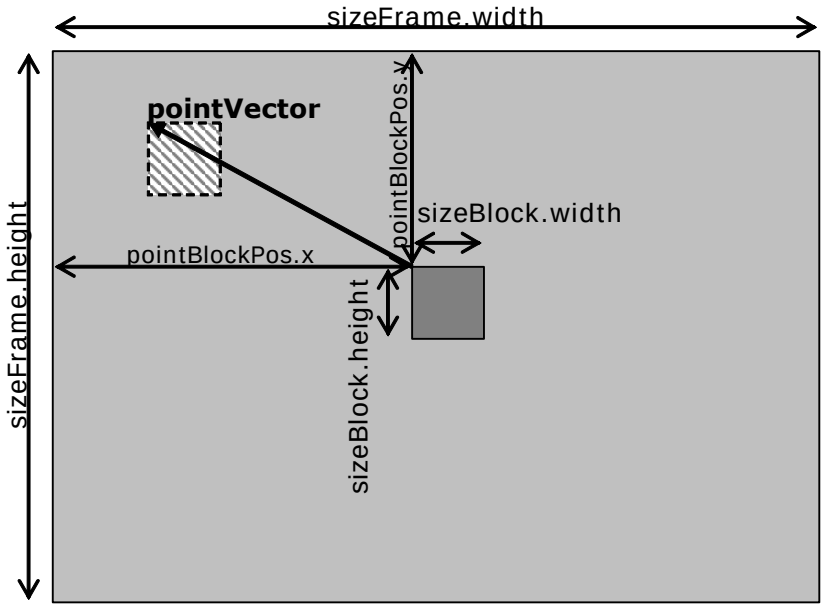
interpolateInfo	Pointer to an instance of the structure holding interpolation parameters. See `IppVCInterpolateBlock_8u` structure.

Description

The function `ippiInterpolateLumaBlock_AVS_8u_P1R` is declared in the `ippvc.h` file. This function performs interpolation (convolution with 6x6 kernel) for motion estimation of the luminance component in accordance with 9.9 of [AVS]. The function uses whole motion

vectors, calculating fractional and integer part of vectors. The function also handles overlapping cases, when the source block pointed by the motion vector lies out of the source frame. Non-existing samples are cloned from the nearest existing samples.

Figure 16-83 Interpolation of Luminance Component Block for Motion Estimation



Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if *`roi`*`.width` or *`roi`*`.height` take values other than {16, 8}.

WeightPrediction_AVS

Applies weighting process to a predicted block.

Syntax

```
IppStatus ippiWeightPrediction_AVS_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep,
    Ipp8u* pDst, Ipp32s dstStep, Ipp32u scale, Ipp32s shift, IppiSize
    sizeBlock);
```

Parameters

pSrc	Pointer to the source data.
srcStep	Distance in bytes between starts of the consecutive lines in the source data.
pDst	Pointer to the destination buffer.
dstStep	Distance in bytes between starts of the consecutive lines in the destination buffer.
scale	Scale value.
shift	Shift.
sizeBlock	Size of the pointed data block. It could be 16 or 8 in each dimension.

Description

The function `ippiWeightPrediction_AVS_8u_C1R` is declared in the `ippvc.h` file. This function applies the weighting process to predicted blocks according to 9.9.3 of [AVS] when `slice_weighting_flag` is equal to 1.

The function uses the following formula:

$$pDst[x, y] = \mathrm{Clip1}((pSrc([x, y] \cdot scale) + 16) \gg 5 + shift)$$

where $x \in [0, roi.\mathrm{width} - 1]$, $y \in [0, roi.\mathrm{height} - 1]$,

$$\mathrm{Clip1}(z) = \begin{cases} 0 & ; \ z < 0 \\ 255 & ; \ z > 255 \\ z & ; \ otherwise \end{cases}$$

Values of $scale$ and $shift$ are parsed from the bitstream.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

Macroblock Reconstruction

ReconstructLumaIntra_AVS

Reconstructs intra luma macroblock.

Syntax

```
IppStatus ippiReconstructLumaIntra_AVS_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
    pSrcDstYPlane, Ipp32s srcDstYStep, const IppIntra8x8PredMode_AVS*
    pMBIntraTypes, const Ipp32s* pSrcNumCoeffs, Ipp32u cbp8x8, Ipp32u QP, Ipp32u
    edgeType);
```

Parameters

ppSrcCoeff	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of entropy decoding (8x8 luminance intra blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in *Y*-plane.
srcDstYStep	*Y*-Plane step.
pMBIntraTypes	Array of Intra_8x8 luma prediction modes for each subblock. *pMBIntraTypes*[*i*] is defined in the same order as the coded subblocks go.
pSrcNumCoeffs	Array of indices of the last coefficient in each subblock. *pSrcNumCoeffs*[*i*] is defined in the same order as the coded subblocks go.
cbp8x8	Coded block pattern. If cbp8x8 & (1<<(1+i)) is not equal to 0 ($0 \le i < 4$), *i*-th 8x8 AC luma block is not zero-filled and it exists in *ppSrcCoeff*.
QP	Quantization parameter (*CurrentQP* in [AVS]). It must be within the range [0;63].

edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero. If the upper-right macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_RIGHT_EDGE` must be non-zero.

Description

The function `ippiReconstructLumaIntra_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs intra luma macroblocks. The process for each 8x8 block in the same order as is shown in Figure 6-7 of [AVS] is described below:

1. Performs scaling, integer inverse transformation, and shift for an 8x8 block in accordance with 9.5 of [AVS].
2. Performs intra prediction for an 8x8 luma component in accordance with 9.4.2 of [AVS].
3. Performs adding of 8x8 prediction block and 8x8 residual block in accordance with 9.10 of [AVS].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*QP* is less than 0 or greater than 63.

ReconstructLumaInter_AVS

Reconstructs inter luma macroblock.

Syntax

```
IppStatus ippiReconstructLumaInter_AVS_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
    pSrcDstYPlane, Ipp32s srcDstYStep, const Ipp32s* pSrcNumCoeffs, Ipp32u
    cbp8x8, Ipp32u QP);
```

Parameters

ppSrcCoeff	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of entropy decoding (8x8 luminance inter blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstYPlane	Pointer to the current macroblock that is reconstructed in *Y*-plane. This macroblock must contain inter prediction samples.
srcDstYStep	*Y*-Plane step.
pSrcNumCoeffs	Array of indices of the last coefficient in each subblock. *pSrcNumCoeffs*[*i*] is defined in the same order as the coded subblocks go.
cbp8x8	Coded block pattern. If `cbp8x8 & (1<<(1+i))` is not equal to 0 ($0 \le i < 4$), *i*-th 8x8 AC luma block is not zero-filled and it exists in *ppSrcCoeff*.
QP	Quantization parameter (*CurrentQP* in [AVS]). It must be within the range [0;63].

Description

The function `ippiReconstructLumaInter_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs inter luma macroblocks:

1. Performs scaling, integer inverse transformation, and shift for each 8x8 block in the same order as is shown in Figure 6-7 of [AVS] in accordance with 9.5 of [AVS].
2. Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 9.10 of [AVS].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*QP* is less than 0 or greater than 63.

ReconstructChromaIntra_AVS

Reconstructs intra chroma macroblock.

Syntax

```
IppStatus ippiReconstructChromaIntra_AVS_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
    pSrcDstUPlane,  Ipp8u* pSrcDstVPlane, Ipp32s srcDstUVStep, const
    IppIntraChromaPredMode_AVS predMode, const Ipp32s* pSrcNumCoeffs, Ipp32u
    cbp8x8, Ipp32u chromaQP, Ipp32u edgeType);
```

Parameters

ppSrcCoeff	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of entropy decoding (8x8 chrominance intra blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to the current macroblock that is reconstructed in *U*-plane.
pSrcDstVPlane	Pointer to the current macroblock that is reconstructed in *V*-plane.
srcDstUVStep	Chrominance planes step.
predMode	Chrominance prediction mode for both subblock.
pSrcNumCoeffs	Array of indices of the last coefficient in each subblock. *pSrcNumCoeffs*[*i*] is defined in the same order as the coded subblocks go.
cbp8x8	Coded block pattern. If `cbp8x8 & (1<<(1+i))` is not equal to 0 ($0 \le i < 2$), *i*-th 8x8 AC chroma block is not zero-filled and it exists in *ppSrcCoeff*.
chromaQP	Quantization parameter (*QP* from Table 9-6 in [AVS]). It must be within the range [0;51].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_EDGE` must be non-zero. If the left macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero. If the upper-left macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero. If the upper-right macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_TOP_RIGHT_EDGE` must be non-zero.

Description

The function `ippiReconstructChromaIntra_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs intra chroma macroblocks. The process for each 8x8 block in the same order as is shown in Figure 6-7 of [AVS] is described below:

1. Performs scaling, integer inverse transformation, and shift for an 8x8 block in accordance with 9.5 of [AVS].
2. Performs intra prediction for an 8x8 chroma component in accordance with 9.4.3 of [AVS].
3. Performs adding of 8x8 prediction block and 8x8 residual block in accordance with 9.10 of [AVS].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*chromaQP* is less than 0 or greater than 51.

ReconstructChromaInter_AVS

Reconstructs inter chroma macroblock.

Syntax

```
IppStatus ippiReconstructChromaInter_AVS_16s8u_C1R(Ipp16s** ppSrcCoeff, Ipp8u*
    pSrcDstUPlane, Ipp8u* pSrcDstVPlane, Ipp32s srcDstUVStep, const Ipp32s*
    pSrcNumCoeffs, Ipp32u cbp8x8, Ipp32u chromaQP);
```

Parameters

ppSrcCoeff	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of entropy decoding (8x8 chrominance inter blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to the blocks for the next macroblock.
pSrcDstUPlane	Pointer to the current macroblock that is reconstructed in *U*-plane.
pSrcDstVPlane	Pointer to the current macroblock that is reconstructed in *V*-plane.
srcDstUVStep	Chrominance planes step.

pSrcNumCoeffs	Array of indices of the last coefficient in each subblock. *pSrcNumCoeffs*[*i*] is defined in the same order as the coded subblocks go.
cbp8x8	Coded block pattern. If `cbp8x8 & (1<<(1+i))` is not equal to 0 ($0 \le i < 4$), *i*-th 8x8 AC chroma block is not zero-filled and it exists in *ppSrcCoeff*.
chromaQP	Quantization parameter (*QP* from Table 9-6 in [AVS]). It must be within the range [0;51].

Description

The function `ippiReconstructChromaInter_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs inter chroma macroblocks:

1. Performs scaling, integer inverse transformation, and shift for each 8x8 block in the same order as is shown in Figures 6-7, 6-8 of [AVS] in accordance with 9.5 of [AVS]. Order of the processed blocks depends on the passed *cbp8x8* value.
2. Performs adding of an 8x8 or 8x16 inter prediction block and an 8x8 or 8x16 residual block in accordance with 9.10 of [AVS]. Size of the processed data depends on the passed *cbp8x8* value.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*chromaQP* is less than 0 or greater than 51.

Deblocking Filtering

FilterDeblockingLuma_VerEdge_AVS

Performs deblocking filtering on the vertical edges of the 16x16 luma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingLuma_VerEdge_AVS_8u_C1IR(IppiFilterDeblock_8u*
    pDeblockInfo);
```

Parameters

pDeblockInfo	Pointer to deblocking parameters

Description

The function `ippiFilterDeblockingLuma_VerEdge_AVS_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the vertical edges of the 16x16 luma macroblock in accordance with 9.11 of [AVS].

`IppiFilterDeblock_8u` structure contains the following fields:

pSrcDstPlane	Pointer to the initial and resultant coefficients.
srcDstStep	Distance between starts of the consecutive lines in the array.
pAlpha	Array of size 2 of Alpha Thresholds (values for external and internal vertical edge).
pBeta	Array of size 2 of Beta Thresholds (values for external and internal vertical edge).
pThresholds	Array of size 2 of Thresholds (filter reduction parameter *C*) (values for external and internal vertical edge).
pBs	Array of size 4 of BS parameters (values for the left edge of each 8x8 block).

The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0], *pThresholds*[0] are values for the external vertical edge, and *pAlpha*[1], *pBeta*[1, *pThresholds*[1]] are values for the internal vertical edge. See Figure 16-84 for the arrangement of *pBs* array elements.

Values of the arrays are calculated as follows:

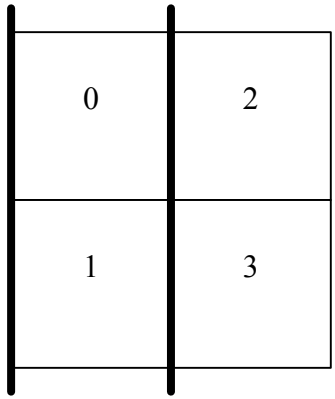
— *pBs* values are calculated as per 9.11.1 of [AVS] and may take the following values: 0 - if no edge is filtered; 1 - if filtering is weak; 2 - if filtering is strong.

— *pAlpha* values are derived from Table 9-9 of [AVS].

— *pBeta* values are derived from Table 9-9 of [AVS].

— *pThresholds*[i] values are derived from Table 9-10 of [AVS].

Figure 16-84 Arrangement of *pBs* Array Elements into a macroblock



Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

FilterDeblockingLuma_HorEdge_AVS

Performs deblocking filtering on the horizontal edges of the 16X16 luma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingLuma_HorEdge_AVS_8u_C1IR(IppiFilterDeblock_8u*
    pDeblockInfo);
```

Parameters

`pDeblockInfo`	Pointer to the deblocking parameters

Description

The function `ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the horizontal edges of the 16x16 luma macroblock in accordance with 9.11 of [AVS].

`IppiFilterDeblock_8u` structure contains the following fields:

pSrcDstPlane	Pointer to the initial and resultant coefficients.
srcDstStep	Distance between starts of the consecutive lines in the array.
pAlpha	Array of size 2 of Alpha Thresholds (values for external and internal vertical edge).
pBeta	Array of size 2 of Beta Thresholds (values for external and internal vertical edge).
pThresholds	Array of size 2 of Thresholds (filter reduction parameter *C*) (values for external and internal vertical edge).
pBs	Array of size 4 of BS parameters (values for the left edge of each 8x8 block).

The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0], *pThresholds*[0] are values for the external horizontal edge, and *pAlpha*[1], *pBeta*[1], *pThresholds*[1] are values for the internal horizontal edge. See Figure 16-85 for the arrangement of *pBs* array elements.

Values of the arrays are calculated as follows:

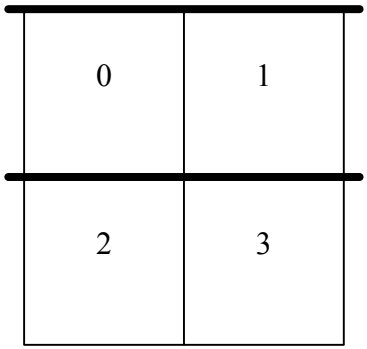
— *pBs* values are calculated as per 9.11.1 of [AVS] and may take the following values: 0 - if no edge is filtered; 1 - if filtering is weak; 2 - if filtering is strong.

— *pAlpha* values are derived from Table 9-9 of [AVS].

— *pBeta* values are derived from Table 9-9 of [AVS].

— *pThresholds*[i] values are derived from Table 9-10 of [AVS].

Figure 16-85 Arrangement of *pBs* Array Elements into a Macroblock



Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.

FilterDeblockingChroma_VerEdge_AVS

Performs deblocking filtering on the vertical edges of 8X8 chroma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingChroma_VerEdge_AVS_8u_C1IR(IppiFilterDeblock_8u*
    pDeblockInfo);
```

Parameters

pDeblockInfo	Pointer to the deblocking parameters

Description

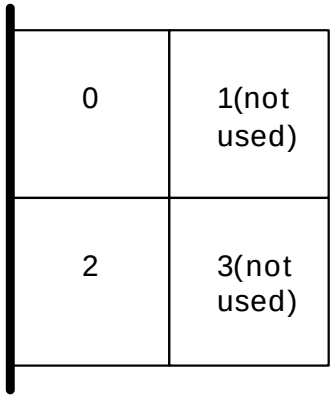
The function `ippiFilterDeblockingChroma_VerEdge_AVS_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the vertical edge of the 8x8 chroma macroblock in accordance with 9.11 of [AVS].

`IppiFilterDeblock_8u` structure contains the following fields:

pSrcDstPlane	Pointer to the initial and resultant coefficients.
srcDstStep	Distance between starts of the consecutive lines in the array.
pAlpha	Array of size 1 of Alpha Thresholds.
pBeta	Array of size 1 of Beta Thresholds.
pThresholds	Array of size 1 of Thresholds (filter reduction parameter *C*).
pBs	Array of size 4 of BS parameters number (values for the left edge of an 8x4 block), but only two of the four elements are used: element 0 and element 2.

The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments. *pAlpha*, *pBeta*, and *pBs* are the same arrays as in `FilterDeblockingLuma_VerEdge_AVS` function. See Figure 16-86 for the arrangement of *pBs* array elements.

Figure 16-86 Arrangement of *pBs* Array Elements into an 8x8 Chroma Block



Values of the arrays are calculated as follows:

— *pBs* values are calculated as per 9.11.1 of [AVS] and may take the following values: 0 - if no edge is filtered; 1 - if filtering is weak; 2 - if filtering is strong.

— *pAlpha* values are derived from Table 9-9 of [AVS].

— *pBeta* values are derived from Table 9-9 of [AVS].

— *pThresholds*[i] values are derived from Table 9-10 of [AVS].

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

FilterDeblockingChroma_HorEdge_AVS

Performs deblocking filtering on the horizontal edges of 8X8 chroma macroblock.

Syntax

```
IppStatus ippiFilterDeblockingChroma_HorEdge_AVS_8u_C1IR(IppiFilterDeblock_8u*
    pDeblockInfo);
```

Parameters

pDeblockInfo	Pointer to the deblocking parameters

Description

The function ippiFilterDeblockingChroma_HorEdge_AVS_8u_C1IR is declared in the ippvc.h file. This function performs Deblocking Filtering on the horizontal edge of the 8x8 chroma macroblock in accordance with 9.11 of [AVS].

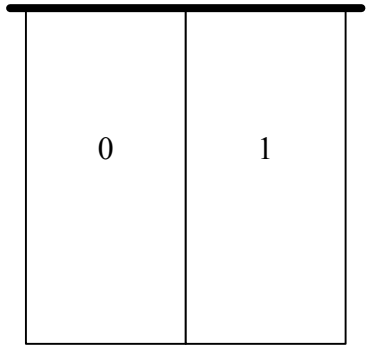
IppiFilterDeblock_8u structure contains the following fields:

pSrcDstPlane	Pointer to the initial and resultant coefficients.
srcDstStep	Distance between starts of the consecutive lines in the array.
pAlpha	Array of size 1 of Alpha Thresholds.
pBeta	Array of size 1 of Beta Thresholds.
pThresholds	Array of size 1 of Thresholds (filter reduction parameter *C*).

pBs	Array of size 2 of BS parameters (values for the upper edge of each 4x8 block).

The function uses arrays *pAlpha*, *pBeta*, *pBs*, *pThresholds* as input arguments. *pAlpha*, *pBeta*, and *pBs* are the same arrays as in FilterDeblockingLuma_HorEdge_AVS function. See Figure 16-87 for the arrangement of *pBs* array elements.

Figure 16-87 Arrangement of *pBs* Array Elements into an 8x8 Chroma Block



Values of the arrays are calculated as follows:

— *pBs* values are calculated as per 9.11.1 of [AVS] and may take the following values: 0 - if no edge is filtered; 1 - if filtering is weak; 2 - if filtering is strong.

— *pAlpha* values are derived from Table 9-9 of [AVS].

— *pBeta* values are derived from Table 9-9 of [AVS].

— *pThresholds*[i] values are derived from Table 9-10 of [AVS].

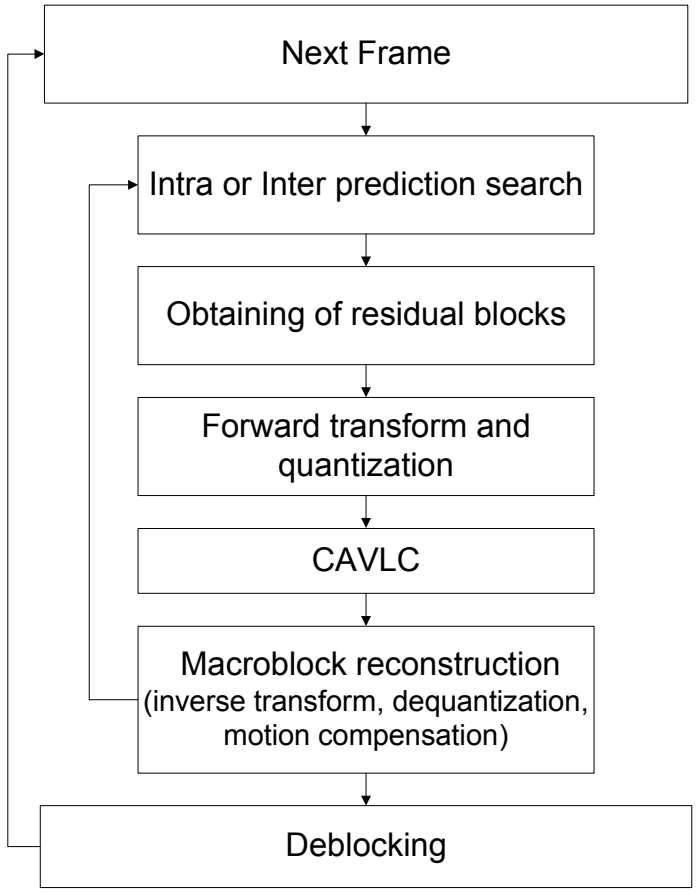
Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.

AVS Encoder Functions

This subsection describes functions for encoding of digital audio-video in accordance with the AVS ([AVS]) standard.

Figure 16-88 AVS Encoder Structure



Other general functions used by the AVS Encoder for predicted blocks estimation and for obtaining residual blocks are SAD Functions, Sum of Differences Evaluation functions, and `GetDiff8x8` function.

The AVS Encoder uses the AVS Decoder functions for calculation of inter and intra predicted blocks, Macroblock Reconstruction, and Deblocking Filtering. Forward transform and quantization, and CAVLC coding are performed by Encoder proper functions, which are described below.

Forward Transform and Quantization

TransformQuant8x8Fwd_AVS

Performs forward transform and quantization for 8X8 residual blocks.

Syntax

```
IppStatus ippiTransformQuant8x8Fwd_AVS_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst,
    Ipp32u* pNumCoeffs, Ipp32u QP, Ipp32u roundMode);
```

Parameters

pSrc	Pointer to a 8x8 residual block - source array of size 64.
pDst	Pointer to a 8x8 residual block - destination array of size 64.
pNumCoeffs	Pointer to a value that contains a number of non-zero elements in the block after quantization. This value is calculated by the function.
QP	Quantization parameter (*CurrentQP* in [AVS]) for luma or chroma, in the range [0;63] or [0, 51].
roundMode	Flag equal to 1 in the case of intra slice, 0 otherwise.

Description

The function `ippiTransformQuant8x8Fwd_AVS_16s_C1` is declared in the `ippvc.h` file. This function performs forward transform and quantization for an 8x8 residual block.

An 8x8 residual block (*pSrc*) is transformed. Quantization is performed on the transformed block. The result is saved to *pDst*.

This function is used in the AVS encoder included into Intel IPP Samples. See introduction to the chapter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	*QP* is less than 0 or greater than 63.

Macroblock Disassembling

DisassembleLumaIntra_AVS

Disassembles intra luma macroblocks.

Syntax

```
IppStatus ippiDisassembleLumaIntra_AVS_16s8u_C1R(const Ipp8u* pSrcPlane, Ipp32s
    srcStep, Ipp8u* pDstPlane, Ipp32s dstStep, Ipp16s** ppDstCoeff, const
    IppIntra8x8PredMode_AVS* pPredModes, Ipp32u* pLumaCBP, Ipp32u QP, Ipp32u
    edgeType);
```

Parameters

pSrcPlane	Pointer to the source macroblock that is being coded in *Y*-plane.
srcStep	Source *Y*-plane step.
pDstPlane	Pointer to the current macroblock that is reconstructed in *Y*-plane.
dstStep	Destination *Y*-plane step.
ppDstCoeff	Pointer to the order of 8x8 blocks of residual coefficients for the macroblock. The pointer is updated by the function and points to free space for the blocks for the next macroblock.
pPredModes	Array of Intra_8x8 luma prediction modes for each subblock.
pLumaCBP	Pointer to the variable to store a coded block pattern. This value is calculated by the function.
QP	Quantization parameter (*CurrentQP* in [AVS]), must be in the range [0;63].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction,

edgeType&IPPVC_TOP_EDGE must be non-zero.
If the left macroblock is not available for 8x8 intra prediction, edgeType&IPPVC_LEFT_EDGE must be non-zero.
If the upper-left macroblock is not available for 8x8Intra prediction, edgeType&IPPVC_TOP_LEFT_EDGE must be non-zero.
If the upper-right macroblock is not available for 8x8Intra prediction, edgeType&IPPVC_TOP_RIGHT_EDGE must be non-zero.

Description

The function ippiDisassembleLumaIntra_AVS_16s8u_C1R is declared in the ippvc.h file. This function disassembles intra luma macroblocks. The residual coefficients are taken as a result of intra-prediction, forward transforming, and forward quantization (8x8 luminance intra blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The process for the 8x8 blocks is described below:

1. Perform intra prediction for an 8x8 luma component in accordance with 9.4.2 of [AVS].
2. Perform subtraction of an 8x8 prediction block from an 8x8 source block.
3. Perform scaling, integer transformation, and forward shift for an 8x8 block.
4. Perform these steps for every available prediction modes for each block and save the best modes to a corresponding variable.

The function calculates the best prediction mode for each subblock, trying to save as much of visual information as possible. *pPredModes*[*i*] is defined in the same order as the coded subblocks go.

If* *pLumaCBP* & (1<<(1+*i*))is not equal to 0 ($0 \le i < 4$), *i*-th 8x8 AC luma block is not zero-filled and it exists in *ppDstCoeff*.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if at least one of the specified pointers is NULL.
ippStsOutOfRangeErr	*QP* is less than 0 or greater than 63.

DisassembleChroma420Intra_AVS

Disassembles intra chroma macroblocks.

Syntax

```
IppStatus ippiDisassembleChroma420Intra_AVS_16s8u_C1R(const Ipp8u*
    pSrcPlane[2], Ipp32s srcStep, Ipp8u* pDstPlane[2], Ipp32s dstStep, Ipp16s**
    ppDstCoeff, const IppIntraChromaPredMode_AVS* pPredModes, Ipp32u*
    pChromaCBP, Ipp32u QP, Ipp32u edgeType);
```

Parameters

pSrcPlane	Array of pointers to the source macroblock that is coded in *U*-plane and *Y*-plane.
srcStep	Source chrominance plane step.
pDstPlane	Pointer to the current macroblock that is reconstructed in *U*-plane and *Y*-plane.
dstStep	Destination chrominance plane step.
ppDstCoeff	Double pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of intra-prediction, forward transforming, and forward quantization (8x8 chrominance intra blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-7 of [AVS]. The pointer is updated by the function and points to free space for the blocks for the next macroblock.
pPredModes	Pointer to the variable to store chrominance prediction mode for both subblocks. The function calculates the best prediction mode for each subblock, trying to save as much of visual information as possible. *pPredModes*[*i*] is defined in the same order as the coded subblocks go.
pChromaCBP	Pointer to the variable to store a coded block pattern. This value is calculated by the function. If* *pChromaCBP* & (1<<(1+*i*)) is not equal to 0 ($0 \le i < 4$), *i*-th 8x8 AC chroma block is not zero-filled and it exists in *ppDstCoeff*.
QP	Quantization parameter (*CurrentQP* in [AVS]), must be within range [0;51].
edgeType	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 intra prediction, edgeType&IPPVC_TOP_EDGE must be non-zero.

If the left macroblock is not available for 8x8 intra prediction, `edgeType&IPPVC_LEFT_EDGE` must be non-zero.
If the upper-left macroblock is not available for 8x8Intra prediction, `edgeType&IPPVC_TOP_LEFT_EDGE` must be non-zero.
If the upper-right macroblock is not available for 8x8Intra prediction, `edgeType&IPPVC_TOP_RIGHT_EDGE` must be non-zero.

Description

The function `ippiDisassembleChroma420Intra_AVS_16s8u_C1R` is declared in the `ippvc.h` file. This function disassembles intra chroma macroblocks. The process for the 8x8 blocks in the same order as is shown in Figure 6-7 of [AVS] is described below:

1. Perform intra prediction for an 8x8 chroma component in accordance with 9.4.2 of [AVS].
2. Perform subtraction of an 8x8 prediction block from an 8x8 source block.
3. Perform scaling, integer transformation, and forward shift for an 8x8 block.
4. Perform these steps for every available prediction modes for each block and save the best modes to a corresponding variable.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if at least one of the specified pointers is `NULL`.
`ippStsOutOfRangeErr`	*QP* is less than 0 or greater than 51.

VC-1

This section describes functions for decoding and encoding of video data in accordance with VC-1 standard ([SMPTE 421M]).

The following enumeration is used for deblocking and smoothing functions to indicate parts of luma blocks edges:

```
enum{

IPPVC_EDGE_QUARTER_1            = 0x01,
IPPVC_EDGE_QUARTER_2            = 0x02,
IPPVC_EDGE_QUARTER_3            = 0x04,
IPPVC_EDGE_QUARTER_4            = 0x08,
IPPVC_EDGE_HALF_1          = IPPVC_EDGE_QUARTER_1 + IPPVC_EDGE_QUARTER_2,
IPPVC_EDGE_HALF_2          = IPPVC_EDGE_QUARTER_3 + IPPVC_EDGE_QUARTER_4,
IPPVC_EDGE_ALL             = IPPVC_EDGE_HALF_1 + IPPVC_EDGE_HALF_2,

};
```

Note that the VC-1 functions with `_16u_` and `_8u_` data types use the following structures:

```
typedef struct _IppVCInterpolate_16u
{
    const Ipp16u* pSrc;
    Ipp32s      srcStep;
    Ipp16u*     pDst;
    Ipp32s      dstStep;
    Ipp32s      dx;
    Ipp32s      dy;
    IppiSize    roiSize;
    Ipp32s      bitDepth;
} IppVCInterpolate_16u;

typedef struct _IppVCInterpolate_8u
{
    const Ipp8u* pSrc;
```

```
    Ipp32s      srcStep;
    Ipp8u*      pDst;
    Ipp32s      dstStep;
    Ipp32s      dx;
    Ipp32s      dy;
    IppiSize    roiSize;
    Ipp32s      bitDepth;
} IppVCInterpolate_8u;
```

where dx, dy are fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).

Figure 16-89 Horizontal Luma Edge

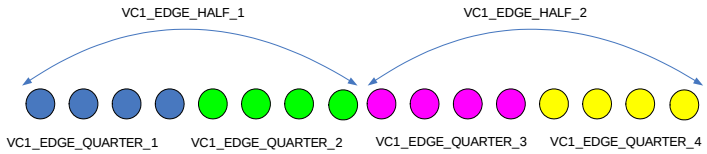


Figure 16-90 Vertical Luma Edge

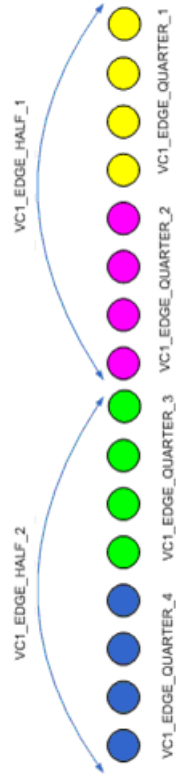


Figure 16-91 Horizontal Chroma Edge

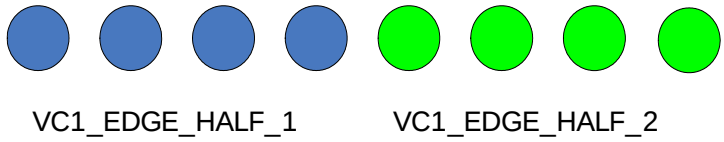
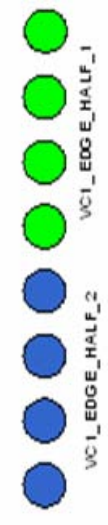


Figure 16-92 Vertical Chroma Edge



VC-1 Decoder

See Figure 16-93 for VC-1 decoding scheme. The functions not in brown color are auxiliary and are called only once in the process of deblocking.

Figure 16-93 VC-1 Decoding Scheme

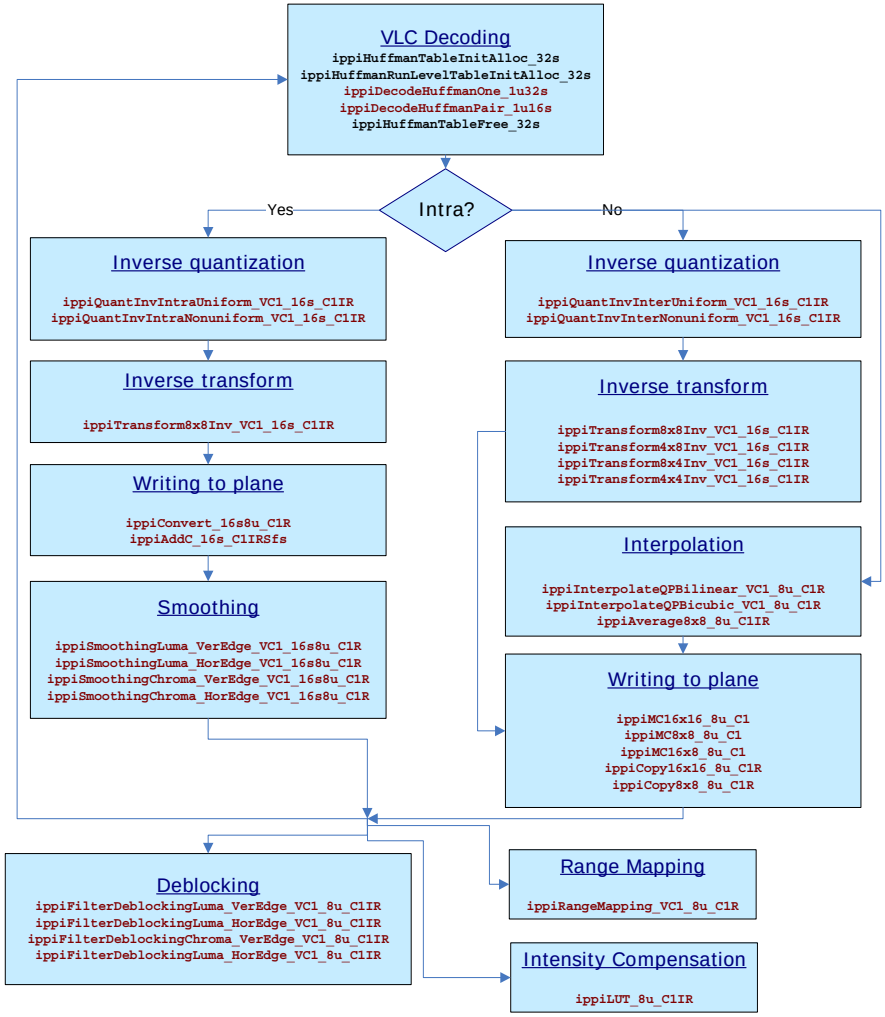


Table 16-31 VC1 Video Decoder Functions

Function Short Name	Description
Inverse Transform	
Transform8x8Inv_VC1, Transform4x8Inv_VC1, Transform8x4Inv_VC1, Transform4x4Inv_VC1	Perform inverse transform of blocks of respective size according to VC-1 standard.
Interpolation	
InterpolateQPBilinear_VC1	Performs bilinear quarter-pel interpolation.
InterpolateQPBicubic_VC1	Performs bicubic quarter-pel interpolation.
Smoothing	
SmoothingLuma_VerEdge_VC1	Performs smoothing filtering on the vertical edge (internal or external) of luma 16X16 macroblocks.
SmoothingLuma_HorEdge_VC1	Performs smoothing filtering on the horizontal edge (internal or external) of luma 16X16 macroblocks.
SmoothingChroma_VerEdge_VC1	Performs smoothing filtering on the vertical edge (internal or external) of chroma 8X8 blocks.
SmoothingChroma_HorEdge_VC1	Performs smoothing filtering on the horizontal edge (internal or external) of chroma 8X8 blocks.
Deblocking	
FilterDeblockingLuma_VerEdge_VC1	Performs deblocking filtering on the vertical edge (internal or external) of luma 16X16 macroblocks.
FilterDeblockingLuma_HorEdge_VC1	Performs deblocking filtering on the horizontal edge (internal or external) of luma 16X16 macroblocks.
FilterDeblockingChroma_VerEdge_VC1	Performs deblocking filtering on the vertical edge (internal or external) of chroma 8X8 macroblocks.
FilterDeblockingChroma_HorEdge_VC1	Performs deblocking filtering on the horizontal edge (internal or external) of chroma 8X8 macroblocks.

Table 16-31 VC1 Video Decoder Functions (continued)

Function Short Name	Description
Dequantization	
QuantInvIntraUniform_VC1, QuantInvIntraNonuniform_VC1	Perform uniform/nonuniform dequantization of 8X8 intra blocks.
QuantInvInterUniform_VC1, QuantInvInterNonuniform_VC1	Perform uniform/nonuniform dequantization of inter blocks.
Range Reduction	
RangeMapping_VC1	Performs range map transformation.

Inverse Transform

Transform8x8Inv_VC1, Transform4x8Inv_VC1, Transform8x4Inv_VC1, Transform4x4Inv_VC1

Perform inverse transform of blocks of respective sizes according to VC-1 standard.

Syntax

```
IppStatus ippiTransform8x8Inv_VC1_16s_C1IR( Ipp16s*  pSrcDst, Ipp32s
    srcDstStep, IppiSize sizeNZ );
IppStatus ippiTransform4x8Inv_VC1_16s_C1IR( Ipp16s*  pSrcDst, Ipp32s
    srcDstStep, IppiSize sizeNZ );
IppStatus ippiTransform8x4Inv_VC1_16s_C1IR( Ipp16s*  pSrcDst, Ipp32s
    srcDstStep, IppiSize sizeNZ );
IppStatus ippiTransform4x4Inv_VC1_16s_C1IR( Ipp16s*  pSrcDst, Ipp32s
    srcDstStep, IppiSize sizeNZ );
IppStatus ippiTransform8x8Inv_VC1_16s_C1R( const Ipp16s*  pSrc, Ipp16s*  pDst,
    Ipp32s srcDstStep, IppiSize sizeNZ );
```

```
IppStatus ippiTransform4x8Inv_VC1_16s_C1R( const Ipp16s*  pSrc, Ipp16s*  pDst,
    Ipp32s srcDstStep, IppiSize sizeNZ );
IppStatus ippiTransform8x4Inv_VC1_16s_C1R( const Ipp16s*  pSrc, Ipp16s*  pDst,
    Ipp32s srcDstStep, IppiSize sizeNZ );
IppStatus ippiTransform4x4Inv_VC1_16s_C1R( const Ipp16s*  pSrc, Ipp16s*  pDst,
    Ipp32s srcDstStep, IppiSize sizeNZ );
```

Parameters

pSrcDst	Pointer to the source and destination block. All samples of the source block should be in range [-2048; 2047). After transforming all samples of the destination block are in range [-512, 511).
pSrc	Pointer to the source block. All samples of the source block should be in range [-2048; 2047).
pDst	Pointer to the destination 8x8 block. After transforming all samples of the destination block are in range [-512, 511).
srcDstStep	Distance between starts of the consecutive lines in the source and destination blocks.
sizeNZ	Size of the top left-hand subblock with non-zero coefficients. This parameter is used by the function for improving performance with fast transforming algorithms if *sizeNZ* is less then block size. If *sizeNZ* is equal to the block size then “full” transforming algorithm is used.

Description

The functions `ippiTransform8x8Inv_VC1_16s`, `ippiTransform4x8Inv_VC1_16s`, `ippiTransform8x4Inv_VC1_16s`, and `ippiTransform4x4Inv_VC1_16s` are declared in the `ippvc.h` file. These functions perform inverse transform in accordance with Annex A: *Transform Specification* of [SMPTE 421M].

`ippiTransform8x8Inv_VC1_16s_C1IR` and `ippiTransform8x8Inv_VC1_16s_C1R` perform inverse transform of 8x8 inter block with 8x8 transformation type or 8x8 intra block.

Example 16-26 ippiTransform8x8Inv_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-2048; 2047)
After transforming all samples of destination block will be in the range [-512,
511)
*/
```

Example 16-26 ippiTransform8x8Inv_VC1_16s_C1IR Usage (continued)

```
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppiSize dstSizeNZ = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform8x8Inv_VC1_16s_C1IR(pSrcDstData, srcDstStep, dstSizeNZ);
/*
Performs inverse transform of 8x8 inter block with 8x8 transformation type or
8x8 intra block in accordance with Annex A "Transform Specification"  SMPTE 421M
*/

}
```

Example 16-27 ippiTransform8x8Inv_VC1_16s_C1R Usage

```
{
Ipp16s*  pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-2048; 2047)
*/
Ipp16s*  pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the range [-512,
511)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/
IppiSize dstSizeNZ = block_size_non_zero;
/*
```

Example 16-27 ippiTransform8x8Inv_VC1_16s_C1R Usage (continued)

```
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/

IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform8x8Inv_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep,
dstSizeNZ );
/*
Performs inverse transform of 8x8 inter block with 8x8 transformation type or
8x8 intra block in accordance with Annex A "Transform Specification"  SMPTE 421M
*/

}
```

`ippiTransform4x8Inv_VC1_16s_C1IR` and `ippiTransform4x8Inv_VC1_16s_C1R` perform inverse transform of 4x8 block into 8x8 inter block with 4x8 transformation type.

Example 16-28 ippiTransform4x8Inv_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-2048; 2047)
After transforming all samples of destination block will be in the range [-512,
511)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppiSize dstSizeNZ[2] = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
```

Example 16-28 ippiTransform4x8Inv_VC1_16s_C1IR Usage (continued)

```
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform4x8Inv_VC1_16s_C1IR(pSrcDstData,srcDstStep,dstSizeNZ[0]);
result = ippiTransform4x8Inv_VC1_16s_C1IR(pSrcDstData+4,srcDstStep,
dstSizeNZ[1]);

/*
Performs inverse transform of 4x8 subblock in 8x8 inter block with 4x8
transformation type in accordance with Annex A "Transform Specification"  SMPTE
421M
*/

}
```

Example 16-29 ippiTransform4x8Inv_VC1_16s_C1R Usage

```
{
Ipp16s*  pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-2048; 2047)
*/
Ipp16s*  pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the range [-512,
511)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/
IppiSize dstSizeNZ[2] = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
```

Example 16-29 ippiTransform4x8Inv_VC1_16s_C1R Usage (continued)

```
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform4x8Inv_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep,
dstSizeNZ[0] );
result = ippiTransform4x8Inv_VC1_16s_C1R( pSrcData+4, srcStep, pDstData+4,
dstStep, dstSizeNZ[1] );

/*
Performs inverse transform of 4x8 subblock in 8x8 inter block with 4x8
transformation type in accordance with Annex A "Transform Specification" SMPTE
421M
*/

}
```

`ippiTransform8x4Inv_VC1_16s_C1IR` and `ippiTransform8x4Inv_VC1_16s_C1R` perform inverse transform of 8x4 subblock into 8x8 inter block with 8x4 transformation type.

Example 16-30 ippiTransform8x4Inv_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-2048; 2047)
After transforming all samples of destination block will be in the range [-512,
511)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppiSize dstSizeNZ[2] = block_size_non_zero;
/*
```

Example 16-30 ippiTransform8x4Inv_VC1_16s_C1IR Usage (continued)

```
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform8x4Inv_VC1_16s_C1IR( pSrcDstData, srcDstStep,
dstSizeNZ[0] );
result = ippiTransform8x4Inv_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2),
srcDstStep, dstSizeNZ[1] );

/*
Performs inverse transform of 8x4 subblock in 8x8 inter block with 8x4
transformation type in accordance with Annex A "Transform Specification"  SMPTE
421M
*/

}
```

Example 16-31 ippiTransform8x4Inv_VC1_16s_C1R Usage

```
{
Ipp16s*  pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-2048; 2047)
*/
Ipp16s*  pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the range [-512,
511)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/
IppiSize dstSizeNZ[2] = block_size_non_zero;
```

Example 16-31 ippiTransform8x4Inv_VC1_16s_C1R Usage (continued)

```
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/

IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform8x4Inv_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep,
dstSizeNZ[0] );
result = ippiTransform8x4Inv_VC1_16s_C1R( pSrcData+(srcStep*2), srcStep,
pDstData+(dstStep*2), dstStep, dstSizeNZ[1] );

/*
Performs inverse transform of 8x4 subblock in 8x8 inter block with 8x4
transformation type in accordance with Annex A "Transform Specification"  SMPTE
421M
*/

}
```

`ippiTransform4x4Inv_VC1_16s_C1IR` and `ippiTransform4x4Inv_VC1_16s_C1R` perform inverse transform of 4x4 block into 8x8 inter block with 4x4 transformation type.

Example 16-32 ippiTransform4x4Inv_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-2048; 2047)
After transforming all samples of destination block will be in the range [-512,
511)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppiSize dstSizeNZ[4] = block_size_non_zero;
```

Example 16-32 ippiTransform4x4Inv_VC1_16s_C1IR Usage (continued)

```
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/
IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform4x4Inv_VC1_16s_C1IR( pSrcDstData, srcDstStep,
dstSizeNZ[0] );
result = ippiTransform4x4Inv_VC1_16s_C1IR( pSrcDstData+4, srcDstStep,
dstSizeNZ[1] );
result = ippiTransform4x4Inv_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2),
srcDstStep, dstSizeNZ[2] );
result = ippiTransform4x4Inv_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2)+4,
srcDstStep, dstSizeNZ[3] );

/*
Performs inverse transform of 4x4 subblock in 8x8 inter block with 4x4
transformation type in accordance with Annex A "Transform Specification"  SMPTE
421M
*/

}
```

Example 16-33 ippiTransform4x4Inv_VC1_16s_C1R Usage

```
{
Ipp16s*  pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-2048; 2047)
*/
Ipp16s*  pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the range [-512,
511)
*/
```

Example 16-33 ippiTransform4x4Inv_VC1_16s_C1R Usage (continued)

```
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/
IppiSize dstSizeNZ[4] = block_size_non_zero;
/*
The size of top left subblock with non-zero coefficients. This parameter is used
by this function for improving performance with the help of "fast" transforming
algorithms in the case of sizeNZ less than block size. If sizeNZ is equal to
block size, then "full" transforming algorithm is used.
Example (8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width=4, dstSizeNZ.height=3
*/

IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform4x4Inv_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep,
dstSizeNZ[0] );
result = ippiTransform4x4Inv_VC1_16s_C1R( pSrcData+4, srcStep, pDstData+4,
dstStep, dstSizeNZ[1] );
result = ippiTransform4x4Inv_VC1_16s_C1R( pSrcData+(srcStep*2), srcStep,
pDstData+(dstStep*2), dstStep,dstSizeNZ[2] );
result = ippiTransform4x4Inv_VC1_16s_C1R( pSrcData+(srcStep*2)+4, srcStep,
pDstData+(dstStep*2)+4, dstStep,dstSizeNZ[3] );

/*
Performs inverse transform of 4x4 subblock in 8x8 inter block with 4x4
transformation type in accordance with Annex A "Transform Specification"  SMPTE
421M
*/

}
```

See below for an example of 8x4 block transform.

Figure 16-94 8x4 Block Inverse Transform

16	0	0	-3	0	0	0	0
1	1	0	1	0	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

In this case `pDstSizeNZ -> width` should be equal to 4, `pDstSizeNZ -> height` should be equal to 3.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when at least one of the specified pointers is `NULL`.

Interpolation

InterpolateQPBilinear_VC1

Performs bilinear quarter-pel interpolation.

Syntax

```
IppStatus ippiInterpolateQPBilinear_VC1_8u_C1R( const IppVCInterpolate_8u*
    pParams );
```

Parameters

pParams	Pointer to the structure that contains parameters for interpolation: *pParams->pSrc* Pointer to the source *pParams->srcStep* Distance in bytes between starts of the consecutive lines in the source block *pParams->pDst* Pointer to the destination *pParams->dstStep* Distance in bytes between starts of the consecutive lines in the destination block

pParams->dx, *pParams->dy* Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3)

pParams->bitDepth Not used by this function, should be equal to 0

pParams->roundControl Frame level rounding control value as described in section 8.3.7 of [SMPTE 421M], should be equal to 0 or 1

pParams->roiSize Block size, should be

16x16:

- Progressive frames/ 1-MV Macroblocks/ luma blocks
- Interlace fields/ 1-MV Macroblocks/ luma blocks

8x8:

- Progressive frames/ 4-MV Macroblocks/ luma and chroma blocks
- Progressive frames/ 4-MV Macroblocks/ chroma blocks
- Interlace fields/ 4-MV Macroblocks/ chroma blocks
- Interlace fields/ 1-MV Macroblocks/ chroma blocks
- Interlace frames/ 1-MV Macroblocks/ chroma blocks

8x4:

- Interlace frames/ 2-MV Macroblocks/ chroma blocks

4x4:

- Interlace frames/ 4-MV Macroblocks/ chroma blocks.

Description

The function `ippiInterpolateQPBilinear_VC1_8u_C1R` is declared in the `ippvc.h` file. The function performs bilinear quarter-pel interpolation in accordance with 8.3.6.5.1 of [SMPTE 421M].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when at least one of the specified pointers is `NULL`.
`ippStsBadArgErr`	Indicates an error condition if *dx* or *dy* values fall outside [0,3].
`ippStsSizeErr`	Indicates an error condition in *pParams->roiSize*.

InterpolateQPBicubic_VC1

Performs bicubic quarter-pel interpolation.

Syntax

```
IppStatus ippiInterpolateQPBicubic_VC1_8u_C1R( const IppVCInterpolate_8u*
    pParams );
```

Parameters

pParams Pointer to the structure that contains parameters for interpolation:

pParams->pSrc Pointer to the source

pParams->srcStep Distance in bytes between starts of the consecutive lines in the source block

pParams->pDst Pointer to the destination

pParams->dstStep Distance in bytes between starts of the consecutive lines in the destination block

pParams->dx, *pParams->dy* Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3)

pParams->bitDepth Not used by this function, should be equal to 0

pParams->roundControl Frame level rounding control value as described in section 8.3.7 of [SMPTE 421M], should be equal to 0 or 1

pParams->roiSize Block size, should be

16x16:

- Progressive frames/ 1-MV Macroblocks/ luma blocks
- Interlace fields/ 1-MV Macroblocks/ luma blocks
- Interlace frames/ 1-MV Macroblocks/ luma blocks

16x8:

- Interlace frames/ 2-MV Macroblocks/ luma blocks

8x8:

- Progressive frames/ 4-MV Macroblocks/ luma blocks
- Interlace fields/ 4-MV Macroblocks/ luma blocks
- Interlace frames/ 4-MV Macroblocks/ luma blocks.

Description

The function `ippiInterpolateQPBicubic_VC1_8u_C1R` is declared in the `ippvc.h` file. The function performs bicubic quarter-pel interpolation in accordance with 8.3.6.5.2 of [SMPTE 421M].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when at least one of the specified pointers is `NULL`.
`ippStsBadArgErr`	Indicates an error condition if *dx* or *dy* values fall outside [0,3].
`ippStsSizeErr`	Indicates an error condition in *pParams->roiSize*.

Smoothing

SmoothingLuma_VerEdge_VC1

Performs smoothing filtering on the vertical edge (internal or external) of luma 16X16 macroblocks.

Syntax

```
IppStatus ippiSmoothingLuma_VerEdge_VC1_16s8u_C1R( Ipp16s* pSrcDstLeft, Ipp32s
    srcDstLeftStep, Ipp16s* pSrcDstRight, Ipp32s srcDstRightStep, Ipp8u* pDst,
    Ipp32s dstStep, Ipp32u fieldNeighbourFlag, Ipp32u edgeDisableFlag );
```

Parameters

pSrcDstLeft	Pointer to the first pixel of the column of the top left block of the left macroblock, from which the smoothing starts.
srcDstLeftStep	Step for transfer to the next row in the left macroblock.
pSrcDstRight	Pointer to the first pixel of the top left block of the right macroblock.
srcDstRightStep	Step for transfer to the next row in the right macroblock.
pDst	Pointer to the first pixel of the right macroblock in the Y-plane.
dstStep	Distance in bytes between starts of the consecutive lines in the Y-plane.

fieldNeighbourFlag Indicates the field macroblock property (2 bits):

if `(fieldNeighbourFlag & VC1_FIELD_LEFT_MB)` – the left macroblock is field decoded,

if `(fieldNeighbourFlag & VC1_FIELD_RIGHT_MB)` – the right macroblock is field decoded.

edgeDisableFlag Flag indicating that

if `(edgeDisableFlag & IPPVC_EDGE_HALF_1)`, then the upper vertical edge is disabled for smoothing,

if `(edgeDisableFlag & IPPVC_EDGE_HALF_2)`, then the bottom vertical edge is disabled for smoothing.

Description

The function `ippiSmoothingLuma_VerEdge_VC1_16s8u_C1R` is declared in the `ippvc.h` file. The function performs smoothing filtering on the vertical edge (internal or external) of the luma 16x16 macroblocks. See 8.5 of [SMPTE 421M].

Figure 16-95 Vertical Smoothing Between Two Macroblocks

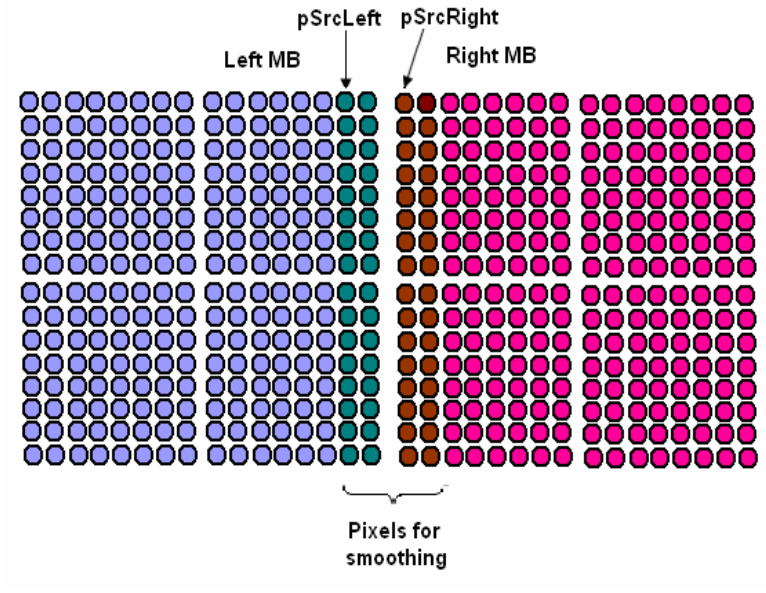
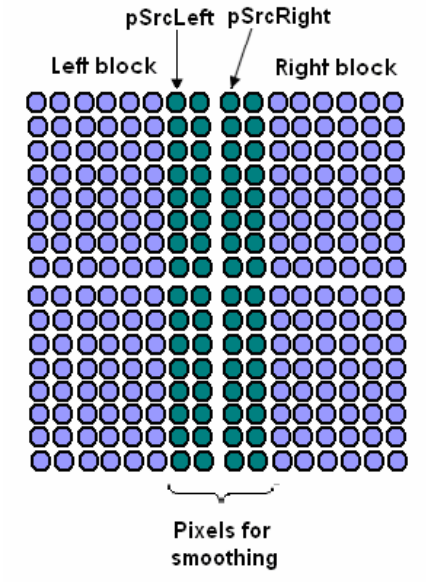


Figure 16-96 Vertical Smoothing Inside One Macroblock



All samples of the source block should be within the range [-2048; 2047). After smoothing, all samples of the destination block are within the range [-512, 511).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when source or desctination pointers are `NULL`.

SmoothingLuma_HorEdge_VC1

Performs smoothing filtering on the horizontal edge (internal or external) of luma 16X16 macroblocks.

Syntax

```
IppStatus ippiSmoothingLuma_HorEdge_VC1_16s8u_C1R( Ipp16s* pSrcDstUpper, Ipp32s
    srcDstUpperStep, Ipp16s* pSrcDstBottom, Ipp32s srcDstBottomStep, Ipp8u*
    pDst, Ipp32s dstStep, Ipp32u edgeDisableFlag );
```

Parameters

pSrcDstUpper	Pointer to the first pixel of the bottom left block row of the upper macroblock, from which the smoothing starts.
srcDstUpperStep	Step for transfer to the next row in the upper macroblock.
pSrcDstBottom	Pointer to the first pixel of the top left block of the bottom macroblock.
srcDstBottomStep	Step for transfer to the next row in the bottom macroblock.
pDst	Pointer to the first pixel of the bottom macroblock in the Y-plane.
dstStep	Distance in bytes between starts of the consecutive lines in the Y-plane.
edgeDisableFlag	Flag indicating that if `(edgeDisableFlag & IPPVC_EDGE_HALF_1)`, then the left horizontal edge is disabled for smoothing, if `(edgeDisableFlag & IPPVC_EDGE_HALF_2)`, then the right horizontal edge is disabled for smoothing.

Description

The function `ippiSmoothingLuma_HorEdge_VC1_16s8u_C1R` is declared in the `ippvc.h` file. The function performs smoothing filtering on the horizontal edge (internal or external) of the luma 16x16 macroblocks. See 8.5 of [SMPTE 421M].

Figure 16-97 Horizontal Smoothing Between Two Macroblocks

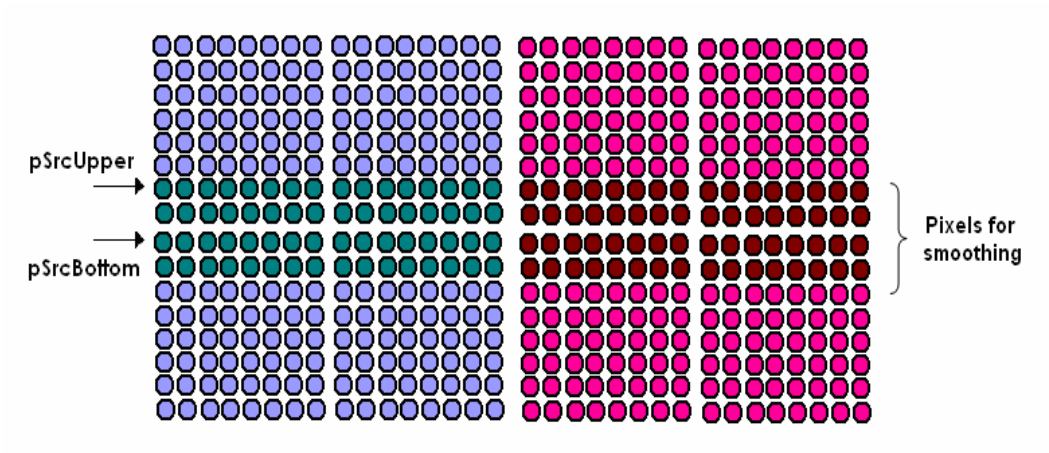
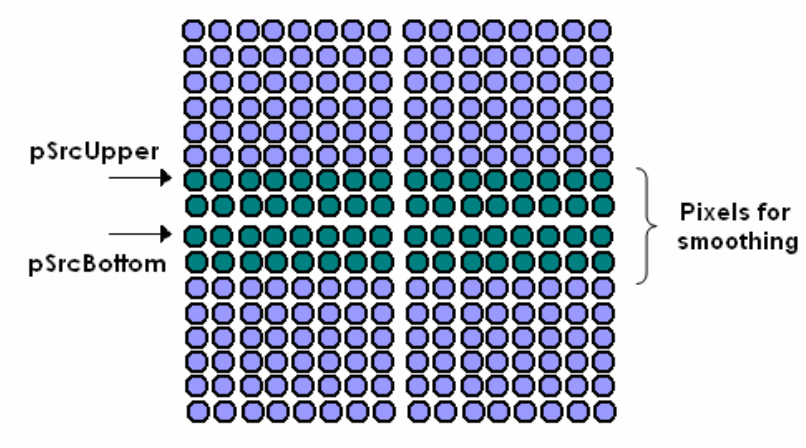


Figure 16-98 Horizontal Smoothing Inside One Macroblock



All samples of the source block should be within the range [-2048; 2047). After smoothing, all samples of the destination block are within the range [-512, 511).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when source or desctination pointers are `NULL`.

SmoothingChroma_VerEdge_VC1

Performs smoothing filtering on the vertical edge (internal or external) of chroma 8X8 blocks.

Syntax

```
IppStatus ippiSmoothingChroma_VerEdge_VC1_16s8u_C1R( Ipp16s* pSrcDstLeft,
    Ipp32s srcDstLeftStep, Ipp16s* pSrcDstRight, Ipp32s srcDstRightStep, Ipp8u*
    pDst, Ipp32s dstStep );
```

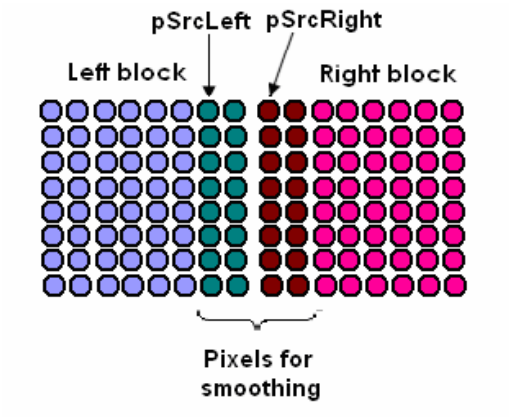
Parameters

pSrcDstLeft	Pointer to the first pixel of the column of the left block, from which the smoothing starts.
srcDstLeftStep	Step for transfer to the next row in the left block.
pSrcDstRight	Pointer to the first pixel of the right block.
srcDstRightStep	Step for transfer to the next row in the right block.
pDst	Pointer to the first pixel of the right block in the Y-plane.
dstStep	Distance in bytes between starts of the consecutive lines in the Y-plane.

Description

The function `ippiSmoothingChroma_VerEdge_VC1_16s8u_C1R` is declared in the `ippvc.h` file. The function performs smoothing filtering on the vertical edge (internal or external) of the chroma 8x8 blocks. See 8.5 of [SMPTE 421M].

Figure 16-99 Vertical Smoothing Between Two Chroma Blocks



All samples of the source block should be within the range [-2048; 2047). After smoothing, all samples of the destination block are within the range [-512, 511).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when source or desctination pointers are NULL.

SmoothingChroma_HorEdge_VC1

Performs smoothing filtering on the horizontal edge (internal or external) of chroma 8X8 blocks.

Syntax

```
IppStatus ippiSmoothingChroma_HorEdge_VC1_16s8u_C1R( Ipp16s* pSrcDstUpper,
    Ipp32s srcDstUpperStep, Ipp16s* pSrcDstBottom, Ipp32s srcDstBottomStep,
    Ipp8u* pDst, Ipp32s dstStep );
```

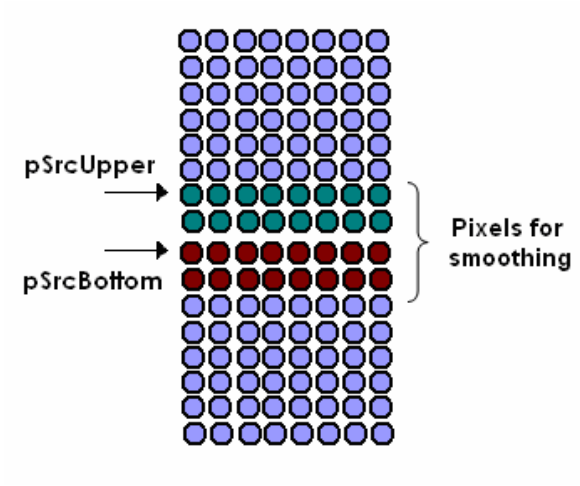
Parameters

pSrcDstUpper	Pointer to the first pixel of therow of the upper block , from which the smoothing starts.
srcDstUpperStep	Step for transfer to the next row in the upper block.
pSrcDstBottom	Pointer to the first pixel of the bottom block.
srcDstBottomStep	Step for transfer to the next row in the bottom block.
pDst	Pointer to the first pixel of the bottom block in the Y-plane.
dstStep	Distance in bytes between starts of the consecutive lines in the Y-plane.

Description

The function ippiSmoothingLuma_HorEdge_VC1_16s8u_C1R is declared in the ippvc.h file. The function performs smoothing filtering on the horizontal edge (internal or external) of the chroma 8x8 blocks. See 8.5 of [SMPTE 421M].

Figure 16-100 Horizontal Smoothing Between Two Chroma Block



All samples of the source block should be within the range [-2048; 2047). After smoothing, all samples of the destination block are within the range [-512, 511).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when source or desctination pointers are `NULL`.

Deblocking

FilterDeblockingLuma_VerEdge_VC1

Performs deblocking filtering on the vertical edge (internal or external) of luma 16X16 macroblocks.

Syntax

```
IppStatus ippiFilterDeblockingLuma_VerEdge_VC1_8u_C1IR( Ipp8u* pSrcDst, Ipp32s
    srcdstStep, Ipp32s pQuant, Ipp32u EdgeDisableFlag );
```

Parameters

pSrcDst	Pointer to the first pixel of the right block in the Y-plane.
srcdstStep	Distance in bytes between starts of the consecutive lines in the Y-plane.
pQuant	Picture quantizer scale.
EdgeDisableFlag	Flag indicating that if `(EdgeDisableFlag & IPPVC_EDGE_QUARTER_1)`, then the first vertical quarter of block edge is disabled for deblocking, if `(EdgeDisableFlag & IPPVC_EDGE_QUARTER_2)`, then the second vertical quarter of block edge is disabled for deblocking, if `(EdgeDisableFlag & IPPVC_EDGE_QUARTER_3)`, then the third vertical quarter of block edge is disabled for deblocking, if `(EdgeDisableFlag & IPPVC_EDGE_QUARTER_4)`, then the fourth vertical quarter of block edge is disabled for deblocking, if `(EdgeDisableFlag & IPPVC_EDGE_HALF_1)`, then the upper vertical edge of the block is disabled for deblocking, if `(EdgeDisableFlag & IPPVC_EDGE_HALF_2)`, then the bottom vertical edge of the block is disabled for deblocking. Conditions 1, 2, 3, 4 can be used for the internal edge of macroblock in the case of progressive P frames. Conditions 5, 6 can be used for other cases.

Description

The function `ippiFilterDeblockingLuma_VerEdge_VC1_8u_C1IR` is declared in the `ippvc.h` file. The function performs deblocking filtering on the vertical edge (internal or external) of the luma 16x16 macroblocks. See 8.6 of [SMPTE 421M].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when at least one of the specified pointers is NULL.

FilterDeblockingLuma_HorEdge_VC1

Performs deblocking filtering on the horizontal edge (internal or external) of luma 16X16 macroblocks.

Syntax

```
IppStatus ippiFilterDeblockingLuma_HorEdge_VC1_8u_C1IR( Ipp8u* pSrcDst, Ipp32s
    srcdstStep, Ipp32s pQuant, Ipp32u EdgeDisableFlag );
```

Parameters

pSrcDst	Pointer to the first pixel of the lower block in the Y-plane.
srcdstStep	Distance between starts of the consecutive lines in the array.
pQuant	Picture quantizer scale.

EdgeDisableFlag	Flag indicating that
	if (`EdgeDisableFlag & IPPVC_EDGE_QUARTER_1`), then the first horizontal quarter of block edge is disabled for deblocking,
	if (`EdgeDisableFlag & IPPVC_EDGE_QUARTER_2`), then the second horizontal quarter of block edge is disabled for deblocking,
	if (`EdgeDisableFlag & IPPVC_EDGE_QUARTER_3`), then the third horizontal quarter of block edge is disabled for deblocking,
	if (`EdgeDisableFlag & IPPVC_EDGE_QUARTER_4`), then the fourth horizontal quarter of block edge is disabled for deblocking,

if (`EdgeDisableFlag & IPPVC_EDGE_HALF_1`), then the left horizontal edge of the block is disabled for deblocking,

if (`EdgeDisableFlag & IPPVC_EDGE_HALF_2`), then the right horizontal edge of the block is disabled for deblocking.

Conditions 1, 2, 3, 4 can be used for the internal edge of macroblock in the case of progressive P frames.

Conditions 5, 6 can be used for other cases.

Description

The function `ippiFilterDeblockingLuma_HorEdge_VC1_8u_C1IR` is declared in the `ippvc.h` file. The function performs deblocking filtering on the horizontal edge (internal or external) of the luma 16x16 macroblocks. See 8.6 of [SMPTE 421M].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when at least one of the specified pointers is NULL.

FilterDeblockingChroma_VerEdge_VC1

Performs deblocking filtering on the vertical edge (internal or external) of chroma 8X8 macroblock.

Syntax

```
IppStatus ippiFilterDeblockingChroma_VerEdge_VC1_8u_C1IR( Ipp8u* pSrcDst,
    Ipp32s srcdstStep, Ipp32s pQuant, Ipp32u EdgeDisableFlag );
```

Parameters

pSrcDst	Pointer to the first pixel of the right block in the U- or Y-plane.
srcdstStep	Distance in bytes between starts of the consecutive lines in the plane.
pQuant	Picture quantizer scale.
EdgeDisableFlag	Flag indicating that if (`EdgeDisableFlag & IPPVC_EDGE_HALF_1`), then the upper vertical edge of the block is disabled for deblocking,

if (EdgeDisableFlag & IPPVC_EDGE_HALF_2), then the bottom vertical edge of the block is disabled for deblocking.

These conditions can be used for the internal edge of macroblock in the case of progressive P frames. In the other cases, the flag should be equal to 0.

Description

The function ippiFilterDeblockingChroma_VerEdge_VC1_8u_C1IR is declared in the ippvc.h file. The function performs deblocking filtering on the vertical edge (internal or external) of the chroma 8x8 macroblock. See 8.6 of [SMPTE 421M].

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when at least one of the specified pointers is NULL.

FilterDeblockingChroma_HorEdge_VC1

Performs deblocking filtering on the horizontal edge (internal or external) of chroma 8X8 macroblock.

Syntax

```
IppStatus ippiFilterDeblockingChroma_HorEdge_VC1_8u_C1IR( Ipp8u* pSrcDst,
    Ipp32s srcdstStep, Ipp32s pQuant, Ipp32u EdgeDisableFlag );
```

Parameters

pSrcDst	Pointer to the first pixel of the lower block in the U- or Y-plane.
srcdstStep	Distance in bytes between starts of the consecutive lines in the plane.
pQuant	Picture quantizer scale.

EdgeDisableFlag	Flag indicating that
	if (EdgeDisableFlag & IPPVC_EDGE_HALF_1), then the upper horizontal edge of the block is disabled for deblocking,
	if (EdgeDisableFlag & IPPVC_EDGE_HALF_2), then the bottom horizontal edge of the block is disabled for deblocking.

These conditions can be used for the internal edge of macroblock in the case of progressive P frames. In the other cases, the flag should be equal to 0.

Description

The function `ippiFilterDeblockingChroma_HorEdge_VC1_8u_C1IR` is declared in the `ippvc.h` file. The function performs deblocking filtering on the horizontal edge (internal or external) of the chroma 8x8 macroblock. See 8.6 of [SMPTE 421M].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when at least one of the specified pointers is NULL.

Dequantization

QuantInvIntraUniform_VC1, QuantInvIntraNonuniform_VC1

Perform uniform/nonuniform dequantization of 8X8 intra blocks.

Syntax

```
IppStatus ippiQuantInvIntraUniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s
    srcDstStep, Ipp32s doubleQuant, IppSize* pDstSizeNZ );
IppStatus ippiQuantInvIntraNonuniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s
    srcDstStep, Ipp32s doubleQuant, IppSize* pDstSizeNZ );
IppStatus ippiQuantInvIntraUniform_VC1_16s_C1R( const Ipp16s* pSrc, Ipp32s
    srcStep, Ipp16s* pDst, Ipp32s dstStep, Ipp32s doubleQuant, IppiSize*
    pDstSizeNZ );
IppStatus ippiQuantInvIntraNonuniform_VC1_16s_C1R( const Ipp16s* pSrc, Ipp32s
    srcStep, Ipp16s* pDst, Ipp32s dstStep, Ipp32s doubleQuant, IppiSize*
    pDstSizeNZ );
```

Parameters

pSrcDst	Pointer to the source and destination block.
srcDstStep	Distance in bytes between starts of the consecutive lines in the source and destination blocks.
doubleQuant	Dequant coefficient; should be within the range [2, 62].
pDstSizeNZ	Pointer to a size of the top left subblock with non-zero coefficients. This value is calculated by this function and could be used for inverse transformation. See Figure 16-101.
pSrc	Pointer to the source block.
srcStep	Distance in bytes between starts of the consecutive lines in the source block.
pDst	Pointer to the destination block.
dstStep	Distance in bytes between starts of the consecutive lines in the destination block.

Description

These functions are declared in the `ippvc.h` file.

The *doubleQuant* parameter should be equal to `2*MQUANT+HALFQP`, where `MQUANT` and `HALFQP` are defined in VC-1 standard.

The functions perform dequantization of all coefficients in intra block (except DC) according to 8.1.2.8 of [SMPTE 421M].

In the case of uniform dequantization:

```
a[i]=quant_coef*a[i];
```

In the case of nonuniform dequantization:

```
a[i]=quant_coef*a[i] + sign(a[i])*(quant_coef >> 1);
```

where `a[i]` is the AC coefficient of the intra block, and `quant_coef` is the dequant coefficient.

Figure 16-101 Intra 8x8 Block Dequantization

16	0	0	**-3**	0	**1**	**1**	0
1	**1**	0	**1**	0	0	0	0
3	0	0	**1**	0	0	0	0
0	0	0	**1**	0	0	0	0
0	0	0	**-1**	0	0	0	0
0	**1**	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

In this case, *pDstSizeNZ* `->` `width` is equal to 7, and *pDstSizeNZ* `->` `height` is equal to 6.

See Example 16-34 and Example 16-35 for usage of the in-place functions

Example 16-34 QuantInvIntraUniform_VC1 Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62];*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppiSize  dstSizeNZ;
/*
The size of top left subblock with non-zero coefficients. This value is
calculated by this function and could be used for inverse transformation.
Example (intra block):
16 0 0 -3 0 1 1 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  1 0 0 0 0
0  0 0 -1 0 0 0 0
0  1 0  0 0 0 0 0
0  0 0  0 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width is equal to 7, dstSizeNZ.height is equal to 6
*/
IppStatus result;
```

Example 16-34 QuantInvIntraUniform_VC1 Usage (continued)

```
/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiQuantInvIntraUniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant, &dstSizeNZ );

/*  Performs uniform dequantization process for all coefficients in the intra
block (except DC) according to 8.1.2.8 "Inverse AC Coefficient Quantization"
specification SMPTE 421M
*/
}
```

Example 16-35 QuantInvIntraNonuniform_VC1 Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62];*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppiSize  dstSizeNZ;
/*
The size of top left subblock with non-zero coefficients. This value is
calculated by this function and could be used for inverse transformation.
Example (intra block):
16 0 0 -3 0 1 1 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  1 0 0 0 0
0  0 0 -1 0 0 0 0
0  1 0  0 0 0 0 0
0  0 0  0 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width is equal to 7, dstSizeNZ.height is equal to 6
*/
IppStatus result;
/*
ippStsNoErr          Indicates no error
ippStsNullPtrErr     Indicates an error when pSrcDstData is NULL
*/

result = ippiQuantInvIntraNonuniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant, &dstSizeNZ );
```

Example 16-35 QuantInvIntraNonuniform_VC1 Usage

```
/*  Performs nonuniform dequantization process for all coefficients in the intra
block (except DC) according to 8.1.2.8 "Inverse AC Coefficient Quantization"
specification SMPTE 421M
*/
}
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when *pSrcDst* or *pDstSizeNZ* is NULL.

QuantInvInterUniform_VC1, QuantInvInterNonuniform_VC1

Perform uniform/nonuniform dequantization of inter blocks.

Syntax

```
IppStatus ippiQuantInvInterUniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s
    srcDstStep, Ipp32s doubleQuant, IppiSize roiSize, IppSize* pDstSizeNZ );
IppStatus ippiQuantInvInterNonuniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s
    srcDstStep, Ipp32s doubleQuant, IppiSize roiSize, IppSize* pDstSizeNZ );
IppStatus ippiQuantInvInterUniform_VC1_16s_C1R( const Ipp16s* pSrc, Ipp32s
    srcStep, Ipp16s* pDst, Ipp32s dstStep, Ipp32s doubleQuant, IppiSize roiSize,
    IppiSize* pDstSizeNZ );
IppStatus ippiQuantInvInterNonuniform_VC1_16s_C1R( const Ipp16s* pSrc, Ipp32s
    srcStep, Ipp16s* pDst, Ipp32s dstStep, Ipp32s doubleQuant, IppiSize roiSize,
    IppiSize* pDstSizeNZ );
```

Parameters

pSrcDst	Pointer to the source and destination block.
srcDstStep	Distance in bytes between starts of the consecutive lines in the source and destination blocks.
doubleQuant	Dequant coefficient; should be within the range [2, 62].
roiSize	Inter block size; should be 8x8, 8x4, 4x8 or 4x4.

pDstSizeNZ	Pointer to a size of the top left subblock with non-zero coefficients. This value is calculated by this function and could be used for inverse transformation. See Figure 16-102.
pSrc	Pointer to the source block.
srcStep	Distance in bytes between starts of the consecutive lines in the source block.
pDst	Pointer to the destination block.
dstStep	Distance in bytes between starts of the consecutive lines in the destination block.

Description

The functions `ippiQuantInvInterUniform_VC1_16s_C1IR` and `ippiQuantInvInterNonuniform_VC1_16s_C1IR` are declared in the `ippvc.h` file.

The *doubleQuant* parameter should be equal to `2*MQUANT+HALFQP`, where `MQUANT` and `HALFQP` are defined in VC-1 standard.

The functions perform dequantization of inter blocks according to 8.1.2.8 of [SMPTE 421M].

In the case of uniform dequantization:

```
a[i]=quant_coef*a[i];
```

In the case of nonuniform dequantization:

```
a[i]=quant_coef*a[i] + sign(a[i])*(quant_coef >> 1);
```

where `a[i]` is the intra block coefficient, and `quant_coef` is the dequant coefficient.

Figure 16-102Inter 8x4 Block Dequantization

16	0	0	-3	0	0	0	0
1	1	0	1	0	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

In this case, *pDstSizeNZ* `->width` is equal to 4, and *pDstSizeNZ* `->height` is equal to 3.

See Example 16-36 and Example 16-37 for usage of in-place functions

Example 16-36 QuantInvInterUniform_VC1 Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62];*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppSize roiSize = inter_block_size;
/* It should be:
8x8 (roiSize.width=8, roiSize.height=8)
4x8 (roiSize.width=4, roiSize.height=8)
8x4 (roiSize.width=8, roiSize.height=4)
4x4 (roiSize.width=4, roiSize.height=4)
*/
IppiSize  dstSizeNZ[2];
/*
The size of top left subblock with non-zero coefficients. This value is
calculated by this function and could be used for inverse transformation.
Example (inter 8x4 block):
16 0 0 -3 0 0 0 0
1  1 0  1 0 0 0 0
3  0 0  1 0 0 0 0
0  0 0  0 0 0 0 0
In this case dstSizeNZ.width is equal to 4, dstSizeNZ.height is equal to 3
*/

IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
ippStsSizeErr       Indicates an error when roiSize.height is not equal 4 or 8
or roiSize.width is not equal 4 or 8
*/
/* in case of 8x4*/
result = ippiQuantInvInterUniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant, &dstSizeNZ[0] );
result = ippiQuantInvInterUniform_VC1_16s_C1IR( pSrcDstData+ (srcDstStep*2),
srcDstStep, doubleQuant, &dstSizeNZ[1] );


/*  Performs uniform dequantization process for inter block according to 8.1.2.8
"Inverse AC Coefficient Quantization" specification SMPTE 421M
*/
}
```

Example 16-37 QuantInvInterNonuniform_VC1 Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62];*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppSize roiSize = inter_block_size;
/* It should be:
8x8 (roiSize.width=8, roiSize.height=8)
4x8 (roiSize.width=4, roiSize.height=8)
8x4 (roiSize.width=8, roiSize.height=4)
4x4 (roiSize.width=4, roiSize.height=4)
*/
IppiSize  dstSizeNZ[4];
/*
The size of top left subblock with non-zero coefficients. This value is
calculated by this function and could be used for inverse transformation.
Example (inter 4x4 block):
16 0 0 -3
1  1 0  1
3  0 0  1
0  0 0  0
In this case dstSizeNZ.width is equal to 4, dstSizeNZ.height is equal to 3
*/
IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
ippStsSizeErr       Indicates an error when roiSize.height is not equal 4 or 8
or roiSize.width is not equal 4 or 8
*/

/* in case of 4x4*/
result = ippiQuantInvInterNonuniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant, &dstSizeNZ[0] );
result = ippiQuantInvInterNonuniform_VC1_16s_C1IR( pSrcDstData+4, srcDstStep,
doubleQuant, &dstSizeNZ[1] );
result = ippiQuantInvInterNonuniform_VC1_16s_C1IR( pSrcDstData+( srcDstStep*2),
srcDstStep, doubleQuant, &dstSizeNZ[2] );
result = ippiQuantInvInterNonuniform_VC1_16s_C1IR(
pSrcDstData+(srcDstStep*2)+4, srcDstStep, doubleQuant, &dstSizeNZ[3] );


/*  Performs nonuniform dequantization process for inter block according to
8.1.2.8 "Inverse AC Coefficient Quantization" specification SMPTE 421M
*/
}
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when *pSrcDst* or *pDstSizeNZ* is NULL.

Range Reduction

RangeMapping_VC1

Performs range map transformation.

Syntax

```
IppStatus ippiRangeMapping_VC1_8u_C1R( const Ipp8u* pSrc, Ipp32s srcStep,
    Ipp8u* pDst, Ipp32s dstStep, IppiSize roiSize, Ipp32s rangeMapParam );
```

Parameters

pSrc	Pointer to the source block. Block coefficient could be within the range of [0, 255].
srcStep	Distance in bytes between starts of the consecutive lines in the source block.
pDst	Pointer to the destination block.
dstStep	Distance in bytes between starts of the consecutive lines in the destination block.
roiSize	Size of the source block.
rangeMapParam	Parameter for the range map; should be within the range of [0, 7].

Description

The function `ippiRangeMapping_VC1_8u_C1R` is declared in the `ippvc.h` file. The function performs range map transformation according to 6.2.15.1 of [SMPTE 421M].

the function calculates each coefficient of the source block as follows:

```
Y[n]=CLIP((((Y[n]-128)*(RANGE_MAPY + 9) + 4) >> 3) + 128);
```

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when *pSrc* or *pDst* is NULL.
`ippStsOutOfRangeErr`	Indicates an error condition if *rangeMapParam* is out of the range [0, 7].

VC-1 Encoder

See Figure 16-103 for VC-1 encoding scheme.

Figure 16-103VC-1 Encoding Scheme

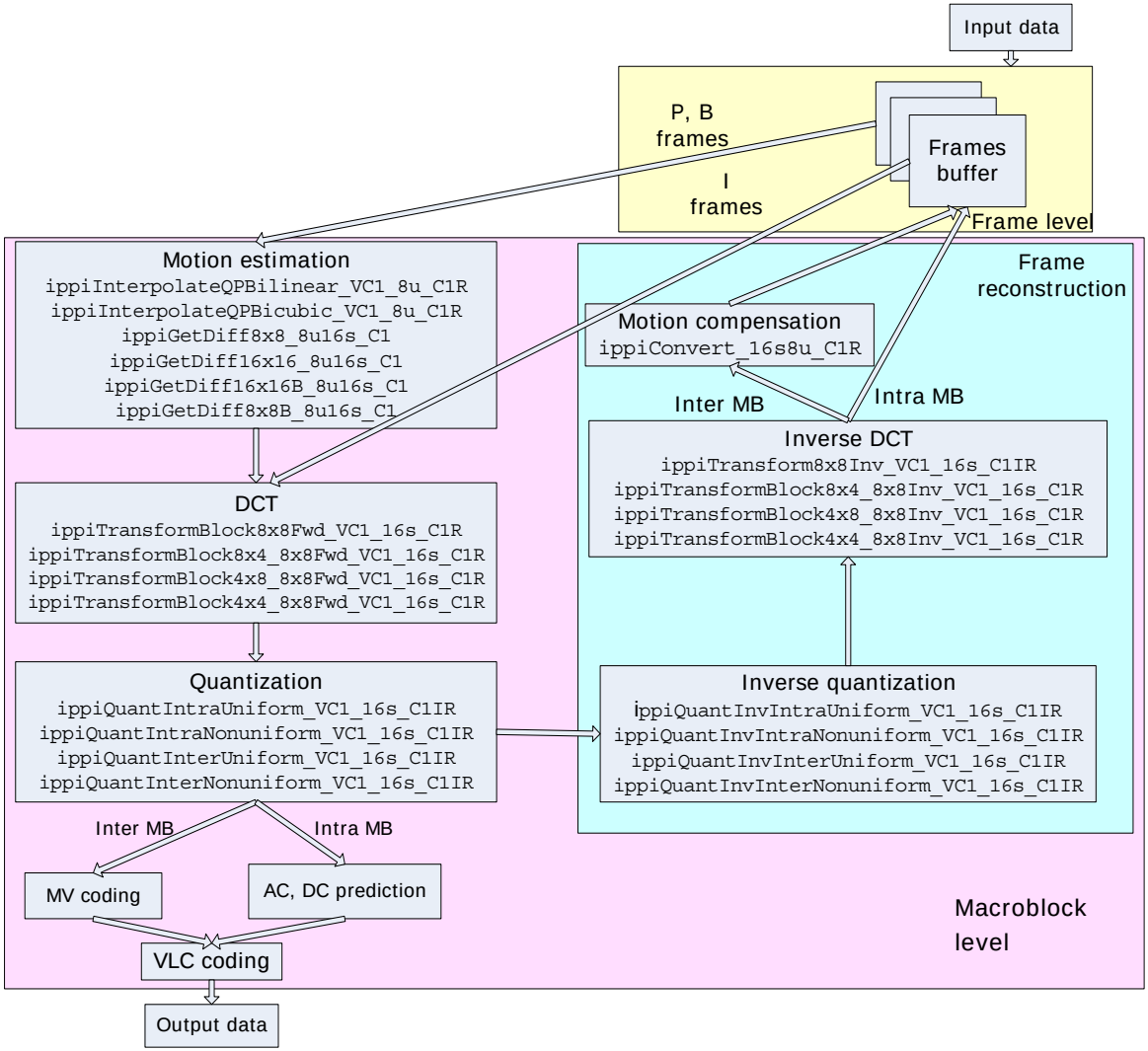


Table 16-32 VC1 Video Encoder Functions

Function Short Name	Description
Forward Transform	
Transform8x8Fwd_VC1, Transform4x8Fwd_VC1, Transform8x4Fwd_VC1, Transform4x4Fwd_VC1	Perform forward transform of blocks of respective size according to VC-1 standard.
Quantization	
QuantIntraUniform_VC1, QuantIntraNonuniform_VC1	Perform uniform/nonuniform quantization of 8X8 intra blocks.
QuantInterUniform_VC1, QuantInterNonuniform_VC1	Perform uniform/nonuniform quantization of inter blocks.

Forward Transform

Transform8x8Fwd_VC1, Transform4x8Fwd_VC1, Transform8x4Fwd_VC1, Transform4x4Fwd_VC1

Perform forward transform of blocks of respective sizes according to VC-1 standard.

Syntax

```
IppStatus ippiTransform8x8Fwd_VC1_16s_C1IR(Ipp16s*  pSrcDst, Ipp32s
    srcDstStep);
IppStatus ippiTransform4x8Fwd_VC1_16s_C1IR(Ipp16s*  pSrcDst, Ipp32s
    srcDstStep);
IppStatus ippiTransform8x4Fwd_VC1_16s_C1IR(Ipp16s*  pSrcDst, Ipp32s
    srcDstStep);
```

```
IppStatus ippiTransform4x4Fwd_VC1_16s_C1IR(Ipp16s*  pSrcDst, Ipp32s
    srcDstStep);
IppStatus ippiTransform8x8Fwd_VC1_16s_C1R(const Ipp16s*  pSrc, Ipp32s srcStep,
    Ipp16s*  pDst, Ipp32s dstStep);
IppStatus ippiTransform4x8Fwd_VC1_16s_C1R(const Ipp16s*  pSrc, Ipp32s srcStep,
    Ipp16s*  pDst, Ipp32s dstStep);
IppStatus ippiTransform8x4Fwd_VC1_16s_C1R(const Ipp16s*  pSrc, Ipp32s srcStep,
    Ipp16s*  pDst, Ipp32s dstStep);
IppStatus ippiTransform4x4Fwd_VC1_16s_C1R(const Ipp16s*  pSrc, Ipp32s srcStep,
    Ipp16s*  pDst, Ipp32s dstStep);
```

Parameters

pSrcDst	Pointer to the source and destination block. All samples of the source block should be in range [-512, 511). After transforming all samples of the destination block are in range [-2048; 2047).
pSrc	Pointer to the source block. All samples of the source block should be in range [-512, 511).
pDst	Pointer to the destination 8x8 block. After transforming all samples of the destination block are in range [-2048; 2047).
srcStep	Distance in bytes between starts of the consecutive lines in the source block.
dstStep	Distance in bytes between starts of the consecutive lines in the destination block.

Description

The functions `ippiTransform8x8Fwd_VC1_16s`, `ippiTransform4x8Fwd_VC1_16s`, `ippiTransform8x4Fwd_VC1_16s`, and `ippiTransform4x4Fwd_VC1_16s` are declared in the `ippvc.h` file. These functions perform forward transform in accordance with Annex A: *Transform Specification* of [SMPTE 421M].

`ippiTransform8x8Fwd_VC1_16s_C1IR` and `ippiTransform8x8Fwd_VC1_16s_C1R` perform transform of 8x8 inter block with 8x8 transformation type or 8x8 intra block.

Example 16-38 ippiTransform8x8Fwd_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-255;255]
After transforming all samples of destination block will be
in the range [-2048; 2047)
*/
```

Example 16-38 ippiTransform8x8Fwd_VC1_16s_C1IR Usage (continued)

```
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform8x8Fwd_VC1_16s_C1IR( pSrcDstData, srcDstStep );
/*
Performs transform of 8x8 inter block with 8x8 transformation type or 8x8 intra
block in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}
```

Example 16-39 ippiTransform8x8Fwd_VC1_16s_C1R Usage

```
{
Ipp16s*  pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-255;255]
*/
Ipp16s*  pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the
range [-2048; 2047)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/

IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform8x8Fwd_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep
);
/*
Performs transform of 8x8 inter block with 8x8 transformation type or 8x8 intra
block in accordance with Annex A "Transform Specification" of SMPTE 421M
*/
```

Example 16-39 ippiTransform8x8Fwd_VC1_16s_C1R Usage (continued)

```
}
```

ippiTransform4x8Fwd_VC1_16s_C1IR and ippiTransform4x8Fwd_VC1_16s_C1R perform transform of 4x8 block into 8x8 inter block with 4x8 transformation type.

Example 16-40 ippiTransform4x8Fwd_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-255;255]
After transforming all samples of destination block will be
in the range [-2048; 2047)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform4x8Fwd_VC1_16s_C1IR(pSrcDstData, srcDstStep);
result = ippiTransform4x8Fwd_VC1_16s_C1IR(pSrcDstData+4, srcDstStep);

/*
Performs transform of 4x8 subblock in 8x8 inter block with 4x8 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}
```

Example 16-41 ippiTransform4x8Fwd_VC1_16s_C1R Usage

```
{
Ipp16s*  pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-255;255]
*/
Ipp16s*  pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be
in the range [-2048; 2047)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes*/
```

Example 16-41 ippiTransform4x8Fwd_VC1_16s_C1R Usage (continued)

```
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/

IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform4x8Fwd_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep
);
result = ippiTransform4x8Fwd_VC1_16s_C1R( pSrcData+4, srcStep, pDstData+4,
dstStep );

/*
Performs transform of 4x8 subblock in 8x8 inter block with 4x8 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}
```

`ippiTransform8x4Fwd_VC1_16s_C1IR` and `ippiTransform8x4Fwd_VC1_16s_C1R` perform transform of 8x4 subblock into 8x8 inter block with 8x4 transformation type.

Example 16-42ippiTransform8x4Fwd_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-255;255]
After transforming all samples of destination block will be
in the range [-2048; 2047)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform8x4Fwd_VC1_16s_C1IR( pSrcDstData, srcDstStep );
result = ippiTransform8x4Fwd_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2),
srcDstStep );

/*
```

Example 16-42ippiTransform8x4Fwd_VC1_16s_C1IR Usage (continued)

```
Performs transform of 8x4 subblock in 8x8 inter block with 8x4 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}
```

Example 16-43 ippiTransform8x4Fwd_VC1_16s_C1R Usage

```
{
Ipp16s*  pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-255;255]
*/
Ipp16s*  pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be in the range [-2048;
2047)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes*/
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/

IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform8x4Fwd_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep
);
result = ippiTransform8x4Fwd_VC1_16s_C1R( pSrcData+(srcStep*2), srcStep,
pDstData+(dstStep*2), dstStep );

/*
Performs transform of 8x4 subblock in 8x8 inter block with 8x4 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}
```

ippiTransform4x4Fwd_VC1_16s_C1IR and ippiTransform4x4Fwd_VC1_16s_C1R perform transform of 4x4 block into 8x8 inter block with 4x4 transformation type.

Example 16-44 ippiTransform4x4Fwd_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
/*
All samples of source block should be in the range [-255;255]
After transforming all samples of destination block will be
 in the range [-2048; 2047)
*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
*/

result = ippiTransform4x4Fwd_VC1_16s_C1IR( pSrcDstData, srcDstStep );
result = ippiTransform4x4Fwd_VC1_16s_C1IR( pSrcDstData+4, srcDstStep );
result = ippiTransform4x4Fwd_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2),
srcDstStep );
result = ippiTransform4x4Fwd_VC1_16s_C1IR( pSrcDstData+(srcDstStep*2)+4,
srcDstStep );

/*
Performs transform of 4x4 subblock in 8x8 inter block with 4x4 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}
```

Example 16-45 ippiTransform4x4Fwd_VC1_16s_C1R Usage

```
{
Ipp16s*  pSrcData = pointer_on_source_block;
/*
All samples of source block should be in the range [-255;255]
*/
Ipp16s*  pDstData = pointer_on_destination_block;
/*
After transforming all samples of destination block will be
in the range [-2048; 2047)
*/
Ipp32s srcStep = source_step; /* Step of the pointer pSrcData (source array) in
bytes*/
```

Example 16-45 ippiTransform4x4Fwd_VC1_16s_C1R Usage (continued)

```
Ipp32s dstStep = destination_step; /* Step of the pointer pDstData (destination
array) in bytes*/

IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcData or pDstData is NULL
*/

result = ippiTransform4x4Fwd_VC1_16s_C1R( pSrcData, srcStep, pDstData, dstStep
);
result = ippiTransform4x4Fwd_VC1_16s_C1R( pSrcData+4, srcStep, pDstData+4,
dstStep );
result = ippiTransform4x4Fwd_VC1_16s_C1R( pSrcData+(srcStep*2), srcStep,
pDstData+(dstStep*2), dstStep);
result = ippiTransform4x4Fwd_VC1_16s_C1R( pSrcData+(srcStep*2)+4, srcStep,
pDstData+(dstStep*2)+4, dstStep );

/*
Performs transform of 4x4 subblock in 8x8 inter block with 4x4 transformation
type in accordance with Annex A "Transform Specification" of SMPTE 421M
*/

}
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when at least one of the specified pointers is NULL.

Quantization

QuantIntraUniform_VC1, QuantIntraNonuniform_VC1

Perform uniform/nonuniform quantization of 8X8 intra blocks.

Syntax

```
IppStatus ippiQuantIntraUniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s
    srcDstStep, Ipp32s doubleQuant );
IppStatus ippiQuantIntraNonuniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s
    srcDstStep, Ipp32s doubleQuant );
```

Parameters

pSrcDst	Pointer to the source and destination block.
srcDstStep	Distance in bytes between starts of the consecutive lines in the source and destination blocks.
doubleQuant	Quant coefficient within the range [2, 62].

Description

The functions `ippiQuantIntraUniform_VC1_16s_C1IR` and `ippiQuantIntraNonuniform_VC1_16s_C1IR` are declared in the `ippvc.h` file. The functions perform quantization of all coefficients in intra block (except DC) according to 8.1.2.8 of [SMPTE 421M].

In the case of uniform quantization:

```
a[i]=a[i]/quant_coef;
```

In the case of nonuniform quantization:

```
a[i]=(a[i]-sign(a[i])*(quant_coef >> 1))/quant_coef;
```

where `a[i]` is the AC coefficient of the intra block, and `quant_coef` is the quant coefficient.

The *doubleQuant* parameter should be equal to 2*MQUANT+HALFQP, where MQUANT and HALFQP are defined in the VC-1 standard. MQUANT should be within the range [1;31], HALFQP should be within the range [0,1] if MQUANT ≤8 and should be equal to 0, if MQUANT > 8.

Example 16-46 ippiQuantIntraUniform_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62];*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
ippStsOutOfRangeErr Indicates an error when quantizer doubleQuant is less than 2
or higher than 62
*/

result = ippiQuantIntraUniform_VC1_16s_C1IR ( pSrcDstData, srcDstStep,
doubleQuant );

/*  Performs uniform quantization process for all coefficients in the intra
block (except DC) according to 8.1.2.8 "Inverse AC Coefficient Quantization"
specification of SMPTE 421M
*/
}
```

Example 16-47 ippiQuantIntraNonuniform_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62];*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
ippStsOutOfRangeErr Indicates an error when quantizer doubleQuant is less than 2
or higher than 62
*/
```

Example 16-47 ippiQuantIntraNonuniform_VC1_16s_C1IR Usage (continued)

```
result = ippiQuantIntraNonuniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant );

/* Performs nonuniform quantization process for all coefficients in the intra
block (except DC) according to 8.1.2.8 "Inverse AC Coefficient Quantization"
specification of SMPTE 421M
*/
}
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when *pSrcDst* is NULL.
ippStsDivByZeroErr	Indicates an error condition when the *doubleQuant* quantizer is equal to zero.

QuantInterUniform_VC1, QuantInterNonuniform_VC1

Perform uniform/nonuniform quantization of inter blocks.

Syntax

```
IppStatus ippiQuantInterUniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s
    srcDstStep, Ipp32s doubleQuant, IppiSize roiSize );
IppStatus ippiQuantInterNonuniform_VC1_16s_C1IR( Ipp16s* pSrcDst, Ipp32s
    srcDstStep, Ipp32s doubleQuant, IppiSize roiSize );
```

Parameters

pSrcDst	Pointer to the source and destination block.
srcDstStep	Distance in bytes between starts of the consecutive lines in the source and destination blocks.
doubleQuant	Quant coefficient; should be an even number within the range [2, 62].
roiSize	Inter block size; should be 8x8, 8x4, 4x8 or 4x4.

Description

The functions `ippiQuantInterUniform_VC1_16s_C1IR` and `ippiQuantInterNonuniform_VC1_16s_C1IR` are declared in the `ippvc.h` file. The functions perform quantization of inter blocks according to 8.1.2.8 of [SMPTE 421M].

In the case of uniform quantization:

```
a[i]=a[i]/quant_coef;
```

In the case of nonuniform quantization:

```
a[i]=(a[i]-sign(a[i])*(quant_coef >> 1))/quant_coef;
```

where `a[i]` is the coefficient of the inter block, and `quant_coef` is the quant coefficient.

The *doubleQuant* parameter should be equal to `2*MQUANT+HALFQP`, where `MQUANT` and `HALFQP` are defined in the VC-1 standard. `MQUANT` should be within the range [1;31], `HALFQP` should be within the range [0,1] if `MQUANT` ≤8 and should be equal to 0, if `MQUANT` > 8.

Example 16-48 ippiQuantInterUniform_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62];*/
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes*/
IppSize roiSize = inter_block_size;
/* It should be:
8x8 (roiSize.width=8, roiSize.height=8)
4x8 (roiSize.width=4, roiSize.height=8)
8x4 (roiSize.width=8, roiSize.height=4)
4x4 (roiSize.width=4, roiSize.height=4)
*/
IppStatus result;

/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
ippStsOutOfRangeErr Indicates an error when quantizer doubleQuant is less than 2
or higher than 62
ippStsSizeErr       Indicates an error when roiSize.height is not equal 4 or 8
or roiSize.width is not equal 4 or 8
*/

result = ippiQuantInterUniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant );
```

Example 16-48 ippiQuantInterUniform_VC1_16s_C1IR Usage (continued)

```
/*  Performs uniform quantization process for inter block according to 8.1.2.8
"Inverse AC Coefficient Quantization" specification of SMPTE 421M
*/
}
```

Example 16-49 ippiQuantInterNonuniform_VC1_16s_C1IR Usage

```
{
Ipp16s*  pSrcDstData = pointer_on_source_and_destination_block;
Ipp32s doubleQuant = quant_coefficient; /* It should be an even number in the
range [2,62]; */
Ipp32s srcDstStep = source_and_destination_step; /* Step of the pointer
pSrcDstData (source and destination array) in bytes */
IppSize roiSize = inter_block_size;
/* It should be:
8x8 (roiSize.width=8, roiSize.height=8)
4x8 (roiSize.width=4, roiSize.height=8)
8x4 (roiSize.width=8, roiSize.height=4)
4x4 (roiSize.width=4, roiSize.height=4)
*/
IppStatus result;
/*
ippStsNoErr         Indicates no error
ippStsNullPtrErr    Indicates an error when pSrcDstData is NULL
ippStsOutOfRangeErr Indicates an error when quantizer doubleQuant is less than 2
or higher than 62
ippStsSizeErr       Indicates an error when roiSize.height is not equal 4 or 8
or roiSize.width is not equal 4 or 8
*/

result = ippiQuantInterNonuniform_VC1_16s_C1IR( pSrcDstData, srcDstStep,
doubleQuant );

/*  Performs nonuniform quantization process for inter block according to
8.1.2.8 "Inverse AC Coefficient Quantization" specification of SMPTE 421M
*/
}
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when *pSrcDst* is NULL.
ippStsDivByZeroErr	Indicates an error condition when the *doubleQuant* quantizer is equal to zero.

Handling of Special Cases

A

Some mathematical functions implemented in Intel IPP are not defined for all possible argument values. This appendix describes how the corresponding Intel IPP image processing functions handle situations when their input arguments fall outside the range of function definition or may lead to ambiguously determined output results.

Table A-1 below summarizes these special cases for different functions and lists result values together with status codes returned by the functions. The status codes ending with `Err` (except for the `ippStsNoErr` status) indicate an error. When an error occurs, the function execution is interrupted. All other status codes indicate that the input argument is outside the range, but the function execution is continued with the corresponding result value.

Table A-1 Special Cases for Intel IPP Image Processing Functions

Function Base Name	Data Type	Case Description	Result Value	Status Code
ippiSqrt	16s	Sqrt (x <0)	0	ippStsSqrtNegArg
	32f	Sqrt (x <0)	NAN_32F	ippStsSqrtNegArg
ippiDiv ippiDiv_Round	8u	Div (0/0)	0	ippStsDivByZero
		Div (x/0)	IPP_MAX_8U	ippStsDivByZero
	16s	Div (0/0)	0	ippStsDivByZero
		Div (x/0), x>0	IPP_MAX_16S	ippStsDivByZero
		Div (x/0), x<0	IPP_MIN_16S	ippStsDivByZero
	32f	Div (0/0)	NAN_32F	ippStsDivByZero
		Div (x/0), x>0	INF_32F	ippStsDivByZero
		Div (x/0), x<0	INF_NEG_32F	ippStsDivByZero

Table A-1 Special Cases for Intel IPP Image Processing Functions (continued)

Function Base Name	Data Type	Case Description	Result Value	Status Code
ippiDiv	16sc	Div (0/0)	0	ppStsDivByZero
		Div (x/0),	0	ippStsDivByZero
	32sc	Div (0/0)	0	ippStsDivByZero
		Div (x/0), x>0	IPP_MAX_32S	ippStsDivByZero
		Div (x/0), x<0	IPP_MIN_32S	ippStsDivByZero
	32fc	Div (0/0)	NAN_32F	ippStsDivByZero
		Div (x/0), x>0	INF_32F	ippStsDivByZero
		Div (x/0), x<0	INF_NEG_32F	ippStsDivByZero
ippiDivC	all	Div(x/const), const=0	-	ippStsDivByZeroErr
ippiLn	8u	Ln (0)	0	ippStsLnZeroArg
	16s	Ln (0)	IPP_MIN_16S	ippStsLnZeroArg
		Ln (x<0)	IPP_MIN_16S	ippStsLnNegArg
	32f	Ln (x<0)	NAN_32F	ippStsLnNegArg
		Ln(x<IPP_MINABS_32F)	INF_NEG_32F	ippStsLnZeroArg
ippiExp	8u	overflow	IPP_MAX_8U	ippStsNoErr
	16s	overflow	IPP_MAX_16S	ippStsNoErr
	32f	overflow	INF_32F	ippStsNoErr

Here x denotes an input value. For the definition of the constants used, see "Image Data Types and Ranges" in chapter 2.

Note that flavors of the same math function operating on different data types may produce unlike results for the equal argument values. However, for a given function and a fixed data type, handling of special cases is the same for all function flavors that have different descriptors in their names. For example, logarithm function `ippiLn` operating on `16s` data treats zero argument values in the same way for all its flavors `ippiLn_16s_C1RSfs`, `ippiLn_16s_C3RSfs`, `ippiLn_16s_C1IRSfs`, and `ippiLn_16s_C3IRSfs`.

Interpolation in Image Geometric Transform Functions

B

This appendix describes the interpolation algorithms used in the geometric transformation functions of the Intel IPP. For more information about each of the geometric transform functions, see Chapter 12, "Image Geometry Transforms".

Overview of Interpolation Modes

In geometric transformations, the grid of input image pixels is not necessarily mapped onto the grid of pixels in the output image. Therefore, to compute the pixel intensities in the output image, the geometric transform functions need to interpolate the intensity values of several input pixels that are mapped to a certain neighborhood of the output pixel.

Geometric transformations can use various interpolation algorithms. When calling the geometric transform functions of the Intel IPP, the application code specifies the interpolation mode (that is, the type of interpolation algorithm) by using the parameter *interpolation*. The library supports the following interpolation modes:

nearest neighbor interpolation (*interpolation* = IPPI_INTER_NN)

linear interpolation (*interpolation* = IPPI_INTER_LINEAR)

cubic interpolation (*interpolation* = IPPI_INTER_CUBIC)

supersampling (*interpolation* = IPPI_INTER_SUPER)

interpolation with Lanczos window function (*interpolation* = IPPI_INTER_LANCZOS)

For certain functions, the above interpolation algorithms can be combined with additional smoothing (antialiasing) of edges to which the original image's borders are transformed. To use this edge smoothing, set the parameter *interpolation* to the bitwise OR of

IPPI_SMOOTH_EDGE and the desired interpolation mode.For example, in order to rotate an image with cubic interpolation and smooth the rotated image's edges, pass to ippiRotate() the following value:

interpolation = IPPI_INTER_CUBIC | IPPI_SMOOTH_EDGE.

Interpolation with edge smoothing option can be used only in those geometric transform functions where this option is explicitly listed in the arguments definition section.

Table B-1 lists the supported interpolation modes for all geometric transform functions that use interpolation.

Table B-1 Interpolation Modes Supported by Image Geometric Transform Functions

Function Base Name	Nearest neighbor	Linear	Cubic	Super-sampling	Lanczos	Edge Smoothing
Resize	x	x	x	x	x	
ResizeCenter	x	x	x	x	x	
ResizeSqrPixel*)	x	x	x	x	x	
GetResizeFract	x	x	x		x	
ResizeShift	x	x	x		x	
Remap	x	x	x			
Rotate	x	x	x			x
RotateCenter	x	x	x			x
Shear	x	x	x			x
WarpAffine	x	x	x			x
WarpAffineBack	x	x	x			
WarpAffineQuad	x	x	x			x
WarpPerspective	x	x	x			x
WarpPerspectiveBack	x	x	x			
WarpPerspectiveQuad	x	x	x			
WarpBilinear	x	x	x			x
WarpBilinearBack	x	x	x			
WarpBilinearQuad	x	x	x			x

*) The function ippiResizeSqrPixel additionally supports three variants of the interpolation with two-parameter cubic filters.

The sections that follow provide more details on each interpolation mode.

Mathematical Notation

In this appendix the following notation is used:

(x_D, y_D)	pixel coordinates in the destination image (integer values);
(x_S, y_S)	the computed coordinates of a point in the source image that is mapped exactly to (x_D, y_D);
$S(x, y)$	pixel value (intensity) in the source image;
$D(x, y)$	pixel value (intensity) in the destination image.

Nearest Neighbor Interpolation

This is the fastest and least accurate interpolation mode. The pixel value in the destination image is set to the value of the source image's pixel closest to the point (x_S, y_S): $D(x_D, y_D) = S(\text{round}(x_S), \text{round}(y_S))$.

To use the nearest neighbor interpolation, set the parameter *interpolation* to IPPI_INTER_NN.

Linear Interpolation

The linear interpolation is slower but more accurate than the nearest neighbor interpolation. On the other hand, it is faster but less accurate than cubic interpolation. The linear interpolation algorithm uses source image intensities at the four pixels (x_{S0}, y_{S0}), (x_{S1}, y_{S0}), (x_{S0}, y_{S1}), (x_{S1}, y_{S1}) which are closest to (x_S, y_S) in the source image:

$x_{S0} = \text{int}(x_S)$, $x_{S1} = x_{S0} + 1$, $y_{S0} = \text{int}(y_S)$, $y_{S1} = y_{S0} + 1$.

First, the intensity values are interpolated along the x-axis to produce two intermediate results I_0 and I_1 (see Figure B-1):

$$I_0 = S(x_S, y_{S0}) = S(x_{S0}, y_{S0})*(x_{S1} - x_S) + S(x_{S1}, y_{S0})*(x_S - x_{S0})$$

$$I_1 = S(x_S, y_{S1}) = S(x_{S0}, y_{S1})*(x_{S1} - x_S) + S(x_{S1}, y_{S1})*(x_S - x_{S0}).$$

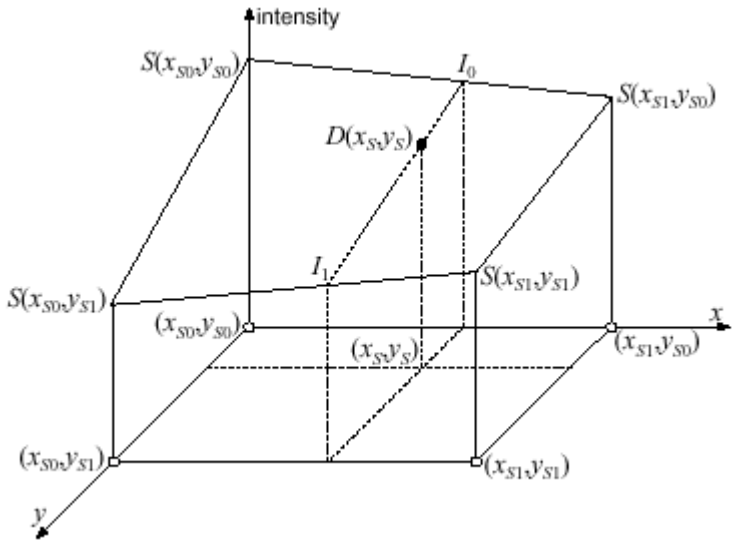
Then, the sought-for intensity $D(x_D, y_D)$ is computed by interpolating the intermediate values I_0 and I_1 along the y-axis:

$$D(x_D, y_D) = I_0*(y_{S1} - y_S) + I_1*(y_S - y_{S0}).$$

To use the linear interpolation, set the parameter *interpolation* to IPPI_INTER_LINEAR.

For images with 8-bit unsigned color channels, the functions ippiWarpAffine, ippiRotate, and ippiShear compute the coordinates (x_S, y_S) with the accuracy 2^{-16} = 1/65536. For images with 16-bit unsigned color channels, these functions compute the coordinates with floating-point precision.

Figure B-1 Linear Interpolation



Cubic Interpolation

The cubic interpolation algorithm (see Figure B-2) uses source image intensities at sixteen pixels in the neighborhood of the point (x_S, y_S) in the source image:

x_{S0} = int(x_S) - 1; x_{S1} = x_{S0} + 1; x_{S2} = x_{S0} + 2; x_{S3} = x_{S0} + 3;

y_{S0} = int(y_S) - 1; y_{S1} = y_{S0} + 1; y_{S2} = y_{S0} + 2; y_{S3} = y_{S0} + 3.

First, for each y_{Sk} the algorithm determines four cubic polynomials $F_0(x)$, $F_1(x)$, $F_2(x)$, and $F_3(x)$:

$F_k(x) = a_k x3 + b_k x2 + c_k x + d_k, \quad 0 \leq k \leq 3$

such that

$$F_k(x_{S0}) = S(x_{S0}, y_{Sk}); \quad F_k(x_{S1}) = S(x_{S1}, y_{Sk}),$$
$$F_k(x_{S2}) = S(x_{S2}, y_{Sk}); \quad F_k(x_{S3}) = S(x_{S3}, y_{Sk}).$$

In Figure B-2, these polynomials are shown by solid curves.

Then, the algorithm determines a cubic polynomial $F_y(y)$ such that

$$F_y(y_{S0}) = F_0(x_S), \; F_y(y_{S1}) = F_1(x_S), \; F_y(y_{S2}) = F_2(x_S), \; F_y(y_{S3}) = F_3(x_S).$$

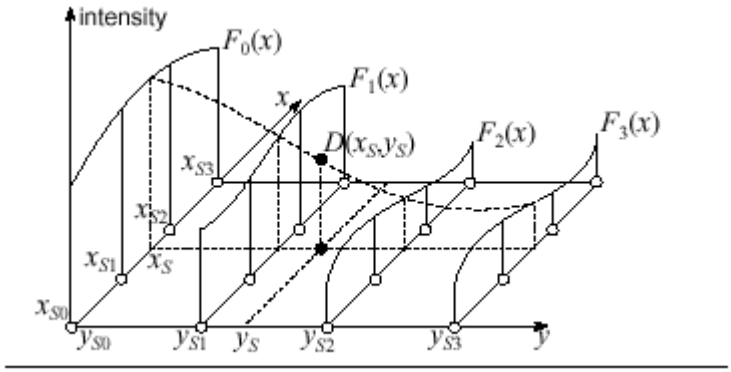
The polynomial $F_y(y)$ is represented by the dashed curve in Figure B-2.

Finally, the sought-for intensity $D(x_D, y_D)$ is set to the value $F_y(y_S)$.

To use the cubic interpolation, set the parameter *interpolation* to `IPPI_INTER_CUBIC`.

For images with 8-bit unsigned color channels, the functions `ippiWarpAffine`, `ippiRotate`, and `ippiShear` compute the coordinates (x_S, y_S) with the accuracy 2^{-16} = 1/65536. For images with 16-bit unsigned color channels, these functions compute the coordinates with floating-point precision.

Figure B-2 Cubic Interpolation



Super Sampling

If the destination image is much smaller than the source image, the above interpolation algorithms may skip some pixels in the source image (that is, these algorithms not necessarily use all source pixels when computing the destination pixels' intensity). In order to use all pixel values of the source image, the `ippiResize` and `ippiResizeCenter` functions support the super-sampling algorithm, which is free of the above drawback.

The super-sampling algorithm is as follows:

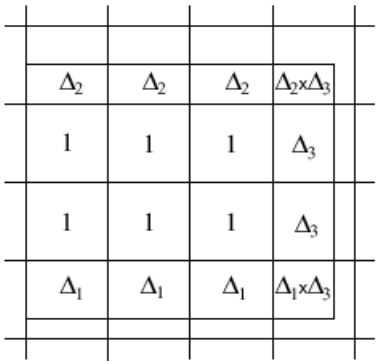
1. Divide the source image's rectangular ROI (or the whole image, if there is no ROI) into equal rectangles, each rectangle corresponding to some pixel in the destination image. Note that each source pixel is represented by a 1x1 square.
2. Compute a weighted sum of source pixel values for all pixels that are in the rectangle or have a non-zero intersection with the rectangle. If a source pixel is fully contained in the rectangle, that pixel's value is taken with weight 1. If the rectangle and the source pixel's square have an intersection of area $a < 1$, that pixel's value is taken with weight a.

   Figure B-3 shows the corresponding weight value for each source pixel intersecting with the rectangle.
3. To compute the pixel value in the destination image, divide this weighted sum by the ratio of the source and destination rectangle areas $S_{Src}/S_{Dst} = 1/xFactor*yFactor$.

   Here $xFactor$, and $yFactor$ are the parameters of the functions that specify the factors by which the x and y dimensions of the source image ROI are changed.

Note that supersampling interpolation can be used only for $xFactor<1$, and $yFactor<1$.

Figure B-3 Supersampling Weights



Lanczos Interpolation

This method is based on the 3-lobed Lanczos window function as the interpolation function.

The interpolation algorithm uses source image intensities at 36 pixels in the neighborhood of the point (x_S, y_S) in the source image:

x_{S0} = int(x_S) - 2; $x_{S1} = x_{S0} + 1$; $x_{S2} = x_{S0} + 2$; $x_{S3} = x_{S0} + 3$; $x_{S3} = x_{S0} + 4$; $x_{S3} = x_{S0} + 5$;

y_{S0} = int(y_S) - 2; $y_{S1} = y_{S0} + 1$; $y_{S2} = y_{S0} + 2$; $y_{S3} = y_{S0} + 3$; $y_{S2} = y_{S0} + 4$; $y_{S2} = y_{S0} + 5$;

First, the intensity values are interpolated along the x-axis to produce six intermediate results I_0 , I_1, ... I_5:

$$I_k = \sum_{i=0}^{5} a_i \cdot S(x_{Si}, y_{Sk}) \quad 0 \le k \le 5.$$

Then the intensity $D(x_D, y_D)$ is computed by interpolating the intermediate values I_k along the y-axis:

$$D(x_D, y_D) = \sum_{k=0}^{5} b_k \cdot I_k$$

Here a_i and b_k are the coefficients defined as

$a_i = L(x_S - x_{Si})$, $b_k = L(y_S - y_{Si})$,

where $L(x)$ is the Lanczos windowed sinc function:

$$L(x) = sinc(x) \cdot Lanczos3(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} \cdot \frac{\sin((\pi x)/3)}{(\pi x)/3} & 0 \le |x| < 3 \\ 0 & 3 \le |x| \end{cases}$$

To use this interpolation, set the parameter *interpolation* to IPPI_INTER_LANCZOS.

Interpolation with Two-Parameter Cubic Filters

The two-parameter family of cubic filters have kernels of the the form:

$$k(x) = \frac{1}{6}\begin{cases} (12-9B-6C)|x|^3 + (-18+12B+6C)|x|^2 + (6-2B) & |x| < 1 \\ (-B-6C)|x|^3 + (6B+30C)|x|^2 + (-12B-48C)|x| + (8B+24C) & 1 \le |x| < 2 \\ 0 & \text{otherwise} \end{cases}$$

where *B* and *C* are two parameters; their variations give different approximation. The Intel IPP function `ippiResizeSqrPixel` uses three members of this family, specifically cubic B-spline (*B*=1, *C*=0), Catmull-Rom spline (*B*=0, *C*=1/2), and special filter with *B*=1/2, *C*=3/10.

Removed Functions

C

This appendix contains Table C-1 that lists the functions that have been removed from the Intel IPP version 5.0. If an application created with the previous versions calls function listed here, then the source code must be modified. The table specifies the corresponding Intel IPP 5.0 functions to substitute for removed functions.

Table C-1 Removed Functions

Removed from 5.0	Substitution or Workaround	Comments
Computer Vision Fucntions		
ippiBlur_32f_C1R	ippiFilterLowpassBorder_32f_C1R	
ippiBlur_8s16s_C1R	ippiConvert_8s32f_C1R, ippiFilterBox_32f_C1R, ippiConvert_32f16s_C1R	
ippiBlur_8u16s_C1R	ippiFilterLowpassBorder_8u_C1R	see also ippiFilterBox
ippiBlurInitAlloc		no structure, useless
ippiConvFree		no structure, useless
ippiDilateStrip_32f_C1R	ippiDilateBorderReplicate_32f_C1R	
ippiDilateStrip_32f_C3R	ippiDilateBorderReplicate_32f_C3R	
ippiDilateStrip_32f_C4R	ippiDilateBorderReplicate_32f_C4R	
ippiDilateStrip_8u_C1R	ippiDilateBorderReplicate_8u_C1R	
ippiDilateStrip_8u_C3R	ippiDilateBorderReplicate_8u_C3R	
ippiDilateStrip_8u_C4R	ippiDilateBorderReplicate_8u_C4R	
ippiDilateStrip_Cross_32f_C1R	ippiDilateBorderReplicate_32f_C1R	
ippiDilateStrip_Cross_32f_C3R	ippiDilateBorderReplicate_32f_C3R	
ippiDilateStrip_Cross_32f_C4R	ippiDilateBorderReplicate_32f_C4R	
ippiDilateStrip_Cross_8u_C1R	ippiDilateBorderReplicate_8u_C1R	
ippiDilateStrip_Cross_8u_C3R	ippiDilateBorderReplicate_8u_C3R	
ippiDilateStrip_Cross_8u_C4R	ippiDilateBorderReplicate_8u_C4R	

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiDilateStrip_Ellipse_32f_C1R	ippiDilateBorderReplicate_32f_C1R	
ippiDilateStrip_Ellipse_32f_C3R	ippiDilateBorderReplicate_32f_C3R	
ippiDilateStrip_Ellipse_32f_C4R	ippiDilateBorderReplicate_32f_C4R	
ippiDilateStrip_Ellipse_8u_C1R	ippiDilateBorderReplicate_8u_C1R	
ippiDilateStrip_Ellipse_8u_C3R	ippiDilateBorderReplicate_8u_C3R	
ippiDilateStrip_Ellipse_8u_C4R	ippiDilateBorderReplicate_8u_C4R	
ippiDilateStrip_Rect_32f_C1R	ippiDilateBorderReplicate_32f_C1R	
ippiDilateStrip_Rect_32f_C3R	ippiDilateBorderReplicate_32f_C3R	
ippiDilateStrip_Rect_32f_C4R	ippiDilateBorderReplicate_32f_C4R	
ippiDilateStrip_Rect_8u_C1R	ippiDilateBorderReplicate_8u_C1R	
ippiDilateStrip_Rect_8u_C3R	ippiDilateBorderReplicate_8u_C3R	
ippiDilateStrip_Rect_8u_C4R	ippiDilateBorderReplicate_8u_C4R	
ippiEigenValsVecs_8s32f_C1R		useless
ippiEigenValsVecsGetSize	ippiEigenValsVecsGetBufferSize_ 8u32f_C1R, ippiEigenValsVecsGetBufferSize_ 32f_C1R	
ippiErodeStrip_32f_C1R	ippiErodeBorderReplicate_32f_C1R	
ippiErodeStrip_32f_C3R	ippiErodeBorderReplicate_32f_C3R	
ippiErodeStrip_32f_C4R	ippiErodeBorderReplicate_32f_C4R	
ippiErodeStrip_8u_C1R	ippiErodeBorderReplicate_8u_C1R	
ippiErodeStrip_8u_C3R	ippiErodeBorderReplicate_8u_C3R	
ippiErodeStrip_8u_C4R	ippiErodeBorderReplicate_8u_C4R	
ippiErodeStrip_Cross_32f_C1R	ippiErodeBorderReplicate_32f_C1R	
ippiErodeStrip_Cross_32f_C3R	ippiErodeBorderReplicate_32f_C3R	
ippiErodeStrip_Cross_32f_C4R	ippiErodeBorderReplicate_32f_C4R	
ippiErodeStrip_Cross_8u_C1R	ippiErodeBorderReplicate_8u_C1R	
ippiErodeStrip_Cross_8u_C3R	ippiErodeBorderReplicate_8u_C3R	
ippiErodeStrip_Cross_8u_C4R	ippiErodeBorderReplicate_8u_C4R	
ippiErodeStrip_Ellipse_32f_C1R	ippiErodeBorderReplicate_32f_C1R	
ippiErodeStrip_Ellipse_32f_C3R	ippiErodeBorderReplicate_32f_C3R	
ippiErodeStrip_Ellipse_32f_C4R	ippiErodeBorderReplicate_32f_C4R	
ippiErodeStrip_Ellipse_8u_C1R	ippiErodeBorderReplicate_8u_C1R	
ippiErodeStrip_Ellipse_8u_C3R	ippiErodeBorderReplicate_8u_C3R	
ippiErodeStrip_Ellipse_8u_C4R	ippiErodeBorderReplicate_8u_C4R	

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiErodeStrip_Rect_32f_C1R	ippiErodeBorderReplicate_32f_C1R	
ippiErodeStrip_Rect_32f_C3R	ippiErodeBorderReplicate_32f_C3R	
ippiErodeStrip_Rect_32f_C4R	ippiErodeBorderReplicate_32f_C4R	
ippiErodeStrip_Rect_8u_C1R	ippiErodeBorderReplicate_8u_C1R	
ippiErodeStrip_Rect_8u_C3R	ippiErodeBorderReplicate_8u_C3R	
ippiErodeStrip_Rect_8u_C4R	ippiErodeBorderReplicate_8u_C4R	
ippiLaplace_32f_C1R	ippiFilterLaplacianBorder_32f_C1R	
ippiLaplace_8s16s_C1R		
ippiLaplace_8u16s_C1R	ippiFilterLaplacianBorder_8u16s_C1R	
ippiLaplaceInitAlloc		no structure, useless
ippiMatchTemplate_Coeff_32f_C1R		useless
ippiMatchTemplate_Coeff_8s32f_C1R		useless
ippiMatchTemplate_Coeff_8u32f_C1R		useless
ippiMatchTemplate_CoeffNormed_ 32f_C1R	ippiCrossCorrValid_NormLevel_ 32f_C1R	
ippiMatchTemplate_CoeffNormed_ 8s32f_C1R	ippiCrossCorrValid_NormLevel_ 8s32f_C1R	
ippiMatchTemplate_CoeffNormed_ 8u32f_C1R	ippiCrossCorrValid_NormLevel_ 8u32f_C1R	
ippiMatchTemplate_Corr_32f_C1R		useless
ippiMatchTemplate_Corr_8s32f_C1R		useless
ippiMatchTemplate_Corr_8u32f_C1R		useless
ippiMatchTemplate_CorrNormed_32f_C1R	ippiCrossCorrValid_Norm_32f_C1R	
ippiMatchTemplate_CorrNormed_ 8s32f_C1R	ippiCrossCorrValid_Norm_8s32f_C1R	
ippiMatchTemplate_CorrNormed_ 8u32f_C1R	ippiCrossCorrValid_Norm_8u32f_C1R	
ippiMatchTemplate_SqDiff_32f_C1R		useless
ippiMatchTemplate_SqDiff_8s32f_C1R		useless
ippiMatchTemplate_SqDiff_8u32f_C1R		useless
ippiMatchTemplate_SqDiffNormed_ 32f_C1R	ippiSqrDistanceValid_Norm_32f_C1R	
ippiMatchTemplate_SqDiffNormed_ 8s32f_C1R	ippiSqrDistanceValid_Norm_8s32f_ C1R	
ippiMatchTemplate_SqDiffNormed_ 8u32f_C1R	ippiSqrDistanceValid_Norm_8u32f_ C1R	

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiMatchTemplateGetBufSize_Coeff		useless
ippiMatchTemplateGetBufSize_CoeffNormed		useless
ippiMatchTemplateGetBufSize_Corr		useless
ippiMatchTemplateGetBufSize_CorrNormed		useless
ippiMatchTemplateGetBufSize_SqDiff		useless
ippiMatchTemplateGetBufSize_ SqDiffNormed		useless
ippiMinEigenVal_8s32f_C1R		
ippiMinEigenValGetSize	ippiMinEigenValGetBufferSize_ 8u32f_C1R, or ippiMinEigenValGetBufferSize_ 32f_C1R	
ippiMorphologyInitAlloc		no structure, useless
ippiScharr_Dx_32f_C1R	ippiFilterScharrHorizBorder_32f_C1R	
ippiScharr_Dx_8s16s_C1R		
ippiScharr_Dx_8u16s_C1R	ippiFilterScharrHorizBorder_8u16s_C1R	
ippiScharr_Dy_32f_C1R	ippiFilterScharrVertBorder_32f_C1R	
ippiScharr_Dy_8s16s_C1R		
ippiScharr_Dy_8u16s_C1R	ippiFilterScharrVertBorder_ 8u16s_C1R	
ippiSobel_32f_C1R	ippiFilterSobelHorizBorder_32f_C1R ippiFilterSobelVertBorder_32f_C1R	
ippiSobel_8s16s_C1R		
ippiSobel_8u16s_C1R	ippiFilterSobelHorizBorder_8u16s_ C1R, or ippiFilterSobelHorizBorder_ 8u16s_C1R	
ippiSobel3x3_D2x_32f_C1R	ippiFilterSobelHorizSecondBorder_ 32f_C1R	
ippiSobel3x3_D2x_8s16s_C1R		
ippiSobel3x3_D2x_8u16s_C1R	ippiFilterSobelHorizSecondBorder_ 8u16s_C1R	
ippiSobel3x3_D2y_32f_C1R	ippiFilterSobelVertSecondBorder_ 32f_C1R	
ippiSobel3x3_D2y_8s16s_C1R		
ippiSobel3x3_D2y_8u16s_C1R	ippiFilterSobelVertSecondBorder_ 8u16s_C1R	
ippiSobel3x3_Dx_32f_C1R	ippiFilterSobelHorizBorder_32f_C1R	

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiSobel3x3_Dx_8s16s_C1R		
ippiSobel3x3_Dx_8u16s_C1R	ippiFilterSobelHorizBorder_ 8u16s_C1R	
ippiSobel3x3_Dy_32f_C1R	ippiFilterSobelVertBorder_32f_C1R	
ippiSobel3x3_Dy_8s16s_C1R		
ippiSobel3x3_Dy_8u16s_C1R	ippiFilterSobelVertBorder_8u16s_C1R	
ippiSobel3x3_DxDy_32f_C1R	ippiFilterSobelCrossBorder_32f_C1R	
ippiSobel3x3_DxDy_8s16s_C1R		
ippiSobel3x3_DxDy_8u16s_C1R	ippiFilterSobelCrossBorder_ 8u16s_C1R	
ippiSobelInitAlloc		no structure, useless
Color Conversion Functions		
ippiJoin420_8u_P2C2R	ippiYCbCr420ToYCbCr422_8u_P2C2R	renamed
ippiJoin420_Filter_8u_P2C2R	ippiYCbCr420ToYCbCr422_Filter_ 8u_P2C2R	renamed
ippiJoin422_8u_P3C2R	ippiYCbCr422_8u_P3C2R	renamed
ippiSplit420_8u_P2P3R	ippiYCbCr420ToYCrCb420_8u_P2P3R	renamed
ippiSplit420_Filter_8u_P2P3R	ippiYCbCr420ToYCrCb420_Filter_ 8u_P2P3R	renamed
ippiSplit422_8u_C2P3R	ippiYCbCr422_8u_C2P3R	renamed
ippiUYToYU422_8u_C2P2R	ippiCbYCr422ToYCbCr420_8u_C2P2R	renamed
ippiUYToYU422_8u_C2R	ippiCbYCr422ToYCbCr422_8u_C2R	renamed
ippiUYToYV422_8u_C2P3R	iippiCbYCr422ToYCrCb420_8u_C2P3R	renamed
ippiYCbCr411ToYCbCr411_8u_P2P3R	ippiYCbCr411_8u_P2P3R	renamed
ippiYCbCr411ToYCbCr411_8u_P3P2R	ippiYCbCr411_8u_P3P2R	renamed
ippiYCbCr420ToYCbCr420_8u_P2P3R	ippiYCbCr420_8u_P2P3R	renamed
ippiYCbCr420ToYCbCr420_8u_P3P2R	ippiYCbCr420_8u_P3P2R	renamed
ippiYCbCr420ToYCbCr422Filter_8u_P2P3R	ippiYCbCr420ToYCbCr422_Filter_ 8u_P2P3R	renamed
ippiYCbCr420ToYCbCr422Filter_8u_P3R	ippiYCbCr420ToYCbCr422_Filter_ 8u_P3R	renamed
ippiYCbCr422ToYCbCr422_8u_C2P3R	ippiYCbCr422_8u_C2P3R	renamed
ippiYCrCb420ToYCbCr422Filter_8u_P3R	ippiYCrCb420ToYCbCr422_Filter_ 8u_P3R	renamed
ippiYUToUY420_8u_P2C2R	ippiYCbCr420ToCbYCr422_8u_P2C2R	renamed
ippiYUToUY422_8u_C2R	ippiYCbCr422ToCbYCr422_8u_C2R	renamed

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiYUToYU422_8u_C2P2R	ippiYCbCr422ToYCbCr420_8u_C2P2R	renamed
ippiYUToYV422_8u_C2P3R	ippiYCbCr422ToYCrCb420_8u_C2P3R	renamed
ippiYVToUY420_8u_P3C2R	ippiYCrCb420ToCbYCr422_8u_P3C2R	renamed
ippiYVToYU420_8u_P3C2R	ippiYCrCb420ToYCbCr422_8u_P3C2R	renamed
ippiYVToYU420_8u_P3P2R	ippiYCrCb420ToYCbCr420_8u_P3P2R	renamed
ippiYCbCr420ToYCbCr420_8u_P2P3R	ippiYCbCr420_8u_P2P3R	renamed
ippiYCbCr420ToYCbCr420_8u_P3P2R	ippiYCbCr420_8u_P3P2R	renamed
ippiYCbCr420ToYCbCr422Filter_8u_P2P3R	ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R	renamed
ippiYCbCr420ToYCbCr422Filter_8u_P3R	ippiYCbCr420ToYCbCr422_Filter_8u_P3R	renamed
Image Processing Functions		
ippiDCTInv_8x8_16s8u	ippiDCT8x8Inv_16s8u_C1R	
ippiDrawText_8u_C3R		empty function
ippiMalloc_16s_P3	ippiMalloc_16s_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_16u_P3	ippiMalloc_16u_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_32f_P3	ippiMalloc_32f_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_32fc_P3	ippiMalloc_32fc_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_32s_P3	ippiMalloc_32s_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_32sc_P3	ippiMalloc_32sc_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_8u_P3	ippiMalloc_8u_C1	3 calls with len, or 1 call with 3*len
Video Coding Functions		
ippiAverageBlock_MPEG4_8u		useless
ippiAverageBlock_MPEG4_8u_I	ippiAverage8x8_8u_C1IR	
ippiAverageMB_MPEG4_8u		useless
ippiAverageMB_MPEG4_8u_I	ippiAverage16x16_8u_C1IR	
ippiBlockMatch_Integer_16x16_MVFAST		functionality on codec level
ippiBlockMatch_Integer_16x16_SEA		functionality on codec level

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiComputeChroma4MV_MPEG4		functionality on codec level
ippiComputeChromaMV_MPEG4		functionality on codec level
ippiComputeTextureErrorBlock_8u16s	ippiSub8x8_8u16s_C1R	
ippiComputeTextureErrorBlock_SAD_8u16s	ippiSubSAD8x8_8u16s_C1R	
ippiCopyApproxHBlock_H263_8u	ippiCopy8x8HP_8u_C1R	
ippiCopyApproxHMB_H263_8u	ippiCopy16x16HP_8u_C1R	
ippiCopyApproxHVBlock_H263_8u	ippiCopy8x8HP_8u_C1R	
ippiCopyApproxHVMB_H263_8u	ippiCopy16x16HP_8u_C1R	
ippiCopyApproxVBlock_H263_8u	ippiCopy8x8HP_8u_C1R	
ippiCopyApproxVMB_H263_8u	ippiCopy16x16HP_8u_C1R	
ippiCopyBlock_16x16_8u	ippiCopy16x16_8u_C1R	
ippiCopyBlock_8x8_8u	ippiCopy8x8_8u_C1R	
ippiCopyBlock_H263_8u	ippiCopy8x8_8u_C1R	
ippiCopyBlockHalfpel_MPEG4_8u	ippiCopy8x8HP_8u_C1R	
ippiCopyMB_H263_8u	ippiCopy16x16_8u_C1R	
ippiCopyMBHalfpel_MPEG4_8u	ippiCopy16x16HP_8u_C1R	
ippiDecodeBlockCoef_AdvIntra_H263_1u8u	ippiDecodeCoefIntra_H263_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_Inter_H263_1u16s	ippiDecodeCoefInter_H263_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_Inter_MPEG4_1u16s	ippiDecodeCoefInter_MPEG4_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_Intra_H263_1u8u	ippiDecodeDCIntra_H263_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_Intra_MPEG4_1u8u	ippiDecodeDCIntra_MPEG4_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_IntraDCOnly_ H263_1u8u	ippiDecodeCoefIntra_H263_1u16s; QuantInv; DCTInv	set of functions

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiDecodeBlockCoef_IntraDCOnly_ MPEG4_1u8u	ippiDecodeACIntra_MPEG4_1u16s; QuantInv: DCTInv	set of functions
ippiDecodeCAEInterH_MPEG4_1u8u		functionality on codec level
ippiDecodeCAEInterV_MPEG4_1u8u		functionality on codec level
ippiDecodeCAEIntraH_MPEG4_1u8u		functionality on codec level
ippiDecodeCAEIntraV_MPEG4_1u8u		functionality on codec level
ippiDecodeCBPY_H263		functionality on codec level
ippiDecodeMCBPCInter_H263		functionality on codec level
ippiDecodeMCBPCIntra_H263		functionality on codec level
ippiDecodeMODB_H263		functionality on codec level
ippiDecodeMV_BVOP_Backward_MPEG4		functionality on codec level
ippiDecodeMV_BVOP_Direct_MPEG4		functionality on codec level
ippiDecodeMV_BVOP_DirectSkip_MPEG4		functionality on codec level
ippiDecodeMV_BVOP_Forward_MPEG4		functionality on codec level
ippiDecodeMV_BVOP_Interpolate_MPEG4		functionality on codec level
ippiDecodeMV_H263		functionality on codec level
ippiDecodeMV_TopBorder_H263		functionality on codec level
ippiDecodeMVS_MPEG4		functionality on codec level
ippiDecodePadMV_PVOP_MPEG4		functionality on codec level
ippiDecodeVLC_IntraDCVLC_MPEG4_1u16s	ippiDecodeDCIntra_MPEG4_1u16s	
ippiDecodeVLCZigzag_Inter_MPEG4_1u16s	ippiDecodeCoefInter_MPEG4_1u16s	

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiDecodeVLCZigzag_IntraACVLC_MPEG4_1u16s	ippiDecodeACIntra_MPEG4_1u16s	
ippiDecodeVLCZigzag_IntraDCVLC_MPEG4_1u16s	ippiDecodeDCIntra_MPEG4_1u16s	
ippiEncode_ACVLC_H263_16s1u	ippiEncodeCoeffIntra_H263_16s1u	
ippiEncode_InterVLC_MPEG4_16s1u	ippiEncodeCoefInter_MPEG4_1u16s	
ippiEncode_IntraACVLC_MPEG4_16s1u	ippiEncodeACIntra_MPEG4_1u16s	
ippiEncode_IntraDCVLC_H263_16s1u	ippiEncodeDCIntra_H263_16s1u	
ippiEncode_IntraDCVLC_MPEG4_16s1u	ippiEncodeDCIntra_MPEG4_1u16s	
ippiEncodeMV_MPEG4_8u16s		functionality on codec level
ippiEncodeVLCZigzag_Inter_MPEG4_16s1u	ippiEncodeCoefInter_MPEG4_1u16s	
ippiEncodeVLCZigzag_IntraACVLC_MPEG4_16s1u	ippiEncodeACIntra_MPEG4_1u16s	
ippiEncodeVLCZigzag_IntraDCVLC_MPEG4_16s1u	ippiEncodeACIntra_MPEG4_1u16s	
ippiExpandFrame_H263_8u		functionality on codec level
ippiFilterDeblocking_HorEdge_H263_8u_I	ippiFilterDeblockingHorEdge_H263_8u_C1IR	
ippiFilterDeblocking_HorEdge_MPEG4_8u_I	ippiFilterDeblockingHorEdge_MPEG4_8u_C1IR	
ippiFilterDeblocking_VerEdge_H263_8u_I	ippiFilterDeblockingVerEdge_H263_8u_C1IR	
ippiFilterDeblocking_VerEdge_MPEG4_8u_I	ippiFilterDeblockingVerEdge_MPEG4_8u_C1IR	
ippiFilterDeringingSmoothBlock_MPEG4_8u	ippiFilterDeringingSmoothBlock_MPEG4_8u_C1R	
ippiFilterDeringingThresholdMB_MPEG4_8u	ippiFilterDeringingThresholdMB_MPEG4_8u_P3R	
ippiFindMVpred_MPEG4		functionality on codec level
ippiLimitMVToRect_MPEG4		functionality on codec level
ippiMCBlock_RoundOff_8u	ippiCopy8x8HP_8u_C1R	
ippiMCBlock_RoundOn_8u	ippiCopy8x8HP_8u_C1R	
ippiMotionEstimation_16x16_MVFAST		functionality on codec level

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiMotionEstimation_16x16_SEA		functionality on codec level
ippiOBMCHalfpel_MPEG4_8u	ippiOBMC8x8HP_MPEG4_8u_C1R	
ippiPadCurrent_16x16_MPEG4_8u_I		functionality on codec level
ippiPadCurrent_8x8_MPEG4_8u_I		functionality on codec level
ippiPadMBGray_MPEG4_8u		functionality on codec level
ippiPadMBHorizontal_MPEG4_8u		functionality on codec level
ippiPadMBOpaque_MPEG4_8u_P4R		functionality on codec level
ippiPadMBPartial_MPEG4_8u_P4R		functionality on codec level
ippiPadMBTransparent_MPEG4_8u_P4R		functionality on codec level
ippiPadMBVertical_MPEG4_8u		functionality on codec level
ippiPadMV_MPEG4		functionality on codec level
ippiPredictBlock_OBMC_8u	ippiOBMC8x8HP_MPEG4_8u_C1R	
ippiPredictReconCoefIntra_MPEG4_16s		functionality on codec level
ippiQuant_H263_C1I	ippiQuantInter_H263_16s_C1I	
ippiQuant_MPEG4_16s_C1I	ippiQuantInter_H263_16s_C1I	
ippiQuantInter_MPEG4_16s_I	ippiQuantInter_H263_16s_C1I	
ippiQuantIntra_H263_C1I	ippiQuantIntra_H263_16s_C1I	
ippiQuantIntra_MPEG4_16s_I	ippiQuantIntra_H263_16s_C1I	
ippiQuantInv_H263_C1I	ippiQuantInvInter_H263_16s_C1I	
ippiQuantInv_MPEG4_16s_C1I	ippiQuantInvInter_MPEG4_16s_C1I	
ippiQuantInvInter_Compact_H263_16s_I	ippiQuantInvInter_H263_16s_C1I	
ippiQuantInvInter_MPEG4_16s_I	ippiQuantInvInter_MPEG4_16s_C1I	
ippiQuantInvInterFirst_MPEG4_16s_I	ippiQuantInvInter_MPEG4_16s_C1I	
ippiQuantInvInterSecond_MPEG4_16s_I	ippiQuantInvInter_MPEG4_16s_C1I	
ippiQuantInvIntra_Compact_H263_16s_I	ippiQuantInvIntra_H263_16s_C1I	
ippiQuantInvIntra_H263_C1I	ippiQuantInvIntra_H263_16s_C1I	

Table C-1 Removed Functions (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiQuantInvIntra_MPEG4_16s_I	ippiQuantInvIntra_MPEG4_16s_C1I	
ippiQuantInvIntraFirst_MPEG4_16s_I	ippiQuantInvIntra_MPEG4_16s_C1I	
ippiQuantInvIntraSecond_MPEG4_16s_I	ippiQuantInvIntra_MPEG4_16s_C1I	
ippiReconBlock_8x8	ippiAdd8x8HP_16s8u_C1RS	
ippiReconBlock_H263	ippiAdd8x8HP_16s8u_C1RS	
ippiReconBlock_H263_I	ippiAdd8x8_16s8u_C1IRS	
ippiReconBlockHalfpel_MPEG4_8u	ippiAdd8x8HP_16s8u_C1R	
ippiReconMB_H263	ippiMC16x16_8u_C1	
ippiReconMB_H263_I	ippiMC16x16_8u_C1	
ippiSumNorm_VOP_MPEG4_8u16u		useless
ippiTransRecBlockCoef_inter_MPEG4	DCT and Quant functions	set of functions
ippiTransRecBlockCoef_intra_MPEG4	DCT and Quant functions	set of functions
ippiUpdateQP_MPEG4		functionality on codec level
ippiUpdateQuant_MQ_H263_1u32s_I		functionality on codec level
ippiUpdateRCModel_MPEG4		functionality on codec level
ippiVCHuffmanDecodeOne_1u32s	ippiDecodeHuffmanOne_1u32s	
ippiVCHuffmanFree_32s	ippiHuffmanTableFree_32s	
ippiVCHuffmanInitAlloc_32s	ippiHuffmanTableInitAlloc_32s	
ippiVCHuffmanInitAllocRL_32s	ippiHuffmanRunLevelTableInitAlloc_32s	
ippiZigzagInv_Horizontal_16s	ippiScanInv_16s_C1I	additional parameters
ippiZigzagInv_Vertical_16s	ippiScanInv_16s_C1I	additional parameters
ippiZigzagInvClassical_Compact_16s	ippiScanInv_16s_C1I	additional parameters
ippiZigzagInvHorizontal_Compact_16s	ippiScanInv_16s_C1I	additional parameters
ippiZigzagInvVertical_Compact_16s	ippiScanInv_16s_C1I	additional parameters

Bibliography

This bibliography provides a list of publications that might be helpful to you in using the image processing subset of Intel IPP. This list is not complete; it serves only as a starting point. The books [Rog85], [Rog90], and [Foley90] are good resources of information on image processing and computer graphics, with mathematical formulas and code examples.

[Aka96] A.Akansu, M.Smith (editors). *Subband and Wavelet transform. Design and Applications*, Kluwer Academic Publishers, 1996.

[AP922] *A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX™ Instructions*, Application Note AP922, Intel Corp. Order number 742474, 1999.

[APMF] *Fast Algorithms for Median Filtering*, Application Note, Intel Corp. Document number 79835, 2001.

[AVS] GB/T 200090.2-2006. China Standard. Information Technology*. Coding of Audio-Visual Objects - Part 2: Visual* (02/2006).

[Bert01] M.Bertalmio, A.L.Bertozzi, G.Sapiro. *Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting.* Proc. ICCV 2001, pp.1335-1362, 2001.

[Bor86] G. Borgefors. *Distance Transformations in Digital Images*. Computer Vision, Graphics, and Image Processing 34, 1986.

[Bou99] J-Y.Bouguet. *Pyramidal Implementation of the Lucas-Kanade Feature Tracker.* OpenCV Documentation, Microprocessor Research Lab, Intel Corporation, 1999.

[Canny86] J. Canny. *A Computational Approach to Edge Detection*, IEEE Trans. on Pattern Analysis and Machine Intelligence 8(6), 1986.

[Davis97] J.Davis and Bobick. *The Representation and Recognition of Action Using Temporal Templates*. MIT Media Lab Technical Report 402, 1997.

[Davis99] J.Davis and G.Bradski. *Real-Time Motion Template Gradients Using Intel® Computer Vision Library*. IEEE ICCV'99 FRAME-RATE WORKSHOP, 1999.

[DICOM] Digital Imaging and Communications in Medicine (DICOM), *published by* National Electrical Manufacturers Association, 2003.

[Feig92] E. Feig and S. Winograd. *Fast Algorithms for the Discrete Cosine Transform*, IEEE Trans. Signal Processing, vol. 40, no. 9, pp. 2174-2193, Sep. 1992.

[Felz04] P. Felzenszwalb, D. Hattenlocher. *Distance Transforms of Sampled Functions*. Cornell Computing and Information Science Technical Report TR2004-1963, September 2004.

[Foley90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics — Principles and Practice,* Second Edition. Addison Wesley, 1990.

[Gon93] R.C. Gonzalez and R.E.Wood. *Digital Image Processing*. Prentice Hall, 1993.

[Hir05] K. Hirakawa, T.W. Parks, *Adaptive Homogeneity-Directed Demosaicing Algorithm*, IEEE Trans. Image Processing, March, 2005.

[IEC61834] IEC 61834. International Electrochemical Commission. *Specifications of Consumer-Use Digital VCRs using 6.3 mm magnetic tape* (*the "Blue Book" DV specification*).

[IEEE] *IEEE Standard Specifications for the Implementations of 8X8 Inverse Discrete Cosine Transform*, IEEE # 1180 (1997).

[IPL] *Intel Image Processing Library Reference Manual.* Intel Corp. Order number 663791, 1999.

[ISO11172] ISOCD 11172. Information Technology. *Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s* (1993).

[ISO13818] ISO/IEC 13818. Information Technology. *Coding of Moving Pictures and Associated Audio Information*, (11/94).

[ISO14496] International Standard ISO/IEC 14496-2. Information Technology*. Coding of Audio-Visual Objects - Part 2: Visual.*

[ISO14496A] ISO/IEC 14496-2:1999/Amd.1:2000(E) . Information Technology. *Coding of Audio-Visual Objects. Part2:Visual. Amendment 1: Visual Extensions* (01/00).

[ISO10918] International Standard ISO/IEC 10918-1*, Digital Compression and Coding of Continuous Tone Still Images,* Appendix A – *Requirements and guidelines*.

[ISO15444] International Standard ISO/IEC 15444-1, *JPEG 2000 Image coding system,* part 1*: Core coding system.*

[ITUH261] ITU-T Recommendation H.261. *Line transmission of non-telephone signals. Video codec for audiovisual services at p x 64 kbits* (03/93).

[ITUH263] ITU-T Recommendation H.263. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Video coding for low bit rate communication* (02/98).

[ITUH264] ITU-T Recommendation H.264. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Advanced video coding for generic audiovisual services* (ITU-T Rec. H.264 | ISO/IEC 14496-10:2005)(03/05).

[ITU709] ITU-R Recommendation BT.709, *Basic Parameter Values for the HDTV Standard for the Studio and International Programme Exchange* [formerly CCIR Rec.709] ITU, Geneva, Switzerland, 1990.

[Jae95] Jaehne, Bernd. *Digital Image Processing,* 3rd Edition, Springer-Verlag, Berlin, 1995.

[Jae97] Jaehne, Bernd. *Practical Handbook on Image Processing for Scientific Applications*, CRC Press, New York, 1997.

[Jack01] Jack, Keith. *Video Demystified: a Handbook for the Digital Engineer,* LLH Technology Publishing, 3rd Edition, 2001.

[JVTG050] JVT-G050. *ITU-T Recommendation and Final Draft International Standard of Joint Video Specification* (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC) (03/03).

[Kadew01] P.KadewTraKuPong, R.Bowden. *An Improved Adaptive Background Mixture Model for Real-Time Tracking with Shadow Detection.* Proc. 2nd European Workshop on Advanced Video-Based Surveilolance Systems, 2001.

[Lein02] R.Leinhart, J.Maydt. *An Extended Set of Haar-like Features for Rapid Object Detection.* IEEE ICIP, vol.1, pp. 900-903, Sep. 2002.

[Li03] L.Li, W.Huang, I.Gu, Q.Tian. *Foreground Object Detection from Videos Containing Complex Background.* Proc.ACM Multimedia Conference, Berkley, 2003.

[Lim90] Jae S.Lim. *Two-Dimensional Signal and Image Processing,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[Lotufo00] R.Lotufo, A.Falcao. *The Ordered Queue and the Optimality of the Watershed Algorithm.* Mathematical Morphology and its Applications to Image and Signal Processing, vol.18, pp.341-350. Kluwer Academic Publishers. Palo Alto, USA, June 2000.

[Lowe04] D.G.Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal of Computer Vision, vol.60, No.2, pp. 91-110, 2004.

[Malvar03] H.S.Malvar, G.J.Sullivan. *Transform, Scaling & Color Space Impact of Professional Extensions*, ISO/IEC JTC/SC29/WG11 and ITU-T SG16 Q.6 Document JVT-H031, Geneva, May 2003.

[Malvar03-1] H.S.Malvar, G.J.Sullivan. *YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range*, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Document No.JVT-1014r3, July 2003.

[Meyer92] F.Meyer. *Color Image Segmentation*, 4th International Conference on Image Processing and its Applications, p.4, Maastricht, April 1993.

[Meyer94] F.Meyer. *Topographic Distance and Watershed Lines.* Signal Processing, No.38, pp.113-125, 1994.

[Myler93] H.Myler, A.Weeks. *Computer Imaging Recipes in C*, Prentice Hall, 1993.

[Otsu79] N. Otsu. *A Threshold Selection Method From Gray Level Histograms*. IEEE Transactions on Systems, Man, and Cybernetics, vol.9, No.1, January 1979, pp. 62-66.

[Puetter05] R.C.Puetter, T.R.Gosnell, and Amos Yahil. *Digital Image Recosntuction: Deblurring and Denoising*, Annual Review of Astronomy and Astrophysics, 2005.

[Randy97] Randy Crane. *A Simplified Approach to Image Processing*, Prentice Hall PTR, 1997.

[Rao90] K.R. Rao and P. Yip. *Discrete Cosine Transform. Algorithms, Advantages, Applications*. Academic Press, Inc, London, 1990.

[Ric72] W.Richardson. *Bayesian-Based Iterative Method of Image Reconstruction*. Journal of the Optical Society of America, vol.62, No.1, January 1972.

[Ritter96] G.Ritter, J.Wilson. *Computer Vision. Algorithms in Image Algebra*. CRC Press, 1996.

[Rog85] David Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.

[Rog90] David Rogers and J.Alan Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill, 1990.

[S3TC] *S3 Texture Compression*. http://en.wikipedia.org/wiki/S3TC

[Sak98] T. Sakamoto, C. Nakanishi and T. Hase, *Software pixel interpolation for digital still cameras suitable for a 32-bit MCU*, IEEE Trans. Consumer Electronics, vol. 44, No. 4, November 1998.

[Serra82] J.Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[Schu94] Dale A. Schumacher. *A comparison of digital halftoning techniques,* Graphic Gems III, Academic Press, 1994, pp. 57–71.

[Sha98] Tom Shanley. *Pentium Pro and Pentium II System Architecture*. Addison-Wesley, 1998.

[SMPTE314M] SMPTE 314M-2005 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video - 25 and 50 Mb/s*. The Society of Motion Picture and Television Engineers (09/05).

[SMPTE370M] SMPTE 370M-2002 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p*. The Society of Motion Picture and Television Engineers (07/02).

[SMPTE370M-06] SMPTE 370M-2006 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p, 720/50p*. The Society of Motion Picture and Television Engineers (04/06).

[SMPTE421M] SMPTE 421M. Final Draft SMPTE Standard - *VC-1 Compressed Video Bitstream Format and Decoding Process* (01/06).

[Telea04] A.Telea. *An Image Inprinting Technique Based on the Fast Marching Method.* Journal of Graphic Tools, vol.9, No.1, ACM Press, 2004.

[Tho91] Spencer W. Thomas and Rod G. Bogart. *Color dithering*, Graphic Gems II, Academic Press, 1991, pp. 72–77.

[Ulichney93] R.Ulichney. *Digital halftoning*. MIT press, 1993.

[Vincent91] L.Vincent, P.Soille. *Watershed in Digital Spaces: An Efficient Algorithm Based on Immersion Simulation*. IEEE Transactions of Pattern Analysis and Machine Intelligence, vol.3, No.6, June 1991, pp.583-598.

[Vincent93] L.Vincent. *Morphological Gray Scale Reconstruction in Image Analysis: Applications and Efficient Algorithms*. IEEE Transactions on Image Processing, vol.2, No.2, April 1993.

[Viola01] P.Viola, M.J.Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001), 2001.

[Wang02] Z.Wang, A.C.Bovik. *A Universal Image Quality Index*. IEEE Signal Processing Letters, vol.9, No.3, March 2002, pp.81-84.

[Wolberg96] G.Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1996.

Glossary

absolute colors	Colors specified by each pixel's coordinates in a color space. Intel Integrated Performance Primitives for image processing use images with absolute colors.
alpha channel	A color channel, also known as the opacity channel, that can be used in color models; for example, the RGBA model.
arithmetic operation	An operation that adds, subtracts, multiplies, divides, or squares the image pixel values.
color-twist matrix	A matrix used to multiply the pixel components in one color space for determining the components in another color space.
DCT	Acronym for the discrete cosine transform. *See* "Discrete Cosine Transforms" in Chapter 10.
dilation	A morphological operation that sets each output pixel to the minimum of the corresponding input pixel and its 8 neighbors.
dyadic operation	An operation that has two input images. It can have other input parameters as well.
element-wise operation	An element-wise operation performs the same operation on each element of a vector, or uses the elements of the same position in multiple vectors as inputs to the operation.
erosion	A morphological operation that sets each output pixel to the maximum of the corresponding input pixel and its 8 neighbors.
four-channel model	A color model that uses four color channels; for example, the RGBA color model.

gray scale image	An image characterized by a single intensity channel so that each intensity value corresponds to a certain shade of gray.
in-place operation	An operation whose output image is one of the input images.
linear filtering	In this manual, 2D convolution operations.
linear image transforms	In this manual, the discrete cosine transform (DCT).
MMX™ technology	An enhancement to the Intel architecture aimed at better performance in multimedia and communications applications. The technology uses four additional data types, eight 64-bit MMX registers, and 57 additional instructions implementing the SIMD (single instruction, multiple data) technique.
monadic operation	An operation that has a single input image. It can have other input parameters as well.
morphological operation	An erosion, dilation, or their combinations.
not-in-place operation	An operation whose output is an image other than the input image(s). *See* in-place operation.
pixel depth	The number of bits determining each channel intensity for a single pixel in the image.
pixel-oriented ordering	Storing the image information in such an order that the values of all color channels for each pixel are clustered; for example, RGBRGB... .
planar-oriented ordering	Storing the image information so that all data of one color channel follow all data of another channel, thus forming a separate “plane” for each channel; for example, RRRRRGGGGGBBBBB...
region of interest	A rectangular image region on which an operation acts (or processing occurs).
RGB	Red-green-blue. A three-channel color model that uses red, green, and blue color channels.
RGBA	Red-green-blue-alpha. A four-channel color model that uses red, green, blue, and alpha (or opacity) channels.
ROI	See *region of interest*.

row-major order	The default storage method for arrays in C. Memory representation is such that the rows of an array are stored contiguously. For example, for the array `a[3][4]`, the element `a[1][0]` immediately follows `a[0][3]`.
saturation	Using saturation arithmetic, when a number exceeds the data-range limit for its data type, it saturates to the upper data-range limit. For example, a signed word greater than `7FFFh` saturates to `7FFFh`. When a number is less than the lower data-range limit, it saturates to the lower data-range. For example, a signed word less than `8000h` saturates to `8000h`.
Streaming SIMD Extensions	The enhancement to Intel architecture instruction set for the next generation processors. It incorporates a group of general-purpose floating-point instructions operating on packed data, additional packed integer instructions, together with cacheability control and state management instructions. These instructions significantly improve performance of applications using compute-intensive processing of floating-point and integer data.
three-channel model	A color model that uses three color channels; for example, the RGB color model.

Index

A

about this manual, 1-4
about this software, 1-1
Abs, 5-40
AbsDiff, 5-41
AbsDiffC, 5-43
absolute color images, 2-20
Add, 5-4
Add128_JPEG, 15-63
Add8x8, 16-56
Add8x8HP, 16-57
AddBackPredPB_H263, 16-301
AddC, 5-8
AddC8x8, 16-58
AddProduct, 5-13
AddRandGauss_Direct, 4-39
AddRandUniform_Direct, 4-37
AddRotateShift, 12-38
AddSquare, 5-11
AddWeighted, 5-14
alpha channel, 2-21
alpha composition functions
    AlphaComp, 5-77
    AlphaCompC, 5-79
    AlphaPremul, 5-82
    AlphaPremulC, 5-84
AlphaCompColorKey, 6-201
anchor cell, in a neighborhood mask, 2-22
And, 5-57
AndC, 5-58
ApplyHaarClassifier, 14-111
arithmetic functions
    absolute value, 5-40
    addition, 5-4
    addition with accumulation, 5-11
    complement, 5-53
    division, 5-32
    exponential, 5-51
    logarithm, 5-48
    multiplication, 5-16
    multiplication with scaling, 5-22
    square, 5-44
    square root, 5-46
    subtraction, 5-26
arithmetic operations, 5-4
audience for this manual, 1-6
Average16x16, 16-59
Average8x8, 16-59
AVS Decoder functions
    FilterDeblockingChroma_HorEdge_AVS, 16-496
    FilterDeblockingChroma_VerEdge_AVS, 16-494
    FilterDeblockingLuma_HorEdge_AVS, 16-492
    FilterDeblockingLuma_VerEdge_AVS, 16-490
    InterpolateLumaBlock_AVS, 16-482
    ReconstructChromaInter_AVS, 16-489
    ReconstructChromaIntra_AVS, 16-488
    ReconstructLumaInter_AVS, 16-486
    ReconstructLumaIntra_AVS, 16-485
    WeightPrediction_AVS, 16-484
AVS Encoder functions
    DisassembleChroma420Intra_AVS, 16-502
    DisassembleLumaIntra_AVS, 16-500
    TransformQuant8x8Fwd_AVS, 16-499

B

BGR555ToYCbCr_JPEG, 15-13
BGR555ToYCbCr411, 6-111
BGR555ToYCbCr411LS_MCU, 15-31
BGR555ToYCbCr420, 6-107
BGR555ToYCbCr422, 6-70
BGR555ToYCbCr422LS_MCU, 15-29
BGR555ToYCbCr444LS_MCU, 15-27
BGR565ToBGR, 6-152
BGR565ToYCbCr_JPEG, 15-13
BGR565ToYCbCr411, 6-111
BGR565ToYCbCr411LS_MCU, 15-31
BGR565ToYCbCr420, 6-107
BGR565ToYCbCr422, 6-70
BGR565ToYCbCr422LS_MCU, 15-29
BGR565ToYCbCr444LS_MCU, 15-27
BGRToBGR565, 6-151
BGRToCbYCr422, 6-74
BGRToCbYCr422_709HDTV, 6-75
BGRToHLS, 6-129
BGRToLab, 6-120
BGRToY_JPEG, 15-6
BGRToYCbCr_JPEG, 15-11
BGRToYCbCr411, 6-108
BGRToYCbCr411LS_MCU, 15-30
BGRToYCbCr420, 6-92
BGRToYCbCr420_709CSC, 6-93
BGRToYCbCr420_709HDTV, 6-95
BGRToYCbCr422, 6-68
BGRToYCbCr422LS_MCU, 15-28
BGRToYCbCr444LS_MCU, 15-26
BGRToYCoCg, 6-135
BGRToYCoCg_Rev, 6-139
BGRToYCrCb420, 6-106
BGRToYCrCb420_709CSC, 6-96
BGRToYUV420, 6-43
Bidir_H264, 16-390
BidirWeight_H264, 16-395
BiDirWeightBlock_H264, 16-395
BiDirWeightBlockImplicit_H264, 16-397
BidirWeightImplicit_H264, 16-397
bit depth, of an image, 2-21
bitstream parsing, 16-239, 16-274, 16-291
borders, in neighborhood operations, 9-5
BoundSegments, 14-91
buffer step, 2-24

C

CalcGlobalMV_MPEG4, 16-249
camera calibration and 3D reconstruction, 14-116
Canny, 14-10
CannyGetSize, 14-9
CbYCr422ToBGR, 6-76
CbYCr422ToBGR_709HDTV, 6-77
CbYCr422ToRGB, 6-73
CbYCr422ToYCbCr411, 6-169
CbYCr422ToYCbCr420, 6-166
CbYCr422ToYCbCr420_Interlace, 6-167
CbYCr422ToYCbCr420_Rotate, 16-119
CbYCr422ToYCbCr422, 6-165
CbYCr422ToYCrCb420, 6-168
CFAToRGB, 6-146
ChangeSpriteBrightness_MPEG4, 16-250
chromaticity diagram, 6-8
CIE Lab color model, 6-22
CIE LUV color model, 6-22
CIE XYZ color model, 6-21
closing, 8-2
CMYK color model, 6-12
CMYKToYCCK_JPEG, 15-16
CMYKToYCCK411LS_MCU, 15-34
CMYKToYCCK422LS_MCU, 15-33
CMYKToYCCK444LS_MCU, 15-32
code samples, 1-4
codecs, 1-4
color conversion functions
    between RGB and HLS, 6-126
    between RGB and HSV, 6-132
    between RGB and Lab, 6-120
    between RGB and LUV, 6-115
    between RGB and XYZ, 6-113
    between RGB and YCbCr, 6-53
    between RGB and YCbCr422, 6-63
    between RGB and YCC, 6-123
    between RGB and YCoCg, 6-135
    between RGB and YCoCg-R, 6-139
    between RGB and YUV, 6-31
    between RGB and YUV420, 6-39
    between RGB and YUV422, 6-35
    color to gray scale, 6-143
    color twist, 6-194, 6-195
    gamma correction, 6-203
color conversion functions, combined, for JPEG

codec, 15-20
color conversion functions, for JPEG codec, 15-3
color gamut, 6-8
color median filter, 9-38
color spaces
CIE Lab, 6-22
CIE LUV, 6-22
CIE XYZ, 6-21
CMYK, 6-12
HLS, 6-18
HSV, 6-18
Photo YCC, 6-15
RGB, 6-10
YCbCr, 6-14
YCCK, 6-14
YCoCg, 6-17
YCoCg-R, 6-17
YUV, 6-12
ColorToGray, 6-145
ColorTwist, 6-195
ColorTwist32f, 6-197
combined quantization, DCT, and level shift functions, for JPEG codec, 15-53
Compare, 7-21
compare functions, 7-21
CompareC, 7-22
CompareEqualEps, 7-25
CompareEqualEpsC, 7-26
CompColorKey, 6-199
Complement, 5-53
compressed macroblock, 16-185
compressed segment, 16-185
ComputeThreshold_Otsu, 7-19
concepts
image data types and ranges, 2-20
major operation models and key parameters, 2-22
of IPP, 2-1
structures and enumerators, 2-14
conventions
font, 1-6
function naming, 2-2
notational, 1-6
conversion functions, for JPEG codec, 15-80
Convert, 4-2
ConvFull, 9-74
ConvValid, 9-77
Copy, 4-11
Copy16x16, 16-53
Copy16x16HP, 16-54
Copy16x16QP_MPEG4, 16-242
Copy16x8HP, 16-54
Copy16x8QP_MPEG4, 16-242
Copy8x4HP, 16-54
Copy8x8, 16-53
Copy8x8HP, 16-54
Copy8x8QP_MPEG4, 16-242
CopyConstBorder, 4-16
CopyManaged, 4-15
CopyReplicateBorder, 4-19
CopySubpix, 4-25
CopySubpixIntersect, 4-27
CopyWrapBorder, 4-22
correction of camera lens distortion
CreateMapCameraUndistort, 14-119
UndistortGetSize, 14-116
UndistortRadial, 14-117
CountInRange, 11-29
CountZeros8x8, 16-222
CplxExtendToPack, 10-37
CreateMapCameraUndistort, 14-119
Cross-Architecture Alignment, 1-2
CrossCorrFull_Norm, 11-98
CrossCorrFull_NormLevel, 11-106
CrossCorrSame_Norm, 11-100
CrossCorrValid, 11-105
CrossCorrValid_Norm, 11-103
CrossCorrValid_NormLevel, 11-111
cross-platform applications, 2-2

D

data exchange functions
adding Gaussian noise, 4-39
adding uniform noise, 4-37
conversion, 4-2
CopyConstBorder, 4-16
copying, 4-11
CopyManaged, 4-15
CopyReplicateBorder, 4-19
CopyWrapBorder, 4-22
duplicate gray scale image, 4-31

initializing, 4-9
scaling, 4-6
swap channels, 4-34
transpose image, 4-32
DCT block, 16-182
DCT2x4x8Frw, 16-221
DCT2x4x8Inv, 16-215
DCT8x4x2To4x4Inv_DV, 16-217
DCT8x8Fwd, 10-51
DCT8x8FwdLS, 10-55
DCT8x8Inv, 10-53
DCT8x8Inv_2x2, 10-57
DCT8x8Inv_4x4, 10-57
DCT8x8Inv_A10, 10-53
DCT8x8Inv_AANTransposed_16s_C1R, 16-166
DCT8x8Inv_AANTransposed_16s_P2C2R, 16-168
DCT8x8Inv_AANTransposed_16s8u_C1R, 16-167
DCT8x8Inv_AANTransposed_16s8u_P2C2R, 16-169
DCT8x8InvLSClip, 10-56
DCT8x8To2x2Inv, 10-58
DCT8x8To4x4Inv, 10-58
DCTFwd, 10-48
DCTFwdFree, 10-45
DCTFwdGetBufSize, 10-46
DCTFwdInitAlloc, 10-43
DCTInv, 10-50
DCTInvFree, 10-46
DCTInvGetBufSize, 10-47
DCTInvInitAlloc, 10-44
DCTQuantFwd8x8_JPEG, 15-54
DCTQuantFwd8x8LS_JPEG, 15-55
DCTQuantInv8x8_JPEG, 15-57
DCTQuantInv8x8LS_1x1_JPEG, 15-59
DCTQuantInv8x8LS_2x2_JPEG, 15-59
DCTQuantInv8x8LS_4x4_JPEG, 15-59
DCTQuantInv8x8LS_JPEG, 15-58
DCTQuantInv8x8To2x2LS_JPEG, 15-60
DCTQuantInv8x8To4x4LS_JPEG, 15-60
DecimateFilterColumn, 9-28
DecimateFilterRow, 9-28
DecodeCAVLCChroma422DcCoeffs_H264, 16-343
DecodeCAVLCChromaDcCoeffs_H264, 16-343
DecodeCAVLCCoeffs_H264, 16-341
DecodeCBProgrAttach_JPEG2K, 15-171
DecodeCBProgrFree_JPEG2K, 15-170
DecodeCBProgrGetStateSize, 15-168
DecodeCBProgrInit_JPEG2K, 15-169
DecodeCBProgrInitAlloc_JPEG2K, 15-169
DecodeChromaBlock_AVS, 16-481
DecodeCodeBlock_JPEG2K, 15-166
DecodeCoeffsInter_H261, 16-278
DecodeCoeffsInter_MPEG4, 16-258
DecodeCoeffsInterRVLCBack_MPEG4, 16-259
DecodeCoeffsIntra_H261, 16-277
DecodeCoeffsIntra_H263, 16-295
DecodeCoeffsIntra_MPEG4, 16-255
DecodeCoeffsIntraRVLCBack_MPEG4, 16-257
DecodeDCIntra_H263, 16-294
DecodeDCIntra_MPEG4, 16-254
DecodeExpGolombOne_H264, 16-345
DecodeGetBufSize_JPEG2K, 15-166
DecodeHuffman8x8_ACFirst_JPEG, 15-113
DecodeHuffman8x8_ACRefine_JPEG, 15-114
DecodeHuffman8x8_DCFirst_JPEG, 15-111
DecodeHuffman8x8_DCRefine_JPEG, 15-112
DecodeHuffman8x8_Direct_JPEG, 15-109
DecodeHuffman8x8_JPEG, 15-108
DecodeHuffmanOne, 16-22
DecodeHuffmanOne_JPEG, 15-123
DecodeHuffmanPair, 16-23
DecodeHuffmanRow_JPEG, 15-125
DecodeHuffmanSpecFree_JPEG, 15-105
DecodeHuffmanSpecGetBufSize_JPEG, 15-102
DecodeHuffmanSpecInit_JPEG, 15-103
DecodeHuffmanSpecInitAlloc_JPEG, 15-104
DecodeHuffmanStateFree_JPEG, 15-107
DecodeHuffmanStateGetBufSize_JPEG, 15-105
DecodeHuffmanStateInit_JPEG, 15-106
DecodeHuffmanStateInitAlloc_JPEG, 15-107
DecodeLumaBlockInter_AVS, 16-480
DecodeLumaBlockIntra_AVS, 16-479
DecodeTextureBlockFromRGBA, 15-190
DeconvFFT, 9-83
DeconvFFTFree, 9-82
DeconvFFTInitAlloc, 9-81
DeconvLR, 9-85
DeconvLRFree, 9-85
DeconvLRInitAlloc, 9-84
deconvolution functions
DeconvFFT, 9-83
DeconvFFTFree, 9-82
DeconvFFTInitAlloc, 9-81

DeconvLR, 9-85
DeconvLRFree, 9-85
DeconvLRInitAlloc, 9-84
deinterlace filtering functions
YCbCr420ToYCrCb420_Filter, 6-180
DeinterlaceBlandInitAlloc, 16-133
DeinterlaceBlend, 16-134
DeinterlaceBlendInitAlloc, 16-132
DeinterlaceEdgeDetect, 16-129
DeinterlaceFilterCAVT, 16-126
DeinterlaceFilterTriangle, 16-125
DeinterlaceMedianThreshold, 16-128
DeinterlaceMotionAdaptive, 16-130
deinterlacing, 16-125
DemosaicAHD, 6-149
denoising functions
DeinterlaceBlendFree, 16-133
DeinterlaceBlendInitAlloc, 16-132
FilterDenoiseAdaptive, 16-141
FilterDenoiseAdaptiveFree, 16-141
FilterDenoiseAdaptiveInitAlloc, 16-140
FilterDenoiseCAST, 16-137
FilterDenoiseCASTInit, 16-136
FilterDenoiseMosquito, 16-145
FilterDenoiseMosquitoFree, 16-145
FilterDenoiseMosquitoInitAlloc, 16-144
FilterDenoiseSmooth, 16-139
dequantization, of the DCT coefficients, 15-52
DequantTransformResidual_H264, 16-349
DequantTransformResidual_SISP_H264, 16-353
DequantTransformResidualAndAdd_H264, 16-350
DFTFree, 10-16
DFTFwd, 10-18
DFTGetBufSize, 10-17
DFTInitAlloc, 10-15
DFTInv, 10-21
DiffPredFirstRow_JPEG, 15-116
DiffPredRow_JPEG, 15-118
Dilate, 8-11
Dilate3x3, 8-7
DilateBorderReplicate, 8-19
dilation, 8-2
DisassembleChroma420Intra_AVS, 16-502
DisassembleLumaIntra_AVS, 16-500
Distance transform, 14-24
DistanceTransform, 14-24
Div, 5-32
Div_Round, 5-35
DivC, 5-37
DotProd, 5-54
DotProdCol, 5-56, 9-67
DownsampleFour_H263, 16-306
DownsampleFour16x16_H263, 16-324
downsampling, image, 6-25
Dup, 4-31
DV, 16-182
DV decoder functions
inverse discrete cosine transformation
DCT2x4x8Inv, 16-215
DCT8x4x2To4x4Inv_DV, 16-217
inverse quantization
QuantInv_DV, 16-204
QuantWeightBlockInv_DV, 16-205
QuantWeightBlockInv_DV100, 16-211
variable length decoding
FreeHuffmanTable_DV, 16-204
HuffmanDecode Segment_DV, 16-196
HuffmanDecode Segment_DV100, 16-201
HuffmanDecodeSegmentOnePass_DV, 16-203
InitAllocHuffmanTable_DV, 16-192
DV encoder functions
color conversion
YCrCb411ToYCbCr422_16x4x5MB_DV, 16-225
YCrCb411ToYCbCr422_5MBDV, 16-224
YCrCb411ToYCbCr422_8x8MB_DV, 16-225
YCrCb411ToYCbCr422_EdgeDV, 16-227
YCrCb411ToYCbCr422_ZoomOut2_5MBDV, 16-224
YCrCb411ToYCbCr422_ZoomOut2_EdgeDV, 16-227
YCrCb411ToYCbCr422_ZoomOut4_5MBDV, 16-224
YCrCb411ToYCbCr422_ZoomOut4_EdgeDV, 16-227
YCrCb411ToYCbCr422_ZoomOut8_5MBDV, 16-224
YCrCb411ToYCbCr422_ZoomOut8_EdgeDV, 16-227
YCrCb420ToYCbCr422_5MBDV, 16-228
YCrCb420ToYCbCr422_8x8x5MB_DV, 16-229
YCrCb420ToYCbCr422_ZoomOut2_5MBDV

, 16-228
YCrCb420ToYCbCr422_ZoomOut4_5MBDV
, 16-228
YCrCb420ToYCbCr422_ZoomOut8_5MBDV
, 16-228
YCrCb422ToYCbCr422_10HalvesMB16x8_
DV100, 16-233
YCrCb422ToYCbCr422_5MBDV, 16-230
YCrCb422ToYCbCr422_8x4x5MB_DV,
16-232
YCrCb422ToYCbCr422_ZoomOut2_5MBDV
, 16-230
YCrCb422ToYCbCr422_ZoomOut4_5MBDV
, 16-230
YCrCb422ToYCbCr422_ZoomOut8_5MBDV
, 16-230
discrete cosine transformation
CountZeros8x8, 16-222
DCT2x4x8Frw, 16-221
DXT1 compression, 15-188
DXT5 compression, 15-188

E

EigenValsVecs, 14-13
EigenValsVecsGetBufferSize, 14-12
EncodeChromaDcCoeffsCAVLC_H264_16s, 16-472
EncodeCoeffsCAVLC_H264_16s, 16-469
EncodeCoeffsInter_H261, 16-285
EncodeCoeffsInter_H263, 16-320
EncodeCoeffsInter_MPEG4, 16-271
EncodeCoeffsIntra_H261, 16-284
EncodeCoeffsIntra_H263, 16-319
EncodeCoeffsIntra_MPEG4, 16-270
EncodeDCIntra_H263, 16-318
EncodeDCIntra_MPEG4, 16-269
EncodeFree_JPEG2K, 15-157
EncodeGetDist_JPEG2K, 15-165
EncodeGetRate_JPEG2K, 15-164
EncodeGetTermPassLen_JPEG2K, 15-163
EncodeHuffman8x8_ACFirst_JPEG, 15-99
EncodeHuffman8x8_ACRefine_JPEG, 15-101
EncodeHuffman8x8_DCFirst_JPEG, 15-97
EncodeHuffman8x8_DCRefine_JPEG, 15-98
EncodeHuffman8x8_Direct_JPEG, 15-92
EncodeHuffman8x8_JPEG, 15-91
EncodeHuffmanOne_JPEG, 15-121
EncodeHuffmanRawTableInit_JPEG, 15-84
EncodeHuffmanRow_JPEG, 15-124
EncodeHuffmanSpecFree_JPEG, 15-88
EncodeHuffmanSpecGetBufSize_JPEG, 15-85
EncodeHuffmanSpecInit_JPEG, 15-86
EncodeHuffmanSpecInitAlloc_JPEG, 15-87
EncodeHuffmanStateFree_JPEG, 15-90
EncodeHuffmanStateGetBufSize_JPEG, 15-88
EncodeHuffmanStateInit_JPEG, 15-89
EncodeHuffmanStateInitAlloc_JPEG, 15-90
EncodeInitAlloc_JPEG2K, 15-157
EncodeLoadCodeBlock_JPEG2K, 15-158
EncodeStoreBits_JPEG2K, 15-161
EncodeTextureBlockFromRGBA, 15-189
enumerators, 2-14
Erode, 8-13
Erode3x3, 8-9
ErodeBorderReplicate, 8-21
erosion, 8-2
error handling, 2-9
error messages, 2-9
Exp, 5-51
ExpandPlane_H264, 16-366

F

feature detection functions
Canny, 14-10
CannyGetSize, 14-9
EigenValsVecs, 14-13
EigenValsVecsGetBufferSize, 14-12
MinEigenVal, 14-17
MinEigenValGetBufferSize, 14-16
FFTFree, 10-7
FFTFwd, 10-9
FFTGetBufSize, 10-8
FFTInitAlloc, 10-6
FFTInv, 10-13
FillterScharrHorizGetBufferSize, 9-115
Filter, 9-40
Filter_Round16s, 9-45
Filter_Round32f, 9-45
Filter_Round32s, 9-45
Filter32f, 9-44
Filter8x8_H261, 16-286
FilterBilateral, 9-24
FilterBilateralGetBufSize, 9-21

FilterBilateralInit, 9-22
FilterBlockBoundaryHorEdge_H263, 16-309
FilterBlockBoundaryVerEdge_H263, 16-309
FilterBox, 9-8
FilterColumn, 9-51
FilterColumn32f, 9-54
FilterColumnPipeline, 9-64
FilterColumnPipeline_Low, 9-64
FilterColumnPipelineGetBufferSize, 9-60
FilterColumnPipelineGetBufferSize_Low, 9-60
FilterDeblocking16x16_HorEdge_H263, 16-312
FilterDeblocking16x16_VerEdge_H263, 16-312
FilterDeblocking8x8HorEdge_H263, 16-310
FilterDeblocking8x8HorEdge_MPEG4, 16-261
FilterDeblocking8x8VerEdge_H263, 16-310
FilterDeblocking8x8VerEdge_MPEG4, 16-261
FilterDeblockingChroma_HorEdge_AVS, 16-496
FilterDeblockingChroma_HorEdge_H264, 16-453
FilterDeblockingChroma_HorEdge_VC1, 16-535
FilterDeblockingChroma_VerEdge_AVS, 16-494
FilterDeblockingChroma_VerEdge_H264, 16-447
FilterDeblockingChroma_VerEdge_MBAFF_H264, 16-451
FilterDeblockingChroma_VerEdge_VC1, 16-534
FilterDeblockingChroma422HorEdge_H264, 16-453
FilterDeblockingChroma422VerEdge_H264, 16-447
FilterDeblockingChroma422VerEdge_MBAFF_H264, 16-451
FilterDeblockingLuma_HorEdge_AVS, 16-492
FilterDeblockingLuma_HorEdge_H264, 16-445
FilterDeblockingLuma_HorEdge_VC1, 16-533
FilterDeblockingLuma_VerEdge_AVS, 16-490
FilterDeblockingLuma_VerEdge_H264, 16-441
FilterDeblockingLuma_VerEdge_MBAFF_H264, 16-443
FilterDeblockingLuma_VerEdge_VC1, 16-532
FilterDenoiseAdaptive, 16-141
FilterDenoiseAdaptiveFree, 16-141
FilterDenoiseAdaptiveInitAlloc, 16-140
FilterDenoiseCAST, 16-137
FilterDenoiseCASTInit, 16-136
FilterDenoiseMosquito, 16-145
FilterDenoiseMosquitoFree, 16-145
FilterDenoiseMosquitoInitAlloc, 16-144
FilterDenoiseSmooth, 16-139
FilterDeringingSmooth8x8_MPEG4, 16-263
FilterDeringingThreshold_MPEG4, 16-262
FilterGauss, 9-107
FilterGaussBorder, 9-142
FilterGaussGetBufferSize, 9-123
FilterGetBufSize, 9-43
FilterHipass, 9-110
filtering functions
    bilateral filter, 9-21
    box filter, 9-8
    color median filter, 9-38
    column filter, 9-51
    decimate filters, 9-28
    Gaussian filter, 9-107
    general rectangular filter, 9-40
    highpass filter, 9-110
    Laplacian filter, 9-105
    lowpass filter, 9-112
    max filter, 9-14
    max filter border replication, 9-19
    max filter working buffer size, 9-17
    median filter, 9-31
    median filter with weighted center pixel, 9-37
    min filter, 9-12
    min filter border replication, 9-18
    min filter working buffer size, 9-16
    Prewitt filter, 9-88
    Roberts filter, 9-103
    row filter, 9-55
    Scharr filter, 9-91
    sharpening filter, 9-113
    Sobel filter, 9-94
FilterLaplace, 9-105
FilterLaplacianBorder, 9-140
FilterLaplacianGetBufferSize, 9-122
FilterLowpass, 9-112
FilterLowpassBorder, 9-143
FilterLowpassGetBufferSize, 9-123
FilterMax, 9-14
FilterMaxBorderReplicate, 9-19
FilterMaxGetBufferSize, 9-17
FilterMedian, 9-31
FilterMedianColor, 9-38
FilterMedianCross, 9-35
FilterMedianHoriz, 9-33
FilterMedianVert, 9-34
FilterMedianWeightedCenter3x3, 9-37

FilterMin, 9-12
FilterMinBorderReplicate, 9-18
FilterMinGetBufferSize, 9-16
FilterPrewittHoriz, 9-88
FilterPrewittVert, 9-89
FilterRobertsDown, 9-103
FilterRobertsUp, 9-104
FilterRoundGetBufSize16s, 9-48
FilterRoundGetBufSize32f, 9-48
FilterRoundGetBufSize32s, 9-48
FilterRow, 9-55
FilterRow32f, 9-57
FilterRowBorderPipeline, 9-61
FilterRowBorderPipeline_Low, 9-61
FilterRowBorderPipelineGetBufferSize, 9-58
FilterRowBorderPipelineGetBufferSize_Low, 9-58
FilterScharrHoriz, 9-91
FilterScharrHorizBorder, 9-125
FilterScharrVert, 9-92
FilterScharrVertBorder, 9-127
FilterScharrVertGetBufferSize, 9-116
FilterSharpen, 9-113
FilterSobelCross, 9-101
FilterSobelCrossBorder, 9-138
FilterSobelCrossGetBufferSize, 9-121
FilterSobelHoriz, 9-94
FilterSobelHorizBorder, 9-128
FilterSobelHorizGetBufferSize, 9-117
FilterSobelHorizMask, 9-94
FilterSobelHorizSecond, 9-97
FilterSobelHorizSecondBorder, 9-133
FilterSobelHorizSecondGetBufferSize, 9-119
FilterSobelNegVertBorder, 9-130
FilterSobelNegVertGetBufferSize, 9-118
FilterSobelVert, 9-95
FilterSobelVertBorder, 9-130
FilterSobelVertGetBufferSize, 9-118
FilterSobelVertMask, 9-96
FilterSobelVertSecond, 9-99
FilterSobelVertSecondBorder, 9-135
FilterSobelVertSecondGetBufferSize, 9-120
FilterWiener, 9-69
FilterWienerGetBufferSize, 9-68
fixed filters, 9-87
fixed filters with border, 9-115
flood fill functions
    FloodFill, 14-38
    FloodFill_Grad, 14-40
    FloodFill_Range, 14-43
    FloodFillGetSize, 14-36
    FloodFillGetSize_Grad, 14-37
FloodFill, 14-38
FloodFill_Grad, 14-40
FloodFill_Range, 14-43
FloodFillGetSize, 14-36
FloodFillGetSize_Grad, 14-37
font conventions, 1-6
ForegroundGaussian, 14-101
ForegroundGaussianFree, 14-100
ForegroundGaussianIInitAlloc, 14-98
ForegroundHistogram, 14-95
ForegroundHistogramFree, 14-94
ForegroundHistogramInitAlloc, 14-92
ForegroundHistogramUpdate, 14-97
format conversion functions
    BGR565ToBGR, 6-152
    BGRToBGR565, 6-151
    CbYCr422ToYCbCr411, 6-169
    CbYCr422ToYCbCr420, 6-166
    CbYCr422ToYCbCr420_Interlace, 6-167
    CbYCr422ToYCbCr422, 6-165
    CbYCr422ToYCrCb420, 6-168
    RGB565ToRGB, 6-152
    RGBToRGB565, 6-151
    YCbCr411, 6-188
    YCbCr411ToYCbCr420, 6-192
    YCbCr411ToYCbCr422, 6-189
    YCbCr411ToYCrCb420, 6-193
    YCbCr411ToYCrCb422, 6-191
    YCbCr420, 6-170
    YCbCr420To422_Interlace, 6-175
    YCbCr420ToCbYCr422, 6-176
    YCbCr420ToCbYCr422_Interlace, 6-177
    YCbCr420ToYCbCr411, 6-181
    YCbCr420ToYCbCr422, 6-172
    YCbCr420ToYCbCr422_Filter, 6-173
    YCbCr420ToYCrCb420, 6-179
    YCbCr420ToYCrCb420_Filter, 6-180
    YCbCr422, 6-153
    YCbCr422To420_Interlace, 6-158
    YCbCr422ToCbYCr422, 6-155
    YCbCr422ToYCbCr411, 6-161

YCbCr422ToYCbCr420, 6-156
YCbCr422ToYCrCb420, 6-160
YCrCb420ToCbYCr422, 6-185
YCrCb420ToYCbCr411, 6-187
YCrCb420ToYCbCr420, 6-186
YCrCb420ToYCbCr422, 6-183
YCrCb420ToYCbCr422_Filter, 6-184
YCrCb422ToYCbCr411, 6-164
YCrCb422ToYCbCr420, 6-163
YCrCb422ToYCbCr422, 6-162
Fourier transforms
DFTFree, 10-16
DFTFwd, 10-18
DFTIGetBufSize, 10-17
DFTInitAlloc, 10-15
DFTInv, 10-21
FFTFree, 10-7
FFTFwd, 10-9
FFTGetBufSize, 10-8
FFTInitAlloc, 10-6
FFTInv, 10-13
FrameFieldSAD16x16, 16-112
FreeHuffmanTable_DV, 16-204
function
context, 2-20
descriptions, 1-5
parameters, 2-6
prototypes, 2-6

G

gamma correction, 6-8
GammaFwd, 6-203
GammaInv, 6-206
GenScaleLevel8x8_H264, 16-468
GenSobelKernel, 9-124
geometric transform functions
affine warp, 12-51
bilinear warp, 12-77
mirroring, 12-28
perspective warp, 12-64
remapping, 12-30
resizing, 12-5
rotation, 12-34
shear, 12-45
GetAffineBound, 12-61
GetAffineQuad, 12-60
GetAffineTransform, 12-62
GetBilinearBound, 12-87
GetBilinearQuad, 12-86
GetBilinearTransform, 12-88
GetCentralMoment, 11-55
GetDiff16x16, 16-63
GetDiff16x16B, 16-70
GetDiff16x8, 16-64
GetDiff16x8B, 16-71
GetDiff4x4, 16-69
GetDiff8x16, 16-66
GetDiff8x16B, 16-74
GetDiff8x4, 16-68
GetDiff8x4B, 16-75
GetDiff8x8, 16-65
GetDiff8x8B, 16-73
GetDistanceTransformMask, 14-27
GetHaarClassifierSize, 14-109
GetHuffmanStatistics8x8_ACFirst_JPEG, 15-95
GetHuffmanStatistics8x8_ACRefine_JPEG, 15-96
GetHuffmanStatistics8x8_DCFirst_JPEG, 15-94
GetHuffmanStatistics8x8_JPEG, 15-93
GetHuffmanStatisticsOne_JPEG, 15-121
GetHuMoments, 11-58
GetLibVersion, 3-2
GetNormalizedCentralMoment, 11-56
GetNormalizedSpatialMoment, 11-54
GetPerspectiveBound, 12-74
GetPerspectiveQuad, 12-72
GetPerspectiveTransform, 12-75
GetPyramidDownROI, 14-68
GetResizeFract, 12-20
GetRotateBound, 12-40
GetRotateQuad, 12-39
GetRotateShift, 12-37
GetShearBound, 12-50
GetShearQuad, 12-49
GetSpatialMoment, 11-53
GradientColorToGray, 14-34
GrayDilateBorder, 8-39, 8-40

H

H.263 decoder inverse quantization
QuantInvInter_H263, 16-300

QuantInvIntra_H263, 16-298
H.263 decoder prediction functions
AddBackPredPB_H263, 16-301
H.263 decoder resampling functions
DownsampleFour_H263, 16-306
Resample_H263, 16-303
SpatialInterpolation_H263, 16-307
UnsampleFour_H263, 16-305
UnsampleFour8x8_H263, 16-307
H.263 Encoder functions
DownsampleFour16x16_H263, 16-324
EncodeCoeffsInter_H263, 16-320
EncodeCoeffsIntra_H263, 16-319
EncodeDCIntra_H263, 16-318
QuantInter_H263, 16-323
QuantIntra_H263, 16-322
H.263+ functions
DownsampleFour_H263, 16-306
FilterBlockBoundaryHorEdge_H263, 16-309
FilterBlockBounderyVerEdge_H263, 16-309
FilterDeblocking16x16HorEdge_H263, 16-312
FilterDeblocking16x16VerEdge_H263, 16-312
FilterDeblocking8x8HorEdge_H263, 16-310
FilterDeblocking8x8VerEdge_H263, 16-310
ReconstructCoeffsInter_H263, 16-315
ReconstructCoeffsIntra_H263, 16-313
Resample_H263, 16-303
SpatialInterpolation_H263, 16-307
UnsampleFour_H263, 16-305
UnsampleFour8x8_H263, 16-307
H.263+ middle-level functions
ReconstructCoeffsInter_H263, 16-315
ReconstructCoeffsIntra_H263, 16-313
H.263+ VLC decoding
DecodeCoeffsInter_H263, 16-296
DecodeCoeffsIntra_H263, 16-295
DecodeDCIntra_H263, 16-294
H263 decoder boundary filtering functions
FilterBlockBounderyHorEdge_H263, 16-309
FilterBlockBounderyVerrEdge_H263, 16-309
FilterDeblocking16x16HorEdge_H263, 16-312
FilterDeblocking16x16VerEdge_H263, 16-312
FilterDeblocking8x8HorEdge_H263, 16-310
FilterDeblocking8x8VerEdge_H263, 16-310
H264 Decoder functions
Bidir_H264, 16-390
BidirWeight_H264, 16-395
BiDirWeightBlock_H264, 16-395
BiDirWeightBlockImplicit_H264, 16-397
BidirWeightImplicit_H264, 16-397
DequantTransformResidual_H264, 16-349
DequantTransformResidual_SISP_H264, 16-353
DequantTransformResidualAndAdd_H264, 16-350
ExpandPlane_H264, 16-366
FilterDeblockingChroma_HorEdge_H264, 16-453
FilterDeblockingChroma_VerEdge_H264, 16-447
FilterDeblockingChroma_VerEdge_MBAFF_H264, 16-451
FilterDeblockingChroma422HorEdge_H264, 16-453
FilterDeblockingChroma422VerEdge_H264, 16-447
FilterDeblockingChroma422VerEdge_MBAFF_H264, 16-451
FilterDeblockingLuma_HorEdge_H264, 16-445
FilterDeblockingLuma_VerEdge_H264, 16-441
FilterDeblockingLuma_VerEdge_MBAFF_H264, 16-443
high profile functions, 16-405, 16-411, 16-413, 16-425, 16-428, 16-430, 16-432, 16-433, 16-435, 16-438
InterpolateBlock_H264, 16-390
InterpolateChroma_H264, 16-379
InterpolateChromaBlock_H264, 16-388
InterpolateChromaBottom_H264, 16-386
InterpolateChromaTop_H264, 16-384
InterpolateLuma_H264, 16-367
InterpolateLumaBlock_H264, 16-377
InterpolateLumaBottom_H264, 16-374
InterpolateLumaTop_H264, 16-371
PredictIntra_16x16_H264, 16-359
PredictIntra_4x4_H264, 16-355
PredictIntraChroma8x8_H264, 16-360
ReconstructChroma422Inter4x4_H264High, 16-409
ReconstructChroma422Intra4x4_H264High, 16-418
ReconstructChroma422IntraHalf4x4_H264High, 16-416
ReconstructChromaInter4x4_H264High, 16-405

ReconstructChromaInter4x4MB_H264, 16-405
ReconstructChromaInterMB_H264, 16-400
ReconstructChromaIntra4x4_H264High, 16-413
ReconstructChromaIntra4x4MB_H264, 16-413
ReconstructChromaIntraHalf4x4High_H264, 16-411
ReconstructChromaIntraHalves4x4MB_H264, 16-411
ReconstructChromaIntraHalvesMB_H264, 16-401
ReconstructChromaIntraMB_H264, 16-403
ReconstructLumaInter4x4_H264High, 16-425
ReconstructLumaInter4x4MB_H264, 16-425
ReconstructLumaInter8x8_H264High, 16-432
ReconstructLumaInter8x8MB_H264, 16-432
ReconstructLumaInterMB_H264, 16-421
ReconstructLumaIntra16x16_H264High, 16-438
ReconstructLumaIntra16x16MB_H264, 16-437, 16-438
ReconstructLumaIntra4x4_H264High, 16-430
ReconstructLumaIntra4x4MB_H264, 16-430
ReconstructLumaIntra8x8_H264High, 16-435
ReconstructLumaIntra8x8MB_H264, 16-435
ReconstructLumaIntraHalf4x4_H264High, 16-428
ReconstructLumaIntraHalf4x4MB_H264, 16-428
ReconstructLumaIntraHalf8x8_H264High, 16-433
ReconstructLumaIntraHalf8x8MB_H264, 16-433
ReconstructLumaIntraHalfMB_H264, 16-422
ReconstructLumaIntraMB_H264, 16-423
TransformDequantChromaDC_H264, 16-347
TransformDequantChromaDC_SISP_H264, 16-354
TransformDequantLumaDC_H264, 16-346
TransformPrediction_H264, 16-352
TransformResidual4x4Inv_H264, 16-348
UnidirWeight_H264, 16-393
UniDirWeightBlock_H264, 16-393
WeightedAverage_H264, 16-392
H264 Encoder functions
EncodeChromaDcCoeffsCAVLC_H264_16s, 16-472
EncodeCoeffsCAVLC_H264_16s, 16-469
GenScaleLevel8x8_H264, 16-468
QuantizeResidual4x4Fwd_H264, 16-463
QuantLuma8x8_H264, 16-467
QuantLuma8x8Inv_H264, 16-473
TransformLuma8x8Fwd_H264, 16-466
TransformLuma8x8InvAddPred_H264, 16-474
TransformQuantChromaDC_H264, 16-459
TransformQuantLumaDC_H264, 16-461
TransformQuantResidual_H264, 16-465
TransformResidual4x4Fwd_H264, 16-462
Haar Features, 14-104
HaarClassifierFree, 14-108
HaarClassifierInitAlloc, 14-106
hardware and software requirements, 1-2
Hint arguments for Image Moment functions, 11-48
HistogramEven, 11-27
HistogramRange, 11-24
HLS color model, 6-18
HLSToBGR, 6-130
HLSToRGB, 6-127
horizontal sampling, in a JPEG codec, 15-64
Hough transform, 14-19
Hough transform functions
HoughLine, 14-20
HoughLine_Region, 14-21
HoughLineGetSize, 14-19
HoughLine, 14-20
HoughLine_Region, 14-21
HoughLineGetSize, 14-19
HSV color model, 6-18
HSVToRGB, 6-133
Huffman codec functions, for JPEG codec, 15-82
HuffmanDecodeSegment_DV, 16-196
HuffmanDecodeSegment_DV100, 16-201
HuffmanDecodeSegmentOnePass_DV, 16-203
HuffmanRunLevelTableInitAlloc, 16-21
HuffmanTableFree, 16-24
HuffmanTableInitAlloc, 16-20

I

ICTFwd_JPEG2K, 15-180
ICTInv_JPEG2K, 15-182
image area extension, 9-6
image data
descriptors, 2-5
range, 2-21
storage, 2-20

image format conversion, 6-151
image segmentation, 14-81
ImageJaehne, 4-40
ImageRamp, 4-42
InitAllocHuffmanTable_DV, 16-192
initialization functions
    creating a test image, 4-40
Inpaint, 14-78
InpaintFree, 14-78
InpaintingInitAlloc, 14-76
Integral, 11-7
Intel Performance Library Suite, 2-1
Intel Performance Primitives software, 1-1
intensity transformation functions
    LUT, 6-211
    LUT_Cubic, 6-219
    LUT_Linear, 6-215
    LUTPalette, 6-224
    LUTPaletteSwap, 6-224
    ReduceBits, 6-209
    ToneMapLinear, 6-227
    ToneMapMean, 6-227
INTER macroblock decoding, 16-276, 16-293
interfield compression, 16-182
InterpolateAverage16x16, 16-55
InterpolateAverage16x8, 16-55
InterpolateAverage8x4, 16-55
InterpolateBlock_H264, 16-390
InterpolateChroma_H264, 16-379
InterpolateChromaBlock_H264, 16-388
InterpolateChromaBottom_H264, 16-386
InterpolateChromaTop_H264, 16-384
InterpolateLuma_H264, 16-367
InterpolateLumaBlock_AVS, 16-482
InterpolateLumaBlock_H264, 16-377
InterpolateLumaBottom_H264, 16-374
InterpolateLumaTop_H264, 16-371
InterpolateQPBicubic_VC1, 16-522
InterpolateQPBilinear_VC1, 16-520
interpolation method, 12-3
INTRA macroblock decoding, 16-275, 16-292
inverse discrete cosine transformation, 16-166
ipiFloodFill, 14-38
IPP functionality
    arithmetic and logical operations, 5-1
    color conversion, 6-1
    computer vision, 14-1
    data exchange functions, 4-1
    filtering operations, 9-1
    geometric transforms, 12-1
    H.261 video decoder, 16-274
    H.263 video decoder, 16-291
    image statistics, 11-1
    intensity transformation, 6-209
    linear transforms, 10-1
    morphological operations, 8-1
    MPEG-4 video decoder, 16-239
    threshold and compare functions, 7-1
ippGetStatusString, 3-4
ippiAbs, 5-40
ippiAbsDiff, 5-41
ippiAbsDiffC, 5-43
ippiAdd, 5-4
ippiAdd128_JPEG, 15-63
ippiAdd8x8, 16-56
ippiAdd8x8HP, 16-57
ippiAddBackPredPB_H263, 16-301
ippiAddC, 5-8
ippiAddC8x8, 16-58
ippiAddProduct, 5-13
ippiAddRandGauss_Direct, 4-39
ippiAddRandUniform_Direct, 4-37
ippiAddRotateShift, 12-38
ippiAddSquare, 5-11
ippiAddWeighted, 5-14
ippiAlphaComp, 5-77
ippiAlphaCompC, 5-79
ippiAlphaCompColorKey, 6-201
ippiAlphaPremul, 5-82
ippiAlphaPremulC, 5-84
ippiAnd, 5-57
ippiAndC, 5-58
ippiAverage16x16, 16-59
ippiAverage8x8, 16-59
ippiBGR555ToYCbCr_JPEG, 15-13
ippiBGR555ToYCbCr411, 6-111
ippiBGR555ToYCbCr411LS_MCU, 15-31
ippiBGR555ToYCbCr420, 6-107
ippiBGR555ToYCbCr422, 6-70
ippiBGR555ToYCbCr422LS_MCU, 15-29
ippiBGR555ToYCbCr444LS_MCU, 15-27
ippiBGR565ToBGR, 6-152

ippiBGR565ToYCbCr411, 6-111
ippiBGR565ToYCbCr411LS_MCU, 15-31
ippiBGR565ToYCbCr420, 6-107
ippiBGR565ToYCbCr422, 6-70
ippiBGR565ToYCbCr422LS_MCU, 15-29
ippiBGR565ToYCbCr444LS_MCU, 15-27
ippiBGRToBGR565, 6-151
ippiBGRToCbYCr422, 6-74
ippiBGRToCbYCr422_709HDTV, 6-75
ippiBGRToHLS, 6-129
ippiBGRToLab, 6-120
ippiBGRToY_JPEG, 15-6
ippiBGRToYCbCr_JPEG, 15-11
ippiBGRToYCbCr411LS_MCU, 15-30
ippiBGRToYCbCr420, 6-92
ippiBGRToYCbCr420_709CSC, 6-93
ippiBGRToYCbCr420_709HDTV, 6-95
ippiBGRToYCbCr422, 6-68
ippiBGRToYCbCr422LS_MCU, 15-28
ippiBGRToYCbCr444LS_MCU, 15-26
ippiBGRToYCoCg, 6-135
ippiBGRToYCoCg_Rev, 6-139
ippiBGRToYCrCb420, 6-106
ippiBGRToYCrCb420_709CSC, 6-96
ippiBGRToYUV420, 6-43
ippiBidir_H264, 16-390
ippiBidirWeight_H264, 16-395
ippiBiDirWeightBlock_H264, 16-395
ippiBiDirWeightBlockImplicit_H264, 16-397
ippiBidirWeightImplicit_H264, 16-397
ippiBoundSegments, 14-91
ippiCalcGlobalMV_MPEG4, 16-249
ippiCanny, 14-10
ippiCannyGetSize, 14-9
ippiCbYCr422ToBGR, 6-76
ippiCbYCr422ToBGR_709HDTV, 6-77
ippiCbYCr422ToRGB, 6-73
ippiCbYCr422ToYCbCr411, 6-169
ippiCbYCr422ToYCbCr420, 6-166
ippiCbYCr422ToYCbCr420_Interlace, 6-167
ippiCbYCr422ToYCbCr422, 6-165
ippiCbYCr422ToYCbCr422_Rotate, 16-119
ippiCbYCr422ToYCrCb420, 6-168
ippiCFAToRGB, 6-146
ippiChangeSpriteBrightness_MPEG4, 16-250
ippiCMYKToYCCK_JPEG, 15-16
ippiCMYKToYCCK411LS_MCU, 15-34
ippiCMYKToYCCK422LS_MCU, 15-33
ippiCMYKToYCCK444LS_MCU, 15-32
ippiColorToGray, 6-145
ippiColorTwist, 6-195
ippiColorTwist32f, 6-197
ippiColumnPipeline_Low, 9-64
ippiCompare, 7-21
ippiCompareC, 7-22
ippiCompareEqualEps, 7-25
ippiCompareEqualEpsC, 7-26
ippiCompColorKey, 6-199
ippiComplement, 5-53
ippiComputeThreshold_Otsu, 7-19
ippiConvert, 4-2
ippiConvFull, 9-74
ippiConvValid, 9-77
ippiCopy, 4-11
ippiCopy16x16, 16-53
ippiCopy16x16HP, 16-54
ippiCopy16x16QP_MPEG4, 16-242
ippiCopy16x8HP, 16-54
ippiCopy16x8QP_MPEG4, 16-242
ippiCopy8x4HP, 16-54
ippiCopy8x8, 16-53
ippiCopy8x8HP, 16-54
ippiCopy8x8QP_MPEG4, 16-242
ippiCopyConstBorder, 4-16
ippiCopyManaged, 4-15
ippiCopyReplicateBorder, 4-19
ippiCopySubpix, 4-25
ippiCopySubpixIntersect, 4-27
ippiCopyWrapBorder, 4-22
ippiCountInRange, 11-29
ippiCountZeros8x8, 16-222
ippiCreateMapCameraUndistort, 14-119
ippiCrossCorrFull_Norm, 11-98
ippiCrossCorrFull_NormLevel, 11-106
ippiCrossCorrSame_Norm, 11-100
ippiCrossCorrValid, 11-105
ippiCrossCorrValid_Norm, 11-103
ippiCrossCorrValid_NormLevel, 11-111
ippiDCT2x4x8Frw, 16-221
ippiDCT2x4x8Inv, 16-215
ippiDCT8x4x2To4x4Inv_DV, 16-217
ippiDCT8x8Fwd, 10-51

ippiDCT8x8FwdLS, 10-55
ippiDCT8x8Inv, 10-53
ippiDCT8x8Inv_2x2, 10-57
ippiDCT8x8Inv_4x4, 10-57
ippiDCT8x8Inv_A10, 10-53
ippiDCT8x8Inv_AANTransposed_16s_C1R, 16-166
ippiDCT8x8Inv_AANTransposed_16s_P2C2R, 16-168
ippiDCT8x8Inv_AANTransposed_16s8u_C1R, 16-167
ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R, 16-169
ippiDCT8x8InvLSClip, 10-56
ippiDCT8x8To2x2Inv, 10-58
ippiDCT8x8To4x4Inv, 10-58
ippiDCTFwd, 10-48
ippiDCTFwdFree, 10-45
ippiDCTFwdGetBufSize, 10-46
ippiDCTFwdInitAlloc, 10-43
ippiDCTInv, 10-50
ippiDCTInvFree, 10-46
ippiDCTInvGetBufSize, 10-47
ippiDCTInvInitAlloc, 10-44
ippiDCTQuantFwd8x8_JPEG, 15-54
ippiDCTQuantFwd8x8LS_JPEG, 15-55
ippiDCTQuantInv8x8_4x4LS_JPEG, 15-60
ippiDCTQuantInv8x8_JPEG, 15-57
ippiDCTQuantInv8x8LS_1x1_JPEG, 15-59
ippiDCTQuantInv8x8LS_2x2_JPEG, 15-59
ippiDCTQuantInv8x8LS_4x4_JPEG, 15-59
ippiDCTQuantInv8x8LS_JPEG, 15-58
ippiDCTQuantInv8x8To2x2LS_JPEG, 15-60
ippiDCTQuantInv8x8To4x4LS_JPEG, 15-60
ippiDecimateFilterColumn, 9-28
ippiDecimateFilterRow, 9-28
ippiDecodeCAVLCChroma422DcCoeffs_H264, 16-343
ippiDecodeCAVLCChromaDcCoeffs_H264, 16-343
ippiDecodeCAVLCCoeffs_H264, 16-341
ippiDecodeCBProgrAttach_JPEG2K, 15-171
ippiDecodeCBProgrFree_JPEG2K, 15-170
ippiDecodeCBProgrGetStateSize, 15-168
ippiDecodeCBProgrInit_JPEG2K, 15-169
ippiDecodeCBProgrInitAlloc_JPEG2K, 15-169
ippiDecodeChromaBlock_AVS, 16-481
ippiDecodeCodeBlock_JPEG2K, 15-166
ippiDecodeCoeffsInter_H261, 16-278
ippiDecodeCoeffsInter_MPEG4, 16-258
ippiDecodeCoeffsInterRVLCBack_MPEG4, 16-259
ippiDecodeCoeffsIntra_H261, 16-277
ippiDecodeCoeffsIntra_H263, 16-295
ippiDecodeCoeffsIntra_MPEG4, 16-255
ippiDecodeCoeffsIntraRVLCBack_MPEG4, 16-257
ippiDecodeDCIntra_H263, 16-294
ippiDecodeDCIntra_MPEG4, 16-254
ippiDecodeExpGolombOne_H264, 16-345
ippiDecodeGetBufSize_JPEG2K, 15-166
ippiDecodeHuffman8x8_ACFirst_JPEG, 15-113
ippiDecodeHuffman8x8_ACRefine_JPEG, 15-114
ippiDecodeHuffman8x8_DCFirst_JPEG, 15-111
ippiDecodeHuffman8x8_DCRefine_JPEG, 15-112
ippiDecodeHuffman8x8_Direct_JPEG, 15-109
ippiDecodeHuffman8x8_JPEG, 15-108
ippiDecodeHuffmanOne, 16-22
ippiDecodeHuffmanOne_JPEG, 15-123
ippiDecodeHuffmanPair, 16-23
ippiDecodeHuffmanRow_JPEG, 15-125
ippiDecodeHuffmanSpecFree_JPEG, 15-105
ippiDecodeHuffmanSpecGetBufSize_JPEG, 15-102
ippiDecodeHuffmanSpecInit_JPEG, 15-103
ippiDecodeHuffmanSpecInitAlloc_JPEG, 15-104
ippiDecodeHuffmanStateFree_JPEG, 15-107
ippiDecodeHuffmanStateGetBufSize_JPEG, 15-105
ippiDecodeHuffmanStateInit_JPEG, 15-106
ippiDecodeHuffmanStateInitAlloc_JPEG, 15-107
ippiDecodeLumaBlockInter_AVS, 16-480
ippiDecodeLumaBlockIntra_AVS, 16-479
ippiDecodeTextureBlockFromRGBA, 15-190
ippiDeconvFFT, 9-83
ippiDeconvFFTFree, 9-82
ippiDeconvFFTInitAlloc, 9-81
ippiDeconvLR, 9-85
ippiDeconvLRFree, 9-85
ippiDeconvLRInitAlloc, 9-84
ippiDeinterlaceBlend, 16-134
ippiDeinterlaceBlendFree, 16-133
ippiDeinterlaceBlendInitAlloc, 16-132
ippiDeinterlaceEdgeDetect, 16-129
ippiDeinterlaceFilterCAVT, 16-126
ippiDeinterlaceFilterTriangle, 16-125
ippiDeinterlaceMedianThreshold, 16-128
ippiDeinterlaceMotionAdaptive, 16-130

ippiDemosaicAHD, 6-149
ippiDequantTransformResidual_H264, 16-349
ippiDequantTransformResidual_SISP_H264, 16-353
ippiDequantTransformResidualAndAdd_H264, 16-350
ippiDFTFree, 10-16
ippiDFTFwd, 10-18
ippiDFTGetBufSize, 10-17
ippiDFTInitAlloc, 10-15
ippiDFTInv, 10-21
ippiDiffPredFirstRow_JPEG, 15-116
ippiDiffPredRow_JPEG, 15-118
ippiDilate, 8-11
ippiDilate3x3, 8-7
ippiDilateBorderReplicate, 8-19
ippiDisassembleChroma420Intra_AVS, 16-502
ippiDisassembleLumaIntra_AVS, 16-500
ippiDistanceTransform, 14-24
ippiDiv, 5-32
ippiDiv_Round, 5-35
ippiDivC, 5-37
ippiDotProd, 5-54
ippiDotProdCol, 5-56, 9-67
ippiDownsampleFour_H263, 16-306
ippiDownsampleFour16x16_H263, 16-324
ippiDup, 4-31
ippiEigenValsVecs, 14-13
ippiEigenValsVecsGetBufferSize, 14-12
ippiEncodeChromaDcCoeffsCAVLC_H264_16s, 16-472
ippiEncodeCoeffsCAVLC_H264_16s, 16-469
ippiEncodeCoeffsInter_H261, 16-285
ippiEncodeCoeffsInter_H263, 16-320
ippiEncodeCoeffsInter_MPEG4, 16-271
ippiEncodeCoeffsIntra_H261, 16-284
ippiEncodeCoeffsIntra_H263, 16-319
ippiEncodeCoeffsIntra_MPEG4, 16-270
ippiEncodeDCIntra_H263, 16-318
ippiEncodeDCIntra_MPEG4, 16-269
ippiEncodeFree_JPEG2K, 15-157
ippiEncodeGetDist_JPEG2K, 15-165
ippiEncodeGetRate_JPEG2K, 15-164
ippiEncodeGetTermPassLen_JPEG2K, 15-163
ippiEncodeHuffman8x8_ACFirst_JPEG, 15-99
ippiEncodeHuffman8x8_ACRefine_JPEG, 15-101
ippiEncodeHuffman8x8_DCFirst_JPEG, 15-97
ippiEncodeHuffman8x8_DCRefine_JPEG, 15-98
ippiEncodeHuffman8x8_Direct_JPEG, 15-92
ippiEncodeHuffman8x8_JPEG, 15-91
ippiEncodeHuffmanOne_JPEG, 15-121
ippiEncodeHuffmanRawTableInit_JPEG, 15-84
ippiEncodeHuffmanRow_JPEG, 15-124
ippiEncodeHuffmanSpecFree_JPEG, 15-88
ippiEncodeHuffmanSpecGetBufSize_JPEG, 15-85
ippiEncodeHuffmanSpecInit_JPEG, 15-86
ippiEncodeHuffmanSpecInitAlloc_JPEG, 15-87
ippiEncodeHuffmanStateFree_JPEG, 15-90
ippiEncodeHuffmanStateGetBufSize_JPEG, 15-88
ippiEncodeHuffmanStateInit_JPEG, 15-89
ippiEncodeHuffmanStateInitAlloc_JPEG, 15-90
ippiEncodeInitAlloc_JPEG2K, 15-157
ippiEncodeLoadCodeBlock_JPEG2K, 15-158
ippiEncodeStoreBits_JPEG2K, 15-161
ippiEncodeTableInitAlloc, 16-178
ippiEncodeTextureBlockFromRGBA, 15-189
ippiErode, 8-13
ippiErode3x3, 8-9
ippiErodeBorderReplicate, 8-21
ippiExp, 5-51
ippiExpandPlane_H264, 16-366
ippiFFTFree, 10-7
ippiFFTFwd, 10-9
ippiFFTGetBufSize, 10-8
ippiFFTInitAlloc, 10-6
ippiFFTInv, 10-13
ippiFilter, 9-40
ippiFilter_Round16s, 9-45
ippiFilter_Round32f, 9-45
ippiFilter_Round32s, 9-45
ippiFilter32f, 9-44
ippiFilter8x8_H261, 16-286
ippiFilterBilateral, 9-24
ippiFilterBilateralGetBufSize, 9-21
ippiFilterBilateralInit, 9-22
ippiFilterBlockBoundaryHorEdge_H263, 16-309
ippiFilterBlockBoundaryVerEdge_H263, 16-309
ippiFilterBox, 9-8
ippiFilterColumn, 9-51
ippiFilterColumn32f, 9-54
ippiFilterColumnPipeline, 9-64
ippiFilterColumnPipeline_Low, 9-66
ippiFilterColumnPipelineGetBufferSize, 9-60

ippiFilterColumnPipelineGetBufferSize_Low, 9-60
ippiFilterDeblocking16x16_HorEdge_H263, 16-312
ippiFilterDeblocking16x16_VerEdge_H263, 16-312
ippiFilterDeblocking8x8HorEdge_H263, 16-310
ippiFilterDeblocking8x8HorEdge_MPEG4, 16-261
ippiFilterDeblocking8x8VerEdge_H263, 16-310
ippiFilterDeblocking8x8VerEdge_MPEG4, 16-261
ippiFilterDeblockingChroma_HorEdge_AVS, 16-496
ippiFilterDeblockingChroma_HorEdge_H264, 16-453
ippiFilterDeblockingChroma_HorEdge_VC1, 16-535
ippiFilterDeblockingChroma_VerEdge_AVS, 16-494
ippiFilterDeblockingChroma_VerEdge_H264, 16-447
ippiFilterDeblockingChroma_VerEdge_MBAFF_H264, 16-451
ippiFilterDeblockingChroma_VerEdge_VC1, 16-534
ippiFilterDeblockingChroma422HorEdge_H264, 16-453
ippiFilterDeblockingChroma422VerEdge_H264, 16-447
ippiFilterDeblockingChroma422VerEdge_MBAFF_H264, 16-451
ippiFilterDeblockingLuma_HorEdge_AVS, 16-492
ippiFilterDeblockingLuma_HorEdge_H264, 16-445
ippiFilterDeblockingLuma_HorEdge_VC1, 16-533
ippiFilterDeblockingLuma_VerEdge_AVS, 16-490
ippiFilterDeblockingLuma_VerEdge_H264, 16-441
ippiFilterDeblockingLuma_VerEdge_MBAFF_H264, 16-443
ippiFilterDeblockingLuma_VerEdge_VC1, 16-532
ippiFilterDenoiseAdaptive, 16-141
ippiFilterDenoiseAdaptiveFree, 16-141
ippiFilterDenoiseAdaptiveInitAlloc, 16-140
ippiFilterDenoiseCAST, 16-137
ippiFilterDenoiseCASTInit, 16-136
ippiFilterDenoiseMosquito, 16-145
ippiFilterDenoiseMosquitoFree, 16-145
ippiFilterDenoiseMosquitoInitAlloc, 16-144
ippiFilterDenoiseSmooth, 16-139
ippiFilterDeringingSmooth8x8_MPEG4, 16-263
ippiFilterDeringingThreshold_MPEG4, 16-262
ippiFilterGauss, 9-107
ippiFilterGaussBorder, 9-142
ippiFilterGetBufSize, 9-43
ippiFilterHipass, 9-110
ippiFilterLaplace, 9-105
ippiFilterLaplacianBorder, 9-140
ippiFilterLowpass, 9-112
ippiFilterLowpassBorder, 9-143
ippiFilterLowpassGetBufferSize, 9-123
ippiFilterMax, 9-14
ippiFilterMaxBorderReplicate, 9-19
ippiFilterMaxGetBufferSize, 9-17
ippiFilterMedian, 9-31
ippiFilterMedianColor, 9-38
ippiFilterMedianCross, 9-35
ippiFilterMedianHoriz, 9-33
ippiFilterMedianVert, 9-34
ippiFilterMedianWeightedCenter3x3, 9-37
ippiFilterMin, 9-12
ippiFilterMinBorderReplicate, 9-18
ippiFilterMinGetBufferSize, 9-16
ippiFilterPrewittHoriz, 9-88
ippiFilterPrewittVert, 9-89
ippiFilterRobertsDown, 9-103
ippiFilterRobertsUp, 9-104
ippiFilterRoundGetBufSize16s, 9-48
ippiFilterRoundGetBufSize32f, 9-48
ippiFilterRoundGetBufSize32s, 9-48
ippiFilterRow, 9-55
ippiFilterRow32f, 9-57
ippiFilterRowBorderPipeline, 9-61
ippiFilterRowBorderPipeline_Low, 9-61
ippiFilterRowBorderPipelineGetBufferSize, 9-58
ippiFilterRowBorderPipelineGetBufferSize_Low, 9-58
ippiFilterScharrHoriz, 9-91
ippiFilterScharrHorizBorder, 9-125
ippiFilterScharrHorizGetBufferSize, 9-115
ippiFilterScharrVert, 9-92
ippiFilterScharrVertBorder, 9-127
ippiFilterScharrVertGetBufferSize, 9-116
ippiFilterSharpen, 9-113
ippiFilterSobelCross, 9-101
ippiFilterSobelCrossBorder, 9-138
ippiFilterSobelCrossGetBufferSize, 9-121
ippiFilterSobelHoriz, 9-94
ippiFilterSobelHorizBorder, 9-128
ippiFilterSobelHorizGetBufferSize, 9-117
ippiFilterSobelHorizMask, 9-94
ippiFilterSobelHorizSecond, 9-97
ippiFilterSobelHorizSecondBorder, 9-133

ippiFilterSobelHorizSecondGetBufferSize, 9-119
ippiFilterSobelNegVertBorder, 9-130
ippiFilterSobelNegVertGetBufferSize, 9-118
ippiFilterSobelVert, 9-95
ippiFilterSobelVertBorder, 9-130
ippiFilterSobelVertGetBufferSize, 9-118
ippiFilterSobelVertMask, 9-96
ippiFilterSobelVertSecond, 9-99
ippiFilterSobelVertSecondBorder, 9-135
ippiFilterSobelVertSecondGetBufferSize, 9-120
ippiFilterWiener, 9-69
ippiFilterWienerGetBufferSize, 9-68
ippiFloodFill_Grad, 14-40
ippiFloodFill_Range, 14-43
ippiFloodFillGetSize, 14-36
ippiFloodFillGetSize_Grad, 14-37
ippiForegroundGaussian, 14-101
ippiForegroundGaussianFree, 14-100
ippiForegroundGaussianIInitAlloc, 14-98
ippiForegroundHistogram, 14-95
ippiForegroundHistogramFree, 14-94
ippiForegroundHistogramInitAlloc, 14-92
ippiForegroundHistogramUpdate, 14-97
ippiFrameFieldSAD16x16, 16-112
ippiFree, 3-6
ippiFreeHuffmanTable_DV, 16-204
ippiGammaFwd, 6-203
ippiGammaInv, 6-206
ippiGenScaleLevel8x8_H264, 16-468
ippiGenSobelKernel, 9-124
ippiGetAffineBound, 12-61
ippiGetAffineQuad, 12-60
ippiGetAffineTransform, 12-62
ippiGetBilinearBound, 12-87
ippiGetBilinearQuad, 12-86
ippiGetBilinearTransform, 12-88
ippiGetCentralMoment, 11-55
ippiGetDiff16x16, 16-63
ippiGetDiff16x16B, 16-70
ippiGetDiff16x8, 16-64
ippiGetDiff16x8B, 16-71
ippiGetDiff4x4, 16-69
ippiGetDiff8x16, 16-66
ippiGetDiff8x16B, 16-74
ippiGetDiff8x4, 16-68
ippiGetDiff8x4B, 16-75
ippiGetDiff8x8, 16-65
ippiGetDiff8x8B, 16-73
ippiGetDistanceTransformMask, 14-27
ippiGetHaarClassifierSize, 14-109
ippiGetHuffmanStatistics8x8_ACFirst_JPEG, 15-95
ippiGetHuffmanStatistics8x8_ACRefine_JPEG, 15-96
ippiGetHuffmanStatistics8x8_DCFirst_JPEG, 15-94
ippiGetHuffmanStatistics8x8_JPEG, 15-93
ippiGetHuffmanStatisticsOne_JPEG, 15-121
ippiGetHuMoments, 11-58
ippiGetLibVersion, 3-2
ippiGetNormalizedCentralMoment, 11-56
ippiGetNormalizedSpatialMoment, 11-54
ippiGetPerspectiveBound, 12-74
ippiGetPerspectiveQuad, 12-72
ippiGetPerspectiveTransform, 12-75
ippiGetPyramidDownROI, 14-68
ippiGetResizeFract, 12-20
ippiGetRotateBound, 12-40
ippiGetRotateQuad, 12-39
ippiGetRotateShift, 12-37
ippiGetShearBound, 12-50
ippiGetShearQuad, 12-49
ippiGetSpatialMoment, 11-53
ippiGradientColorToGray, 14-34
ippiGrayDilateBorder, 8-39, 8-40
ippiHaarClassifierFree, 14-108
ippiHaarClassifierInitAlloc, 14-106
ippiHistogramEven, 11-27
ippiHistogramRange, 11-24
ippiHLSToBGR, 6-130
ippiHLSToRGB, 6-127
ippiHoughLine, 14-20
ippiHoughLine_Region, 14-21
ippiHoughLineGetSize, 14-19
ippiHSVToRGB, 6-133
ippiHuffmanDecodeSegment_DV, 16-196
ippiHuffmanDecodeSegment_DV100, 16-201
ippiHuffmanDecodeSegmentOnePass_DV, 16-203
ippiHuffmanRunLevelTableInitAlloc, 16-21
ippiHuffmanTableFree, 16-24
ippiHuffmanTableInitAlloc, 16-20
ippiICTFwd_JPEG2K, 15-180
ippiICTInv_JPEG2K, 15-182
ippiImageJaehne, 4-40

ippiImageRamp, 4-42
ippiInitAllocHuffmanTable_DV, 16-192
ippiInpaint, 14-78
ippiInpaintFree, 14-78
ippiInpaintInitAlloc, 14-76
ippiIntegral, 11-7
ippiInterpolateAverage16x16, 16-55
ippiInterpolateAverage16x8, 16-55
ippiInterpolateAverage8x4, 16-55
ippiInterpolateAverage8x8, 16-55
ippiInterpolateBlock_H264, 16-390
ippiInterpolateChroma_H264, 16-379
ippiInterpolateChromaBlock_H264, 16-388
ippiInterpolateChromaBottom_H264, 16-386
ippiInterpolateChromaTop_H264, 16-384
ippiInterpolateLuma_H264, 16-367
ippiInterpolateLumaBlock_AVS, 16-482
ippiInterpolateLumaBlock_H264, 16-377
ippiInterpolateLumaBottom_H264, 16-374
ippiInterpolateLumaTop_H264, 16-371
ippiInterpolateQPBicubic_VC1, 16-522
ippiInterpolateQPBilinear_VC1, 16-520
ippiJoin422LS_MCU, 15-81
ippiLabelMarkers, 14-82
ippiLabelMarkersGetBufferSize, 14-81
ippiLabToBGR, 6-121
ippiFilterGaussGetBufferSize, 9-123
ippiFilterLaplacianGetBufferSize, 9-122
ippiLn, 5-48
ippiLShiftC, 5-71
ippiLUT, 6-211
ippiLUT_Cubic, 6-219
ippiLUT_Linear, 6-215
ippiLUTPalette, 6-224
ippiLUTPaletteSwap, 6-224
ippiLUVToRGB, 6-118
ippiMagnitude, 10-29
ippiMagnitudePack, 10-30
ippiMalloc, 3-5
ippiMax, 11-34
ippiMaxIndx, 11-35
ippiMC16x16, 16-26
ippiMC16x16B, 16-39
ippiMC16x4B, 16-50
ippiMC16x8, 16-27
ippiMC16x8B, 16-40
ippiMC16x8BUV, 16-52
ippiMC16x8UV, 16-37
ippiMC2x2, 16-35
ippiMC2x2B, 16-49
ippiMC2x4, 16-33
ippiMC2x4B, 16-47
ippiMC4x2, 16-34
ippiMC4x2B, 16-48
ippiMC4x4, 16-32
ippiMC4x4B, 16-46
ippiMC4x8, 16-31
ippiMC4x8B, 16-44
ippiMC8x16, 16-28
ippiMC8x16B, 16-41
ippiMC8x4, 16-30, 16-36
ippiMC8x4B, 16-43
ippiMC8x8, 16-29
ippiMC8x8B, 16-42
ippiMean, 11-15
ippiMean_StdDev, 11-18
ippiMeanAbsDev16x16, 16-90
ippiMeanAbsDev8x8, 16-89
ippiMedian, 16-127
ippiMin, 11-30
ippiMinEigenVal, 14-17
ippiMinEigenValGetBufferSize, 14-16
ippiMinIndx, 11-32
ippiMinMax, 11-37
ippiMinMaxIndx, 11-38
ippiMirror, 12-28
ippiMomentFree, 11-49
ippiMomentGetStateSize, 11-50
ippiMomentInit, 11-50
ippiMomentInitAlloc, 11-48
ippiMoments, 11-51
ippiMorpAdvGetSize, 8-26
ippiMorpGrayGetSize, 8-38
ippiMorphAdvFree, 8-24
ippiMorphAdvInit, 8-25, 8-37
ippiMorphAdvInitAlloc, 8-23, 8-35
ippiMorphBlackhatBorder, 8-32
ippiMorphGradientBorder, 8-33
ippiMorphGrayFree, 8-36
ippiMorphologyFree, 8-16
ippiMorphologyGetSize, 8-18
ippiMorphologyInit, 8-17

ippiMorphologyInitAlloc, 8-15
ippiMorphOpenBorder, 8-27, 8-29
ippiMorphReconstructErode, 8-46
ippiMorphReconstructGetBufferSize, 8-42
ippiMorphTophatBorder, 8-30
ippiMul, 5-16
ippiMulC, 5-18
ippiMulCScale, 5-24
ippiMulPack, 10-23
ippiMulPackConj, 10-27
ippiMulScale, 5-22
ippiNorm_Inf, 11-60
ippiNorm_L1, 11-62
ippiNorm_L2, 11-65
ippiNormDiff_Inf, 11-67
ippiNormDiff_L1, 11-70
ippiNormDiff_L2, 11-73
ippiNormRel_Inf, 11-76
ippiNormRel_L1, 11-78
ippiNormRel_L2, 11-81
ippiNot, 5-61
ippiOBMC16x16HP_MPEG4, 16-243
ippiOBMC8x8HP_MPEG4, 16-243
ippiOBMC8x8QP_MPEG4, 16-243
ippiOpticalFlowPyrLKFree, 14-49
ippiOpticalFlowPyrLKInitAlloc, 14-48
ippiOr, 5-62
ippiOrC, 5-64
ippiPackToCplxExtend, 10-35
ippiPhase, 10-31
ippiPhasePack, 10-32
ippiPredictIntra_16x16_H264, 16-359
ippiPredictIntra_4x4_H264, 16-355
ippiPredictIntraChroma8x8_H264, 16-360
ippiPutIntraBlock, 16-179
ippiPutNonIntraBlock, 16-180
ippiPyramidFree, 14-62
ippiPyramidInitAlloc, 14-61
ippiPyramidLayerDown, 14-70
ippiPyramidLayerDownFree, 14-65
ippiPyramidLayerUp, 14-72
ippiPyramidLayerUpFree, 14-67
ippiPyramidLayerUpInitAlloc, 14-65
ippiPyrDown, 14-57
ippiPyrDownGetBufSize, 14-55
ippiPyrUp, 14-58
ippiPyrUpGetBufSize, 14-56
ippiQualityIndex, 11-86
ippiQuant_MPEG2, 16-177
ippiQuantFwd8x8_JPEG, 15-49
ippiQuantFwdRawTableInit_JPEG, 15-47
ippiQuantFwdTableInit_JPEG, 15-48
ippiQuantInter_H263, 16-323
ippiQuantInter_MPEG4, 16-267
ippiQuantInterGetSize_MPEG4, 16-267
ippiQuantInterInit_MPEG4, 16-266
ippiQuantInterNonuniform_VC1, 16-557
ippiQuantInterUniform_VC1, 16-557
ippiQuantIntra_H263, 16-322
ippiQuantIntra_MPEG2, 16-176
ippiQuantIntra_MPEG4, 16-267
ippiQuantIntraGetSize_MPEG4, 16-267
ippiQuantIntraInit_MPEG4, 16-266
ippiQuantIntraNonuniform_VC1, 16-555
ippiQuantIntraUniform_VC1, 16-555
ippiQuantInv_DV, 16-204
ippiQuantInv_MPEG2, 16-166
ippiQuantInv8x8_JPEG, 15-52
ippiQuantInvInter_H263, 16-300
ippiQuantInvInter_MPEG4, 16-253
ippiQuantInvInterGetSize_MPEG4, 16-252
ippiQuantInvInterInit_MPEG4, 16-251
ippiQuantInvInterNonuniform_VC1, 16-540
ippiQuantInvInterUniform_VC1, 16-540
ippiQuantInvIntra_H263, 16-298
ippiQuantInvIntra_MPEG2, 16-165
ippiQuantInvIntra_MPEG4, 16-253
ippiQuantInvIntraGetSize_MPEG4, 16-252
ippiQuantInvIntraInit_MPEG4, 16-251
ippiQuantInvIntraNonuniform_VC1, 16-536
ippiQuantInvIntraUniform_VC1, 16-536
ippiQuantInvTableInit_JPEG, 15-51
ippiQuantizeResidual4x4Fwd_H264, 16-463
ippiQuantLuma8x8_H264, 16-467
ippiQuantLuma8x8Inv_H264, 16-473
ippiQuantWeightBlockInv_DV, 16-205
ippiQuantWeightBlockInv_DV100, 16-211
ippiRangeMapping_VC1, 16-544
ippiRCTFwd_JPEG2K, 15-176
ippiRCTInv_JPEG2K, 15-178
ippiReconstructChroma422Inter4x4_H264High, 16-409

ippiReconstructChroma422Intra4x4_H264High, 16-418
ippiReconstructChroma422IntraHalf4x4_H264High, 16-416
ippiReconstructChromaInter_AVS, 16-489
ippiReconstructChromaInter4x4_H264High, 16-405
ippiReconstructChromaInter4x4MB_H264, 16-405
ippiReconstructChromaInterMB_H264, 16-400
ippiReconstructChromaIntra_AVS, 16-488
ippiReconstructChromaIntra4x4_H264High, 16-413
ippiReconstructChromaIntra4x4MB_H264, 16-413
ippiReconstructChromaIntraHalf4x4High_H264, 16-411
ippiReconstructChromaIntraHalves4x4MB_H264, 16-411
ippiReconstructChromaIntraHalvesMB_H264, 16-401
ippiReconstructChromaIntraMB_H264, 16-403
ippiReconstructCoeffsInter_H261, 16-281
ippiReconstructCoeffsInter_H263, 16-315
ippiReconstructCoeffsInter_MPEG4, 16-260
ippiReconstructCoeffsIntra_H261, 16-279
ippiReconstructCoeffsIntra_H263, 16-313
ippiReconstructDCTBlock_MPEG1, 16-157
ippiReconstructDCTBlock_MPEG2, 16-160
ippiReconstructDCTBlockIntra_MPEG1, 16-158
ippiReconstructDCTBlockIntra_MPEG2, 16-163
ippiReconstructLumaInter_AVS, 16-486
ippiReconstructLumaInter4x4_H264High, 16-425
ippiReconstructLumaInter4x4MB_H264, 16-425
ippiReconstructLumaInter8x8_H264High, 16-432
ippiReconstructLumaInter8x8MB_H264, 16-432
ippiReconstructLumaIntera4x4MB_H264, 16-430
ippiReconstructLumaInteraHalf4x4MB_H264, 16-428
ippiReconstructLumaInterMB_H264, 16-421
ippiReconstructLumaIntra_AVS, 16-485
ippiReconstructLumaIntra16x16_H264High, 16-438
ippiReconstructLumaIntra16x16MB_H264, 16-437, 16-438
ippiReconstructLumaIntra4x4_H264High, 16-430
ippiReconstructLumaIntra8x8_H264High, 16-435
ippiReconstructLumaIntra8x8MB_H264, 16-435
ippiReconstructLumaIntraHalf4x4_H264High, 16-428
ippiReconstructLumaIntraHalf8x8_H264High, 16-433
ippiReconstructLumaIntraHalf8x8MB_H264, 16-433
ippiReconstructLumaIntraHalfMB_H264, 16-422
ippiReconstructLumaIntraMB_H264, 16-423
ippiReconstructPredFirstRow_JPEG, 15-119
ippiReconstructPredRow, 15-120
ippiRectStsDev, 11-20
ippiReduceBits, 6-209
ippiRemap, 12-30
ippiResample_H263, 16-303
ippiResampleRow, 4-52
ippiResampleRowGetBorderWidth, 4-51
ippiResampleRowGetSize, 4-49
ippiResampleRowInit, 4-50
ippiResampleRowReplicateBorder, 4-52
ippiResize, 12-5
ippiResizeCCRotate, 16-122
ippiResizeCenter, 12-8
ippiResizeFilter, 12-27
ippiResizeFilterGetSize, 12-25
ippiResizeFilterInit, 12-26
ippiResizeShift, 12-21
ippiResizeSqrPixel, 12-12
ippiResizeSqrPixelGetBufSize, 12-19
ippiResizeYUV422, 12-23
ippiRGB555ToYCbCr_JPEG, 15-9
ippiRGB565ToRGB, 6-152
ippiRGB565ToYCbCr_JPEG, 15-9
ippiRGB565ToYUV420, 6-45
ippiRGB565ToYUV422, 6-38
ippiRGBToCbYCr422, 6-71
ippiRGBToCbYCr422Gamma, 6-71
ippiRGBToGray, 6-143
ippiRGBToHLS, 6-126
ippiRGBToHSV, 6-132
ippiRGBToLUV, 6-115
ippiRGBToRGB565, 6-151
ippiRGBToXYZ, 6-113
ippiRGBToY_JPEG, 15-4
ippiRGBToYCbCr, 6-53
ippiRGBToYCbCr_JPEG, 15-7
ippiRGBToYCbCr411LS_MCU, 15-25
ippiRGBToYCbCr420, 6-85, 6-90
ippiRGBToYCbCr422, 6-63
ippiRGBToYCbCr422LS_MCU, 15-23
ippiRGBToYCbCr444LS_MCU, 15-22

ippiRGBToYCC, 6-123
ippiRGBToYUV, 6-31
ippiRGBToYUV420, 6-39
ippiRGBToYUV422, 6-35
ippiRotate, 12-34
ippiRotateCenter, 12-42
ippiRShiftC, 5-73
ippiSAD16x16, 16-92
ippiSAD16x16Blocks4x4, 16-100
ippiSAD16x16Blocks8x8, 16-99
ippiSAD16x8, 16-93
ippiSAD4x4, 16-98
ippiSAD4x8, 16-97
ippiSAD8x16, 16-94
ippiSAD8x4, 16-96
ippiSAD8x8, 16-95
ippiSampleDown411LS_MCU, 15-76
ippiSampleDown422LS_MCU, 15-75
ippiSampleDown444LS_MCU, 15-74
ippiSampleDownH2V1_JPEG, 15-65
ippiSampleDownH2V2_JPEG, 15-66
ippiSampleDownRowH2V1_Box_JPEG, 15-67
ippiSampleDownRowH2V2_Box_JPEG, 15-68
ippiSampleLine, 4-44
ippiSampleUp411LS_MCU, 15-79
ippiSampleUp444LS_MCU, 15-77
ippiSampleUpH2V1_JPEG, 15-69
ippiSampleUpH2V2_JPEG, 15-71
ippiSampleUpRowH2V1_Triangle_JPEG, 15-72
ippiSampleUpRowH2V2_Triangle_JPEG, 15-73
ippiSAT8x8D, 16-110
ippiSATD16x16, 16-101
ippiSATD16x8, 16-103
ippiSATD4x4, 16-109
ippiSATD4x8, 16-108
ippiSATD8x16, 16-104
ippiSATD8x4, 16-106
ippiSATD8x8, 16-105
ippiSBGRToYCoCg, 6-136
ippiSBGRToYCoCg_Rev, 6-140
ippiScale, 4-6
ippiScanFwd, 16-118
ippiScanInv, 16-117
ippiSegmentGradient, 14-89
ippiSegmentGradientGetBufferSize, 14-88
ippiSegmentWatershed, 14-84
ippiSegmentWatershedGetBufferSize, 14-83
ippiSet, 4-9
ippiShear, 12-45
ippiSmoothingChroma_HorEdge_VC1, 16-530
ippiSmoothingChroma_VerEdge_VC1, 16-528
ippiSmoothingLuma_HorEdge_VC1, 16-526
ippiSmoothingLuma_VerEdge_VC1, 16-523
ippiSpatialInterpolation_H263, 16-307
ippiSplit422LS_MCU, 15-80
ippiSqr, 5-44
ippiSqrDiff16x16, 16-79
ippiSqrDiff16x16B, 16-80
ippiSqrDistanceFull_Norm, 11-91
ippiSqrDistanceSame_Norm, 11-94
ippiSqrDistanceValid_Norm, 11-96
ippiSqrIntegral, 11-9
ippiSqrt, 5-46
ippiSRHNInitAlloc_PSF2x2, 14-124
ippiSRHNInitAlloc_PSF3x3, 14-124
ippiSSD4x4, 16-82
ippiSSD8x8, 16-81
ippiSub, 5-26
ippiSub128_JPEG, 15-62
ippiSub16x16, 16-76, 16-78
ippiSub8x8, 16-76
ippiSubC, 5-29
ippiSubSAD8x8, 16-78
ippiSum, 11-5
ippiSumsDiff16x16Blocks4x4, 16-113
ippiSumsDiff16x16Blocks4x4_8u16s_C1, 16-113
ippiSumsDiff8x8Blocks4x4, 16-115
ippiSumsDiff8x8Blocks4x4_8u16s_C1, 16-115
ippiSumWindowColumn, 9-11
ippiSumWindowRow, 9-10
ippiSwapChannels, 4-34
ippiThreshold, 7-2
ippiThreshold_GT, 7-5
ippiThreshold_GTVal, 7-11
ippiThreshold_LT, 7-7
ippiThreshold_LTVal, 7-13
ippiThreshold_LTValGTVal, 7-16
ippiThreshold_Val, 7-9
ippiTiltedHaarClassifierInitAlloc, 14-107
ippiTiltedIntegral, 11-11
ippiTiltedRectStdDev, 11-22
ippiTiltedSqrIntegral, 11-13

ippiToneMapMean, 6-227
ippiTransform4x4Fwd_VC1_16s_C1IR, 16-548
ippiTransform4x4Fwd_VC1_16s_C1R, 16-548
ippiTransform4x4Inv_VC1_16s_C1IR, 16-510
ippiTransform4x4Inv_VC1_16s_C1R, 16-511
ippiTransform4x8Fwd_VC1_16s_C1IR, 16-547
ippiTransform4x8Fwd_VC1_16s_C1R, 16-548
ippiTransform4x8Inv_VC1_16s_C1IR, 16-510
ippiTransform4x8Inv_VC1_16s_C1R, 16-511
ippiTransform8x4Fwd_VC1_16s_C1IR, 16-547
ippiTransform8x4Fwd_VC1_16s_C1R, 16-548
ippiTransform8x4Inv_VC1_16s_C1IR, 16-510
ippiTransform8x4Inv_VC1_16s_C1R, 16-511
ippiTransform8x8Fwd_VC1_16s_C1IR, 16-547
ippiTransform8x8Fwd_VC1_16s_C1R, 16-548
ippiTransform8x8Inv_VC1_16s_C1IR, 16-510
ippiTransform8x8Inv_VC1_16s_C1R, 16-510
ippiTransformDequantChromaDC_H264, 16-347
ippiTransformDequantChromaDC_SISP_H264, 16-354
ippiTransformDequantLumaDC_H264, 16-346
ippiTransformLuma8x8Fwd_H264, 16-466
ippiTransformLuma8x8InvAddPred_H264, 16-474
ippiTransformPrediction_H264, 16-352
ippiTransformQuant8x8Fwd_AVS, 16-499
ippiTransformQuantChromaDC_H264, 16-459
ippiTransformQuantLumaDC_H264, 16-461
ippiTransformQuantResidual_H264, 16-465
ippiTransformResidual4x4Fwd_H264, 16-462
ippiTransformResidual4x4Inv_H264, 16-348
ippiTranspose, 4-32
ippiUndistortGetSize, 14-116
ippiUndistortRadial, 14-117
ippiUnidirWeight_H264, 16-393
ippiUniDirWeightBlock_H264, 16-393
ippiUpdateMotionHistory, 14-47
ippiUpsampleFour_H263, 16-305
ippiUpsampleFour8x8_H263, 16-307
ippiVariance16x16, 16-88, 16-91
ippiVarMean8x8_8u32s, 16-83
ippiVarMeanDiff16x16, 16-85
ippiVarMeanDiff16x8, 16-86
ippiWarpAffine, 12-51
ippiWarpAffineBack, 12-54
ippiWarpAffineQuad, 12-57
ippiWarpBilinear, 12-77
ippiWarpBilinearBack, 12-80
ippiWarpBilinearQuad, 12-83
ippiWarpChroma_MPEG4, 16-248
ippiWarpGetSize_MPEG4, 16-246
ippiWarpInit_MPEG4, 16-245
ippiWarpLuma_MPEG4, 16-247
ippiWarpPerspective, 12-64
ippiWarpPerspectiveBack, 12-67
ippiWarpPerspectiveQuad, 12-70
ippiWeightedAverage_H264, 16-392
ippiWeightPrediction_AVS, 16-484
ippiWinBartlett, 10-38
ippiWinHumming, 10-40
ippiWinHummingSep, 10-40
ippiWTFwd, 13-9
ippiWTFwd_B53_JPEG2K, 15-148
ippiWTFwdCol_B53_JPEG2K, 15-132
ippiWTFwdCol_D97_JPEG2K, 15-141
ippiWTFwdColLift_B53_JPEG2K, 15-134
ippiWTFwdColLift_D97_JPEG2K, 15-142
ippiWTFwdFree, 13-7
ippiWTFwdGetBufSize, 13-8
ippiWTFwdInitAlloc, 13-5
ippiWTFwdRow_B53_JPEG2K, 15-129
ippiWTFwdRow_D97_JPEG2K, 15-138
ippiWTInv, 13-17
ippiWTInv_B53_JPEG2K, 15-150
ippiWTInv_D97_JPEG2K, 15-154
ippiWTInvCol_B53_JPEG2K, 15-135
ippiWTInvCol_D97_JPEG2K, 15-144
ippiWTInvColLift_B53_JPEG2K, 15-136
ippiWTInvColLift_D97_JPEG2K, 15-145
ippiWTInvFree, 13-15
ippiWTInvGetBufSize, 13-16
ippiWTInvInitAlloc, 13-13
ippiWTInvRow_B53_JPEG2K, 15-131
ippiWTInvRow_D97_JPEG2K, 15-139
ippiXor, 5-67
ippiXorC, 5-68
ippiXYZToRGB, 6-114
ippiYCbCr411, 6-188
ippiYCbCr411ToBGR, 6-109
ippiYCbCr411ToBGR555, 6-112
ippiYCbCr411ToBGR555LS_MCU, 15-43
ippiYCbCr411ToBGR565, 6-112
ippiYCbCr411ToBGR565LS_MCU, 15-43

ippiYCbCr411ToBGRLS_MCU, 15-42
ippiYCbCr411ToRGBLS_MCU, 15-37
ippiYCbCr411ToYCbCr420, 6-192
ippiYCbCr411ToYCbCr422, 6-189
ippiYCbCr411ToYCrCb420, 6-193
ippiYCbCr411ToYCrCb422, 6-191
ippiYCbCr420, 6-170
ippiYCbCr420To422_Interlace, 6-175
ippiYCbCr420ToBGR, 6-97
ippiYCbCr420ToBGR_709CSC, 6-99
ippiYCbCr420ToBGR_709HDTV, 6-100
ippiYCbCr420ToBGR444Dither, 6-105
ippiYCbCr420ToBGR555Dither, 6-105
ippiYCbCr420ToBGR565Dither, 6-105
ippiYCbCr420ToCbYCr422, 6-176
ippiYCbCr420ToCbYCr422_Interlace, 6-177
ippiYCbCr420ToRGB, 6-86
ippiYCbCr420ToRGB444Dither, 6-89
ippiYCbCr420ToRGB555Dither, 6-89
ippiYCbCr420ToRGB565Dither, 6-89
ippiYCbCr420ToYCbCr411, 6-181
ippiYCbCr420ToYCbCr422, 6-172
ippiYCbCr420ToYCbCr422_Filter, 6-173
ippiYCbCr420ToYCrCb420, 6-179
ippiYCbCr420ToYCrCb420_Filter, 6-180
ippiYCbCr422, 6-153
ippiYCbCr422To420_Interlace, 6-158
ippiYCbCr422ToBGR, 6-69
ippiYCbCr422ToBGR444, 6-82
ippiYCbCr422ToBGR444Dither, 6-83
ippiYCbCr422ToBGR555, 6-82
ippiYCbCr422ToBGR555Dither, 6-83
ippiYCbCr422ToBGR555LS_MCU, 15-41
ippiYCbCr422ToBGR565, 6-82
ippiYCbCr422ToBGR565Dither, 6-83
ippiYCbCr422ToBGR565LS_MCU, 15-41
ippiYCbCr422ToBGRLS_MCU, 15-40
ippiYCbCr422ToCbYCr422, 6-155
ippiYCbCr422ToRGB, 6-64, 6-65
ippiYCbCr422ToRGB_JPEG, 15-15
ippiYCbCr422ToRGB444, 6-79
ippiYCbCr422ToRGB444Dither, 6-80
ippiYCbCr422ToRGB555, 6-79
ippiYCbCr422ToRGB555Dither, 6-80
ippiYCbCr422ToRGB565, 6-79
ippiYCbCr422ToRGB565Dither, 6-80
ippiYCbCr422ToRGBLS_MCU, 15-36
ippiYCbCr422ToYCbCr411, 6-161
ippiYCbCr422ToYCbCr420, 6-156
ippiYCbCr422ToYCrCb420, 6-160
ippiYCbCr444ToBGR555LS_MCU, 15-39
ippiYCbCr444ToBGR565LS_MCU, 15-39
ippiYCbCr444ToBGRLS_MCU, 15-38
ippiYCbCr444ToRGBLS_MCU, 15-35
ippiYCbCrToBGR, 6-56
ippiYCbCrToBGR_709CSC, 6-101
ippiYCbCrToBGR_JPEG, 15-12
ippiYCbCrToBGR444, 6-60
ippiYCbCrToBGR444Dither, 6-61
ippiYCbCrToBGR555, 6-60
ippiYCbCrToBGR555_JPEG, 15-14
ippiYCbCrToBGR555Dither, 6-61
ippiYCbCrToBGR565, 6-60
ippiYCbCrToBGR565_JPEG, 15-14
ippiYCbCrToBGR565Dither, 6-61
ippiYCbCrToRGB, 6-54
ippiYCbCrToRGB_JPEG, 15-8
ippiYCbCrToRGB444, 6-57
ippiYCbCrToRGB444Dither, 6-58
ippiYCbCrToRGB555, 6-57
ippiYCbCrToRGB555_JPEG, 15-10
ippiYCbCrToRGB555Dither, 6-58
ippiYCbCrToRGB565, 6-57
ippiYCbCrToRGB565_JPEG, 15-10
ippiYCbCrToRGB565Dither, 6-58
ippiYCCK444ToCMYKLS_MCU, 15-44
ippiYCCKToCMYK_JPEG, 15-18
ippiYCCKToCMYK411LS_MCU, 15-46
ippiYCCKToCMYK422LS_MCU, 15-45
ippiYCCToRGB, 6-124
ippiYCiCgToSBGR, 6-138
ippiYCoCgToBGR, 6-137
ippiYCoCgToBGR_Rev, 6-141
ippiYCoCgToSBGR_Rev, 6-142
ippiYCrCb411ToYCbCr422_16x4x5MB_DV, 16-225
ippiYCrCb411ToYCbCr422_5MBDV, 16-224
ippiYCrCb411ToYCbCr422_8x8MB_DV, 16-225
ippiYCrCb411ToYCbCr422_EdgeDV, 16-227
ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV, 16-224
ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV, 16-227

ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV, 16-224
ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV, 16-227
ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV, 16-224
ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV, 16-227
ippiYCrCb420ToCbYCr422, 6-185
ippiYCrCb420ToRGB, 6-91
ippiYCrCb420ToYCbCr411, 6-187
ippiYCrCb420ToYCbCr420, 6-186
ippiYCrCb420ToYCbCr422, 6-183
ippiYCrCb420ToYCbCr422_5MBDV, 16-228
ippiYCrCb420ToYCbCr422_8x8x5MB_DV, 16-229
ippiYCrCb420ToYCbCr422_Filter, 6-184
ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV, 16-228
ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV, 16-228
ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV, 16-228
ippiYCrCb422ToRGB, 6-66
ippiYCrCb422ToYCbCr411, 6-164
ippiYCrCb422ToYCbCr420, 6-163
ippiYCrCb422ToYCbCr422, 6-162
ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100, 16-233
ippiYCrCb422ToYCbCr422_5MBDV, 16-230
ippiYCrCb422ToYCbCr422_8x4x5MB_DV, 16-232
ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV, 16-230
ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV, 16-230
ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV, 16-230
ippiYUV420ToBGR, 6-44
ippiYUV420ToBGR444, 6-50
ippiYUV420ToBGR444Dither, 6-51
ippiYUV420ToBGR555, 6-50
ippiYUV420ToBGR555Dither, 6-51
ippiYUV420ToBGR565, 6-50
ippiYUV420ToBGR565Dither, 6-51
ippiYUV420ToRGB, 6-41
ippiYUV420ToRGB444, 6-46
ippiYUV420ToRGB444Dither, 6-48
ippiYUV420ToRGB555, 6-46
ippiYUV420ToRGB555Dither, 6-48
ippiYUV420ToRGB565, 6-46
ippiYUV420ToRGB565Dither, 6-48
ippiYUV422ToRGB, 6-37
ippiYUVToRGB, 6-34
ippiZigzagFwd8x8, 4-45
ippiZigzagInv8x8, 4-47
ippjGetLibVersion, 15-2
ippsApplyHaarClassifier, 14-111

J

Join422LS_MCU, 15-81
JPEG 2000 Codec
  entropy coding and decoding functions, 15-155 thru ??
  wavelet transform functions, 15-127 thru ??
JPEG Codec
  color conversion functions, 15-3 thru 15-18
  combined color conversion functions, 15-20 thru 15-46
  combined quantization, DCT, and level shift functions, 15-53 thru 15-61
  entropy coding and decoding functions, ?? thru 15-168
  Huffman codec functions, 15-82 thru 15-115
  level shift functions, 15-62 thru 15-64
  library version, 15-2
  quantization functions, 15-47 thru 15-52
  sampling functions, 15-64 thru 15-72
  wavelet transform functions, ?? thru 15-155
JPEG coding
  color conversion functions
    BGR555ToYCbCr_JPEG, 15-13
    BGR565ToYCbCr_JPEG, 15-13
    BGRToY_JPEG, 15-6
    BGRToYCbCr_JPEG, 15-11
    CMYKToYCCK_JPEG, 15-16
    RGB555ToYCbCr_JPEG, 15-9
    RGB565ToYCbCr_JPEG, 15-9
    RGBToY_JPEG, 15-4
    RGBToYCbCr_JPEG, 15-7
    YCbCr422ToRGB_JPEG, 15-15
    YCbCrToBGR_JPEG, 15-12
    YCbCrToBGR555_JPEG, 15-14
    YCbCrToBGR565_JPEG, 15-14
    YCbCrToRGB_JPEG, 15-8
    YCbCrToRGB555_JPEG, 15-10

YCbCrToRGB565_JPEG, 15-10
YCCKToCMYK_JPEG, 15-18
combined DCT functions
DCTQuantFwd8x8_JPEG, 15-54
DCTQuantFwd8x8LS_JPEG, 15-55
DCTQuantInv8x8_JPEG, 15-57
DCTQuantInv8x8LS_1x1_JPEG, 15-59
DCTQuantInv8x8LS_2x2_JPEG, 15-59
DCTQuantInv8x8LS_4x4_JPEG, 15-59
DCTQuantInv8x8LS_JPEG, 15-58
DCTQuantInv8x8To2x2LS_JPEG, 15-60
DCTQuantInv8x8To4x4LS_JPEG, 15-60
level shift functions
Add128_JPEG, 15-63
Sub128_JPEG, 15-62
JPEG coding, lossless, 15-116

L

LabelMarkers, 14-82
LabelMarkersGetBufferSize, 14-81
LabToBGR, 6-121
level shift functions, for JPEG codec, 15-62
lines padding, 2-24
Ln, 5-48
logical functions
bitwise AND, 5-57
bitwise NOT, 5-61
bitwise OR, 5-62
bitwise XOR, 5-67
left shift, 5-71
right shift, 5-73
logical operations, 5-57
LShiftC, 5-71
LUT, 6-211
LUT_Cubic, 6-219
LUT_Linear, 6-215
LUTPalette, 6-224
LUTPaletteSwap, 6-224
LUVToRGB, 6-118

M

macroblock, 16-183, 16-276
Magnitude, 10-29
MagnitudePack, 10-30
Malloc, 3-5
manual organization, 1-4
mapping, of data range, 4-7
mask, in median filtering, 9-30
masked operation, 4-14
Max, 11-34
MaxIndx, 11-35
MC16x16, 16-26
MC16x16B, 16-39
MC16x4, 16-36
MC16x4B, 16-50
MC16x8, 16-27
MC16x8B, 16-40
MC16x8BUV, 16-52
MC16x8UV, 16-37
MC2x2, 16-35
MC2x2B, 16-49
MC2x4, 16-33
MC2x4B, 16-47
MC4x2, 16-34
MC4x2B, 16-48
MC4x4, 16-32
MC4x4B, 16-46
MC4x8, 16-31
MC4x8B, 16-44
MC8x16, 16-28
MC8x16B, 16-41
MC8x4, 16-30
MC8x4B, 16-43
MC8x8, 16-29
MC8x8B, 16-42
MCU, minimum coded units, 15-21
Mean, 11-15
Mean_StdDev, 11-18
MeanAbsDev16x16, 16-90
MeanAbsDev8x8, 16-89
Median, 16-127
median filters, 9-30
Memory allocation functions, 3-5 thru ??
memory allocation functions
ippiFree, 3-6
ippiMalloc, 3-5
Min, 11-30
MinEigenVal, 14-17
MinEigenValGetBufferSize, 14-16
MinIndx, 11-32
MinMax, 11-37
MinMaxIndx, 11-38

Mirror, 12-28
MMX technology, 1-1
MomentFree, 11-49
MomentGetStateSize, 11-50
MomentInit, 11-50
MomentInitAlloc, 11-48
Moments, 11-51
moments, of an image, 11-47
MorphAdvFree, 8-24
MorphAdvGetSize, 8-26
MorphAdvInit, 8-25
MorphAdvInitAlloc, 8-23
MorphBlackhatBorder, 8-32
MorphCloseBorder, 8-29
MorphGradientBorder, 8-33
MorphGrayFree, 8-36
MorphGrayGetSize, 8-38
MorphGrayInit, 8-37
MorphGrayInitAlloc, 8-35
morphological functions
  DilateBorderReplicate, 8-19
  dilation, 8-7
  ErodeBorderReplicate, 8-21
  erosion, 8-9
  GrayDilateBorder, 8-39, 8-40
  MorphAdvFree, 8-24
  MorphAdvGetSize, 8-26
  MorphBlackhatBorder, 8-32
  MorphCloseBorder, 8-29
  MorphGradientBorder, 8-33
  MorphGrayFree, 8-36
  MorphGrayGetSize, 8-38
  MorphologyFree, 8-16
  MorphOpenBorder, 8-27
  MorphReconstructErode, 8-46
  MorphReconstructGetBufferSize, 8-42
  MorphTophatBorder, 8-30
MorphologyFree, 8-16
MorphologyGetSize, 8-18
MorphologyInit, 8-17
MorphologyInitAlloc, 8-15
MorphOpenBorder, 8-27
MorphReconstructErode, 8-46
MorphReconstructGetBufferSize, 8-42
MorphTophatBorder, 8-30
motion Analysis and object tracking, 14-45
motion analysis and object tracking functions
  OpticalFlowPyrLK, 14-50
  OpticalFlowPyrLKFree, 14-49
  OpticalFlowPyrLKInitAlloc, 14-48
  UpdateMotionHistory, 14-47
motion compensation adding functions, 16-56, 16-57, 16-58
motion compensation averaging functions, 16-59
motion compensation copying functions, 16-53, 16-54
motion compensation interpolating functions, 16-55
motion estimation subtracting functions, 16-76, 16-78
MPEG4 video decoder functions
  CalcGlobalMV_MPEG4, 16-249
  ChangeSpriteBrightness_MPEG4, 16-250
  Copy16x16QP_MPEG4, 16-242
  Copy16x8QP_MPEG4, 16-242
  Copy8x8QP_MPEG4, 16-242
  DecodeCoeffsInter_MPEG4, 16-258
  DecodeCoeffsInterRVLCBack_MPEG4, 16-259
  DecodeCoeffsIntra_MPEG4, 16-255
  DecodeCoeffsIntraRVLCBack_MPEG4, 16-257
  DecodeDCIntra_MPEG4, 16-254
  EncodeCoeffsInter_MPEG4, 16-271
  EncodeCoeffsIntra_MPEG4, 16-270
  EncodeDCIntra_MPEG4, 16-269
  FilterDeblocking8x8HorEdge_MPEG4, 16-261
  FilterDeblocking8x8VerEdge_MPEG4, 16-261
  FilterDeringingSmooth8x8_MPEG4, 16-263
  FilterDeringingThreshold_MPEG4, 16-262
  OBMC16x16HP_MPEG4, 16-243
  OBMC8x8HP_MPEG4, 16-243
  OBMC8x8QP_MPEG4, 16-243
  QuantInter_MPEG4, 16-267
  QuantInterGetSize_MPEG4, 16-267
  QuantInterInit_MPEG4, 16-266
  QuantIntra_MPEG4, 16-267
  QuantIntraGetSize_MPEG4, 16-267
  QuantIntraInit_MPEG4, 16-266
  QuantInvInter_MPEG4, 16-253
  QuantInvInterGetSize_MPEG4, 16-252
  QuantInvInterInit_MPEG4, 16-251
  QuantInvIntra_MPEG4, 16-253
  QuantInvIntraGetSize_MPEG4, 16-252
  QuantInvIntraInit_MPEG4, 16-251

ReconstructCoeffsInter_MPEG4, 16-260
WarpChroma_MPEG4, 16-248
WarpGetSize_MPEG4, 16-246
WarpInit_MPEG4, 16-245
WarpLuma_MPEG4, 16-247
Mul, 5-16
MulC, 5-18
MulCScale, 5-24
MulPack, 10-23
MulPackConj, 10-27
MulScale, 5-22

N

naming conventions, 1-7
neigborhood operations, 2-22
Norm_Inf, 11-60
Norm_L1, 11-62
Norm_L2, 11-65
NormDiff_Inf, 11-67
NormDiff_L1, 11-70
NormDiff_L2, 11-73
NormRel_Inf, 11-76
NormRel_L1, 11-78
NormRel_L2, 11-81
norms, of an image, 11-60
Not, 5-61
notational conventions, 1-6

O

object detection functions
HaarClassifierFree, 14-108
HaarClassifierInitAlloc, 14-106
TiltedHaarClassifierInitAlloc, 14-107
OBMC16x16HP_MPEG4, 16-243
OBMC8x8HP_MPEG4, 16-243
OBMC8x8QP_MPEG4, 16-243
opening, 8-2
operands, 2-6
operation model, of image processing, 2-22
optical flow, 14-48
OpticalFlowPyrLK, 14-50
OpticalFlowPyrLKFree, 14-49
OpticalFlowPyrLKInitAlloc, 14-48
Or, 5-62
OrC, 5-64

P

PackToCplxExtend, 10-35
pattern recognition, 14-104
Phase, 10-31
PhasePack, 10-32
Photo YCC color model, 6-15
pixel order images, 2-20
planar order images, 2-20
platforms supported, 1-2
pOffset, 16-19
point operations, 2-22
ppBitStream, 16-19
PredictIntra_16x16_H264, 16-359
PredictIntra_4x4_H264, 16-355
PredictIntraChroma8x8_H264, 16-360
prefix
in data types, 2-5
in function names, 1-7
PutIntraBlock, 16-179
PutNonIntraBlock, 16-180
PyramidFree, 14-62
PyramidInitAlloc, 14-61
PyramidLayerDown, 14-70
PyramidLayerDownFree, 14-65
PyramidLayerUp, 14-72
PyramidLayerUpFree, 14-67
PyramidLayerUpInitAlloc, 14-65
pyramids function
PyramidFree, 14-62
PyramidLayerDownFree, 14-65
PyramidLayerUp, 14-72
PyramidLayerUpFree, 14-67
PyramidLayerUpInitAlloc, 14-65
PyrDown, 14-57
PyrDownGetBufSize, 14-55
PyrUp, 14-58
PyrUpGetBufSize, 14-56

Q

quality factor, in JPEG compression, 15-47
QualityIndex, 11-86
Quant_MPEG2, 16-177
QuantFwd8x8_JPEG, 15-49
QuantFwdRawTableInit_JPEG, 15-47
QuantFwdTableInit_JPEG, 15-48

QuantInter_H263, 16-323
QuantInter_MPEG4, 16-267
QuantInterGetSize_MPEG4, 16-267
QuantInterInit_MPEG4, 16-266
QuantInterNonuniform_VC1, 16-557
QuantInterUniform_VC1, 16-557
QuantIntra_H263, 16-322
QuantIntra_MPEG2, 16-176
QuantIntra_MPEG4, 16-267
QuantIntraGetSize_MPEG4, 16-267
QuantIntraInit_MPEG4, 16-266
QuantIntraNonuniform_VC1, 16-555
QuantIntraUniform_VC1, 16-555
QuantInv_DV, 16-204
QuantInv_MPEG2, 16-166
QuantInv8x8_JPEG, 15-52
QuantInvInter_H263, 16-300
QuantInvInter_MPEG4, 16-253
QuantInvInterGetSize_MPEG4, 16-252
QuantInvInterInit_MPEG4, 16-251
QuantInvInterNonuniform_VC1, 16-540
QuantInvInterUniform_VC1, 16-540
QuantInvIntra_H263, 16-298
QuantInvIntra_MPEG2, 16-165
QuantInvIntra_MPEG4, 16-253
QuantInvIntraGetSize_MPEG4, 16-252
QuantInvIntraInit_MPEG4, 16-251
QuantInvIntraNonuniform_VC1, 16-536
QuantInvIntraUniform_VC1, 16-536
QuantInvTableInit_JPEG, 15-51
quantization functions, for JPEG codec, 15-47
quantization table, 15-48
QuantizeResidual4x4Fwd_H264, 16-463
QuantLuma8x8_H264, 16-467
QuantLuma8x8Inv_H264, 16-473
QuantWeightBlockInv_DV, 16-205
QuantWeightBlockInv_DV100, 16-211

R

RangeMapping_VC1, 16-544
RCTFwd_JPEG2K, 15-176
RCTInv_JPEG2K, 15-178
ReconstructChroma422Inter4x4_H264High, 16-409
ReconstructChroma422Intra4x4_H264High, 16-418
ReconstructChroma422IntraHalf4x4_H264High, 16-416
ReconstructChromaInter_AVS, 16-489
ReconstructChromaInter4x4_H264High, 16-405
ReconstructChromaInter4x4MB_H264, 16-405
ReconstructChromaInterMB_H264, 16-400
ReconstructChromaIntra_AVS, 16-488
ReconstructChromaIntra4x4MB_H264, 16-413
ReconstructChromaIntraHalf4x4High_H264, 16-411
ReconstructChromaIntraHalves4x4MB_H264, 16-411
ReconstructChromaIntraHalvesMB_H264, 16-401
ReconstructChromaIntraMB_H264, 16-403
ReconstructCoeffsInter_H261, 16-281
ReconstructCoeffsInter_H263, 16-315
ReconstructCoeffsInter_MPEG4, 16-260
ReconstructCoeffsIntra_H261, 16-279
ReconstructCoeffsIntra_H263, 16-313
ReconstructDCTBlock_MPEG1, 16-157
ReconstructDCTBlock_MPEG2, 16-160
ReconstructDCTBlockIntra_MPEG1, 16-158
ReconstructDCTBlockIntra_MPEG2, 16-163
ReconstructLumaInter_AVS, 16-486
ReconstructLumaInter4x4_H264High, 16-425
ReconstructLumaInter4x4MB_H264, 16-425
ReconstructLumaInter8x8_H264High, 16-432
ReconstructLumaInter8x8MB_H264, 16-432
ReconstructLumaInterMB_H264, 16-421
ReconstructLumaIntra_AVS, 16-485
ReconstructLumaIntra16x16_H264High, 16-438
ReconstructLumaIntra16x16MB_H264, 16-437, 16-438
ReconstructLumaIntra4x4_H264High, 16-430
ReconstructLumaIntra4x4MB_H264, 16-430
ReconstructLumaIntra8x8_H264High, 16-435
ReconstructLumaIntra8x8MB_H264, 16-435
ReconstructLumaIntraHalf4x4_H264High, 16-428
ReconstructLumaIntraHalf4x4MB_H264, 16-428
ReconstructLumaIntraHalf8x8_H264High, 16-433
ReconstructLumaIntraHalf8x8MB_H264, 16-433
ReconstructLumaIntraHalfMB_H264, 16-422
ReconstructLumaIntraMB_H264, 16-423
ReconstructPredFirstRow_JPEG, 15-119
ReconstructPredRow, 15-120
RectStdDev, 11-20

ReduceBits, 6-209
region of interest, 2-22
    in source and destination image, 2-23
    offset, 2-24
    size, 2-23
related publications, 1-6
Remap, 12-30
Resample_H263, 16-303
ResampleRow, 4-52
ResampleRowGetBorderWidth, 4-51
ResampleRowGetSize, 4-49
ResampleRowInit, 4-50
ResampleRowReplicateBorder, 4-52
Resize, 12-5
ResizeCCRotate, 16-122
ResizeCenter, 12-8
ResizeFilter, 12-27
ResizeFilterGetSize, 12-25
ResizeFilterInit, 12-26
ResizeShift, 12-21
ResizeSqrPixel, 12-12
ResizeSqrPixelGetBufSize, 12-19
ResizeYUV422, 12-23
RGB color model, 6-10
RGB555ToYCbCr_JPEG, 15-9
RGB565ToRGB, 6-152
RGB565ToYCbCr_JPEG, 15-9
RGB565ToYUV420, 6-45
RGB565ToYUV422, 6-38
RGBToCbYCr422, 6-71
RGBToCbYCr422Gamma, 6-71
RGBToGray, 6-143
RGBToHLS, 6-126
RGBToHSV, 6-132
RGBToLUV, 6-115
RGBToRGB565, 6-151
RGBToXYZ, 6-113
RGBToY_JPEG, 15-4
RGBToYCbCr, 6-53
RGBToYCbCr_JPEG, 15-7
RGBToYCbCr411LS_MCU, 15-25
RGBToYCbCr420, 6-85, 6-90
RGBToYCbCr422, 6-63
RGBToYCbCr422LS_MCU, 15-23
RGBToYCbCr444LS_MCU, 15-22
RGBToYCC, 6-123
RGBToYCrCb422, 6-65
RGBToYUV, 6-31
RGBToYUV420, 6-39
RGBToYUV422, 6-35
RLE coding functions, 15-183
    JoinRow_TIFF, 15-187
    PackBitsRow_TIFF, 15-184
    SplitRow_TIFF, 15-186
    UnpackBitsRow_TIFF, 15-185
ROI processing, in geometric transforms, 12-4
ROI. See region of interest
Rotate, 12-34
RotateCenter, 12-42
RShiftC, 5-73

S

S3 Texture Compression, 15-188
SAD16x16, 16-92
SAD16x16Blocks4x4, 16-100
SAD16x16Blocks8x8, 16-99
SAD16x8, 16-93
SAD4x4, 16-98
SAD4x8, 16-97
SAD8x16, 16-94
SAD8x4, 16-96
SAD8x8, 16-95
SampleDown411LS_MCU, 15-76
SampleDown422LS_MCU, 15-75
SampleDown444LS_MCU, 15-74
SampleDownH2V1_JPEG, 15-65
SampleDownH2V2_JPEG, 15-66
SampleDownRowH2V1_Box_JPEG, 15-67
SampleDownRowH2V2_Box_JPEG, 15-68
SampleLine, 4-44
SampleUp411LS_MCU, 15-79
SampleUp444LS_MCU, 15-77
SampleUpH2V1_JPEG, 15-69
SampleUpH2V2_JPEG, 15-71
SampleUpRowH2V1_Triangle_JPEG, 15-72
SampleUpRowH2V2_Triangle_JPEG, 15-73
sampling functions, for JPEG codec, 15-64
SAT8x8D, 16-110
SATD16x16, 16-101
SATD16x8, 16-103
SATD4x4, 16-109

SATD4x8, 16-108
SATD8x16, 16-104
SATD8x4, 16-106
SATD8x8, 16-105
SBGRToYCoCg, 6-136
SBGRToYCoCg_Rev, 6-140
Scale, 4-6
scaling, of output results, 2-8
ScanFwd, 16-118
ScanInv, 16-117
scanline padding, 12-4
scanning
    ScanFwd, 16-118
    ScanInv, 16-117
segment, 16-185
SegmentGradient, 14-89
SegmentGradientGetBufferSize, 14-88
SegmentWatershed, 14-84
SegmentWatershedGetBufferSize, 14-83
Set, 4-9
Shear, 12-45
SIMD instructions, 1-1
SmoothingChroma_HorEdge_VC1, 16-530
SmoothingChroma_VerEdge_VC1, 16-528
SmoothingLuma_HorEdge_VC1, 16-526
SmoothingLuma_VerEdge_VC1, 16-523
SpatialInterpolation_H263, 16-307
Split422LS_MCU, 15-80
Sqr, 5-44
SqrDiff16x16, 16-79
SqrDiff16x16B, 16-80
SqrDistanceFull_Norm, 11-91
SqrDistanceSame_Norm, 11-94
SqrDistanceValid_Norm, 11-96
SqrIntegral, 11-9
Sqrt, 5-46
SRHNInitAlloc_PSF2x2, 14-124
SRHNInitAlloc_PSF3x3, 14-124
SSD4x4, 16-82
SSD8x8, 16-81
statistics functions
    computing image moments, 11-51
    context initializing, 11-48
    deallocation, 11-49
    HistogramEven, 11-27
    HistogramRange, 11-24
    image norms
        infinity norm, 11-60
        infinity norm of differences, 11-67
        L1- norm, 11-62
        L1- norm of differences, 11-70
        L2- norm, 11-65
        L2- norm of differences, 11-73
        relative error, 11-76
    integral of pixel squares, 11-9
    integral representation, 11-7
    maximum, 11-34
    mean and stdev value, 11-18
    mean value, 11-15
    minimum, 11-30
    minimum and maximum, 11-37
    pixels count, 11-29
    retrieving central moments, 11-55
    retrieving Hu moments, 11-58
    retrieving spatial moments, 11-53
    standard deviation of integral images, 11-20
    standard deviation of tilted integral images, 11-22
    sum, 11-5
    tilted integral of pixel squares, 11-13
    tilted integral representation, 11-11
status codes, 2-9
status information function
    ippGetStatusString, 3-4
step
    through the buffer, 2-23
Streaming SIMD Extensions, 1-1
structures in IPP, 2-14
Sub, 5-26
Sub128_JPEG, 15-62
Sub16x16, 16-76, 16-78
Sub8x8, 16-76
SubC, 5-29
SubSAD8x8, 16-78
Sum, 11-5
SumsDiff16x16Blocks4x4, 16-113
SumsDiff8x8Blocks4x4, 16-115
SumWindowColumn, 9-11
SumWindowRow, 9-10
superblock, 16-184
support functions, 3-1
SwapChannels, 4-34

T

texture compression, 15-188
Threshold, 7-2
threshold functions, 7-2
Threshold_GT, 7-5
Threshold_GTVal, 7-11
Threshold_LT, 7-7
Threshold_LTVal, 7-13
Threshold_LTValGTVal, 7-16
Threshold_Val, 7-9
tiled image processing, 2-26
Tilted RectStdDev, 11-22
TiltedHaarClassifierInitAlloc, 14-107
TiltedIntegral, 11-11
TiltedSqrIntegral, 11-13
ToneMapMean, 6-227
Transform4x4Fwd_VC1, 16-547
Transform4x4Inv_VC1, 16-510
Transform4x8Fwd_VC1, 16-547
Transform4x8Inv_VC1, 16-510
Transform8x4Fwd_VC1, 16-547
Transform8x4Inv_VC1, 16-510
Transform8x8Fwd_VC1, 16-547
Transform8x8Inv_VC1, 16-510
TransformDequantChromaDC_H264, 16-347
TransformDequantChromaDC_SISP_H264, 16-354
TransformDequantLumaDC_H264, 16-346
TransformLuma8x8Fwd_H264, 16-466
TransformLuma8x8InvAddPred_H264, 16-474
TransformPrediction_H264, 16-352
TransformQuant8x8Fwd_AVS, 16-499
TransformQuantChromaDC_H264, 16-459
TransformQuantLumaDC_H264, 16-461
TransformQuantResidual_H264_16s_C1I, 16-465
TransformResidual4x4Fwd_H264, 16-462
TransformResidual4x4Inv_H264, 16-348
Transpose, 4-32

U

UndistortGetSize, 14-116
UndistortRadial, 14-117
UnidirWeight_H264, 16-393
UniDirWeightBlock_H264, 16-393
uniform chromaticity scale diagram, 6-23
universal pyramids, 14-61
UpdateMotionHistory, 14-47
UpsampleFour_H263, 16-305
UpsampleFour8x8_H263, 16-307

V

Variance16x16, 16-88, 16-91
VarMean8x8, 16-83
VarMeanDiff16x16, 16-85
VarMeanDiff16x8, 16-86
VC-1 decoder functions
    bicubic interpolation, 16-522
    bilinear interpolation, 16-520
    deblocking filtering
        chroma blocks, 16-534, 16-535
        luma blocks, 16-532, 16-533
    dequantization
        inter functions, 16-540
        intra functions, 16-536
    range reduction, 16-544
    smoothing filtering
        chroma blocks, 16-528, 16-530
        luma blocks, 16-523, 16-526
VC1 decoder functions
    inverse transform, 16-510
VC-1 encoder functions
    quantization
        inter functions, 16-557
        intra functions, 16-555
VC1 encoder functions
    forward transform, 16-547
version information function
    GetLibVersion, 3-2
version information, for JPEG library, 15-2
vertical sampling, in a JPEG codec, 15-64
video coding color conversion, 16-119
video encoder functions, motion estimation
    MeanAbsDev16x16, 16-90
    MeanAbsDev8x8, 16-89
video processing functions
    general color conversion
        CbYCr422ToYCbCr420_Rotate, 16-119
        ResizeCCRotate, 16-122
video processing functions, general
    DeinterlaceBlend, 16-134
    DeinterlaceBlendFree, 16-133
    DeinterlaceBlendInitAlloc, 16-132

DeinterlaceEdgeDetect, 16-129
DeinterlaceFilterCAVT, 16-126
DeinterlaceFilterTriangle, 16-125
DeinterlaceMedianThreshold, 16-128
DeinterlaceMotionAdaptive, 16-130
FilterDenoiseAdaptive, 16-141
FilterDenoiseAdaptiveFree, 16-141
FilterDenoiseAdaptiveInitAlloc, 16-140
FilterDenoiseCAST, 16-137
FilterDenoiseCASTInit, 16-136
FilterDenoiseMosquito, 16-145
FilterDenoiseMosquitoFree, 16-145
FilterDenoiseMosquitoInitAlloc, 16-144
FilterDenoiseSmooth, 16-139
Median, 16-127
VOP, video object plane, 16-239

W

WarpAffine, 12-51
WarpAffineBack, 12-54
WarpAffineQuad, 12-57
WarpBilinear, 12-77
WarpBilinearBack, 12-80
WarpBilinearQuad, 12-83
WarpChroma_MPEG4, 16-248
WarpGetSize_MPEG4, 16-246
WarpInit_MPEG4, 16-245
WarpLuma_MPEG4, 16-247
WarpPerspective, 12-64
WarpPerspectiveBack, 12-67
WarpPerspectiveQuad, 12-70
wavelet transform functions
buffer size calculation, 13-8, 13-16
deallocation, 13-7, 13-15
forward transform, 13-9
initialization, 13-5, 13-13
inverse transform, 13-17
wavelet transforms, for lossless compression, 15-127
WeightedAverage_H264, 16-392
WeightPrediction_AVS, 16-484
WinBartlett, 10-38
WinBartlettSep, 10-38
windowing functions
Bartlett window, 10-38
Hamming window, 10-40
WinHamming, 10-40
WinHummingSep, 10-40
WTFwd, 13-9
WTFwd_B53_JPEG2K, 15-148
WTFwdCol_B53_JPEG2K, 15-132
WTFwdCol_D97_JPEG2K, 15-141
WTFwdColLift_B53_JPEG2K, 15-134
WTFwdColLift_D97_JPEG2K, 15-142
WTFwdFree, 13-7
WTFwdGetBufSize, 13-8
WTFwdInitAlloc, 13-5
WTFwdRow_B53_JPEG2K, 15-129
WTFwdRow_D97_JPEG2K, 15-138
WTInv, 13-17
WTInv_B53_JPEG2K, 15-150
WTInv_D97_JPEG2K, 15-154
WTInvCol_B53_JPEG2K, 15-135
WTInvCol_D97_JPEG2K, 15-144
WTInvColLift_B53_JPEG2K, 15-136
WTInvColLift_D97_JPEG2K, 15-145
WTInvFree, 13-15
WTInvGetBufSize, 13-16
WTInvInitAlloc, 13-13
WTInvRow_B53_JPEG2K, 15-131
WTInvRow_D97_JPEG2K, 15-139

X

Xor, 5-67
XorC, 5-68
XYZToRGB, 6-114

Y

YCbC422ToBGR555Dither, 6-83
YCbC422ToBGR565Dither, 6-83
YCbCr411, 6-188
YCbCr411ToBGR, 6-109
YCbCr411ToBGR555, 6-112
YCbCr411ToBGR555LS_MCU, 15-43
YCbCr411ToBGR565, 6-112
YCbCr411ToBGR565LS_MCU, 15-43
YCbCr411ToBGRLS_MCU, 15-42
YCbCr411ToRGBLS_MCU, 15-37
YCbCr411ToYCbCr420, 6-192
YCbCr411ToYCbCr422, 6-189

YCbCr411ToYCrCb420, 6-193
YCbCr411ToYCrCb422, 6-191
YCbCr420, 6-170
YCbCr420To422_Interlace, 6-175
YCbCr420ToBGR, 6-97
YCbCr420ToBGR_709CSC, 6-99
YCbCr420ToBGR_709HDTV, 6-100
YCbCr420ToBGR444Dither, 6-105
YCbCr420ToBGR555Dither, 6-105
YCbCr420ToBGR565Dither, 6-105
YCbCr420ToCbYCr422, 6-176
YCbCr420ToCbYCr422_Interlace, 6-177
YCbCr420ToRGB, 6-86
YCbCr420ToRGB444Dither, 6-89
YCbCr420ToRGB555Dither, 6-89
YCbCr420ToRGB565Dither, 6-89
YCbCr420ToYCbCr411, 6-181
YCbCr420ToYCbCr422, 6-172
YCbCr420ToYCbCr422_Filter, 6-173
YCbCr420ToYCrCb420, 6-179
YCbCr420ToYCrCb420_Filter, 6-180
YCbCr422, 6-153
YCbCr422To420_Interlace, 6-158
YCbCr422ToBGR, 6-69
YCbCr422ToBGR444, 6-82
YCbCr422ToBGR444Dither, 6-83
YCbCr422ToBGR555, 6-82
YCbCr422ToBGR555LS_MCU, 15-41
YCbCr422ToBGR565, 6-82
YCbCr422ToBGR565LS_MCU, 15-41
YCbCr422ToBGRLS_MCU, 15-40
YCbCr422ToCbYCr422, 6-155
YCbCr422ToRGB, 6-64
YCbCr422ToRGB_JPEG, 15-15
YCbCr422ToRGB444, 6-79
YCbCr422ToRGB444Dither, 6-80
YCbCr422ToRGB555, 6-79
YCbCr422ToRGB555Dither, 6-80
YCbCr422ToRGB565, 6-79
YCbCr422ToRGB565Dither, 6-80
YCbCr422ToRGBLS_MCU, 15-36
YCbCr422ToYCbCr411, 6-161
YCbCr422ToYCbCr420, 6-156
YCbCr422ToYCrCb420, 6-160
YCbCr444ToBGR555LS_MCU, 15-39
YCbCr444ToBGR565LS_MCU, 15-39
YCbCr444ToBGRLS_MCU, 15-38
YCbCr444ToRGBLS_MCU, 15-35
YCbCrToBGR, 6-56
YCbCrToBGR_709CSC, 6-101
YCbCrToBGR_JPEG, 15-12
YCbCrToBGR444, 6-60
YCbCrToBGR444Dither, 6-61
YCbCrToBGR555, 6-60
YCbCrToBGR555_JPEG, 15-14
YCbCrToBGR555Dither, 6-61
YCbCrToBGR565, 6-60
YCbCrToBGR565_JPEG, 15-14
YCbCrToBGR565Dither, 6-61
YCbCrToRGB, 6-54
YCbCrToRGB_JPEG, 15-8
YCbCrToRGB444, 6-57
YCbCrToRGB444Dither, 6-58
YCbCrToRGB555, 6-57
YCbCrToRGB555_JPEG, 15-10
YCbCrToRGB555Dither, 6-58
YCbCrToRGB565, 6-57
YCbCrToRGB565_JPEG, 15-10
YCbCrToRGB565Dither, 6-58
YCCK color model, 6-14
YCCK444ToCMYKLS_MCU, 15-44
YCCKToCMYK_JPEG, 15-18
YCCKToCMYK411LS_MCU, 15-46
YCCKToCMYK422LS_MCU, 15-45
YCCToRGB, 6-124
YCnCr color model, 6-14
YCoCg color model, 6-17
YCoCgToBGR, 6-137
YCoCgToBGR_Rev, 6-141
YCoCgToSBGR, 6-138
YCoCgToSBGR_Rev, 6-142
YCrCb411ToYCbCr422_16x4x5MB_DV, 16-225
YCrCb411ToYCbCr422_5MBDV, 16-224
YCrCb411ToYCbCr422_EdgeDV, 16-227
YCrCb411ToYCbCr422_ZoomOut2_5MBDV, 16-224
YCrCb411ToYCbCr422_ZoomOut2_EdgeDV, 16-227
YCrCb411ToYCbCr422_ZoomOut4_5MBDV, 16-224
YCrCb411ToYCbCr422_ZoomOut4_EdgeDV, 16-227
YCrCb411ToYCbCr422_ZoomOut8_5MBDV, 16-224
YCrCb411ToYCbCr422_ZoomOut8_EdgeDV, 16-227
YCrCb420ToCbYCr422, 6-185
YCrCb420ToRGB, 6-91

YCrCb420ToYCbCr411, 6-187
YCrCb420ToYCbCr420, 6-186
YCrCb420ToYCbCr422, 6-183
YCrCb420ToYCbCr422_5MBDV, 16-228
YCrCb420ToYCbCr422_8x8x5MB_DV, 16-229
YCrCb420ToYCbCr422_Filter, 6-184
YCrCb420ToYCbCr422_ZoomOut2_5MBDV, 16-228
YCrCb420ToYCbCr422_ZoomOut4_5MBDV, 16-228
YCrCb420ToYCbCr422_ZoomOut8_5MBDV, 16-228
YCrCb422ToRGB, 6-66
YCrCb422ToYCbCr411, 6-164
YCrCb422ToYCbCr420, 6-163
YCrCb422ToYCbCr422, 6-162
YCrCb422ToYCbCr422_10HalvesMB16x8_DV100, 16-233
YCrCb422ToYCbCr422_5MBDV, 16-230
YCrCb422ToYCbCr422_8x4x5MB_DV, 16-232
YCrCb422ToYCbCr422_ZoomOut2_5MBDV, 16-230
YCrCb422ToYCbCr422_ZoomOut4_5MBDV, 16-230
YCrCb422ToYCbCr422_ZoomOut8_5MBDV, 16-230
YUV color model, 6-12
YUV420ToBGR, 6-44
YUV420ToBGR444, 6-50
YUV420ToBGR444Dither, 6-51
YUV420ToBGR555, 6-50
YUV420ToBGR555Dither, 6-51
YUV420ToBGR565, 6-50
YUV420ToBGR565Dither, 6-51
YUV420ToRGB, 6-41
YUV420ToRGB444, 6-46
YUV420ToRGB444Dither, 6-48
YUV420ToRGB555, 6-46
YUV420ToRGB555Dither, 6-48
YUV420ToRGB565, 6-46
YUV420ToRGB565Dither, 6-48
YUV422ToRGB, 6-37
YUVToRGB, 6-34

Z

zigzag order of elements, 15-48
ZigzagFwd8x8, 4-45
ZigzagInv8x8, 4-47