

# Horus Global Function Guide

Version 2.0 - Jan 2003

Erik Engbers

Intelligent Sensory Information Systems  
University of Amsterdam, Faculty of Science  
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands  
koelma@science.uva.nl  
<http://www.science.uva.nl/~horus/>

---

# Contents

<b>1</b>	<b>General Documentation</b>	<b>2</b>
1.1	Border handling . . . . .	2
1.2	Precision . . . . .	2
1.3	Interpolation . . . . .	2
1.4	Image signatures . . . . .	2
<b>2</b>	<b>Function Documentation</b>	<b>3</b>
2.1	HxAbs . . . . .	4
2.2	HxAcos . . . . .	5
2.3	HxAdd . . . . .	7
2.4	HxAddVal . . . . .	10
2.5	HxAnd . . . . .	11
2.6	HxAndVal . . . . .	13
2.7	HxArg . . . . .	14
2.8	HxAsin . . . . .	15
2.9	HxAtan . . . . .	17
2.10	HxCannyEdgeMap . . . . .	19
2.11	HxCannyThreshold . . . . .	21
2.12	HxCannyThresholdAlt . . . . .	23
2.13	HxCeil . . . . .	24
2.14	HxColorSpace . . . . .	25
2.15	HxComplement . . . . .	26
2.16	HxConjugate . . . . .	28
2.17	HxContrastStretch . . . . .	29
2.18	HxConvGauss2d . . . . .	31
2.19	HxConvGauss3d . . . . .	33
2.20	HxConvKernelSeparated2d . . . . .	35
2.21	HxConvKernelSeparated . . . . .	38

---

2.22 HxConvolution1d . . . . .	40
2.23 HxConvolution . . . . .	43
2.24 HxCos . . . . .	46
2.25 HxCosh . . . . .	48
2.26 HxCross . . . . .	50
2.27 HxCrossVal . . . . .	51
2.28 HxDistanceTransform . . . . .	52
2.29 HxDiv . . . . .	53
2.30 HxDivVal . . . . .	55
2.31 HxDot . . . . .	56
2.32 HxDotVal . . . . .	57
2.33 HxEqual . . . . .	58
2.34 HxEqualVal . . . . .	59
2.35 HxExp . . . . .	60
2.36 HxExportByteData . . . . .	61
2.37 HxExportDoubleData . . . . .	62
2.38 HxExportFloatData . . . . .	63
2.39 HxExportIntData . . . . .	64
2.40 HxExportMatlabPixels . . . . .	65
2.41 HxExportShortData . . . . .	66
2.42 HxExportSI . . . . .	67
2.43 HxExtend . . . . .	68
2.44 HxExtendVal . . . . .	69
2.45 HxFloor . . . . .	70
2.46 HxGauss . . . . .	71
2.47 HxGaussDerivative2d . . . . .	72
2.48 HxGaussDerivative3d . . . . .	73
2.49 HxGreaterEqual . . . . .	74
2.50 HxGreaterEqualVal . . . . .	75
2.51 HxGreaterThan . . . . .	76
2.52 HxGreaterThanVal . . . . .	79
2.53 HxImageAsByte . . . . .	80
2.54 HxImageAsDouble . . . . .	81
2.55 HxImageAsFloat . . . . .	82
2.56 HxImageAsInt . . . . .	83
2.57 HxImageAsShort . . . . .	84

2.58 HxImageAsVec2Byte . . . . .	85
2.59 HxImageAsVec2Double . . . . .	86
2.60 HxImageAsVec2Float . . . . .	87
2.61 HxImageAsVec2Int . . . . .	88
2.62 HxImageAsVec2Short . . . . .	89
2.63 HxImageAsVec3Byte . . . . .	90
2.64 HxImageAsVec3Double . . . . .	91
2.65 HxImageAsVec3Float . . . . .	92
2.66 HxImageAsVec3Int . . . . .	93
2.67 HxImageAsVec3Short . . . . .	94
2.68 HxImageMaxSize . . . . .	95
2.69 HxImageMinSize . . . . .	96
2.70 HxInf . . . . .	97
2.71 HxInfVal . . . . .	99
2.72 HxInverseProjectRange . . . . .	100
2.73 HxLeftShift . . . . .	101
2.74 HxLeftShiftVal . . . . .	102
2.75 HxLessEqual . . . . .	103
2.76 HxLessEqualVal . . . . .	104
2.77 HxLessThan . . . . .	105
2.78 HxLessThanVal . . . . .	108
2.79 HxLog10 . . . . .	109
2.80 HxLog . . . . .	111
2.81 HxMakeFromByteData . . . . .	113
2.82 HxMakeFromDoubleData . . . . .	114
2.83 HxMakeFromFile . . . . .	115
2.84 HxMakeFromFileSI . . . . .	116
2.85 HxMakeFromFloatData . . . . .	117
2.86 HxMakeFromIntData . . . . .	118
2.87 HxMakeFromMatlab . . . . .	119
2.88 HxMakeFromShortData . . . . .	120
2.89 HxMakeFromSI . . . . .	121
2.90 HxMakeFromSignature . . . . .	122
2.91 HxMakeFromValue . . . . .	123
2.92 HxMax . . . . .	124
2.93 HxMaxVal . . . . .	125

---

2.94 HxMin	126
2.95 HxMinVal	127
2.96 HxMod	128
2.97 HxModVal	129
2.98 HxMul	130
2.99 HxMulVal	132
2.100HxNegate	133
2.101HxNorm1	134
2.102HxNorm2	135
2.103HxNorm2Sqr	136
2.104HxNormInf	137
2.105HxNotEqual	138
2.106HxNotEqualVal	139
2.107HxOr	140
2.108HxOrVal	142
2.109HxPixInf	143
2.110HxPixMax	144
2.111HxPixMin	145
2.112HxPixProduct	146
2.113HxPixSum	147
2.114HxPixSup	148
2.115HxPow	149
2.116HxPowVal	150
2.117HxProjectRange	151
2.118HxRestrict	152
2.119HxRightShift	153
2.120HxRightShiftVal	154
2.121HxRotate	155
2.122HxRound	156
2.123HxScale	157
2.124HxSin	158
2.125HxSinh	160
2.126HxSqrt	162
2.127HxSquaredDistance	164
2.128HxSub	165
2.129HxSubVal	168

---

2.130HxSup	169
2.131HxSupVal	171
2.132HxTan	172
2.133HxTanh	174
2.134HxThreshold	176
2.135HxTriStateThreshold	178
2.136HxUnaryMax	179
2.137HxUnaryMin	180
2.138HxUnaryProduct	181
2.139HxUnarySum	182
2.140HxUniform	183
2.141HxWriteFile	184
2.142HxXor	185
2.143HxXorVal	187

---

# **Chapter 1**

## **General Documentation**

**1.1 Border handling**

**1.2 Precision**

**1.3 Interpolation**

**1.4 Image signatures**

---

---

## **Chapter 2**

# **Function Documentation**

---

## 2.1 HxAbs

### Synopsis

HxImageRep HxAbs (HxImageRep img)

### Input

HxImageRep img    The image you want to take the absolute value of.

### Return value

HxImageRep    The absolute value of the input image.

### Description

The function HxAbs takes the absolute value of each pixel value. The absolute value of a vector is defined by taking the absolute value of the components of the vector, i.e.  $|(a_1, a_2, \dots, a_n)| = (|a_1|, |a_2|, \dots, |a_n|)$ .

### Remarks

*No possible errors*    This function can not result in any errors.

### See also

**HxNegate** (p. 133), **HxNorm1** (p. 134), **HxNorm2** (p. 135), **HxNorm2Sqr** (p. 136), **HxNormInf** (p. 137),

### Keywords

Unary, Arithmetic,

## 2.2 HxAcos

### Synopsis

HxImageRep HxAcos (HxImageRep img)

### Input

HxImageRep img   The image you want to take the arccosine of.

### Return value

HxImageRep   The arccosine of the input image.

### Description

This function takes the arccosine (or inverse cosine) of every pixel value of the input image. For vector images, the arccosine of every element of the vector is taken.

### Remarks

*Valid input images*   The pixel values of the input image should be larger or equal to -1 and smaller or equal to 1.

### Examples

Taking the arccosine of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxAcosExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxContrastStretch(img, 2.0);
    img = HxAddVal(img, -1.0);
    img = HxAcos(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);
    return img;
}

int main(int argc, char* argv[])
{
    // HxImageRep im1 = HxMakeFromFile("../Images/exampleImage1.tif");
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxAcosExample1(im1);
    // HxWriteFile(im1, "HxAcosResult1.tif");
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.1: Input image for the HxAcos example.



Figure 2.2: Output image of the HxAcos example.

This example illustrates the result of taking the arccosine of an image.

**See also**

**HxCos** (p. 46), **HxTan** (p. 172), **HxAsin** (p. 15), **HxCosh** (p. 48),

**Keywords**

Unary, Trigonometric,

## 2.3 HxAdd

### Synopsis

```
HxImageRep HxAdd(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The sum of the input images

### Description

The function HxAdd adds two images on a pixel-by-pixel basis. Vector pixel types of the same dimension are summed component wise, while pixel types of different dimensions cannot be summed.

### Remarks

**Admissable image types** The two input images should have the same dimensionality, the same pixel-dimensionality and the same size.

**Warning for possible overflow** When two byte images are added, where the sum of the maximums of the images is larger than 255, a warning is given that an overflow could have occurred. In case the sum of two bytes is larger than 255, the value of that byte pixel will be the sum modulo 256.

### Examples

Adding two images

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxAddExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxImageAsDouble(im1);
    im2 = HxImageAsDouble(im2);
    im1 = HxAdd(im1, im2);
    im1 = HxContrastStretch(im1, 255);
    im1 = HxImageAsByte(im1);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxAddExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.3: First input image for the HxAdd example.

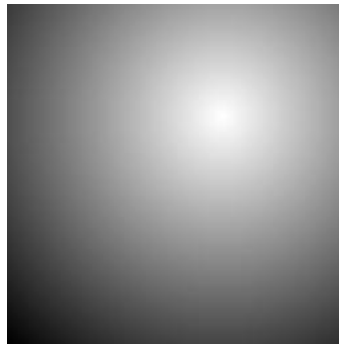


Figure 2.4: Second input image for the HxAdd example.



Figure 2.5: Output image of the HxAdd example.

In this example we add two (scalar) images. First we convert the image to double images to prevent overflow as much as possible. After adding the images, we stretch the result and convert it to byte for visualization purposes.

**See also**

**HxSub** (p. 165), **HxMul** (p. 130), **HxDiv** (p. 53),

**Keywords**

Binary, Arithmetic,

## 2.4 HxAddVal

### Synopsis

```
HxImageRep HxAddVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The input value

### Return value

HxImageRep    The sum of the input image and the input value.

### Description

The function HxAddVal adds the HxValue *val* to every pixel value in the input image and stores the result in the corresponding pixel of the output image. The dimensionality of '*val*' and the *pixeldimensionality* of the input image must be the same. If '*val*' is a vector and the pixels of the input image are vectors, the values are summed component wise.

### Remarks

**Admissable image and value types**    The dimensionality of '*val*' and the *pixeldimensionality* of the input image must be the same.

**Overflow error**    The function HxAddVal can result in an overflow error.

### See also

**HxAdd** (p. 7), **HxSubVal** (p. 168), **HxDivVal** (p. 55), **HxMulVal** (p. 132),

### Keywords

Binary value, Arithmetic,

## 2.5 HxAnd

### Synopsis

```
HxImageRep HxAnd(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1   The first input image.

HxImageRep im2   The second input image.

### Return value

HxImageRep   The logical ‘and’ of the input images.

### Description

The function HxAnd takes the logical ‘and’ of corresponding pixels in the two images. The ‘and’ of two vectors is taken component wise.

### Remarks

**Valid input images**   The input images should have the same dimensionality, the same size, the same pixel dimensionality and the same pixel type. The numerical values of the pixels, or its components, should be HxShort, HxByte, HxInt .

### Examples

Taking the logical ‘and’

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxAndExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxImageAsDouble(im1);
    im2 = HxImageAsDouble(im2);
    im1 = HxContrastStretch(im1, 1.0);
    im2 = HxContrastStretch(im2, 1.0);
    im1 = HxImageAsByte(im1);
    im2 = HxImageAsByte(im2);
    im1 = HxAnd(im1, im2);
    im1 = HxContrastStretch(im1, 255);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxAndExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```

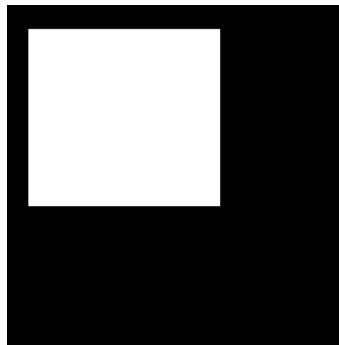


Figure 2.6: First input image for the HxAnd example.

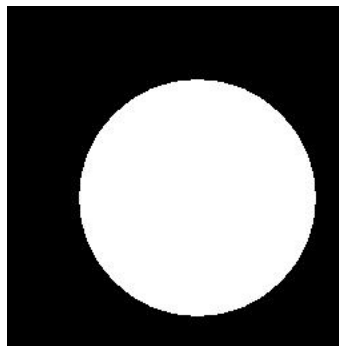


Figure 2.7: Second input image for the HxAnd example.



Figure 2.8: Output image of the HxAnd example.

In this example we take the logical ‘and’ of the two input images. We first convert them to binary format after which the logical ‘and’ is taken.

**See also**

**HxOr** (p. 140), **HxXor** (p. 185), **HxAndVal** (p. 13),

**Keywords**

Binary, Logical,

## 2.6 HxAndVal

### Synopsis

```
HxImageRep HxAndVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image.

HxValue *val*    The input value.

### Return value

HxImageRep    The logical ‘and’ of the input image and the input value.

### Description

The function HxAndVal takes the logical ‘and’ of the pixels from the input image with the input value on a pixel-by-pixel basis. The ‘and’ of two vectors is taken component wise.

### Remarks

*Valid input images and input values*    The input value should have the same dimensionality as the pixel dimensionality of the input image. The pixel type and the type of the input value should be, HxShort, HxByte, HxInt . The types should be equal.

### See also

**HxOrVal** (p. 142), **HxXorVal** (p. 187), **HxAnd** (p. 11),

### Keywords

Binary value, Logical,

## 2.7 HxArg

### Synopsis

```
HxImageRep HxArg (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the complex argument of.

### Return value

HxImageRep    The pixelwise complex argument of the input image.

### Description

The function HxArg takes the argument of each pixel value. The input image should be a complex value image, i.e. a vec2 image. The argument is the counterclockwise angle from the positive real axis. When the complex number  $x+iy$  is represented by  $(x,y)$  then for the argument  $a$  holds that  $y = \cos a$ . The resulting image is scalar.

### Remarks

*Valid input images*    Only vec2 images are valid input images for this function.

### See also

**HxConjugate** (p. 28),

### Keywords

Unary, Complex, Arithmetic,

## 2.8 HxAsin

### Synopsis

```
HxImageRep HxAsin (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the arcsine of.

### Return value

HxImageRep    The arcsine of the input image.

### Description

This function takes the arcsine (or inverse sine) of every pixel value of the input image. For vector images, the arcsine of every element of the vector is taken.

### Remarks

*Valid input images*    The pixel values of the input image should be larger or equal to -1 and smaller or equal to 1.

### Examples

Taking the arcsine of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxAsinExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxContrastStretch(img, 2.0);
    img = HxAddVal(img, -1.0);
    img = HxAsin(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);
    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxAsinExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.9: Input image for the HxAsin example.



Figure 2.10: Output image of the HxAsin example.

This example illustrates the result of taking the arcsine of an image.

**See also**

**HxSin** (p. 158), **HxTan** (p. 172), **HxAcos** (p. 5), **HxSinh** (p. 160),

**Keywords**

Unary, Trigonometric,

## 2.9 HxAtan

### Synopsis

```
HxImageRep HxAtan (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the arctangent of.

### Return value

HxImageRep    The pixelwise arctangent of the input image.

### Description

This function takes the arctangent of every pixel value of the input image. For vector images, the arctangent of every element of the vector is taken.

### Remarks

*No possible errors*    This function can not result in any errors.

### Examples

Taking the arctangent of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxAtanExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxContrastStretch(img, 10.0);
    img = HxAddVal(img, -5.0);
    img = HxAtan(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);
    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxAtanExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.11: Input image for the HxAtan example.



Figure 2.12: Output image for the HxAtan example.

This example illustrates the result of taking the arctangent of an image.

**See also**

**HxTan** (p. 172), **HxAsin** (p. 15), **HxTanh** (p. 174),

**Keywords**

Unary, Trigonometric,

## 2.10 HxCannyEdgeMap

### Synopsis

```
HxImageRep HxCannyEdgeMap(HxImageRep img, double sigma)
```

### Input

`HxImageRep img` The image you want to determine the edges of.

`double sigma` The standard deviation of the Gaussian kernel that is used to calculate the canny edge map.

### Return value

`HxImageRep` The output image is a 2D image, with `HxVec2Double` pixels. The first component of the pixel is the magnitude of the gradient in the X-direction, the second component of the pixel is the magnitude of the gradient in the Y-direction.

### Description

The function `HxCannyEdgeMap` computes the magnitude of the gradient of `img` in the X- and Y-direction. For every position in `img`, the magnitude in the X-direction is put in the first element of the output-vector, while the magnitude in the Y-direction is put in the second element of the output-vector.

The magnitudes of the gradient in the X- and Y-direction are calculated by convolving `img` with the derivative of a 1D-Gaussian in both directions. The used Gaussians have a standard deviation of `sigma`.

### Remarks

**Valid input images** This function is only properly defined for 2D images with scalar pixels.

**Valid values for sigma** This function is only properly defined for `sigma` greater than 0.

**Border handling** This function uses MIRRORED border handling, see section **Border handling** (p. 2).

### Examples

Calculating the norm of the output of `HxCannyEdgeMap`

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxCannyEdgeMapExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxCannyEdgeMap(img, 1.0);
    img = HxNorm1(img);
    img = HxContrastStretch(img, 255);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxCannyEdgeMapExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.13: Input image for the HxCannyEdgeMap example.



Figure 2.14: Output image of the HxCannyEdgeMap example.

In this example we apply HxCannyEdgeMap on a grey value image. We take the size of the resulting vector pixels (HxNorm1), after which we stretch the image for proper visualisation (HxContrastStretch).

**See also**

**HxCannyThreshold** (p. 21), **HxCannyThresholdAlt** (p. 23),

**Keywords**

Edge detection, Filter, Canny,

## 2.11 HxCannyThreshold

### Synopsis

```
HxImageRep HxCannyThreshold (HxImageRep img, double sigma, double level)
```

### Input

`HxImageRep img` The 2D scalar image you want to determine the edges of.

`double sigma` The standard deviation of the Gaussian kernel that is used to calculate the canny edge map.

`double level` The threshold level for the grayvalue of the edges.

### Return value

`HxImageRep` The resulting 2D integer image.

### Description

The function `HxCannyThresholds` computes the magnitude of the gradient of `img` in the X- and Y-direction. Then the Euclidean norm of these gradient vectors is determined, after which it is thresholded with the double 'level'.

The magnitudes of the gradient in the X- and Y-direction are calculated by convolving `img` with the derivative of a 1D-Gaussian in both directions. The used Gaussians have a standard deviation of `sigma`.

### Remarks

**Valid input images** This function is only properly defined for 2D images with scalar pixels.

**Valid values for sigma and level** This function is only properly defined for `sigma` greater than 0. 'Level' can have any double value.

**Border handling** This function uses MIRRORED border handling, see the section on **Border handling** (p. 2).

**Possible overflow** For very large values of the level parameter, this function can result in an overflow.

### Examples

Finding edges with `HxCannyThreshold`

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxCannyThresholdExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxCannyThreshold(img, 3.0, 5.0);
    img = HxContrastStretch(img, 255);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
```

```
HxImageRep im1 = HxMakeFromFile(argv[1]);
im1 = HxCannyThresholdExample1(im1);
HxWriteFile(im1, argv[2]);

return 0;
}
```



Figure 2.15: Input image for the HxCannyThreshold example.



Figure 2.16: Output image of the HxCannyThreshold example.

In this example we use `HxCannyThreshold` to determine the edges in a 2D scalar image. For sigma we take 3.0, for the level we take 5.0. The resulting image is stretched for proper visualisation (`HxContrastStretch`).

**See also**

**`HxCannyEdgeMap`** (p. 19), **`HxCannyThresholdAlt`** (p. 23),

**Keywords**

Edge detection, Filter, Canny,

## 2.12 HxCannyThresholdAlt

### Synopsis

```
HxImageRep HxCannyThresholdAlt (HxImageRep img, double sigma, double level)
```

### Description

This is an alternative implementation of the function **HxCannyThreshold** (p. 21).

## 2.13 HxCeil

### Synopsis

```
HxImageRep HxCeil(HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to take the ceiling of.

### Return value

HxImageRep    The ceiling value of the input image.

### Description

The function HxCeil takes the ceiling value of each pixel value, where the ceiling of a number is the smallest integer that is larger than the pixel value. The ceiling of a vector is defined by separately taking the ceiling of the components of the vector.

### Remarks

*No possible errors*    This function can not result in any errors.

### See also

**HxMax** (p. 124), **HxMin** (p. 126), **HxFloor** (p. 70),

### Keywords

Unary, Arithmetic,

## 2.14 HxColorSpace

### Synopsis

```
HxImageRep HxColorSpace (HxImageRep img, HxColorModel fromColorSpace,  
HxColorModel toColorSpace)
```

### Input

`HxImageRep img` The image you want to convert to another color space.

`HxColorModel fromColorSpace` The color space of the input image.

`HxColorModel toColorSpace` The color space you want to convert the input image to.

### Return value

`HxImageRep` The result of converting the input image. The result is always of the type `Vec3Double`.

### Description

The function `HxColorSpace` converts the input image to `toColorSpace`, assuming that the `colorModel` of the input image is `fromColorSpace`.

### Remarks

*Valid input images* All `Vec3` images are valid as input for this function.

*Valid color models* The valid color models are: RGB, HSI

### See also

- ,

### Keywords

Color, Conversion,

## 2.15 HxComplement

### Synopsis

```
HxImageRep HxComplement(HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the complement of.

### Return value

HxImageRep    The complement of the input image.

### Description

The function HxComplement takes the complement of every pixel value, where the complement is the bit-wise complement. HxComplement is only defined for integer image types.

### Remarks

*Valid input images*    Only integer images types are valid as input for this function.

### Examples

Taking the complement of an image

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxComplementExample1(HxImageRep img) {
    img = HxImageAsByte(img);
    img = HxComplement(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxComplementExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.17: Input image for the HxComplement example.



Figure 2.18: Output image of the HxComplement example.

In this example we take the complement of an image. The input image must be of an integer type.

**See also**

**HxNegate** (p. 133),

**Keywords**

Unary, Arithmetic,

## 2.16 HxConjugate

### Synopsis

```
HxImageRep HxConjugate(HxImageRep img)
```

### Input

HxImageRep *img* The image you want to take the complex conjugate of.

### Return value

HxImageRep The pixelwise complex conjugate of the input image.

### Description

The function HxConjugate takes the conjugate of each pixel value. The input image should be a complex value image, i.e. a vec2 image. If the pixel value of the input image is given as  $x+iy$  (represented by  $(x,y)$ ) then the corresponding pixel in the output image is set to  $x-iy$  (represented by  $(x,-y)$ ).

### Remarks

*Valid input images* Only vec2 images are valid input images for this function.

### See also

[HxArg](#) (p. 14),

### Keywords

Unary, Complex, Arithmetic,

## 2.17 HxContrastStretch

### Synopsis

```
HxImageRep HxContrastStretch (HxImageRep img, double mFactor)
```

### Input

HxImageRep *img* The image you want to improve the contrast of.

double *mFactor* The maximum value in the output image. The minimum value is zero.

### Return value

HxImageRep An image with a range as given by *mFactor* for the pixel values.

### Description

This function stretches the contrast of the function, where the range of the pixel value in the output image is given by the interval [0, *mFactor*]. The pixel value 'y' in the output image is calculated from the original pixel value 'x' as follows:  $y = mFactor \frac{x - I_{min}}{I_{max} - I_{min}}$ . Where  $I_{min}$  and  $I_{max}$  denote the minimum and maximum value of the input image. This function is defined for scalar images that are not constant.

### Remarks

**Valid input images** Only scalar images are valid, where the minimum and the maximum value in the image may not be equal.

**Valid value for mFactor** *mFactor* should be larger than zero.

### Examples

Example of stretching the contrast of an image

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxContrastStretchExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxContrastStretch(img, 150.0);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxContrastStretchExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.19: Input image for the HxContrastStretch example.



Figure 2.20: Output image of the HxContrastStretch example.

In this example we stretch the pixel values to a range from 0 to 150.

**See also**

- ,

**Keywords**

Enhancement,

## 2.18 HxConvGauss2d

### Synopsis

```
HxImageRep HxConvGauss2d (HxImageRep img, double sigmax, int orderDerivx, double truncationx, double sigmay, int orderDerivy, double truncationy)
```

### Input

`HxImageRep img` The image you want to convolve with a Gaussian kernel. The used Gaussian kernel is composed from two separate kernels, one for the x-direction and one for the y-direction.

`double sigmax` The sigma of the Gaussian kernel in the x-direction.

`int orderDerivx` The order of the derivative of the Gaussian kernel in the x-direction.

`double truncationx` This parameter determines at how many sigma the Gaussian filter is clipped in the x-direction.

`double sigmay` The sigma of the Gaussian kernel in the y-direction.

`int orderDerivy` The order of the derivative of the Gaussian kernel in the y-direction.

`double truncationy` This parameter determines at how many sigma the Gaussian filter is clipped in the y-direction.

### Return value

`HxImageRep` The result of convolving the input image with a Gaussian kernel as described by the parameters `sigmax`, `orderDerivx`, `truncationx`, `sigmay`, `orderDerivy` and `truncationy`.

### Description

The function `HxConvGauss2d` convolves 2D-images with a Gaussian filter function. The filter function is separated in the x- and y-direction, where the sigma and the order of the derivative in both directions can be given separately. The truncation (in both directions) determines the size of the filter in sigma (of the particular direction), where the filter size, in sigma, is 2 times the truncation plus 1. For vector images, each channel is convolved separately with the Gaussian kernel.

### Remarks

**Valid image types** All 2D types of images are allowed for this function.

**Valid values for the parameters** `sigmax`, `sigmay`, `truncationx` and `truncationy` should both be greater than zero. `orderDerivx` and `orderDerivy` should be larger or equal to zero.

**Border handling** This function uses `MIRRORED` border handling, see the section on **Border handling** (p. 2).

### Examples

Convoluting with a Gaussian kernel using `HxConvGauss2d`

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxConvGauss2dExample1(HxImageRep img) {
```

```
img = HxImageAsDouble(img);
img = HxConvGauss2d(img, 3.0, 0.0, 3.0, 3.0, 1.0, 3.0);
img = HxContrastStretch(img, 255.0);
img = HxImageAsByte(img);

return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxConvGauss2dExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.21: Input image for the HxConvGauss2d example.



Figure 2.22: Output image of the HxConvGauss2d example.

In this example we convolve the input image with a Gaussian filter. The sigma of this filter in the x-direction is 3.0, the order of the derivative in this direction is 0 and the truncation width is 3.0 (in sigmas). In the y-direction the sigma of the filter is 3.0, the order of the derivative is 1 and the truncation width is 3.0.

**See also**

[HxGaussDerivative2d](#) (p. 72), [HxConvolution](#) (p. 43), [HxGauss](#) (p. 71), [HxConvGauss3d](#) (p. 33),

**Keywords**

Filter, Convolution, Gauss,

## 2.19 HxConvGauss3d

### Synopsis

```
HxImageRep HxConvGauss3d (HxImageRep img, double sigmax, int orderDerivx, double truncationx, double sigmay, int orderDerivy, double truncationy, double sigmaz, int orderDerivz, double truncationz)
```

### Input

`HxImageRep img` The image you want to convolve with a Gaussian kernel. The used Gaussian kernel is composed from three separate kernels, one for the x-direction, one for the y-direction and one for the z-direction.

`double sigmax` The sigma of the Gaussian kernel in the x-direction.

`int orderDerivx` The order of the derivative of the Gaussian kernel in the x-direction.

`double truncationx` This parameter determines at how many sigma the Gaussian filter is clipped in the x-direction.

`double sigmay` The sigma of the Gaussian kernel in the y-direction.

`int orderDerivy` The order of the derivative of the Gaussian kernel in the y-direction.

`double truncationy` This parameter determines at how many sigma the Gaussian filter is clipped in the y-direction.

`double sigmaz` The sigma of the Gaussian kernel in the z-direction.

`int orderDerivz` The order of the derivative of the Gaussian kernel in the z-direction.

`double truncationz` This parameter determines at how many sigma the Gaussian filter is clipped in the z-direction.

### Return value

`HxImageRep` The result of convolving the input image with a Gaussian kernel as described by the parameters `sigmax`, `orderDerivx`, `truncationx`, `sigmay`, `orderDerivy`, `truncationy`, `sigmaz`, `orderDerivz` and `truncationz`.

### Description

The function `HxConvGauss3d` convolves 3D-images with a Gaussian filter function. The filter function is separated in the x-, y- and z-direction, where the sigma and the order of the derivative in both directions can be given separately. The truncation (in all directions) determines the size of the filter in sigma (of the particular direction), where the filter size, in sigma, is 2 times the truncation plus 1. For vector images, each channel is convolved separately with the Gaussian kernel.

### Remarks

**Valid image types** All 3D types of images are allowed for this function.

**Valid values for the parameters** `sigmax`, `sigmay`, `sigmaz`, `truncationx`, `truncationy` and `truncationz` should all be greater than zero. `orderDerivx`, `orderDerivy` and `orderDerivz` should be larger or equal to zero.

**Border handling** This function uses MIRRORED border handling, see the section on **Border handling** (p. 2).

**See also**

**HxGaussDerivative3d** (p. 73), **HxConvolution** (p. 43), **HxGauss** (p. 71), **HxConvGauss2d** (p. 31),

**Keywords**

Filter, Convolution, Gauss,

## 2.20 HxConvKernelSeparated2d

### Synopsis

```
HxImageRep HxConvKernelSeparated2d (HxImageRep img, HxImageRep kernelX, HxImageRep kernelY, ResultPrecision resPrec = DEFAULT_PREC)
```

### Input

`HxImageRep img` The image you want to convolve with the kernel described by the `kernelX` and `kernelY` images.

`HxImageRep kernelX` The kernel for the convolution in the x-direction. The dimensions of this image should be 1 by x, where x is an arbitrary positive integer.

`HxImageRep kernelY` The kernel for the convolution in the y-direction. The dimensions of this image should be 1 by y, where y is an arbitrary positive integer.

`ResultPrecision resPrec` This parameter determines the precision of the resulting image. The available precision values are `DEFAULT_PREC`, `SOURCE_PREC`, `ARITH_PREC`, `SMALL_PREC`. See the section on **Precision** (p. 2) for a more detailed description about precision.

### Return value

`HxImageRep` The result of convolving the input image with `kernelX` and `kernelY`.

### Description

The function `HxConvKernelSeparated2d` convolves the input image with a 1 by x filter in the x-direction, after which the result of that convolution is convolved with a 1 by y filter in the y-direction. The convolution kernels can be either a scalar or vector images. If the kernels are scalar images, the input image may be a scalar or a vector image. In case of scalar kernels and a vector input image, the convolution (in each direction) is taken separately for each channel. If both kernel and image are vector images, the pixel dimension should be the same. In the latter case, each channel of the input image is convolved with the corresponding channel of the kernels. The kernels should have the same pixel dimensionality. Note that both kernels are given in a '1 by q' format.

### Remarks

**Valid input image types** All 2D types of images are allowed for this function.

**Valid kernel image types** The `kernelX` image should be an image of size 1 by x, where x should be smaller or equal to the size of the input image in the x-direction. The `kernelY` image should be an image of size 1 by y, where y should be smaller or equal to the size of the input image in the y-direction. If the input image is a vector image, the kernels should be vector images with the same pixel dimension as the input image. The kernels should have the same pixel dimensionality.

**Valid precision values** The available precision values are `DEFAULT_PREC`, `SOURCE_PREC`, `ARITH_PREC`, `SMALL_PREC`. See the section on **Precision** (p. 2) for a more detailed description about precision.

**Border handling** This function uses `MIRRORED` border handling, see the section on **Border handling** (p. 2).

### Examples

Convoluting an image with two separate 1d kernels.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxConvKernelSeparated2dExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    HxImageRep kernelX = HxMakeGaussian1d(3.0, 0, 5.0, 10, -1);
    HxImageRep kernelY = HxMakeParabola1d(1.0, 5.0, 10, -1);
    img = HxConvKernelSeparated2d(img, kernelX, kernelY);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxConvKernelSeparated2dExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.23: Input image for the HxConvKernelSeparated2d example.

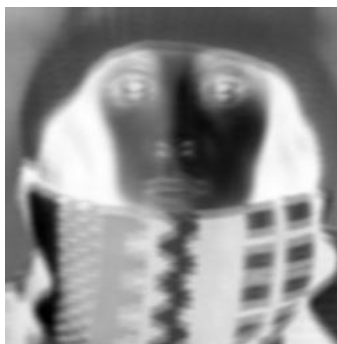


Figure 2.24: Output image of the HxConvKernelSeparated2d example.

In this example we generate a 1d Gaussian kernel for the x-direction and a 1d parabola kernel for the y-direction. The input image in this example code is then convolved with the two generated kernels.

**See also**

**HxGaussDerivative2d** (p. 72), **HxConvolution** (p. 43), **HxConvGauss2d** (p. 31) , **HxConvKernelSeparated** (p. 38) ,

**Keywords**

Filter, Convolution,

## 2.21 HxConvKernelSeparated

### Synopsis

```
HxImageRep HxConvKernelSeparated (HxImageRep img, HxImageRep kernel,
HxImageRep::ResultPrecision resPrec)
```

### Input

`HxImageRep img` The image you want to convolve with the kernel defined by the 1d kernel image.

`HxImageRep kernel` The kernel for the convolution in all image dimensions. The dimensions of this image should be 1xn, where n is an arbitrary positive integer.

`ResultPrecision resPrec` This parameter determines the precision of the resulting image. The available precision values are `DEFAULT_PREC`, `SOURCE_PREC`, `ARITH_PREC`, `SMALL_PREC`. See the section on **Precision** (p. 2) for a more detailed description about precision.

### Return value

`HxImageRep` The result of convolving the input image with the one dimensional kernel 'kernel' for every dimension.

### Description

The function `HxConvKernelSeparated` convolves the input image with a 1xn filter in every direction (dimension) of the image. The convolution kernel can be either a scalar or vector image. If the kernel is a scalar image, the input image may be a scalar or a vector image. In case of a scalar kernel and an vector input image, the convolution (in every direction) is taken separately for each channel. If both kernel and image are vector images, the pixel dimension should be the same. In the latter case, each channel of the input image is convolved with the corresponding channel of the kernel.

### Remarks

**Valid input image types** All types of images are allowed for this function.

**Valid kernel image types** The kernel image should be an image of size 1 by x, where x should be smaller or equal to the size of the input image in any direction. When the kernel is a vector image, the pixel dimension of the kernel should be equal to the pixel dimension of the input image.

**Valid precision values** The available precision values are `DEFAULT_PREC`, `SOURCE_PREC`, `ARITH_PREC`, `SMALL_PREC`. See the section on **Precision** (p. 2) for a more detailed description about precision.

**Border handling** This function uses `MIRRORED` border handling, see the section on **Border handling** (p. 2).

### Examples

Convoluting an image with one 1d kernel in all directions.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxConvKernelSeparatedExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    HxImageRep kernel = HxMakeGaussian1d(3.0, 0, 5.0, 10, -1);
```

```
img = HxConvKernelSeparated(img, kernel, HxImageRep::DEFAULT_PREC);
img = HxContrastStretch(img, 255.0);
img = HxImageAsByte(img);

return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxConvKernelSeparatedExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.25: Input image for the HxConvKernelSeparated example.

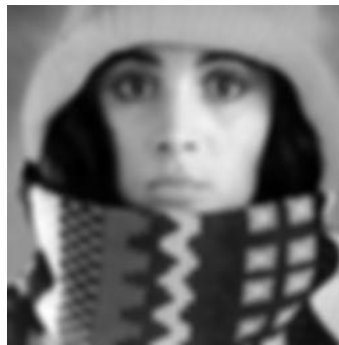


Figure 2.26: Output image of the HxConvKernelSeparated example.

In this example we generate a 1d Gaussian kernel, after which the input image in this example code is convolved with the kernel in all directions.

#### See also

[HxConvolution](#) (p. 43), [HxConvGauss2d](#) (p. 31), [HxConvKernelSeparated2d](#) (p. 35),

#### Keywords

Filter, Convolution,

## 2.22 HxConvolution1d

### Synopsis

```
HxImageRep HxConvolution1d (HxImageRep img, HxImageRep kernel, int
dimension, HxImageRep::ResultPrecision resPrec)
```

### Input

`HxImageRep img` The image you want to convolve with the kernel described by the `kernelX` and `kernelY` images.

`HxImageRep kernel` The kernel for the convolution. The dimensions of this image should be `1xn`, where `n` is an arbitrary positive integer.

`int dimension` The dimension of the image that should be convolved with the given kernel.

`ResultPrecision resPrec` This parameter determines the precision of the resulting image. The available precision values are `DEFAULT_PREC`, `SOURCE_PREC`, `ARITH_PREC`, `SMALL_PREC`. See the section on **Precision** (p. 2) for a more detailed description about precision.

### Return value

`HxImageRep` The result of convolving the input image with the kernel in the given dimension.

### Description

The function `HxConvolution1d` convolves the input image with a `1xn` filter in the dimension as given by the integer 'dimension'. The convolution kernel can be either a scalar or vector image. If the kernel is scalar, the input image may be a scalar or a vector image. In case of scalar kernels and a vector input image, the convolution is taken separately for each channel. If both kernel and image are vector images, the pixel dimension should be the same. In the latter case, each channel of the input image is convolved with the corresponding channel of the kernel. The kernel should have the same pixel dimensionality.

### Remarks

**Valid input image types** All images are allowed for this function.

**Valid kernel image types** The kernel image should be an image of size `1 by x`, where `x` should be smaller or equal to the size of the input image in direction as given by the integer 'dimension'. The pixel dimensionality of the kernel should either be `1` or equal to the pixel dimensionality of the input image.

**Valid dimension values** The dimension should be at least one and at most equal to the dimensionality of the input image.

**Valid precision values** The available precision values are `DEFAULT_PREC`, `SOURCE_PREC`, `ARITH_PREC`, `SMALL_PREC`. See the section on **Precision** (p. 2) for a more detailed description about precision.

**Border handling** This function uses `MIRRORED` border handling, see the section on **Border handling** (p. 2).

### Examples

Convoluting an image with 1d kernels in one dimension.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxConvolution1dExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    HxImageRep kernel = HxMakeGaussian1d(3.0, 0, 5.0, 10, -1);
    img = HxConvolution1d(img, kernel, 2, HxImageRep::DEFAULT_PREC);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep img = HxMakeFromFile(argv[1]);
    img = HxConvolution1dExample1(img);
    HxWriteFile(img, argv[2]);

    return 0;
}
```



Figure 2.27: Input image for the HxConvolution1d example.

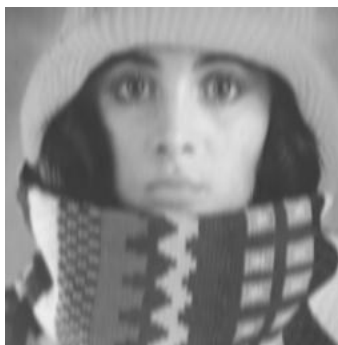


Figure 2.28: Output image of the HxConvolution1d example.

In this example we generate a 1d Gaussian kernel and convolve the input image with this kernel in the second dimension of this input image.

**See also**

**HxConvolution** (p. 43), **HxConvGauss2d** (p. 31), **HxConvKernelSeparated** (p. 38), **HxConvKernelSeparated2d** (p. 35),

**Keywords**

Filter, Convolution,

## 2.23 HxConvolution

### Synopsis

```
HxImageRep HxConvolution (HxImageRep img, HxImageRep kernel, HxImageRep::ResultPrecision resPrec)
```

### Input

`HxImageRep img` The image you want to convolve with the kernel image.

`HxImageRep kernel` The kernel for the convolution.

`ResultPrecision resPrec` This parameter determines the precision of the resulting image. The available precision values are `DEFAULT_PREC`, `SOURCE_PREC`, `ARITH_PREC`, `SMALL_PREC`. See the section on **Precision** (p. 2) for a more detailed description about precision.

### Return value

`HxImageRep` The result of convolving the input image with the kernel image.

### Description

The function `HxConvolution` convolves the input image with the kernel image. The convolution kernel can be either a scalar or vector image. If the kernel is a scalar image, the input image may be a scalar or a vector image. In case of a scalar kernel and an vector input image, the convolution is taken separately for each channel. If both kernel and image are vector images, the pixel dimension should be the same. In the latter case, each channel of the input image is convolved with the corresponding channel of the kernel.

### Remarks

**Valid input image types** All types of images are allowed for this function.

**Valid kernel image types** The size of the kernel in any dimension should be less or equal to the size of the input image in the corresponding dimension. The dimensionality of the kernel image should be equal to the dimensionality of the input image. The pixel dimensionality of the kernel should be either 1 or equal to the pixel dimensionality of the input image.

**Valid precision values** The available precision values are `DEFAULT_PREC`, `SOURCE_PREC`, `ARITH_PREC`, `SMALL_PREC`. See the section on **Precision** (p. 2) for a more detailed description about precision.

**Border handling** This function uses `MIRRORED` border handling, see the section on **Border handling** (p. 2).

### Examples

Convolving an image with a kernel.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxConvolutionExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxImageAsDouble(im1);
    im2 = HxImageAsDouble(im2);
    im1 = HxConvolution(im1, im2, HxImageRep::DEFAULT_PREC);
    im1 = HxContrastStretch(im1, 255.0);
    im1 = HxImageAsByte(im1);
}
```

```
    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxConvolutionExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.29: First input image for the HxConvolution example.

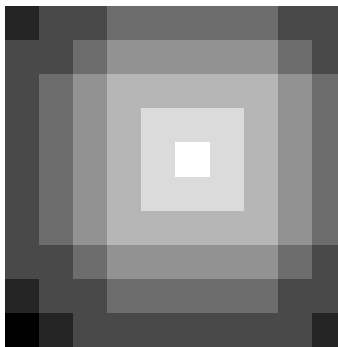


Figure 2.30: Second input image for the HxConvolution example.

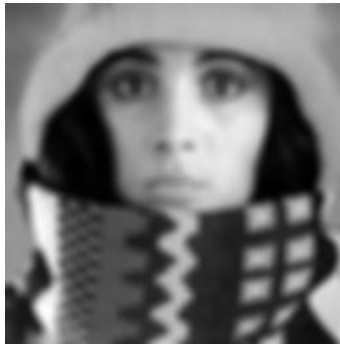


Figure 2.31: Output image of the HxConvolution example.

In this example an image is convolved with the upper left corner from that image. The example code only works on 2D images.

**See also**

**HxConvKernelSeparated** (p. 38), **HxConvGauss2d** (p. 31), **HxConvGauss3d** (p. 33), **HxConvKernelSeparated2d** (p. 35),

**Keywords**

Filter, Convolution,

## 2.24 HxCos

### Synopsis

```
HxImageRep HxCos (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the cosine of.

### Return value

HxImageRep    The pixelwise cosine of the input image.

### Description

This function takes the cosine of every pixel value of the input image. For vector images, the cosine of every element of the vector is taken.

### Remarks

*No possible errors*    This function can not result in any errors.

### Examples

Taking the cosine of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxCosExample1() {
    HxSizes sizes(256, 256, 0);
    double data[256*256];
    for(int i=0; i<256; i++)
        for(int j=0; j<256; j++) {
            data[i+j*256]=(i/32.0)*(i/32.0);
        }
    HxImageRep img = HxMakeFromDoubleData(1, 2, sizes, data);
    img = HxCos(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep img = HxCosExample1();
    HxWriteFile(img, argv[1]);

    return 0;
}
```



Figure 2.32: Output image of the HxCos example.

This example generates an images with a number of stripes, using the cosine function.

**See also**

**HxSin** (p. 158), **HxTan** (p. 172), **HxAcos** (p. 5), **HxCosh** (p. 48),

**Keywords**

Unary, Trigonometric,

## 2.25 HxCosh

### Synopsis

HxImageRep HxCosh (HxImageRep img)

### Input

HxImageRep img   The image you want to take the hyperbolic cosine of.

### Return value

HxImageRep   The pixelwise hyperbolic cosine of the input image.

### Description

This function takes the hyperbolic cosine of every pixel value of the input image. For vector images, the hyperbolic cosine of every element of the vector is taken.

### Remarks

*Overflow error*   The function HxCosh can result in an overflow error.

### Examples

Taking the hyperbolic cosine of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxCoshExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxContrastStretch(img, 4.0);
    img = HxAddVal(img, -2.0);
    img = HxCosh(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);
    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxCoshExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.33: Input image for the HxCosh example.



Figure 2.34: Output image of the HxCosh example.

This example illustrates the result of taken the hyperbolic cosine of an image.

**See also**

**HxCos** (p. 46), **HxSinh** (p. 160), **HxAcos** (p. 5),

**Keywords**

Unary, Trigonometric,

## 2.26 HxCross

### Synopsis

```
HxImageRep HxCross(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The cross product of the input images

### Description

The function HxCross takes the cross product of the two input images on a pixel-by-pixel basis. If pixel one is given by (a,b,c) and pixel two is given by (d,e,f), the cross product is defined as (bf-ce, cd-af, ae-bd). Only vector pixel types of the dimension 3 are valid.

### Remarks

**Valid image types**    The two input images should have the same dimensionality, a pixeldimensionality of 3 and the same size.

**Overflow error**    The function HxCross can result in an overflow error.

### See also

**HxAdd** (p. 7), **HxMul** (p. 130), **HxDiv** (p. 53),

### Keywords

Binary, Arithmetic,

## 2.27 HxCrossVal

### Synopsis

```
HxImageRep HxCrossVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The input value

### Return value

HxImageRep    The cross product of the input image with the input value, pixel wise.

### Description

The function HxCrossVal takes the cross product of the input image pixels with the input value. pixel-by-pixel basis. If pixel one is given by (a,b,c) and the input value is given by (d,e,f), the cross product is defined as (bf-ce, cd-af, ae-bd). Only vector pixel types of dimension 3 are valid. The dimensionality of the input value should also be 3.

### Remarks

*Valid image types and input values*    The dimensionality of the input value and the pixel dimensionality of the input image should both be 3.

*Overflow error*    The function HxCrossVal can result in an overflow error.

### See also

[HxAddVal](#) (p. 10), [HxMulVal](#) (p. 132), [HxDivVal](#) (p. 55), [HxCross](#) (p. 50),

### Keywords

Binary value, Arithmetic,

## 2.28 HxDistanceTransform

### Synopsis

HxImageRep HxDistanceTransform (HxImageRep img)

### Return value

HxImageRep The distance transform of the input image.

### Description

The function HxDistanceTransform calculates the shortest distance (approximately) to the background for every object pixel, through a recursive morphological operation. Background pixels are defined as pixels with value 0, while object pixels are defined as pixels with value 1. The input image should be a 2D scalar image of the type short, byte or integer. The image should also contain at least one background pixel. The resulting image is a scalar double 2D image.

### Remarks

**Valid image types** This function only accepts 2D scalar image, with short, byte or integer types, containing solely pixel with value 0 or value 1. The input image should at least contain one pixel with value 0.

### See also

anchor\_HxParabolicErosion, anchor\_HxParabolicDilation,

### Keywords

Morphology,

## 2.29 HxDiv

### Synopsis

```
HxImageRep HxDiv(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1   The first input image

HxImageRep im2   The second input image

### Return value

HxImageRep   The division of the input images

### Description

The function HxDiv divides two images on a pixel-by-pixel basis. Vector pixel types of the same dimension are divided component wise. Pixel types of different dimensions can be divided, as long as the pixel dimensionality of one of the input images is 1. The image with pixeldimensionality 1 is then treated as an image with the same pixeldimensionality as the other image (with equal values per pixel). The pixel values of the second input image may not be zero, or have zero valued components.

### Remarks

**Admissable image types** The two input images should have the same dimensionality, the same pixeldimensionality (or the pixeldimensionality of one of the pixels should be 1) and the same size.

**Restriction on the second input image** The pixel values of the second input image may not be zero, or have zero valued components.

**Overflow error** The function HxDiv can result in an overflow error.

### Examples

#### Dividing two images

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxDivExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxImageAsDouble(im1);
    im2 = HxImageAsDouble(im2);
    im2 = HxDivVal(im2, 150.0);
    im2 = HxAddVal(im2, 1.0);
    im1 = HxDiv(im1, im2);
    im1 = HxContrastStretch(im1, 255.0);
    im1 = HxImageAsByte(im1);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxDivExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.35: First input image for the HxDiv example.

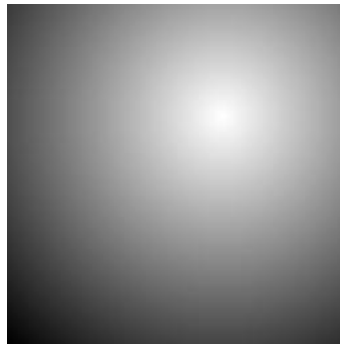


Figure 2.36: Second input image for the HxDiv example.



Figure 2.37: Output image of the HxDiv example.

In this example we divide two (scalar) images. After dividing the images, we stretch the result for visualization purposes.

**See also**

**HxSub** (p. 165), **HxMul** (p. 130), **HxAdd** (p. 7),

**Keywords**

Binary, Arithmetic,

## 2.30 HxDivVal

### Synopsis

```
HxImageRep HxDivVal(HxImageRep img, HxValue val)
```

### Input

`HxImageRep img` The input image you want to divide by a value.

`HxValue val` The input value.

### Return value

`HxImageRep` The input image divided by the input value, pixel wise.

### Description

The function `HxDivVal` divides the values of the input image by the the input value on a pixel-by-pixel basis. If the pixel type of the input image and the type of the value are both vectors with the same dimensionality, they are treated component wise. If the input image has vector type pixels and the dimensionality of the input value is 1, every component of a pixel vector is divided by the input value.

### Remarks

***Valid input images and values*** The dimensionality of the input value should be 1 or equal to the pixel dimensionality of the image. The input value may not be zero, nor may it contain zero components.

***Overflow error*** The function `HxDivVal` can result in an overflow error.

### See also

`HxMulVal` (p. 132), `HxPowVal` (p. 150), `HxDiv` (p. 53),

### Keywords

Binary value, Arithmetic,

## 2.31 HxDot

### Synopsis

```
HxImageRep HxDot (HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The dot product of the input images

### Description

The function HxDot takes the dot product of two images on a pixel-by-pixel basis. The dot product of the scalar values is defined as the product of the values. The dot product of two vector of the same dimensionality is the sum of product of the corresponding components of the vectors. The dot product is only valid for image with the same pixel dimensionality.

### Remarks

*Admissable image types*    The two input images should have the same dimensionality, the same pixel dimensionality and the same size.

*Overflow error*    The function HxDot can result in an overflow error.

### See also

**HxAdd** (p. 7), **HxCross** (p. 50), **HxMul** (p. 130),

### Keywords

Binary, Arithmetic,

## 2.32 HxDotVal

### Synopsis

```
HxImageRep HxDotVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The input value

### Return value

HxImageRep    The dot product of the input image and the input value.

### Description

The function HxDotVal takes the dot product of the HxValue *val* with every pixel value in the input image and stores the result in the corresponding pixel of the output image. The dimensionality of ‘*val*’ and the *pixeldimensionality* of the input image must be the same. If ‘*val*’ is a scalar and the pixels of the input image are scalars, the values are multiplied.

### Remarks

*Admissable image and value types*    The dimensionality of ‘*val*’ and the *pixeldimensionality* of the input image must be the same.

*Overflow error*    The function HxDotVal can result in an overflow error.

### See also

**HxDot** (p. 56), **HxAddVal** (p. 10), **HxDivVal** (p. 55), **HxMulVal** (p. 132),

### Keywords

Binary value, Arithmetic,

## 2.33 HxEqual

### Synopsis

```
HxImageRep HxEqual(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The result of comparing whether the first input image is equal to the second input input, pixelwise. When a pixel from the first input image is equal to the corresponding pixel of the second input image, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function HxEqual compares the corresponding pixels from the input images. If the value of the pixel of the first input image is equal to the value of the pixel of the second input pixel, the corresponding pixel in the result image will be 1, otherwise it will be 0. The function is equivalently defined for vector images.

The dimensionality, sizes and pixel dimensionality of the input images must equal. The resulting images is an image with the same dimensions as the input images, with integer pixel values.

### Remarks

*Valid input images*    This function is only properly defined images with equal sizes, equal dimensionality and equal pixeldimensionality.

### See also

**HxNotEqual** (p. 138), **HxEqualVal** (p. 59), **HxNotEqualVal** (p. 139),

### Keywords

Comparison, Binary,

## 2.34 HxEqualVal

### Synopsis

```
HxImageRep HxEqualVal(HxImageRep im1, HxValue val)
```

### Input

HxImageRep im1    The first input image.

HxValue val    The input value.

### Return value

HxImageRep    The result of comparing whether the input image is equal to the input value, pixelwise. When a pixel from the input image is equal to input value, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function HxEqualVal compares the pixels from the input image to the HxValue 'val'. If the value of the pixel of the input image is equal to 'val', the corresponding pixel in the result image will be 1, otherwise it will be 0. The function is equivalently defined for vector images.

The pixel dimensionality of the input image must be equal to the dimensionality of 'val'. The resulting images is an image with the same dimensions as the input image, with integer pixel values.

### Remarks

*Valid input image and value 'val'*    The pixel dimensionality of the input image must be equal to the dimensionality of 'val'.

### See also

[HxEqual](#) (p. 58), [HxNotEqualVal](#) (p. 139), [HxNotEqual](#) (p. 138),

### Keywords

Comparison, Binary value,

## 2.35 HxExp

### Synopsis

```
HxImageRep HxExp (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to take the exponent of.

### Return value

HxImageRep    The pixelwise exponent of the input image.

### Description

The function `HxExp` takes the exponent of every pixel value in the input value and stores it in the corresponding pixels of the return image.

### Remarks

*Overflow error*    The function `HxExp` can result in an overflow error.

### See also

`HxLog` (p. 111), `HxLog10` (p. 109), `HxPow` (p. 149), `HxPowVal` (p. 150),

### Keywords

Unary, Arithmetic,

## 2.36 HxExportByteData

### Synopsis

```
void HxExportByteData (HxImageRep img, HxByte *data)
```

### Input

HxImageRep img    The image you want to convert into an array of HxBytes

### Output

HxByte \*data    The array of HxByte you want to store the image contents in.

### Description

The function HxExportByteData places the data of the image img in the array 'data'. Starting with the upper left pixel it stores every row from left to right. The array should have a size equal to the dimensionality of the pixels times the number of pixels in the image. The array should already be allocated.

### Remarks

*Valid image types*    This function works on byte images.

### See also

[HxExportDoubleData](#) (p. 62), [HxExportFloatData](#) (p. 63), [HxMakeFromByteData](#) (p. 113),

### Keywords

Export, Conversion,

## 2.37 HxExportDoubleData

### Synopsis

```
void HxExportByteData (HxImageRep img, double *data)
```

### Input

HxImageRep img    The image you want to convert into an array of HxBytes

### Output

double \*data    The array of doubles you want to store the image contents in.

### Description

The function HxExportDoubleData places the data of the image img in the array 'data'. Starting with the upper left pixel it stores every row from left to right. The array should have a size equal to the dimensionality of the pixels times the number of pixels in the image. The array should already be allocated.

### Remarks

*Valid image types*    This function works on double images.

### See also

[HxExportByteData](#) (p. 61), [HxExportFloatData](#) (p. 63), [HxMakeFromDoubleData](#) (p. 114),

### Keywords

Export, Conversion,

## 2.38 HxExportFloatData

### Synopsis

```
void HxExportFloatData (HxImageRep img, float *data)
```

### Input

HxImageRep img    The image you want to convert into an array of floats.

### Output

float \*data    The array of floats you want to store the image contents in.

### Description

The function HxExportFloatData places the data of the image img in the array 'data'. Starting with the upper left pixel it stores every row from left to right. The array should have a size equal to the dimensionality of the pixels times the number of pixels in the image. The array should already be allocated.

### Remarks

*Valid image types*    This function works on float images.

### See also

[HxExportDoubleData](#) (p. 62), [HxExportByteData](#) (p. 61), [HxMakeFromFloatData](#) (p. 117),

### Keywords

Export, Conversion,

## 2.39 HxExportIntData

### Synopsis

```
void HxExportIntData (HxImageRep img, int *data)
```

### Input

HxImageRep img    The image you want to convert into an array of HxBytes

### Output

int \*data    The array of integers you want to store the image contents in.

### Description

The function HxExportIntData places the data of the image img in the array 'data'. Starting with the upper left pixel it stores every row from left to right. The array should have a size equal to the dimensionality of the pixels times the number of pixels in the image. The array should already be allocated.

### Remarks

*Valid image types*    This function works on integers images.

### See also

**HxExportDoubleData** (p. 62), **HxExportByteData** (p. 61), **HxMakeFromIntData** (p. 118),

### Keywords

Export, Conversion,

## 2.40 HxExportMatlabPixels

### Synopsis

```
void HxExportMatlabPixels (HxImageRep img, double *pixels)
```

### Input

HxImageRep img    The image you want to convert into an array of HxBytes

### Output

double \*pixels    The array of Matlab pixels (i.e. doubles) you want to store the image contents in.

### Description

The function HxExportMatlabPixels places the data of the image `img` in the array `'pixels'`. Starting with the upper left pixel it stores every row from left to right. The array should have a size equal to the dimensionality of the pixels times the number of pixels in the image. The array should already be allocated.

### Remarks

*Valid image types*    This function works on all types of images.

### See also

[HxExportByteData](#) (p. 61), [HxExportFloatData](#) (p. 63), [HxMakeFromDoubleData](#) (p. 114),

### Keywords

Export, Conversion,

## 2.41 HxExportShortData

### Synopsis

```
void HxExportShortData (HxImageRep img, short *data)
```

### Input

HxImageRep img    The image you want to convert into an array of shorts.

### Output

short \*data    The array of shorts you want to store the image contents in.

### Description

The function HxExportShortData places the data of the image img in the array 'data'. Starting with the upper left pixel it stores every row from left to right. The array should have a size equal to the dimensionality of the pixels times the number of pixels in the image. The array should already be allocated.

### Remarks

*Valid image types*    This function works on short images.

### See also

[HxExportDoubleData](#) (p. 62), [HxExportByteData](#) (p. 61), [HxMakeFromShortData](#) (p. 120),

### Keywords

Export, Conversion,

## 2.42 HxExportSI

### Synopsis

```
IMAGE* HxExportSI (HxImageRep img)
```

### Input

HxImageRep img    The image you want to convert to the ScilImage format.

### Return value

IMAGE\*    The ScilImage image pointer that should refer to the image data in ScilImage format.

### Description

The function HxExportSI converts the input image to an image in ScilImage format. An image in ScilImage format can be converted to an image in Horus format by using the constructor of HxImageRep with as input an IMAGE\* or by using the function **HxMakeFromSI** (p. 121)

### Remarks

*No possible errors*    This function can not result in any errors.

### See also

**HxExportByteData** (p. 61), **HxExportDoubleData** (p. 62), **HxMakeFromFileSI** (p. 116), [<reference>HxMakeFromSI</reference>](#),

### Keywords

Export, Conversion,

## 2.43 HxExtend

### Synopsis

```
HxImageRep HxExtend (HxImageRep img, HxImageRep background, HxPoint  
begin)
```

### Input

HxImageRep *img*    The image that you want to place in the background image.

HxImageRep *background*    The background image where you want to place the image *img* in.

### Return value

HxImageRep    The background image with the image *img* overlaid at position 'begin'.

### Description

The function HxExtend places the image *img* in the background image at position 'begin', i.e. the upperleft corner of image *img* is placed at position 'begin'.

### Remarks

**Valid image types**    The images *img* and *background* should have the same dimensionality and the same pixel dimensionality.

**Valid values for 'begin'**    'begin' should be a valid position in the background image.

**Valid size for image *img***    The size of image *img* should be such that all pixels have a valid position in the background image, when placing the upperleft corner of the image at position 'begin' in the background image.

### See also

**HxExtendVal** (p. 69), **HxRestrict** (p. 152),

### Keywords

Geometric,

## 2.44 HxExtendVal

### Synopsis

```
HxImageRep HxExtendVal (HxImageRep img, HxSizes newSize, HxValue background, HxPoint begin)
```

### Input

`HxImageRep img` The image that you want to place in a larger image.

`HxSizes newSize` The size of the new image.

`HxValue background` The value of the background of the new image, where the background is that part of the image that does not originate from the input image.

`HxPoint begin` The position of the upperleft corner of the image ‘img’ in the background image.

### Return value

`HxImageRep` The background image (as described by ‘newSize’ and ‘background’) with the image img overlaid at position ‘begin’.

### Description

The function `HxExtendVal` places the image `img` in the background image at position ‘begin’, i.e. the upperleft corner of image `img` is placed at position ‘begin’. The background image has a constant value ‘background’ and its size is as described by ‘newSize’.

### Remarks

**Valid image types** The images `img` and the image constructed from ‘newSize’ and ‘background’ should have the same dimensionality and the same pixel dimensionality.

**Valid values for ‘newSize’ and ‘background’** The size of the background image, ‘newSize’, should be larger than the image size for every dimension. ‘newSize’ should have the same dimensionality as the input image. The value ‘background’ should have the same dimensionality as the pixel dimensionality of the input image.

**Valid values for ‘begin’** ‘begin’ should be a valid position in the background image.

**Valid size for image `img`** The size of image `img` should be such that all pixels have a valid position in the background image, when placing the upperleft corner of the image at position ‘begin’ in the background image.

### See also

[HxExtend](#) (p. 68), [HxRestrict](#) (p. 152),

### Keywords

Geometric,

## 2.45 HxFloor

### Synopsis

```
HxImageRep HxFloor(HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to take the floor of.

### Return value

HxImageRep    The floor value of the input image.

### Description

The function HxFloor takes the floor value of each pixel value, where the floor of a number is the largest integer that is smaller than the pixel value. The floor of a vector is defined by separately taking the floor of the components of the vector.

### Remarks

*No possible errors*    This function can not result in any errors.

### See also

**HxPixMax** (p. 144), **HxPixMin** (p. 145), **HxCeil** (p. 24),

### Keywords

Unary, Arithmetic,

## 2.46 HxGauss

### Synopsis

```
HxImageRep HxGauss (HxImageRep img, double sigma, double truncation = 3.0)
```

### Input

`HxImageRep img` The image you want to convolve with a Gaussian.

`double sigma` The sigma of the Gaussian, used in the convolution.

`double truncation` The truncation determines at how many sigma the Gaussian filter is clipped.

### Return value

`HxImageRep` The result of convolving the input image `img` with a Gaussian. The result has double precision pixels.

### Description

The function `HxGauss` convolves the input image with a Gaussian filter function that has a sigma 'sigma'. The truncation determines the size of the filter in sigma, where the filter size, in sigma, is 2 times the truncation plus 1. `HxGauss` convolves the image with a 1 dimensional Gaussian kernel in every dimension. For vector images, each channel is convolved separately with the Gaussian kernel.

### Remarks

**Valid image types** All types of images are allowed for this function. For vector images, each channel is treated separately.

**Valid values for the parameters** sigma and truncation should both be greater than zero.

**Border handling** This function uses `MIRRORED` border handling, see the section on **Border handling** (p. 2).

### See also

`HxGaussDerivative2d` (p. 72), `HxConvolution` (p. 43),

### Keywords

Filter, Convolution, Gauss,

## 2.47 HxGaussDerivative2d

### Synopsis

```
HxImageRep HxGaussDerivative2d (HxImageRep img, double sigma, int
orderDerivx, int orderDerivy, double truncation = 3.0)
```

### Input

`HxImageRep img` The image you want to convolve with a Gaussian kernel. The used Gaussian kernel is composed from two separate kernels, one for the x-direction and one for the y-direction.

`double sigma` The sigma of the Gaussian kernel in both directions.

`int orderDerivx` The order of the derivative of the Gaussian kernel in the x-direction.

`int orderDerivy` The order of the derivative of the Gaussian kernel in the y-direction.

`double truncation` This parameter determines at how many sigma the Gaussian filter is clipped in both directions.

### Return value

`HxImageRep` The result of convolving the input image with a Gaussian kernel as described by the parameters `sigma`, `orderDerivx`, `orderDerivy` and `truncation`.

### Description

The function `HxGaussDerivative2d` convolves 2D-images with a Gaussian filter function. The filter function is separated in the x- and y-direction, where the order of the derivative in both directions can be given separately. The truncation (in both directions) determines the size of the filter in sigma (of the particular direction), where the filter size, in sigma, is 2 times the truncation plus 1. Sigma is equal for both directions and is given by 'sigma'. For vector images, each channel is convolved separately with the Gaussian kernel.

### Remarks

**Valid image types** All 2D types of images are allowed for this function.

**Valid values for the parameters** `sigma` and `truncation` should both be greater than zero. `orderDerivx` and `orderDerivy` should be larger or equal to zero.

**Border handling** This function uses MIRRORED border handling, see the section on **Border handling** (p. 2).

### See also

**HxGauss** (p. 71), **HxConvGauss2d** (p. 31), **HxGaussDerivative3d** (p. 73),

### Keywords

Filter, Convolution, Gauss, Gaussian derivative,

## 2.48 HxGaussDerivative3d

### Synopsis

```
HxImageRep HxGaussDerivative3d (HxImageRep img, double sigma, int
orderDerivx, int orderDerivy, int orderDerivz, double truncation =
3.0)
```

### Input

`HxImageRep img` The image you want to convolve with a Gaussian kernel.

`double sigma` The sigma of the Gaussian kernel (in both directions).

`int orderDerivx` The order of the derivative of the Gaussian kernel in the x-direction.

`int orderDerivy` The order of the derivative of the Gaussian kernel in the y-direction.

`int orderDerivz` The order of the derivative of the Gaussian kernel in the z-direction.

`double truncation` This parameter determines at how many sigma the Gaussian filter is clipped.

### Return value

`HxImageRep` The result of convolving the input image with a Gaussian kernel as described by the parameters `sigma`, `orderDerivx`, `orderDerivy`, `orderDerivz` and `truncation`.

### Description

The function `HxGaussDerivative3d` convolves 2D-images with a Gaussian filter function. The filter function is separated in the x-, y- and z-direction, where the order of the derivative in all directions can be given separately. The truncation (in all directions) determines the size of the filter in sigma (of the particular direction), where the filter size, in sigma, is 2 times the truncation plus 1. Sigma is equal for all directions and is given by 'sigma'. For vector images, each channel is convolved separately with the Gaussian kernel.

### Remarks

**Valid image types** All 3D types of images are allowed for this function.

**Valid values for the parameters** `sigma` and `truncation` should both be greater than zero. `orderDerivx`, `orderDerivy` and `orderDerivz` should be larger or equal to zero.

**Border handling** This function uses MIRRORED border handling, see the section on **Border handling** (p. 2).

### See also

`HxGauss` (p. 71), `HxConvGauss3d` (p. 33), `HxGaussDerivative2d` (p. 72),

### Keywords

Filter, Convolution, Gauss, Gaussian derivative,

## 2.49 HxGreaterEqual

### Synopsis

```
HxImageRep HxGreaterEqual(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The result of comparing whether the first input image is greater than or equal to the second input input, pixelwise. When a pixel from the first input image is greater than or equal to the corresponding pixel of the second input image, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function HxGreaterEqual compares the corresponding pixels from the input images, which must be scalar. If the value of the pixel of the first input image is greater than or equal to the value of the pixel of the second input pixel, the corresponding pixel in the result image will be 1, otherwise it will be 0.

The dimensions of the input images must equal, but the pixel types can be anything, as long as they are scalar. The resulting images is an image with the same dimensions as the input images, with integer pixel values.

### Remarks

*Valid input images*    This function is only properly defined for scalar input images with equal sizes and equal dimensionality.

### See also

**HxLessThan** (p. 105), **HxGreaterThan** (p. 76), **HxLessEqual** (p. 103), **HxEqual** (p. 58), **HxGreaterEqualVal** (p. 75),

### Keywords

Comparison, Binary,

## 2.50 HxGreaterEqualVal

### Synopsis

```
HxImageRep HxGreaterEqualVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The value you want the compare the image values with.

### Return value

HxImageRep    When a pixel from the first input image is greater or equal to the given value input image, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function HxGreaterEqualVal compares the pixel values of the input image with the value 'val'. When the value of the pixel is greater or equal than the value, the corresponding pixel in the output image has value 1, otherwise the value of the pixel is 0. This operation is only defined for scalar images (and values).

### Remarks

*Valid input images*    Only scalar images are valid. The image and value do not have to be of the same type.

*Valid 'val' types*    Only scalar values are valid.

### See also

**HxLessEqualVal** (p. 104), **HxLessThanVal** (p. 108), **HxGreaterThanVal** (p. 79), **HxGreaterEqual** (p. 74),

### Keywords

Binary value, Comparison,

## 2.51 HxGreaterThan

### Synopsis

```
HxImageRep HxGreaterThan(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1   The first input image.

HxImageRep im2   The second input image.

### Return value

HxImageRep   The result of comparing whether the first input image is greater than the second input input, pixelwise. When a pixel from the first input image is greater than the corresponding pixel of the second input image, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function HxGreaterThan compares the corresponding pixels from the input images, which must be scalar. If the value of the pixel of the first input image is greater than the value of the pixel of the second input pixel, the corresponding pixel in the result image will be 1, otherwise it will be 0.

The dimensions of the input images must equal, but the pixel types can be anything, as long as they are scalar. The resulting images is an image with the same dimensions as the input images, with integer pixel values.

### Remarks

*Valid input images*   This function is only properly defined for scalar input images with equal sizes and equal dimensionality.

### Examples

Example of using HxGreaterThan

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxGreaterThanExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxGreaterThan(im1, im2);
    im1 = HxContrastStretch(im1, 255);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxGreaterThanExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.38: First input image for the HxGreaterThan example.

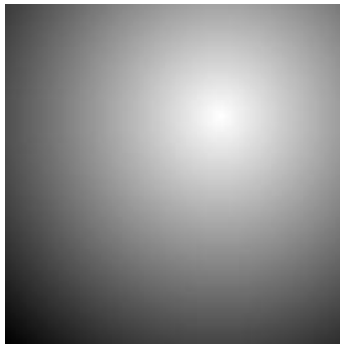


Figure 2.39: Second input image for the HxGreaterThan example.

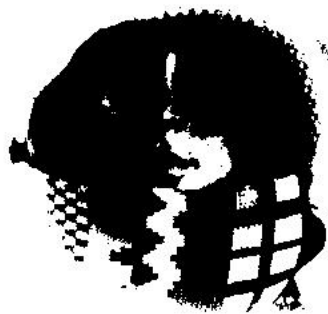


Figure 2.40: Output image of the HxGreaterThan example.

In the example code we use `HxGreaterThan` to compare the two input images. The contrast stretch is used for visualization purposes.

**See also**

**HxGreaterEqual** (p. 74), **HxLessThan** (p. 105), **HxLessEqual** (p. 103), **HxEqual** (p. 58), **HxGreater-ThanVal** (p. 79),

**Keywords**

Comparison, Binary,

## 2.52 HxGreaterThanVal

### Synopsis

```
HxImageRep HxGreaterThanVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The value you want the compare the image values with.

### Return value

HxImageRep    When a pixel from the first input image is greater than the given value input image, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function HxLessThanVal compares the pixel values of the input image with the value 'val'. When the value of the pixel is greater than the value, the corresponding pixel in the output image has value 1, otherwise the value of the pixel is 0. This operation is only defined for scalar images (and values).

### Remarks

*Valid input images*    Only scalar images are valid. The image and value do not have to be of the same type.

*Valid value types*    Only scalar values are valid.

### See also

**HxGreaterEqualVal** (p. 75), **HxLessThanVal** (p. 108), **HxLessEqualVal** (p. 104), **HxEqualVal** (p. 59), **HxGreaterThan** (p. 76),

### Keywords

Binary value, Comparison,

## 2.53 HxImageAsByte

### Synopsis

```
HxImageRep HxImageAsByte (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with scalar byte pixel values.

### Description

The function `HxImageAsByte` converts the input image to an image with scalar byte pixel values. The input image should be a scalar image.

### Remarks

*Valid input images*    All scalar images are valid input images.

### See also

[HxImageAsVec2Byte](#) (p. 85), [HxImageAsVec3Byte](#) (p. 90), [HxImageAsShort](#) (p. 84), [HxImageAsInt](#) (p. 83), [HxImageAsFloat](#) (p. 82), [HxImageAsDouble](#) (p. 81),

### Keywords

Conversion,

## 2.54 HxImageAsDouble

### Synopsis

```
HxImageRep HxImageAsDouble (HxImageRep img)
```

### Input

HxImageRep img    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with scalar double pixel values.

### Description

The function `HxImageAsDouble` converts the input image to an image with scalar double pixel values. The input image should be a scalar image.

### Remarks

*Valid input images*    All scalar images are valid input images.

### See also

[HxImageAsVec2Double](#) (p. 86), [HxImageAsVec3Double](#) (p. 91), [HxImageAsByte](#) (p. 80), [HxImageAsShort](#) (p. 84), [HxImageAsInt](#) (p. 83), [HxImageAsFloat](#) (p. 82),

### Keywords

Conversion,

## 2.55 HxImageAsFloat

### Synopsis

```
HxImageRep HxImageAsFloat (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with scalar float pixel values.

### Description

The function `HxImageAsFloat` converts the input image to an image with scalar float pixel values. The input image should be a scalar image.

### Remarks

*Valid input images*    All scalar images are valid input images.

### See also

[HxImageAsVec2Float](#) (p. 87), [HxImageAsVec3Float](#) (p. 92), [HxImageAsByte](#) (p. 80), [HxImageAsShort](#) (p. 84), [HxImageAsInt](#) (p. 83), [HxImageAsDouble](#) (p. 81),

### Keywords

Conversion,

## 2.56 HxImageAsInt

### Synopsis

```
HxImageRep HxImageAsInt (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with scalar int pixel values.

### Description

The function `HxImageAsInt` converts the input image to an image with scalar int pixel values. The input image should be a scalar image.

### Remarks

*Valid input images*    All scalar images are valid input images.

### See also

[HxImageAsVec2Int](#) (p. 88), [HxImageAsVec3Int](#) (p. 93), [HxImageAsByte](#) (p. 80), [HxImageAsShort](#) (p. 84), [HxImageAsFloat](#) (p. 82), [HxImageAsDouble](#) (p. 81),

### Keywords

Conversion,

## 2.57 HxImageAsShort

### Synopsis

```
HxImageRep HxImageAsShort (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with scalar short pixel values.

### Description

The function `HxImageAsShort` converts the input image to an image with scalar short pixel values. The input image should be a scalar image.

### Remarks

*Valid input images*    All scalar images are valid input images.

### See also

[HxImageAsVec2Short](#) (p. 89), [HxImageAsVec3Short](#) (p. 94), [HxImageAsByte](#) (p. 80), [HxImageAsInt](#) (p. 83), [HxImageAsFloat](#) (p. 82), [HxImageAsDouble](#) (p. 81),

### Keywords

Conversion,

## 2.58 HxImageAsVec2Byte

### Synopsis

```
HxImageRep HxImageAsVec2Byte (HxImageRep img)
```

### Input

HxImageRep img    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with byte vector pixels with dimensionality 2.

### Description

The function `HxImageAsVec2Byte` converts the input image to an image with vector byte pixel values, with dimensionality 2. The input image should be either a scalar image or a vector image with pixel dimensionality 2. If the input image is a scalar image the pixel value of that pixel is converted to byte and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 2 are valid input images.

### See also

[HxImageAsByte](#) (p. 80), [HxImageAsVec3Byte](#) (p. 90), [HxImageAsVec2Short](#) (p. 89), [HxImageAsVec2Int](#) (p. 88), [HxImageAsVec2Float](#) (p. 87), [HxImageAsVec2Double](#) (p. 86),

### Keywords

Conversion,

## 2.59 HxImageAsVec2Double

### Synopsis

HxImageRep HxImageAsVec2Double (HxImageRep img)

### Input

HxImageRep img    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with double vector pixels with dimensionality 2.

### Description

The function `HxImageAsVec2Double` converts the input image to an image with vector double pixel values, with dimensionality 2. The input image should be either a scalar image or a vector image with pixel dimensionality 2. If the input image is a scalar image the pixel value of that pixel is converted to double and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 2 are valid input images.

### See also

[HxImageAsDouble](#) (p. 81), [HxImageAsVec3Double](#) (p. 91), [HxImageAsVec2Byte](#) (p. 85), [HxImageAsVec2Short](#) (p. 89), [HxImageAsVec2Int](#) (p. 88), [HxImageAsVec2Float](#) (p. 87),

### Keywords

Conversion,

## 2.60 HxImageAsVec2Float

### Synopsis

```
HxImageRep HxImageAsVec2Float (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with float vector pixels with dimensionality 2.

### Description

The function `HxImageAsVec2Float` converts the input image to an image with vector float pixel values, with dimensionality 2. The input image should be either a scalar image or a vector image with pixel dimensionality 2. If the input image is a scalar image the pixel value of that pixel is converted to float and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 2 are valid input images.

### See also

[HxImageAsFloat](#) (p. 82), [HxImageAsVec3Float](#) (p. 92), [HxImageAsVec2Byte](#) (p. 85), [HxImageAsVec2Short](#) (p. 89), [HxImageAsVec2Int](#) (p. 88), [HxImageAsVec2Double](#) (p. 86),

### Keywords

Conversion,

## 2.61 HxImageAsVec2Int

### Synopsis

```
HxImageRep HxImageAsVec2Int (HxImageRep img)
```

### Input

HxImageRep img    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with int vector pixels with dimensionality 2.

### Description

The function `HxImageAsVec2Int` converts the input image to an image with vector int pixel values, with dimensionality 2. The input image should be either a scalar image or a vector image with pixel dimensionality 2. If the input image is a scalar image the pixel value of that pixel is converted to int and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 2 are valid input images.

### See also

[HxImageAsInt](#) (p. 83), [HxImageAsVec3Int](#) (p. 93), [HxImageAsVec2Byte](#) (p. 85), [HxImageAsVec2Short](#) (p. 89), [HxImageAsVec2Float](#) (p. 87), [HxImageAsVec2Double](#) (p. 86),

### Keywords

Conversion,

## 2.62 HxImageAsVec2Short

### Synopsis

```
HxImageRep HxImageAsVec2Short (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with short vector pixels with dimensionality 2.

### Description

The function `HxImageAsVec2Short` converts the input image to an image with vector short pixel values, with dimensionality 2. The input image should be either a scalar image or a vector image with pixel dimensionality 2. If the input image is a scalar image the pixel value of that pixel is converted to short and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 2 are valid input images.

### See also

[HxImageAsShort](#) (p. 84), [HxImageAsVec3Short](#) (p. 94), [HxImageAsVec2Byte](#) (p. 85), [HxImageAsVec2Int](#) (p. 88), [HxImageAsVec2Float](#) (p. 87), [HxImageAsVec2Double](#) (p. 86),

### Keywords

Conversion,

## 2.63 HxImageAsVec3Byte

### Synopsis

```
HxImageRep HxImageAsVec3Byte (HxImageRep img)
```

### Input

HxImageRep img    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with byte vector pixels with dimensionality 3.

### Description

The function `HxImageAsVec3Byte` converts the input image to an image with vector byte pixel values, with dimensionality 3. The input image should be either a scalar image or a vector image with pixel dimensionality 3. If the input image is a scalar image the pixel value of that pixel is converted to byte and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 3 are valid input images.

### See also

[HxImageAsByte](#) (p. 80), [HxImageAsVec2Byte](#) (p. 85), [HxImageAsVec3Short](#) (p. 94), [HxImageAsVec3Int](#) (p. 93), [HxImageAsVec3Float](#) (p. 92), [HxImageAsVec3Double](#) (p. 91),

### Keywords

Conversion,

## 2.64 HxImageAsVec3Double

### Synopsis

```
HxImageRep HxImageAsVec3Double (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with double vector pixels with dimensionality 3.

### Description

The function `HxImageAsVec3Double` converts the input image to an image with vector double pixel values, with dimensionality 3. The input image should be either a scalar image or a vector image with pixel dimensionality 3. If the input image is a scalar image the pixel value of that pixel is converted to double and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 3 are valid input images.

### See also

[HxImageAsDouble](#) (p. 81), [HxImageAsVec2Double](#) (p. 86), [HxImageAsVec3Byte](#) (p. 90), [HxImageAsVec3Short](#) (p. 94), [HxImageAsVec3Int](#) (p. 93), [HxImageAsVec3Float](#) (p. 92),

### Keywords

Conversion,

## 2.65 HxImageAsVec3Float

### Synopsis

```
HxImageRep HxImageAsVec3Float (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with float vector pixels with dimensionality 3.

### Description

The function `HxImageAsVec3Float` converts the input image to an image with vector float pixel values, with dimensionality 3. The input image should be either a scalar image or a vector image with pixel dimensionality 3. If the input image is a scalar image the pixel value of that pixel is converted to float and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 3 are valid input images.

### See also

[HxImageAsFloat](#) (p. 82), [HxImageAsVec2Float](#) (p. 87), [HxImageAsVec3Byte](#) (p. 90), [HxImageAsVec3Short](#) (p. 94), [HxImageAsVec3Int](#) (p. 93), [HxImageAsVec3Double](#) (p. 91),

### Keywords

Conversion,

## 2.66 HxImageAsVec3Int

### Synopsis

```
HxImageRep HxImageAsVec3Int (HxImageRep img)
```

### Input

HxImageRep img    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with int vector pixels with dimensionality 3.

### Description

The function `HxImageAsVec3Int` converts the input image to an image with vector int pixel values, with dimensionality 3. The input image should be either a scalar image or a vector image with pixel dimensionality 3. If the input image is a scalar image the pixel value of that pixel is converted to int and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 3 are valid input images.

### See also

`HxImageAsInt` (p. 83), `HxImageAsVec2Int` (p. 88), `HxImageAsVec3Byte` (p. 90), `HxImageAsVec3Short` (p. 94), `HxImageAsVec3Float` (p. 92), `HxImageAsVec3Double` (p. 91),

### Keywords

Conversion,

## 2.67 HxImageAsVec3Short

### Synopsis

```
HxImageRep HxImageAsVec3Short (HxImageRep img)
```

### Input

HxImageRep *img*    The image you want to convert

### Return value

HxImageRep    The input image converted to an image with short vector pixels with dimensionality 3.

### Description

The function `HxImageAsVec3Short` converts the input image to an image with vector short pixel values, with dimensionality 3. The input image should be either a scalar image or a vector image with pixel dimensionality 3. If the input image is a scalar image the pixel value of that pixel is converted to short and copied to both vector components of the corresponding pixel in the output image.

### Remarks

*Valid input images*    All scalar images and images with pixel dimensionality 3 are valid input images.

### See also

[HxImageAsShort](#) (p. 84), [HxImageAsVec2Short](#) (p. 89), [HxImageAsVec3Byte](#) (p. 90), [HxImageAsVec3Int](#) (p. 93), [HxImageAsVec3Float](#) (p. 92), [HxImageAsVec3Double](#) (p. 91),

### Keywords

Conversion,

## 2.68 HxImageMaxSize

### Synopsis

```
int HxImageMaxSize (HxImageRep img)
```

### Input

`HxImageRep img` The image you want to determine the maximum size of, i.e. the size in the dimension that is the largest.

### Return value

`int` The maximum size of the image.

### Description

Suppose we have a three dimensional image with sizes  $256 \times 512 \times 128$ . For this image, the function `HxImageMaxSize` returns 512, i.e. the size in the dimension that is the largest.

### Remarks

*No possible errors* This function cannot result in any errors.

### See also

`HxImageMinSize` (p. 96),

### Keywords

Inquiry,

## 2.69 HxImageMinSize

### Synopsis

```
int HxImageMinSize (HxImageRep img)
```

### Input

`HxImageRep img` The image you want to determine the minimum size of, i.e. the size in the dimension that is the largest.

### Return value

`int` The minimum size of the image.

### Description

Suppose we have a three dimensional image with sizes  $256 \times 512 \times 128$ . For this image, the function `HxImageMinSize` returns 128, i.e. the size in the dimension that is the smallest.

### Remarks

*No possible errors*

### See also

`HxImageMaxSize` (p. 95),

### Keywords

Inquiry,

## 2.70 HxInf

### Synopsis

```
HxImageRep HxInf(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The pixelwise infimum of the input images

### Description

The function HxInf takes the infimum of two images on a pixel-by-pixel basis. The infimum of two vector values of equal dimensionality is defined as the component wise infimum resulting in a vector with the same dimensionality. The infimum of pixel types with different dimensionalities cannot be taken.

### Remarks

**Admissable image types**    The two input images should have the same dimensionality, the same pixel dimensionality and the same size.

### Examples

Taking the pixelwise infimum of two images

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxInfExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxInf(im1, im2);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxInfExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.41: First input image for the HxInf example.

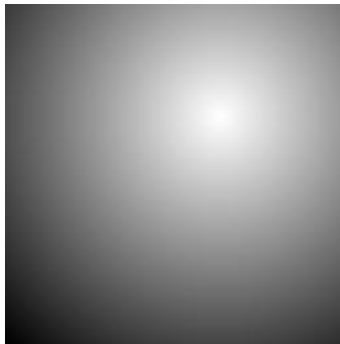


Figure 2.42: Second input image for the HxInf example.



Figure 2.43: Output image of the HxInf example.

In this example we take the infimum of two (scalar) images.

**See also**

**HxMax** (p. 124), **HxMin** (p. 126), **HxSup** (p. 169),

**Keywords**

Binary, Arithmetic,

## 2.71 HxInfVal

### Synopsis

```
HxImageRep HxInfVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The input value

### Return value

HxImageRep    The infimum of the input image and the input value.

### Description

The function HxInfVal takes the infimum of HxValue 'val' and every pixel value in the input image (for every pixel again) and stores the result in the corresponding pixel of the output image. The dimensionality of 'val' and the pixeldimensionality of the input image must be the same. If 'val' is a vector and the pixels of the input image are vectors, the infimum is taken component wise.

### Remarks

*Admissable image and value types*    The dimensionality of 'val' and the pixeldimensionality of the input image must be the same.

### See also

**HxInf** (p. 97), **HxSupVal** (p. 171), **HxSup** (p. 169),

### Keywords

Binary value, Arithmetic,

## 2.72 HxInverseProjectRange

### Synopsis

```
HxImageRep HxInverseProjectRange (HxImageRep img, int dimension, Hx-  
ImageRep arg)
```

### Input

HxImageRep *img*    The first input image, which is scalar.

int *dimension*    The (pixel) dimension of the second input image at which the first input image is copied.

HxImageRep *arg*    The second input image.

### Return value

HxImageRep    The result of copying the input image in the given dimension of the second input image.

### Description

The function `HxInverseProjectRange` copies the values of the first scalar input image in the second input image. If the second input image is a vector image, the value is put in the component of the vector as given by the dimension. The images should have the same size and dimensionality.

### Remarks

**Valid input images**    The first input image should be a scalar image, the second input image should be an image with the same dimensionality and size as the first image.

**Valid values for 'dimension'**    The parameter 'dimension' should be at least 1 and at most equal to the pixel dimensionality of the second input image.

### See also

[anchor\\_HxMakeFrom2Images](#), [anchor\\_HxMakeFrom3Images](#), [HxProjectRange](#) (p. 151),

### Keywords

Binary, Projection,

## 2.73 HxLeftShift

### Synopsis

```
HxImageRep HxLeftShift (HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The left shifted version of the first input image.

### Description

The function HxLeftShift shifts the bits of the pixel values of the first input image of the number of places to the left as indicated by the corresponding value in the second input image. For vector images, the each component of a vector is shifted the number of places as indicated by the same component of the corresponding vector value. Only integer (HxShort, HxByte, HxInt ) type images can be used as input images.

### Remarks

*Valid input images*    The input images should have the same dimensionality, the same size and the same pixel dimensionality. The numerical type of the pixels, or its components, should be HxShort, HxByte, HxInt .

### See also

**HxRightShift** (p. 153), **HxAnd** (p. 11), **HxLeftShiftVal** (p. 102),

### Keywords

Binary, Logical,

## 2.74 HxLeftShiftVal

### Synopsis

```
HxImageRep HxLeftShiftVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image you want to left shift the pixel values of.

HxValue *val*    The input value.

### Return value

HxImageRep    The left shifted version of the input image.

### Description

The function HxLeftShiftVal shifts the bits of the pixel values of the input image of the number of places to the left as indicated by the input value. For vector images and vector values of the same dimensionality, each component of a vector is shifted the number of places as indicated by the corresponding component of the value vector. Only integer (HxShort, HxByte, HxInt ) type images and values can be used.

### Remarks

***Valid input images and input values***    The dimensionality of the input value should be equal to the pixel dimensionality of the input image. The image pixels and input value (or their components) should be HxShort, HxByte, HxInt .

### See also

**HxRightShiftVal** (p. 154), **HxAndVal** (p. 13), **HxLeftShift** (p. 101),

### Keywords

Binary value, Logical,

## 2.75 HxLessEqual

### Synopsis

```
HxImageRep HxLessEqual(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The result of comparing whether the first input image is less than or equal to the second input image, pixelwise. When a pixel from the first input image is less than or equal to the corresponding pixel of the second input image, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function `HxLessEqual` compares the corresponding pixels from the input images, which must be scalar. If the value of the pixel of the first input image is less than or equal to the value of the pixel of the second input image, the corresponding pixel in the result image will be 1, otherwise it will be 0.

The dimensions of the input images must equal, but the pixel types can be anything, as long as they are scalar. The resulting image is an image with the same dimensions as the input images, with integer pixel values.

### Remarks

*Valid input images*    This function is only properly defined for scalar input images with equal sizes and equal dimensionality.

### See also

`HxLessThan` (p. 105), `HxGreaterThan` (p. 76), `HxGreaterEqual` (p. 74), `HxEqual` (p. 58), `HxLessEqualVal` (p. 104),

### Keywords

Comparison, Binary,

## 2.76 HxLessEqualVal

### Synopsis

```
HxImageRep HxLessEqualVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The value you want the compare the image values with.

### Return value

HxImageRep    When a pixel from the first input image is less or equal to the given value input image, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function HxLessEqualVal compares the pixel values of the input image with the value 'val'. When the value of the pixel is less or equal than the value, the corresponding pixel in the output image has value 1, otherwise the value of the pixel is 0. This operation is only defined for scalar images (and values).

### Remarks

*Valid input images*    Only scalar images are valid. The image and value do not have to be of the same type.

*Valid <val> types*    Only scalar values are valid.

### See also

**HxLessThanVal** (p. 108), **HxGreaterThanVal** (p. 79), **HxGreaterEqualVal** (p. 75), **HxEqualVal** (p. 59), **HxLessEqual** (p. 103),

### Keywords

Comparison,

## 2.77 HxLessThan

### Synopsis

```
HxImageRep HxLessThan(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The result of comparing whether the first input image is less than the second input input, pixelwise. When a pixel from the first input image is less than the corresponding pixel of the second input image, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function HxLessThan compares the corresponding pixels from the input images, which must be scalar. If the value of the pixel of the first input image is less than the value of the pixel of the second input pixel, the corresponding pixel in the result image will be 1, otherwise it will be 0.

The dimensions of the input images must equal, but the pixel types can be anything, as long as they are scalar. The resulting images is an image with the same dimensions as the input images, with integer pixel values.

### Remarks

*Valid input images*    This function is only properly defined for scalar input images with equal sizes and equal dimensionality.

### Examples

Example of using HxLessThan

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxLessThanExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxLessThan(im1, im2);
    im1 = HxContrastStretch(im1, 255);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxLessThanExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.44: First input image for the HxLessThan example.

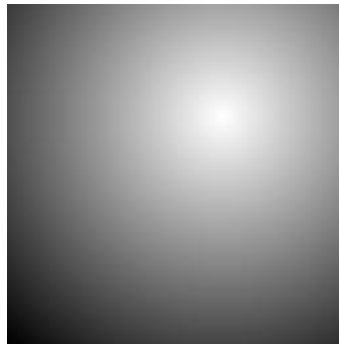


Figure 2.45: Second input image for the HxLessThan example.

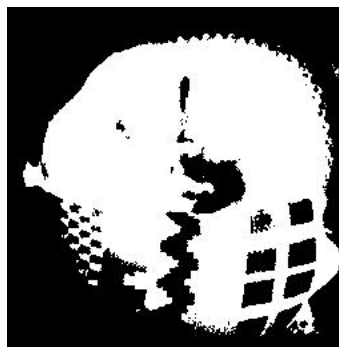


Figure 2.46: Output image of the HxLessThan example.

In the example code we use `HxLessThan` to compare the two input images. The contrast stretch is used for visualization purposes.

**See also**

`HxLessEqual` (p. 103), `HxGreaterThan` (p. 76), `HxGreaterEqual` (p. 74), `HxEqual` (p. 58), `HxLessThanVal` (p. 108),

**Keywords**

Comparison, Binary,

## 2.78 HxLessThanVal

### Synopsis

```
HxImageRep HxLessThanVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The value you want the compare the image values with.

### Return value

HxImageRep    When a pixel from the first input image is less than the given value input image, the value of the corresponding pixel in the result is 1, otherwise it is 0.

### Description

The function `HxLessThanVal` compares the pixel values of the input image with the value 'val'. When the value of the pixel is less than the value, the corresponding pixel in the output image has value 1, otherwise the value of the pixel is 0. This operation is only defined for scalar images (and values).

### Remarks

*Valid input images*    Only scalar images are valid. The image and value do not have to be of the same type.

*Valid value types*    Only scalar values are valid.

### See also

`HxLessEqualVal` (p. 104), `HxGreaterThanVal` (p. 79), `HxGreaterEqualVal` (p. 75), `HxEqualVal` (p. 59), `HxLessThan` (p. 105),

### Keywords

Binary value, Comparison,

## 2.79 HxLog10

### Synopsis

```
HxImageRep HxLog10 (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the  $\log_{10}$  of.

### Return value

HxImageRep    The  $\log_{10}$  of the input image.

### Description

The function HxLog10 takes the logarithm, with base 10, of each pixel value. The logarithm of a vector is defined by taking the logarithm of the components of the vector, i.e.  $\log_{10}(a_1, a_2, \dots, a_n) = (\log_{10} |a_1|, \log_{10} |a_2|, \dots, \log_{10} |a_n|)$ .

### Remarks

**Valid input images**    Only image with values greater than zero are valid. If one of the values is less or equal than zero (or one of the components of a vector is less or equal than zero), the input image is not valid.

### Examples

Taking the logarithm of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxLog10Example1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxLog10(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep img = HxMakeFromFile(argv[1]);
    img = HxLog10Example1(img);
    HxWriteFile(img, argv[2]);

    return 0;
}
```



Figure 2.47: Input image for the HxLog10 example.



Figure 2.48: Output image of the HxLog10 example.

In this example we first convert the input image to a double valued image. We take the logarithm (with base 10), after which we stretch the image to let it contain values ranges from 0.0 to 255.0 for visualization purposes.

**See also**

**HxLog** (p. 111), **HxExp** (p. 60), **HxPow** (p. 149),

**Keywords**

Unary, Arithmetic,

## 2.80 HxLog

### Synopsis

```
HxImageRep HxLog (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the log of.

### Return value

HxImageRep    The log of the input image.

### Description

The function HxLog takes the logarithm of each pixel value. The logarithm of a vectro is defined by taking the logarithm of the components of the vector, i.e.  $\log(a_1, a_2, \dots, a_n) = (\log |a_1|, \log |a_2|, \dots, \log |a_n|)$ .

### Remarks

*Valid input images*    Only image with values greater than zero are valid. If one of the values is less or equal than zero (or one of the components of a vector is less or equal than zero), the input image is not valid.

### Examples

Taking the logarithm of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxLogExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxLog(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep img = HxMakeFromFile(argv[1]);
    img = HxLogExample1(img);
    HxWriteFile(img, argv[2]);

    return 0;
}
```



Figure 2.49: Input image for the HxLog example.



Figure 2.50: Output image of the HxLog example.

In this example we first convert the input image to a double valued image. We take the logarithm, after which we stretch the image to let it contain values ranging from 0.0 to 255.0 for visualization purposes.

**See also**

**HxLog10** (p. 109), **HxExp** (p. 60),

**Keywords**

Unary, Arithmetic,

## 2.81 HxMakeFromByteData

### Synopsis

```
HxImageRep HxMakeFromByteData (int pixelDimensionality, int
dimensions, HxSizes sizes, HxByte *data)
```

### Input

`int pixelDimensionality` The dimensionality of the pixels of the image you want to create.

`int dimensions` The number of dimensions of the image you want to create.

`HxSizes sizes` The size of every dimension of the image you want to create.

`HxByte *data` The array of HxByte that you want to create the image from.

### Return value

`HxImageRep` The image created from the array 'data'.

### Description

This function creates a byte image from an array of HxBytes. The size of the image is given by 'sizes', the dimensionality of the pixels by `pixelDimensionality` and the dimensionality of the image is given by 'dimensions'. The array 'data' should contain at least a number of bytes equal to the number of pixels in the image (determined by the sizes and the dimensionality) times the pixel dimensionality.

### Remarks

**Valid input parameters** `pixelDimensionality` should be 1, 2 or 3. 'dimensions' should be 2 or 3. 'sizes' should have a dimensionality equal to 'dimensions'.

**Valid arrays** The array should contain at least a number of bytes equal to the number of pixels in the (proposed) image times the proposed pixel dimensionality.

### See also

[HxMakeFromDoubleData](#) (p. 114), [HxMakeFromFloatData](#) (p. 117), [HxExportByteData](#) (p. 61),

### Keywords

Generation,

## 2.82 HxMakeFromDoubleData

### Synopsis

```
HxImageRep HxMakeFromDoubleData (int pixelDimensionality, int
dimensions, HxSizes sizes, double *data)
```

### Input

`int pixelDimensionality` The dimensionality of the pixels of the image you want to create.

`int dimensions` The number of dimensions of the image you want to create.

`HxSizes sizes` The size of every dimension of the image you want to create.

`double *data` The array of doubles that you want to create the image from.

### Return value

`HxImageRep` The image created from the array 'data'.

### Description

This function creates a double image from an array of doubles. The size of the image is given by 'sizes', the dimensionality of the pixels by `pixelDimensionality` and the dimensionality of the image is given by 'dimensions'. The array 'data' should contain at least a number of doubles equal to the number of pixels in the image (determined by the sizes and the dimensionality) times the pixel dimensionality.

### Remarks

*Valid input parameters* `pixelDimensionality` should be 1, 2 or 3. 'dimensions' should be 2 or 3. 'sizes' should have a dimensionality equal to 'dimensions'.

### See also

**HxMakeFromByteData** (p. 113), **HxMakeFromFloatData** (p. 117), **HxExportDoubleData** (p. 62),

### Keywords

Generation,

## 2.83 HxMakeFromFile

### Synopsis

```
HxImageRep HxmakeFromFile(HxString fileName)
```

### Return value

HxImageRep The image you have loaded from file.

### Description

The function HxMakeFromFile reads an image from file. The supported image file formats are: tif, jpg, ppm . The format of the filename itself, accepts slashes as well as backslashes.

### Remarks

*Valid file formats* This function supports the following file formats: tif, jpg, ppm /

*Valid filenames* Of course the file to which the filename refers must exist. The filename format accepts slashes as well as backslashes.

### See also

**HxWriteFile** (p. 184), **HxMakeFromFileSI** (p. 116),

### Keywords

File I/O, Import,

## 2.84 HxMakeFromFileSI

### Synopsis

```
HxImageRep HxmakeFromFileSI (HxString fileName)
```

### Return value

HxImageRep The image you have loaded from file.

### Description

The function HxMakeFromFileSi reads a ScilImage image from file and returns it as an Horus image.

### Remarks

*Valid file formats* This function only reads ScilImage image files.

*Valid filenames* Of course the file to which the filename refers must exist. The filename format accepts slashes as well as backslashes.

### See also

**HxMakeFromFile** (p. 115), **HxWriteFile** (p. 184),

### Keywords

File I/O, Import,

## 2.85 HxMakeFromFloatData

### Synopsis

```
HxImageRep HxMakeFromFloatData (int pixelDimensionality, int
dimensions, HxSizes sizes, float *data)
```

### Input

`int pixelDimensionality` The dimensionality of the pixels of the image you want to create.

`int dimensions` The number of dimensions of the image you want to create.

`HxSizes sizes` The size of every dimension of the image you want to create.

`float *data` The array of floats that you want to create the image from.

### Return value

`HxImageRep` The image created from the array 'data'.

### Description

This function creates a float image from an array of floats. The size of the image is given by 'sizes', the dimensionality of the pixels by `pixelDimensionality` and the dimensionality of the image is given by 'dimensions'. The array 'data' should contain at least a number of floats equal to the number of pixels in the image (determined by the sizes and the dimensionality) times the pixel dimensionality.

### Remarks

**Valid input parameters** `pixelDimensionality` should be 1, 2 or 3. 'dimensions' should be 2 or 3. 'sizes' should have a dimensionality equal to 'dimensions'.

**Valid arrays** The array should contain at least a number of floats equal to the number of pixels in the (proposed) image times the proposed pixel dimensionality.

### See also

[HxMakeFromDoubleData](#) (p. 114), [HxMakeFromByteData](#) (p. 113), [HxExportFloatData](#) (p. 63),

### Keywords

Generation,

## 2.86 HxMakeFromIntData

### Synopsis

```
HxImageRep HxMakeFromIntData (int pixelDimensionality, int dimensions,  
HxSizes sizes, int *data)
```

### Input

`int pixelDimensionality` The dimensionality of the pixels of the image you want to create.

`int dimensions` The number of dimensions of the image you want to create.

`HxSizes sizes` The size of every dimension of the image you want to create.

`int *data` The array of integers that you want to create the image from.

### Return value

`HxImageRep` The image created from the array 'data'.

### Description

This function creates an integer image from an array of integers. The size of the image is given by 'sizes', the dimensionality of the pixels by `pixelDimensionality` and the dimensionality of the image is given by 'dimensions'. The array 'data' should contain at least a number of integers equal to the number of pixels in the image (determined by the sizes and the dimensionality) times the pixel dimensionality.

### Remarks

**Valid input parameters** `pixelDimensionality` should be 1, 2 or 3. 'dimensions' should be 2 or 3. 'sizes' should have a dimensionality equal to 'dimensions'.

**Valid arrays** The array should contain at least a number of integers equal to the number of pixels in the (proposed) image times the proposed pixel dimensionality.

### See also

[HxMakeFromByteData](#) (p. 113), [HxMakeFromDoubleData](#) (p. 114), [HxExportIntData](#) (p. 64),

### Keywords

Generation,

## 2.87 HxMakeFromMatlab

### Synopsis

```
HxImageRep HxMakeFromMatlab (const HxImageSignature & signature, Hx-  
Sizes sizes, double *pixels)
```

### Input

`const HxImageSignature & signature` The signature of the image you want to construct from the matlab data.

`HxSizes sizes` The size of every dimension of the image you want to create.

`double *pixels` The array of doubles from which you create your image.

### Return value

`HxImageRep` The image created from the array 'pixels'.

### Description

This function creates an image with the given signature and sizes from an array of doubles, i.e. the way Matlab stores its images. The type of the image is given by the signature, see the section on image **Image signatures** (p. 2). The size of the image is given by 'sizes'. The array 'pixels' should contain at least a number of doubles equal to the number of pixels in the image (determined by the sizes and the dimensionality from the signature) times the pixel dimensionality (from the signature).

### Remarks

**Valid input parameters** The image signature should be one of the existing image signatures (see the section on image **Image signatures** (p. 2)). Sizes should have a dimensionality equal to the dimension as mentioned in the signature. The size in every dimension should be at least 1.

**Valid arrays** The array should contain at least a number of doubles equal to the number of pixels in the (proposed) image times the proposed pixel dimensionality.

### See also

**HxMakeFromDoubleData** (p. 114), **HxExportMatlabPixels** (p. 65),

### Keywords

Generation, Conversion,

## 2.88 HxMakeFromShortData

### Synopsis

```
HxImageRep HxMakeFromShortData (int pixelDimensionality, int
dimensions, HxSizes sizes, short *data)
```

### Input

`int pixelDimensionality` The dimensionality of the pixels of the image you want to create.

`int dimensions` The number of dimensions of the image you want to create.

`HxSizes sizes` The size of every dimension of the image you want to create.

`short *data` The array of shorts that you want to create the image from.

### Return value

`HxImageRep` The image created from the array 'data'.

### Description

This function creates a short image from an array of `HxBytes`. The size of the image is given by 'sizes', the dimensionality of the pixels by `pixelDimensionality` and the dimensionality of the image is given by 'dimensions'. The array 'data' should contain at least a number of shorts equal to the number of pixels in the image (determined by the sizes and the dimensionality) times the pixel dimensionality.

### Remarks

*Valid input parameters* `pixelDimensionality` should be 1, 2 or 3. 'dimensions' should be 2 or 3. 'sizes' should have a dimensionality equal to 'dimensions'.

*Valid arrays* The array should contain at least a number of shorts equal to the number of pixels in the (proposed) image times the proposed pixel dimensionality.

### See also

[HxMakeFromByteData](#) (p. 113), [HxMakeFromIntData](#) (p. 118), [HxExportShortData](#) (p. 66),

### Keywords

Generation,

## 2.89 HxMakeFromSI

### Synopsis

HxImageRep HxMakeFromSI (IMAGE\* p)

### Input

IMAGE\* p The pointer to the ScilImage image that you want to convert to a Horus image.

### Return value

HxImageRep The input image in Horus format.

### Description

The function HxMakeFromSI converts a ScilImage image to Horus image format.

### Remarks

*Valid pointers p* The input pointer should refer to a valid ScilImage image.

### See also

**HxExportSI** (p. 67), <reference>HxMakeFromFileSI</reference>,

### Keywords

Conversion,

## 2.90 HxMakeFromSignature

### Synopsis

```
HxImageRep HxMakeFromSignature (const HxImageSignature & signature,  
HxSizes sizes)
```

### Input

`const HxImageSignature & signature` The signature of the image you want to construct.

`HxSizes sizes` The size of every dimension of the image you want to create.

### Return value

`HxImageRep` The created image.

### Description

This function creates an image with the given signature and sizes. The image is not initialized yet! The type of the image is given by the signature, see the section on image **Image signatures** (p. 2). The size of the image is given by 'sizes'. The dimensionality of 'sizes' should be equal to the image dimensionality as given in the image signature.

### Remarks

*Valid input parameters* The image signature should be one of the existing image signatures (see the section on image **Image signatures** (p. 2)). Sizes should have a dimensionality equal to the dimension as mentioned in the signature. The size in every dimension should be at least 1.

### See also

**HxMakeFromDoubleData** (p. 114), **HxMakeFromValue** (p. 123),

### Keywords

Generation,

## 2.91 HxMakeFromValue

### Synopsis

```
HxImageRep HxMakeFromValue (const HxImageSignature & signature, Hx-  
Sizes sizes, HxValue value)
```

### Input

`const HxImageSignature & signature` The signature of the image you want to construct.

`HxSizes sizes` The size of every dimension of the image you want to create.

`HxValue value` The value you want to assign to every pixel in the image.

### Return value

`HxImageRep` The created image.

### Description

This function creates an image with the given signature and sizes and gives every pixel the value as given by `HxValue value`. The type of the image is given by the signature, see the section on image **Image signatures** (p. 2). The size of the image is given by 'sizes'. The dimensionality of `HxValue value` should be equal to the pixel dimensionality as given in the image signature.

### Remarks

**Valid input parameters** The image signature should be one of the existing image signatures (see the section on image **Image signatures** (p. 2)). Sizes should have a dimensionality equal to the dimension as mentioned in the signature. The size in every dimension should be at least 1. The dimensionality of the `HxValue value` should be equal to the pixel dimensionality as given in the image signature.

### See also

**HxMakeFromDoubleData** (p. 114),

### Keywords

Generation,

## 2.92 HxMax

### Synopsis

```
HxImageRep HxMax(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The pixelwise maximum of the input images

### Description

The function `HxMax` takes the maximum of two images on a pixel-by-pixel basis, i.e. the value of a pixel in the output image is the maximum of the corresponding pixels of the input images. This function is only defined for scalar images that have the same dimensionality and size.

### Remarks

*Admissable image types*    The two input images should be scalar and should have the same dimensionality and size.

### See also

**HxSup** (p. 169), **HxInf** (p. 97), **HxMin** (p. 126),

### Keywords

Binary, Arithmetic,

## 2.93 HxMaxVal

### Synopsis

```
HxImageRep HxMaxVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The input value

### Return value

HxImageRep    The pixelwise maximum of the input image and the input value.

### Description

The function `HxMaxVal` takes the maximum of the input image and the input value on a pixel-by-pixel basis, i.e. the value of a pixel in the output image is the maximum of the corresponding pixel in the input images and the input value. This function is only defined for scalar images and values.

### Remarks

*Valid image types and value types*    The input image and the input value should be scalars.

### See also

`HxMinVal` (p. 127), `HxMax` (p. 124),

### Keywords

Binary value, Arithmetic,

## 2.94 HxMin

### Synopsis

```
HxImageRep HxMin(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The pixelwise minimum of the input images

### Description

The function HxMin takes the minimum of two images on a pixel-by-pixel basis, i.e. the value of a pixel in the output image is the minimum of the corresponding pixels of the input images. This function is only defined for scalar images that have the same dimensionality and size.

### Remarks

*Admissable image types*    The two input images should be scalar and should have the same dimensionality and size.

### See also

**HxInf** (p. 97), **HxSup** (p. 169), **HxMax** (p. 124),

### Keywords

Binary, Arithmetic,

## 2.95 HxMinVal

### Synopsis

```
HxImageRep HxMinVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The input value

### Return value

HxImageRep    The pixelwise minimum of the input image and the input value.

### Description

The function `HxMinVal` takes the minimum of the input image and the input value on a pixel-by-pixel basis, i.e. the value of a pixel in the output image is the minimum of the corresponding pixel in the input images and the input value. This function is only defined for scalar images and values.

### Remarks

*Valid image types and value types*    The input image and the input value should be scalars.

### See also

`HxMaxVal` (p. 125), `HxMin` (p. 126),

### Keywords

Binary value, Arithmetic,

## 2.96 HxMod

### Synopsis

```
HxImageRep HxMod(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The first input image modulo the second input image, pixel wise.

### Description

The function HxMod takes the values of the first input image modulo the corresponding values of the second input image. For vector images, the modulo is taken component wise. Only integer (HxShort, HxByte, HxInt ) type images can be used as input images.

### Remarks

*Valid input images*    The input images should have the same dimensionality, the same size and the same pixel dimensionality. The numerical type of the pixels, or its components, should be HxShort, HxByte, HxInt .

### See also

**HxAnd** (p. 11), **HxModVal** (p. 129),

### Keywords

Binary, Logical,

## 2.97 HxModVal

### Synopsis

```
HxImageRep HxModVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep `img` The input image.

HxValue `val` The input value.

### Return value

HxImageRep The input image modulo the input value, pixel wise.

### Description

The function `HxModVal` takes the values of the first input image modulo the input value. If the pixel values and the input value are vectors of the same dimensionality, the modulo is taken component wise. Only integer (`HxShort`, `HxByte`, `HxInt`) type pixel values and input values can be used.

### Remarks

*Valid input images and input values* The input value should have the same dimensionality as the input image pixels. The numerical type of the pixels and input value, or their components, should be `HxShort`, `HxByte`, `HxInt`.

### See also

`HxAndVal` (p. 13), `HxMod` (p. 128),

### Keywords

Binary value, Logical,

## 2.98 HxMul

### Synopsis

```
HxImageRep HxMul(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The (Hadamard) product of the input images

### Description

The function HxMul multiplies two images on a pixel-by-pixel basis. Vector pixel types of the same dimension are multiplied component wise. Pixel types of different dimensions can be multiplied, as long as the pixel dimensionality of one of the input images is 1. The image with pixel dimensionality 1 is then treated as an image with the same pixel dimensionality as the other image (with equal values per pixel).

### Remarks

**Valid input images**    The two input images should have the same dimensionality, the same pixel dimensionality (or the pixel dimensionality of one of the pixels should be 1) and the same size.

**Overflow error**    The function HxMul can result in an overflow error.

### Examples

Multiplying two images

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxMulExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxImageAsDouble(im1);
    im2 = HxImageAsDouble(im2);
    im1 = HxMul(im1, im2);
    im1 = HxContrastStretch(im1, 255.0);
    im1 = HxImageAsByte(im1);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxMulExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.51: First input image for the HxMul example.

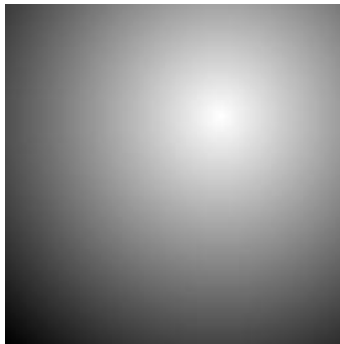


Figure 2.52: First input image for the HxMul example.



Figure 2.53: Output image of the HxMul example.

In this example we multiply two (scalar) images. After multiplying the images, we stretch the result for visualization purposes.

**See also**

**HxDiv** (p. 53), **HxAdd** (p. 7),

**Keywords**

Binary, Arithmetic,

## 2.99 HxMulVal

### Synopsis

```
HxImageRep HxMulVal(HxImageRep img, HxValue val)
```

### Input

`HxImageRep img` The input image you want to multiply with a value.

`HxValue val` The input value.

### Return value

`HxImageRep` The input image multiplied with the input value, pixel wise.

### Description

The function `HxMulVal` multiplies the values of the input image with the input value on a pixel-by-pixel basis. If the pixel type of the input image and the type of the value are both vectors with the same dimensionality, they are treated component wise. If the input image has vector type pixels and the dimensionality of the input value is 1, every component of a pixel vector is multiplied with the input value.

### Remarks

***Valid input images and values*** The dimensionality of the input value should be 1 or equal to the pixel dimensionality of the image.

***Overflow error*** The function `HxMulVal` can result in an overflow error.

### See also

**`HxPowVal`** (p. 150), **`HxDivVal`** (p. 55), **`HxMul`** (p. 130),

### Keywords

Binary value, Arithmetic,

## 2.100 HxNegate

### Synopsis

```
HxImageRep HxNegate (HxImageRep img)
```

### Input

HxImageRep *img* The image you want to take the negation of.

### Return value

HxImageRep The negation of the input image.

### Description

The function HxNegate negates all pixel values. The negation of a vector is taken by negating every component separately. The function HxNegate does not work for byte images.

### Remarks

*Valid input images* All images types are valid as input for this function, except byte images.

### See also

**HxComplement** (p. 26),

### Keywords

Unary, Arithmetic,

## 2.101 HxNorm1

### Synopsis

HxImageRep HxNorm1 (HxImageRep img)

### Return value

HxImageRep The result of taking the city block distance norm of the input image.

### Description

The function HxUnarySum takes the city block distance norm of every pixel value in the input image and stores it in the corresponding pixel of the output image. The city block distance norm of a scalar equals the absolute value of the scalar itself, while the city block distance norm of a vector equals the sum of the absolute value of the components of the vector. If the input is a vector image containing HxBytes, the output image is of the integer type.

### Remarks

*Possible errors* HxNorm1 can result in an overflow error.

### Keywords

Unary, Norm, Arithmetic,

## 2.102 HxNorm2

### Synopsis

HxImageRep HxNorm2 (HxImageRep img)

### Return value

HxImageRep The result of taking the Euclidian norm of the input image.

### Description

The function HxNorm2 takes the Euclidian norm of every pixel value in the input image and stores it in the corresponding pixel of the output image. The Euclidian norm of a scalar equals the absolute value of the scalar itself, while the Euclidian norm of a vector equals the square root of the sum of the squared values of the components of the vector. If the input is a vector image containing HxBytes, the output image is of the integer type.

### Remarks

*Possible errors* HxNorm2 can result in an overflow error.

### Keywords

Unary, Norm, Arithmetic,

## 2.103 HxNorm2Sqr

### Synopsis

HxImageRep HxNorm2Sqr (HxImageRep img)

### Return value

HxImageRep The result of taking the squared Euclidian norm of the input image.

### Description

The function HxNorm2Sqr takes the squared Euclidian norm of every pixel value in the input image and stores it in the corresponding pixel of the output image. The squared Euclidian norm of a scalar equals square of the scalar itself, while the squared Euclidian norm of a vector equals the sum of the squared values of the components of the vector. If the input is a vector image containing HxBytes, the output image is of the integer type.

### Remarks

*Possible errors* HxNorm2Sqr can result in an overflow error.

### Keywords

Unary, Norm, Arithmetic,

## 2.104 HxNormInf

### Synopsis

HxImageRep HxNormInf (HxImageRep img)

### Return value

HxImageRep The result of taking the chessboard norm of the input image.

### Description

The function HxNormInf takes the city chessboard norm of every pixel value in the input image and stores it in the corresponding pixel of the output image. The chessboard norm of a scalar equals the absolute value of the scalar itself, while the chessboard norm of a vector equals infimum of the absolute value of the components of the vector. If the input is a vector image containing HxBytes, the output image is of the integer type.

### Remarks

*No possible errors* HxNormInf can not result in any error.

### Keywords

Unary, Norm, Arithmetic,

## 2.105 HxNotEqual

### Synopsis

```
HxImageRep HxNotEqual(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The result of comparing whether the first input image is (not) equal to the second input input, pixelwise. When a pixel from the first input image is equal to the corresponding pixel of the second input image, the value of the corresponding pixel in the result is 0, otherwise it is 1.

### Description

The function HxEqual compares the corresponding pixels from the input images. If the value of the pixel of the first input image is equal to the value of the pixel of the second input pixel, the corresponding pixel in the result image is 0, otherwise it is 1. The function is equivalently defined for vector images.

The dimensionality, sizes and pixel dimensionality of the input images must equal. The resulting images is an image with the same dimensions as the input images, with integer pixel values.

### Remarks

*Valid input images*    This function is only properly defined images with equal sizes, equal dimensionality and equal pixeldimensionality.

### See also

[HxEqual](#) (p. 58), [HxEqualVal](#) (p. 59), [HxNotEqualVal](#) (p. 139),

### Keywords

Comparison, Binary,

## 2.106 HxNotEqualVal

### Synopsis

```
HxImageRep HxNotEqualVal(HxImageRep im1, HxValue val)
```

### Input

HxImageRep im1   The first input image.

HxValue val    The input value.

### Return value

HxImageRep    The result of comparing whether the input image is (not) equal to the input value, pixel-wise. When a pixel from the input image is equal to input value, the value of the corresponding pixel in the result is 0, otherwise it is 1.

### Description

The function HxEqualVal compares the pixels from the input image to the HxValue 'val'. If the value of the pixel of the input image is equal to 'val', the corresponding pixel in the result image is 0, otherwise it is 1. The function is equivalently defined for vector images.

The pixel dimensionality of the input image must be equal to the dimensionality of 'val'. The resulting images is an image with the same dimensions as the input image, with integer pixel values.

### Remarks

*Valid input image and value 'val'*   The pixel dimensionality of the input image must be equal to the dimensionality of 'val'.

### See also

**HxNotEqual** (p. 138), **HxEqualVal** (p. 59), **HxEqual** (p. 58),

### Keywords

Comparison, Binary value,

## 2.107 HxOr

### Synopsis

```
HxImageRep HxOr(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The logical ‘or’ of the input images.

### Description

The function HxOr takes the logical ‘or’ of corresponding pixels in the two images. The ‘or’ of two vectors is taken component wise.

### Remarks

**Valid input images**    The input images should have the same dimensionality, the same size, the same pixel dimensionality and the same pixel type. The numerical values of the pixels, or its components, should be HxShort, HxByte, HxInt .

### Examples

Taking the logical ‘or’

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxOrExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxImageAsDouble(im1);
    im2 = HxImageAsDouble(im2);
    im1 = HxContrastStretch(im1, 1.0);
    im2 = HxContrastStretch(im2, 1.0);
    im1 = HxImageAsByte(im1);
    im2 = HxImageAsByte(im2);
    im1 = HxOr(im1, im2);
    im1 = HxContrastStretch(im1, 255);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxOrExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.54: First input image for the HxOr example.

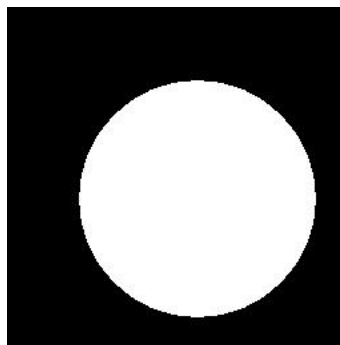


Figure 2.55: Second input image for the HxOr example.

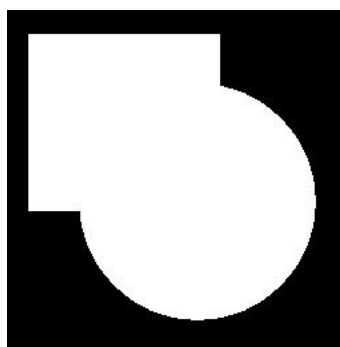


Figure 2.56: Output image of the HxOr example.

In this example we take the logical 'or' of the two input images.

**See also**

**HxAnd** (p. 11), **HxXor** (p. 185), **HxOrVal** (p. 142),

**Keywords**

Binary, Logical,

## 2.108 HxOrVal

### Synopsis

```
HxImageRep HxOrVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image.

HxValue *val*    The input value.

### Return value

HxImageRep    The logical 'or' of the input image and the input value.

### Description

The function HxOrVal takes the logical 'or' of the pixels from the input image with the input value on a pixel-by-pixel basis. The 'or' of two vectors is taken component wise.

### Remarks

*Valid input images and input values*    The input value should have the same dimensionality as the pixel dimensionality of the input image. The pixel type and the type of the input value should be, HxShort, HxByte, HxInt . The types should be equal.

### See also

[HxAndVal](#) (p. 13), [HxXorVal](#) (p. 187), [HxOr](#) (p. 140),

### Keywords

Binary value, Logical,

## 2.109 HxPixInf

### Synopsis

```
HxValue HxPixInf (HxImageRep img)
```

### Input

HxImageRep *img* The image you want to determine the infimum of.

### Return value

HxValue The infimum of the pixel values of the input image.

### Description

The function `HxPixInf` determines the pixel-wise minimum of the input image. For scalar image, this results in the minimum over all pixel values. For vector images, this function returns a vector with components equal to the minimum value for each component of the pixel value vectors.

### Remarks

*No possible errors* This function can not result in any errors.

### See also

[HxPixSup](#) (p. 148), [HxPixMax](#) (p. 144), [HxPixMin](#) (p. 145),

### Keywords

Reduction,

## 2.110 HxPixMax

### Synopsis

```
HxValue HxPixMax (HxImageRep img)
```

### Input

HxImageRep img    The image you want to determine the maximum of.

### Return value

HxValue    The maximum of the pixel values of the input image.

### Description

The function HxPixMax determines the pixel-wise maximum of the input image.

### Remarks

*Valid input images*    This function accepts all scalar image types.

### See also

[HxPixInf](#) (p. 143), [HxPixSup](#) (p. 148), [HxPixMin](#) (p. 145),

### Keywords

Reduction,

## 2.111 HxPixMin

### Synopsis

```
HxValue HxPixMin (HxImageRep img)
```

### Input

HxImageRep *img* The image you want to determine the minimum of.

### Return value

HxValue The minimum of the pixel values of the input image.

### Description

The function HxPixMin determines the pixel-wise minimum of the input image.

### Remarks

*Valid input images* This function accepts all scalar image types.

### See also

[HxPixSup](#) (p. 148), [HxPixMax](#) (p. 144), [HxPixInf](#) (p. 143),

### Keywords

Reduction,

## 2.112 HxPixProduct

### Synopsis

HxValue HxPixProduct (HxImageRep img)

### Input

HxImageRep img The image you want to determine the product of all pixel values of.

### Return value

HxValue The product of all the pixel values of the input image.

### Description

The function HxPixProduct determines the pixel-wise product of the input image, i.e. the product of all the pixel values of the input image. For vector images the Hadamard product is used, i.e.  $\vec{x} * \vec{y} = \{x_1y_1, x_2y_2, \dots, x_ny_n\}$ .

### Remarks

*Overflow* This function can result in an overflow error.

### See also

[HxPixSum](#) (p. 147), [HxPixMax](#) (p. 144), [HxPixMin](#) (p. 145),

### Keywords

Reduction,

## 2.113 HxPixSum

### Synopsis

```
HxValue HxPixSum (HxImageRep img)
```

### Input

HxImageRep *img* The image you want to determine the sum of all pixel values of.

### Return value

HxValue The sum of all the pixel values of the input image.

### Description

The function HxPixSum determines the pixel-wise sum of the input image, i.e. the sum of all the pixel values of the input image.

### Remarks

*Overflow* This function can result in an overflow error.

### See also

**HxPixProduct** (p. 146), **HxPixMax** (p. 144), **HxPixMin** (p. 145),

### Keywords

Reduction,

## 2.114 HxPixSup

### Synopsis

```
HxValue HxPixSup (HxImageRep img)
```

### Input

HxImageRep *img* The image you want to determine the supremum of.

### Return value

HxValue The supremum of the pixel values of the input image.

### Description

The function HxPixSup determines the pixel-wise maximum of the input image. For scalar image, this results in the maximum over all pixel values. For vector images, this function returns a vector with components equal to the maximum value for each component of the pixel value vectors.

### Remarks

*No possible errors* This function can not result in any errors.

### See also

[HxPixInf](#) (p. 143), [HxPixMax](#) (p. 144), [HxPixMin](#) (p. 145),

### Keywords

Reduction,

## 2.115 HxPow

### Synopsis

```
HxImageRep HxPow(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The first input image to the power of the second input image, pixel wise.

### Description

The function HxPow take the values of the first input image to the power of the second image on a pixel-by-pixel basis. Vector pixel types of the same dimension are treated component wise. Pixel types of different dimensions can be processed by this function as long as the pixel dimensionality of one of the input images is 1. The image with pixel dimensionality 1 is then treated as an image with the same pixel dimensionality as the other image (with equal values per pixel).

### Remarks

*Valid input images*    The two input images should have the same dimensionality, the same pixel dimensionality (or the pixel dimensionality of one of the pixels should be 1) and the same size.

*Overflow error*    The function HxPow can result in an overflow error.

### See also

[HxMul](#) (p. 130), [HxDiv](#) (p. 53), [HxPowVal](#) (p. 150),

### Keywords

Binary, Arithmetic,

## 2.116 HxPowVal

### Synopsis

```
HxImageRep HxPowVal(HxImageRep img, HxValue val)
```

### Input

`HxImageRep img` The input image you want to take the power of.

`HxValue val` The input value.

### Return value

`HxImageRep` The input image to the power of the input value, pixel wise.

### Description

The function `HxPowVal` takes the values of the input image to the power of the input value on a pixel-by-pixel basis. If the pixel type of the input image and the type of the value are both vectors with the same dimensionality, they are treated component wise. If the input image has vector type pixels and the dimensionality of the input value is 1, every component of a pixel vector is taken to the power of the value.

### Remarks

***Valid input images and values*** The dimensionality of the input value should be 1 or equal to the pixel dimensionality of the image.

***Overflow error*** The function `HxPowVal` can result in an overflow error.

### See also

**`HxMulVal`** (p. 132), **`HxDivVal`** (p. 55), **`HxPow`** (p. 149),

### Keywords

Binary value, Arithmetic,

## 2.117 HxProjectRange

### Synopsis

```
HxImageRep HxProjectRange (HxImageRep img, int dimension)
```

### Return value

HxImageRep The result of taking the project range of the input image.

### Description

The function HxProjectRange takes a component of the vector (which component is determined by 'dimension') and assigns it to the corresponding pixel in the output image. The function HxProjectRange only works on vector images. The input parameter 'dimension' should be an integer larger than zero and smaller or equal to the pixel dimension of the input image. The result of this function is a scalar image.

### Remarks

*Valid input images* Only vector images are valid input images.

*Valid values for 'dimension'* The input parameter 'dimension' should be an integer larger than zero and smaller or equal to the pixel dimension of the input image.

### Keywords

Unary,

## 2.118 HxRestrict

### Synopsis

```
HxImageRep HxRestrict (HxImageRep img, HxPoint begin, HxPoint end)
```

### Input

`HxImageRep img` The image you want to restrict to a part of the image.

`HxPoint begin` The begin point of the area that should be cut out of the input image.

`HxPoint end` The end point of the area that should be cut out of the input image.

### Return value

`HxImageRep` An image containing the area (as described by 'begin' and 'end') from the input image.

### Description

The function `HxRestrict` cuts an area from the input image and returns that area as an image. This rectangular area is characterized by 'begin' and 'end', where every component of 'end' should be larger or equal to the corresponding component of 'begin'. 'begin' and 'end' should be valid positions in the input image.

### Remarks

*Valid image types* This function accepts all types of images.

*Valid parameters* 'begin' and 'end' should have the same dimensionality as the pixel dimensionality of the input image. Both points should be valid positions in the input image. Every component of 'end' should be larger or equal to the corresponding component of 'begin'.

### See also

[HxExtend](#) (p. 68), [HxExtendVal](#) (p. 69),

### Keywords

Geometric,

## 2.119 HxRightShift

### Synopsis

```
HxImageRep HxRightShift (HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The right shifted version of the first input image.

### Description

The function HxRightShift shifts the bits of the pixel values of the first input image of the number of places to the right as indicated by the corresponding value in the second input image. For vector images, the each component of a vector is shifted the number of places as indicated by the same component of the corresponding vector value. Only integer (HxShort, HxByte, HxInt ) type images can be used as input images.

### Remarks

*Valid input images*    The input images should have the same dimensionality, the same size and the same pixel dimensionality. The numerical type of the pixels, or its components, should be HxShort, HxByte, HxInt .

### See also

**HxLeftShift** (p. 101), **HxAnd** (p. 11), **HxRightShiftVal** (p. 154),

### Keywords

Binary, Logical,

## 2.120 HxRightShiftVal

### Synopsis

```
HxImageRep HxRightShiftVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img* The input image you want to right shift the pixel values of.

HxValue *val* The input value.

### Return value

HxImageRep The right shifted version of the input image.

### Description

The function HxRightShiftVal shifts the bits of the pixel values of the input image of the number of places to the right as indicated by the input value. For vector images and vector values of the same dimensionality, each component of a vector is shifted the number of places as indicated by the corresponding component of the value vector. Only integer (HxShort, HxByte, HxInt ) type images and values can be used.

### Remarks

**Valid input images and input values** The dimensionality of the input value should be equal to the pixel dimensionality of the input image. The image pixels and input value (or their components) should be HxShort, HxByte, HxInt .

### See also

**HxLeftShiftVal** (p. 102), **HxAndVal** (p. 13), **HxRightShift** (p. 153),

### Keywords

Binary value, Logical,

## 2.121 HxRotate

### Synopsis

```
HxImageRep HxRotate (HxImageRep img, double alpha, HxGeoIntType gi =  
LINEAR, int adjustSize = 1, HxValue background = HxValue(0));
```

### Input

HxImageRep *img*    The image you want to rotate.

double *alpha*    The angle you want to rotate the image over. A positive angle stands for a counterclockwise rotation.

HxGeoIntType *gi*    The type of interpolation that is used by the function. Possible values: NEAREST, LINEAR, see the section on **Interpolation** (p. 2).

int *adjustSize*    This parameter determines whether the resulting image has the same size as the input image (*adjustSize* = 0) or whether the sizes are adjusted (*adjustSize* not equal to 0).

HxValue *background*    The value of that should be given to pixels in the result image that do not originate from the input image.

### Return value

HxImageRep    The rotated image.

### Description

The function HxRotate rotates the image over the center of the image, over an angle of *alpha* degrees. A positive angle stands for a counterclockwise rotation.

Pixels in the result image are calculated by interpolating between the rotated pixels of the input image. The type of interpolation is determined by HxGeoIntType *gi*, see the section on **Interpolation** (p. 2). Pixels in the result image that do not originate from pixels in the input image, will get the value as given by HxValue *background*.

Since the rotated image will normally not fit its original rectangle, you can choose to adjust the size of the output image, such that the complete rotated version fits in (*adjustSize* = 1). When *adjustSize* does not equal 1, the size of the result image will be the same as the input image, as is the scale of the rotated image, resulting in 'chopped of corners' in the result image.

### Remarks

*Valid input images*    This function is properly defined for all 2D images.

*Valid background values*    The background should have the same pixeltype as the pixeltype of the input image.

### See also

anchor.HxReflect, **HxScale** (p. 157),

### Keywords

Geometric,

## 2.122 HxRound

### Synopsis

```
HxImageRep HxRound(HxImageRep img)
```

### Input

HxImageRep *img* The image you want to round the values of.

### Return value

HxImageRep The rounded value of the input image.

### Description

The function HxRound takes the rounded value of each pixel value, where the round of a number is the value of the nearest integer. The round of a vector is defined by separately taking the round of the components of the vector.

### Remarks

*No possible errors* This function can not result in any errors.

### See also

[HxFloor](#) (p. 70), [HxCeil](#) (p. 24),

### Keywords

Unary, Arithmetic,

## 2.123 HxScale

### Synopsis

```
HxImageRep HxScale (HxImageRep img, double sx, double sy, double sz,  
HxGeoIntType gi, int adjustSize)
```

### Input

HxImageRep *img*    The image you want to rotate.

double *sx*    The scaling factor for the x-dimension.

double *sy*    The scaling factor for the y-dimension.

double *sz*    The scaling factor for the z-dimension.

HxGeoIntType *gi*    The type of interpolation that is used by the function. Possible values: NEAREST, LINEAR, see the section on **Interpolation** (p. 2).

int *adjustSize*    This parameter determines whether the resulting image has the same size as the input image (*adjustSize* = 0) or whether the sizes are adjusted (*adjustSize* not equal to 0).

### Return value

HxImageRep    The scaled version of the input image.

### Description

The function `HxScale` scales the image, with factors given for each dimension. *sx*, *sy* and *sz* respectively give the scaling factors for the x-, y- and z-direction. *sx*, *sy* and *sz* should be larger or equal to zero, with the restriction that the resulting image should have at least a size 1 for every dimension that the input image has.

Pixels in the result image are calculated by interpolating between the placed pixels of the input image. The type of interpolation is determined by HxGeoIntType *gi*, see the section on **Interpolation** (p. 2). Pixels in the result image that do not originate from pixels in the input image, will get the value '0'.

Since the scaled image will normally not have the size of the input image, you can choose to adjust the size of the output image, such that the complete scaled version fits in (*adjustSize* = 1). When *adjustSize* does not equal 1, the size of the result image will be the same as the input image. In this case there is either part of the resulting image that remains 'empty' (when scaling factors are smaller than 1) or only the top left corner (in case of 2D) of the image appears in the resulting image (when scaling factors are larger than 1).

### Remarks

**Valid input images**    This function is properly defined for all types of images.

**Valid scaling factors**    All factors should be larger or equal to zero. Additionally, the factors should be such that the resulting image has a size of at least 1 in every direction. So for 2D images, *sx* and *sy* should be large enough. For 3D images, *sz* also has to be large enough.

### See also

`anchor_HxReflect`, **HxRotate** (p. 155), **HxExtend** (p. 68), **HxRestrict** (p. 152),

### Keywords

Geometric,

## 2.124 HxSin

### Synopsis

```
HxImageRep HxSin (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the sine of.

### Return value

HxImageRep    The pixelwise sine of the input image.

### Description

This function takes the sine of every pixel value of the input image. For vector images, the sine of every element of the vector is taken.

### Remarks

*No possible errors*    This function can not result in any errors.

### Examples

Taking the sine of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxSinExample1() {
    HxSizes sizes(256, 256, 0);
    double data[256*256];
    for(int i=0; i<256; i++)
        for(int j=0; j<256; j++) {
            data[i+j*256]=(i/32.0)*(i/32.0);
        }
    HxImageRep img = HxMakeFromDoubleData(1, 2, sizes, data);
    img = HxSin(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep img = HxSinExample1();
    HxWriteFile(img, argv[1]);

    return 0;
}
```

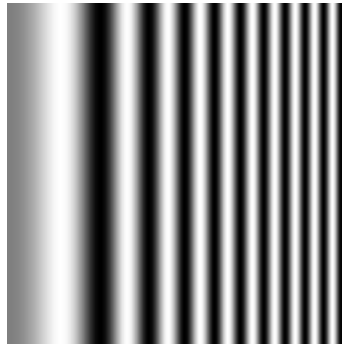


Figure 2.57: Output image of the HxSin example.

This example generates an images with a number of stripes, using the sine function.

**See also**

**HxCos** (p. 46), **HxTan** (p. 172), **HxAsin** (p. 15), **HxSinh** (p. 160),

**Keywords**

Unary, Trigonometric,

## 2.125 HxSinh

### Synopsis

```
HxImageRep HxSinh (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the hyperbolic sine of.

### Return value

HxImageRep    The pixelwise hyperbolic sine of the input image.

### Description

This function takes the hyperbolic sine of every pixel value of the input image. For vector images, the hyperbolic sine of every element of the vector is taken.

### Remarks

**Overflow error**    The function HxSinh can result in an overflow error.

### Examples

Taking the hyperbolic sine of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxSinhExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxContrastStretch(img, 4.0);
    img = HxAddVal(img, -2.0);
    img = HxSinh(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);
    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxSinhExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.58: Input image for the HxSinh example.



Figure 2.59: Output image of the HxSinh example.

This example illustrates the result of taken the hyperbolic sine of an image.

**See also**

**HxSin** (p. 158), **HxCosh** (p. 48), **HxAsin** (p. 15),

**Keywords**

Unary, Trigonometric,

## 2.126 HxSqrt

### Synopsis

```
HxImageRep HxSqrt (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the square root of.

### Return value

HxImageRep    The square root of the input image.

### Description

The function HxSqrt takes the square root of each pixel value. The square root of a vector is defined by taking the square root of the components of the vector.

### Remarks

*Valid input images*    Only image with values greater than or equal to zero are valid. If one of the values is less than zero (or one of the components of a vector is less than zero), the input image is not valid.

### Examples

Taking the square root of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxSqrtExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxSqrt(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep img = HxMakeFromFile(argv[1]);
    img = HxSqrtExample1(img);
    HxWriteFile(img, argv[2]);

    return 0;
}
```



Figure 2.60: Input image for the HxSqrt example.



Figure 2.61: Output image of the HxSqrt example.

In this example we first convert the input image to a double valued image. We take the square root, after which we stretch the image to let it contain values ranges from 0.0 to 255.0 for visualization purposes.

**See also**

**HxPow** (p. 149),

**Keywords**

Unary, Arithmetic,

## 2.127 HxSquaredDistance

### Synopsis

```
HxImageRep HxSquaredDistance (HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The squared Euclidian distance between the two input images.

### Description

The function `HxSquaredDistance` takes the squared Euclidian distance between every two corresponding pixels of the input images. The value is stored in the corresponding pixel of the output image. The pixel precision of the output image equals the highest precision of the input images. The squared distance of two integer images, results in an integer result, while the squared distance of an integer and a double image, results in a double image.

### Remarks

*Valid input images*    The input images should have the same dimensionality, the same size and the same pixel dimensionality.

*Possible overflow*    This function can result in an overflow.

### See also

`HxSqrt` (p. 162), `HxMul` (p. 130),

### Keywords

Binary, Distance,

## 2.128 HxSub

### Synopsis

```
HxImageRep HxSub(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The subtraction of the input images

### Description

The function HxSub subtracts two images on a pixel-by-pixel basis. Vector pixel types of the same dimension are subtracted component wise, while pixel types of different dimensions cannot be summed.

### Remarks

**Admissable image types**    The two input images should have the same dimensionality, the same pixel dimensionality and the same size.

**Warning for possible overflow**    Subtraction can result in an overflow error.

### Examples

Subtracting two images

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxSubExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxImageAsDouble(im1);
    im2 = HxImageAsDouble(im2);
    im1 = HxSub(im1, im2);
    im1 = HxContrastStretch(im1, 255);
    im1 = HxImageAsByte(im1);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxSubExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.62: First input image for the HxSub example.

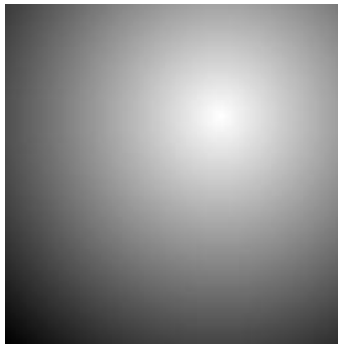


Figure 2.63: Second input image for the HxSub example.



Figure 2.64: Output image of the HxSub example.

In this example we subtract two (scalar) images. First we convert the image to double images to prevent overflow as much as possible. After subtracting the images, we stretch the result and convert it to byte for visualization purposes.

**See also**

**HxAdd** (p. 7), **HxMul** (p. 130), **HxDiv** (p. 53),

**Keywords**

Binary, Arithmetic,

## 2.129 HxSubVal

### Synopsis

```
HxImageRep HxSubVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The input value

### Return value

HxImageRep    The subtraction of the input image and the input value.

### Description

The function `HxSubVal` subtracts the `HxValue val` from every pixel value in the input image and stores the result in the corresponding pixel of the output image. The dimensionality of 'val' and the `pixeldimensionality` of the input image must be the same. If 'val' is a vector and the pixels of the input image are vectors, the values are subtracted component wise.

### Remarks

**Admissable image and value types**    The dimensionality of 'val' and the `pixeldimensionality` of the input image must be the same.

**Overflow error**    The function `HxSubVal` can result in an overflow error.

### See also

**HxSub** (p. 165), **HxAddVal** (p. 10), **HxDivVal** (p. 55), **HxMulVal** (p. 132),

### Keywords

Binary value, Arithmetic,

## 2.130 HxSup

### Synopsis

```
HxImageRep HxSup(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image

HxImageRep im2    The second input image

### Return value

HxImageRep    The pixelwise supremum of the input images

### Description

The function HxSup takes the supremum of two images on a pixel-by-pixel basis. The supremum of two vector values of equal dimensionality is defined as the component wise supremum resulting in a vector with the same dimensionality. The supremum of pixel types with different dimensionalities cannot be taken.

### Remarks

**Admissable image types**    The two input images should have the same dimensionality, the same pixel dimensionality and the same size.

### Examples

Taking the pixelwise supremum of two images

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxSupExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxSup(im1, im2);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxSupExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.65: First input image for the HxSup example.

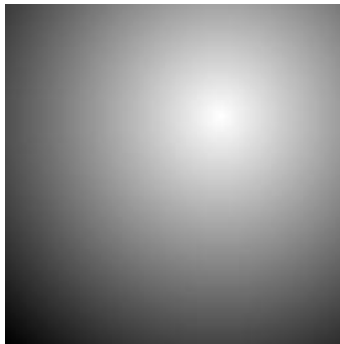


Figure 2.66: Second input image for the HxSup example.



Figure 2.67: Output image of the HxSup example.

In this example take the supremum of two (scalar) images.

**See also**

**HxMax** (p. 124), **HxMin** (p. 126), **HxInf** (p. 97),

**Keywords**

Binary, Arithmetic,

## 2.131 HxSupVal

### Synopsis

```
HxImageRep HxSupVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image

HxValue *val*    The input value

### Return value

HxImageRep    The supremum of the input image and the input value.

### Description

The function HxSupVal takes the supremum of HxValue ‘val’ and every pixel value in the input image (for every pixel again) and stores the result in the corresponding pixel of the output image. The dimensionality of ‘val’ and the pixel dimensionality of the input image must be the same. If ‘val’ is a vector and the pixels of the input image are vectors, the supremum is taken component wise.

### Remarks

*Admissable image and value types*    The dimensionality of ‘val’ and the pixel dimensionality of the input image must be the same.

### See also

**HxSup** (p. 169), **HxInfVal** (p. 99), **HxInf** (p. 97),

### Keywords

Binary value, Arithmetic,

## 2.132 HxTan

### Synopsis

```
HxImageRep HxTan (HxImageRep img)
```

### Input

HxImageRep img    The image you want to take the tangent of.

### Return value

HxImageRep    The pixelwise tangent of the input image.

### Description

This function takes the tangent of every pixel value of the input image. For vector images, the tangent of every element of the vector is taken. Note that the tangent is not defined for number  $\pi + \frac{k}{2}\pi$ , where k is an integer.

### Remarks

**Valid input images**    All pixel values of the input image should be different from  $\pi + \frac{k}{2}\pi$ , where k is an integer. The function does not check on this condition. If a pixel value is invalid a 'not a number' is returned.

**Overflow error**    The function HxTan can result in an overflow error.

### Examples

Taking the tangent of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxTanExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxContrastStretch(img, 3.0);
    img = HxSubVal(img, 1.5);
    img = HxTan(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep img = HxMakeFromFile(argv[1]);
    img = HxTanExample1(img);
    HxWriteFile(img, argv[2]);

    return 0;
}
```



Figure 2.68: Input image for the HxTan example.

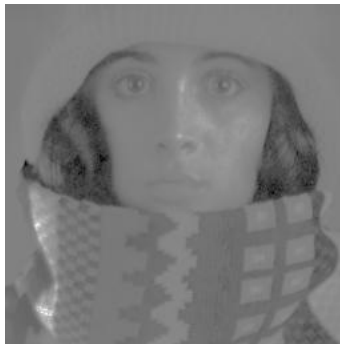


Figure 2.69: Output image of the HxTan example.

This example illustrates the result of taking the tangent of an image

**See also**

**HxSin** (p. 158), **HxCos** (p. 46), **HxAtan** (p. 17), **HxTanh** (p. 174),

**Keywords**

Unary, Trigonometric,

## 2.133 HxTanh

### Synopsis

HxImageRep HxTanh (HxImageRep img)

### Input

HxImageRep img   The image you want to take the hyperbolic tangent of.

### Return value

HxImageRep   The pixelwise hyperbolic tangent of the input image.

### Description

This function takes the hyperbolic tangent of every pixel value of the input image. For vector images, the hyperbolic tangent of every element of the vector is taken.

### Remarks

*No possible errors*   This function can not result in any errors.

### Examples

Taking the hyperbolic tangent of an image.

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxTanhExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxContrastStretch(img, 4.0);
    img = HxAddVal(img, -2.0);
    img = HxTanh(img);
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);
    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    im1 = HxTanhExample1(im1);
    HxWriteFile(im1, argv[2]);

    return 0;
}
```



Figure 2.70: Input image for the HxTanh example.



Figure 2.71: Output image of the HxTanh example.

This example illustrates the result of taking the hyperbolic tangent of an image.

**See also**

**HxTan** (p. 172), **HxSinh** (p. 160), **HxAtan** (p. 17),

**Keywords**

Unary, Trigonometric,

## 2.134 HxThreshold

### Synopsis

```
HxImageRep HxThreshold (HxImageRep img, HxValue level)
```

### Input

HxImageRep img   The image you want to threshold.

HxValue level    The threshold level.

### Description

The function HxThreshold compares each pixel value with the given value of level. If the pixel value is smaller than 'level' the corresponding pixel in the output image has value zero, otherwise the corresponding pixel in the output image has value one. The function HxThreshold only works for scalar images.

### Remarks

*Valid input images*   The input image should be a scalar image.

*Valid level values*   The level should be a scalar.

### Examples

Thresholding an image

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxThresholdExample1(HxImageRep img) {
    img = HxImageAsDouble(img);
    img = HxContrastStretch(img, 255.0);
    img = HxThreshold(img, 32.0); // 128.0
    img = HxContrastStretch(img, 255.0);
    img = HxImageAsByte(img);

    return img;
}

int main(int argc, char* argv[])
{
    HxImageRep img = HxMakeFromFile(argv[1]);
    img = HxThresholdExample1(img);
    HxWriteFile(img, argv[2]);

    return 0;
}
```



Figure 2.72: Input image for the HxThreshold example.



Figure 2.73: Output image of the HxThreshold example.

In this example we threshold an image, with threshold value 128.

**See also**

**HxTriStateThreshold** (p. 178),

**Keywords**

Threshold, Segmentation,

## 2.135 HxTriStateThreshold

### Synopsis

```
HxImageRep HxTriStateThreshold (HxImageRep img, HxValue level, HxValue v1, HxValue v2, HxValue v3)
```

### Input

HxImageRep *img*    The image you want to threshold.

HxValue *level*    The threshold level.

HxValue *v1*    The value assigned to pixels with a value smaller than the threshold level.

HxValue *v2*    The value assigned to pixels with a value equal to the threshold level.

HxValue *v3*    The value assigned to pixel with a value larger than the threshold level.

### Return value

HxImageRep    The thresholded image.

### Description

The function `HxTriStateThreshold` compares each pixel value with the given value of 'level'. If the pixel value is smaller than 'level' the corresponding pixel in the output image has a value equal to `HxValue v1`, if the value is equal to 'level', the corresponding pixel in the output image has a value equal to `HxValue v2`, otherwise the corresponding pixel in the output image has a value equal to `HxValue v3`. The function `HxThreshold` only works for scalar images.

### Remarks

*Valid input images*    The input image should be a scalar image.

*Valid level values*    The level should be a scalar.

### See also

`HxThreshold` (p. 176),

### Keywords

Threshold, Segmentation,

## 2.136 HxUnaryMax

### Synopsis

```
HxImageRep HxUnaryMax (HxImageRep img)
```

### Return value

HxImageRep The result of taking the unary maximum of the input image.

### Description

The function HxUnaryMax takes the maximum of every pixel value in the input image and stores it in the corresponding pixel of the output image. The unary maximum of a scalar equals the scalar itself, while the unary maximum of a vector equals the maximum of the components of the vector. If the input is a scalar image, this function returns the same image.

### Remarks

*No possible errors* HxUnaryMax does not result in any errors.

### Keywords

Unary, Arithmetic,

## 2.137 HxUnaryMin

### Synopsis

```
HxImageRep HxUnaryMin (HxImageRep img)
```

### Return value

HxImageRep The result of taking the unary minimum of the input image.

### Description

The function HxUnaryMin takes the minimum of every pixel value in the input image and stores it in the corresponding pixel of the output image. The unary minimum of a scalar equals the scalar itself, while the unary minimum of a vector equals the minimum of the components of the vector. If the input is a scalar image, this function returns the same image.

### Remarks

*No possible errors* HxUnaryMin does not result in any errors.

### Keywords

Unary, Arithmetic,

## 2.138 HxUnaryProduct

### Synopsis

```
HxImageRep HxUnaryProduct (HxImageRep img)
```

### Return value

HxImageRep The result of taking the unary product of the input image.

### Description

The function HxUnaryProduct takes the unary product of every pixel value in the input image and stores it in the corresponding pixel of the output image. The unary product of a scalar equals the scalar itself, while the unary product of a vector equals the product of the components of the vector. If the input is a vector image containing HxBytes, the output image is of the integer type. If the input is a scalar image, this function returns the same image.

### Remarks

*Possible errors* HxUnaryProduct can result in an overflow error.

### Keywords

Unary, Arithmetic,

## 2.139 HxUnarySum

### Synopsis

```
HxImageRep HxUnarySum (HxImageRep img)
```

### Return value

HxImageRep The result of taking the unary sum of the input image.

### Description

The function HxUnarySum takes the unary sum of every pixel value in the input image and stores it in the corresponding pixel of the output image. The unary sum of a scalar equals the scalar itself, while the unary sum of a vector equals the sum of the components of the vector. If the input is a vector image containing HxBytes, the output image is of the integer type. If the input is a scalar image, this function returns the same image.

### Remarks

*Possible errors* HxUnarySum can result in an overflow error.

### Keywords

Unary, Arithmetic,

## 2.140 HxUniform

### Synopsis

```
HxImageRep HxUniform (HxImageRep img, HxSizes sizes)
```

### Input

`HxImageRep img` The image you want to convolve with a uniform filter.

`HxSizes sizes` The size of the uniform filter.

### Return value

`HxImageRep` The result of convolving the input image with the a uniform filter as described by 'sizes'.

### Description

The function `HxUniform` convolves the input image with a uniform filter with sizes as given by the input parameter 'sizes'. If the input image is a 2D image, 'sizes' should have 2 components. In case of a 3D image, 'sizes' should have 3 components. For vector images, each channel is convolved separately. Note that the sizes of the filter may not exceed the corresponding sizes of the image. The output image has double (vector) values.

### Remarks

**Valid image types** All types of images are allowed for this function. For vector images, each channel is treated separately.

**Border handling** This function uses MIRRORED border handling, see the section on **Border handling** (p. 2).

### See also

`HxGauss` (p. 71), `anchor_HxPercentile`,

### Keywords

Filter, Convolution,

## 2.141 HxWriteFile

### Synopsis

```
bool HxWriteFile (HxImageRep img, HxString fileName)
```

### Input

HxImageRep img    The image you want to save.

### Return value

bool    The function HxWriteFile returns true if the image was successfully written to file, otherwise it returns false.

### Description

The function HxWriteFile writes an image to file. The file extension you use in the filename determines the image format of the file. The supported file formats are: tif, jpg, ppm . The format of the filename itself, accepts slashes as well as backslashes.

### Remarks

*Valid file formats*    Although the different standards for image file formats allow for instance doubles as pixel types, a number of paint programs only accept integer or byte types for the pixel values. When a program fails to load the image you wrote to file with the Horus library, try converting it to byte values first.

### See also

**HxMakeFromFile** (p. 115), anchor\_HxImagesToFile,

### Keywords

File I/O, Export,

## 2.142 HxXor

### Synopsis

```
HxImageRep HxXor(HxImageRep im1, HxImageRep im2)
```

### Input

HxImageRep im1    The first input image.

HxImageRep im2    The second input image.

### Return value

HxImageRep    The logical 'xor' of the input images.

### Description

The function HxXor takes the logical 'xor' of corresponding pixels in the two images. The 'xor' of two vectors is taken component wise.

### Remarks

**Valid input images**    The input images should have the same dimensionality, the same size, the same pixel dimensionality and the same pixel type. The numerical values of the pixels, or its components, should be HxShort, HxByte, HxInt .

### Examples

Taking the logical 'xor'

```
#include "HxImageRepGlobalFuncs.h"
#include "HxImageRep.h"

HxImageRep HxXorExample1(HxImageRep im1, HxImageRep im2) {
    im1 = HxImageAsDouble(im1);
    im2 = HxImageAsDouble(im2);
    im1 = HxContrastStretch(im1, 1.0);
    im2 = HxContrastStretch(im2, 1.0);
    im1 = HxImageAsByte(im1);
    im2 = HxImageAsByte(im2);
    im1 = HxXor(im1, im2);
    im1 = HxContrastStretch(im1, 255);

    return im1;
}

int main(int argc, char* argv[])
{
    HxImageRep im1 = HxMakeFromFile(argv[1]);
    HxImageRep im2 = HxMakeFromFile(argv[2]);
    im1 = HxXorExample1(im1, im2);
    HxWriteFile(im1, argv[3]);

    return 0;
}
```



Figure 2.74: First input image for the HxXor example.

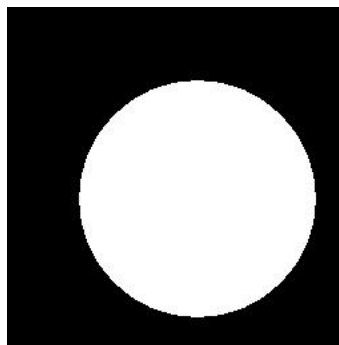


Figure 2.75: Second input image for the HxXor example.

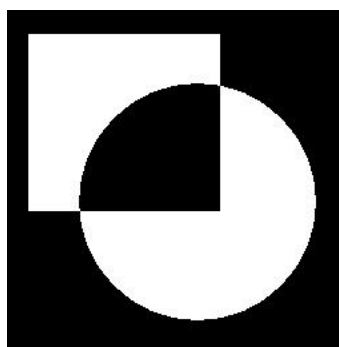


Figure 2.76: Output image of the HxXor example.

In this example we take the logical 'xor' of the two input images.

**See also**

**HxAnd** (p. 11), **HxOr** (p. 140), **HxXorVal** (p. 187),

**Keywords**

Binary, Logical,

## 2.143 HxXorVal

### Synopsis

```
HxImageRep HxXorVal(HxImageRep img, HxValue val)
```

### Input

HxImageRep *img*    The input image.

HxValue *val*    The input value.

### Return value

HxImageRep    The logical 'xor' of the input image and the input value.

### Description

The function HxXorVal takes the logical 'xor' of the pixels from the input image with the input value on a pixel-by-pixel basis. The 'xor' of two vectors is taken component wise.

### Remarks

***Valid input images and input values***    The input value should have the same dimensionality as the pixel dimensionality of the input image. The pixel type and the type of the input value should be, HxShort, HxByte, HxInt . The types should be equal.

### See also

**HxOrVal** (p. 142), **HxAndVal** (p. 13), **HxXor** (p. 185),

### Keywords

Binary value, Logical,

---

# Index

- Arithmetic , 4, 9, 10, 14, 24, 27, 28, 50, 51, 54–57, 60, 70, 98, 99, 110, 112, 124–127, 131–137, 149, 150, 156, 163, 167, 168, 170, 171, 179–182
- Binary , 9, 12, 50, 54, 56, 58, 74, 78, 98, 100, 101, 103, 107, 124, 126, 128, 131, 138, 141, 149, 153, 164, 167, 170, 186
- Binary value , 10, 13, 51, 55, 57, 59, 75, 79, 99, 102, 108, 125, 127, 129, 132, 139, 142, 150, 154, 168, 171, 187
- Canny , 20, 22
- Color , 25
- Comparison , 58, 59, 74, 75, 78, 79, 103, 104, 107, 108, 138, 139
- Complex , 14, 28
- Conversion , 25, 61–67, 80–94, 119, 121
- Convolution , 32, 34, 37, 39, 42, 45, 71–73, 183
- Distance , 164
- Edge detection , 20, 22
- Enhancement , 30
- Export , 61–67, 184
- File I/O , 115, 116, 184
- Filter , 20, 22, 32, 34, 37, 39, 42, 45, 71–73, 183
- Gauss , 32, 34, 71–73
- Gaussian derivative , 72, 73
- Generation , 113, 114, 117–120, 122, 123
- Geometric , 68, 69, 152, 155, 157
- Import , 115, 116
- Inquiry , 95, 96
- Logical , 12, 13, 101, 102, 128, 129, 141, 142, 153, 154, 186, 187
- Morphology , 52
- Norm , 134–137
- Projection , 100
- Reduction , 143–148
- Segmentation , 177, 178
- Threshold , 177, 178
- Trigonometric , 6, 16, 18, 47, 49, 159, 161, 173, 175
- Unary , 4, 6, 14, 16, 18, 24, 27, 28, 47, 49, 60, 70, 110, 112, 133–137, 151, 156, 159, 161, 163, 173, 175, 179–182
-