

Image Processing Operations

Version 0.2

Jeremy Worley, Thomas Robey, Kelly Shuldberg

Khoral Research, Inc.

April 28, 1998



(c) 1997 Khoral Research, Inc. All Rights Reserved. Copies of part or all of this work is granted without fee provided the copies are not made or distributed for profit or commercial advantage and that the copies bear this notice. Khoral Research makes no warranty of any kind with regard to this material and shall not be liable for errors contained herein.

TABLE OF CONTENTS

<i>TABLE OF CONTENTS</i>	<i>i</i>
<i>Introduction</i>	<i>1</i>
Image Processing on Embeddable Systems	1
Abstract Data Type	2
Functional Limitations and Data Organization	2
<i>Histogram Operations</i>	<i>4</i>
vsip_icdfeq_p Cumulative Density Equalization.....	4
vsip_ihisteq_p Histogram Equalization	6
vsip_ihistogram_p Compute Image Histogram.....	8
vsip_ihistfunc_p Moving Histogram Operation.....	10
vsip_imedfilt_p Median Filter	12
<i>Convolution Operators</i>	<i>14</i>
vsip_iconv_p Convolution.....	14
<i>Discrete Difference Operators</i>	<i>16</i>
vsip_isobel_p Sobel Gradient Edge Detection.....	16
vsip_iroberts_p Roberts Gradient Edge Detection	18
vsip_iprewitt_p Prewitt Gradient Edge Detection.....	20
<i>Morphological Operators</i>	<i>22</i>
vsip_idilate_p Grayscale Morphological Dilation.....	22
vsip_ierode_p Grayscale Morphological Erosion	24
vsip_iopen_p Grayscale Morphological Opening	26
vsip_iclose_p Grayscale Morphological Closing.....	28
<i>Resize Operators</i>	<i>30</i>
vsip_iinterp_p Expand Image.....	30
vsip_iresample_p Shrink Image	32
<i>Pad Operators</i>	<i>34</i>
vsip_ipadconst_p Pad Image with Constant.....	34
vsip_ipadreflect_p Pad Image by Reflection	36
vsip_ipadextend_p Pad Image by Extension	38
<i>Other Functions</i>	<i>40</i>
vsip_istats_p Compute Image Statistics.....	40
vsip_idct8x8_p 8x8 Discrete Cosine Transform.....	43
vsip_ilut_p Compute Image using Look Up Table.....	45
vsip_ilabel_p Compute Connected Components Image	46
vsip_ibind_p Create and Bind an Image View	48
vsip_ikreate_p Create Image	51
vsip_idestroy_p Destroy Image View	54
vsip_igetoffset_p Image Get Offset	56
vsip_igetstride1_p Image Get Stride1	58
vsip_igetstride2_p Image Get Stride2	60
vsip_igetlength1_p Image Get Length1	62
vsip_igetlength2_p Image Get Length2	64

Introduction

This chapter presents the image processing component of the VSIP library. As with other components of this library, the image processing functions were designed with embeddable systems in mind, while simultaneously addressing the needs of workstation and supercomputer application developers. The objective is to create a standardized, reusable component library for image processing on embeddable systems. It is intentionally vendor neutral to the extent possible.

While the vector, matrix, and signal processing libraries have a relatively rich history of standardization, image processing is not nearly so standardized. In spite of efforts such as PIK, vendors and tools developers have generally implemented proprietary libraries for image processing that have been tuned for specific application areas such as medical imaging or multimedia. In many ways, the effort discussed here is no different. The application areas being specifically targeted are military applications such as automatic target recognition algorithms running on embeddable systems. However, we have attempted to provide the basic tools in a general purpose format so that the library can be reused in a broader application space than that for which it was specifically designed.

In designing this library, numerous specifications and implementations have been examined to determine the scope and nature of functionality. These include PIK, XIL, mediaLib, CSPI's IPL, Khoros, LNK's proprietary image library, and Datacube's imaging library.

Because the image processing library has been designed for embeddable systems, functionality has been scoped to favor low-level, general-purpose functionality rather than high-level application-specific functionality. For example, 2D orthogonal transforms are available, but colorometric operators are not.

Image Processing on Embeddable Systems

Most of the proposed standards for image processing were designed for workstations and high performance computing platforms. In general, image processing has not yet achieved a wide enough user demand in the military's embedded systems community to motivate most vendors to develop optimized libraries.

Embedded HPC systems differ significantly from the workstation environment. The differences include limited memory, limited power, and limited physical size. Consequently, many embedded applications have performance constraints where the hardware's peak performance is very close to the algorithmic/application processing requirements. Furthermore, most embedded systems vendors have chosen not to invest in sophisticated compiler technology, choosing instead to adapt freely available compilers, building highly optimized libraries that allow an application developer to hand-optimize their own applications. The VSIP API is designed to allow optimized VSIP library implementers to perform those application optimizations "under the hood," thus lowering the complexity of porting a legacy application to a different vendor's hardware.

A significant effort has been expended by the VSIP Forum to allow optimized libraries to exploit techniques such as loop fusion and strip-mining without exposing the underlying architecture to the application developer. In designing the image processing library, we have tried to address the extent to which these kinds of optimizations can be implemented. Given that, on average, the number of FLOPS per logical operation for images is an order of magnitude higher than vectors and signals, it may not be important that two consecutive VSIP operators can be fused. However, the VSIP image library has been designed with the same design factors as the rest of the library, so such optimization should be possible.

Most image processing libraries such as PIK and Khoros use polymorphic operators to reduce the number of the entry points in the library to something closer to the number of logical operations present in the library. Other library standards such as XIL provide separate functions in the interest of producing small binaries. VSIP favors the latter approach, usually creating small non-polymorphic functions that result in smaller binaries.

Abstract Data Type

The data object for images inherits the properties of matrix views, but may also contain additional information. Efficient operation on images requires a greater level of flexibility specifying the organization of the data. For example, specifying interlaced versus non-interlaced images has performance implications for algorithms. Functions that operate on matrix views should also operate on image views.

Manipulation of color images uses multiband data, which could be convenient to handle as a single object. If an implementation supports multiband data, then three-banded data should be defined using a *3* or *4* in the place in the encoder ring where the *c* for *complex* is located.

Time sequences of images, for now, will be handled simply as multiple image views or as an array of image views. This approach eliminates the responsibility of the library implementer from having to manage the complexity of potentially heterogeneous time sequences of images. This technique is believed to be an optimal solution to the problem because most applications will be operating on a single block at any given operational stage of the algorithm. As such, the complexity and computational costs of image sequence management exist only for applications that use such functionality.

Functional Limitations and Data Organization

The image processing library will support unsigned byte, signed short, signed long, float, float complex, double, double complex, and bit data types. The bit data type presents complications because there is no support within C for it. Furthermore, portability represents a big problem due to Endian issues. Consequently, bit data will be implemented as follows:

- The bit data type will be padded at the end of a scan line to the nearest multiple of 32 bits.
- The bit data type will be defined relative to a byte with pixel 0 of a scan line residing in the 2^7 pixel location on “big-bittian” systems.
- The smallest unit of access provided by the library will be a *rowlet*, which is a 32-bit quantity that would be packed into a 32-bit int.

Histogram Operations

vsip_icdfeq_p

Cumulative Density Equalization

Equalize an image's unbinned histogram.

Functionality

Cumulative density equalization is a point-based image enhancement technique in which the gray levels of an image are redistributed over the image's original dynamic range so that they have a uniform density. This is achieved by computing the unbinned cumulative density function (CDF) and using that function as a mapping from the source image to the destination image. The CDF is a monotonically increasing function and thus guarantees preservation of pixel gray level order.

$$C(r) \leftarrow \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \begin{cases} 1 & src(m,n) < r \\ 0.5 & src(m,n) = r \\ 0 & src(m,n) > r \end{cases}$$

$$dest(m,n) \leftarrow C(cdf_src(m,n)) \cdot (src_{max} - src_{min}) + src_{min}$$

where C is the cumulative distribution function (CDF) computed as shown above, src is the input image, cdf_src is the image from which the CDF is to be calculated, $dest$ is the output image, src_{min} is the minimum pixel value in src , and src_{max} is the maximum pixel value in src .

Prototypes

```
void vsip_icdfeq_i(
    const vsip_iview_i *src,
    const vsip_iview_i *cdf_src,
    const vsip_iview_i *dest);

void vsip_icdfeq_f(
    const vsip_iview_f *src,
    const vsip_iview_f *cdf_src,
    const vsip_iview_f *dest);
```

Arguments

<code>*src</code>	View of input object
<code>*cdf_src</code>	View of CDF source object
<code>*dest</code>	View of output object

Return Value

Void

Restrictions**Errors/Exceptions****Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

`vsip_ihisteq_p`

vsip_ihisteq_p**Histogram Equalization**

Equalize an image's binned histogram.

Functionality

Histogram equalization is a point-based enhancement technique in which the gray levels of an image are redistributed over the image's original dynamic range so that each bin has a uniform density. This is achieved by computing the binned cumulative density function (CDF) from the histogram and using that function as a mapping from the source image to the destination image. The CDF is a monotonically increasing function and, as such, guarantees preservation of pixel gray level order.

$$C(src(m,n)) \leftarrow \frac{\left[\sum_{r=0}^P P(r) + \frac{src(m,n) - src_{min}}{length(hst_src)} P(p+1) \right]}{\frac{length(hst_src)}{\sum_{r=0} P(r)}}, P = \left\lfloor \frac{src(m,n) - src_{min}}{src_{max} - src_{min}} length(hst_src) \right\rfloor$$

$$dest(m,n) = C(src(m,n)) \cdot (src_{max} - src_{min}) + src_{min}$$

where C is the cumulative distribution function (CDF), src is the input image, hst_src is the input histogram, $dest$ is the output image, src_{min} is the minimum pixel value in src , and src_{max} is the maximum pixel in src .

Prototypes

```
void vsip_ihisteq_byte (
    const vsip_iview_byte *src,
    const vsip_vview_i *hst_src,
    const vsip_iview_byte *dest);
```

```
void vsip_ihisteq_i (
    const vsip_iview_i *src,
    const vsip_vview_i *hst_src,
    const vsip_iview_i *dest);
```

```
void vsip_ihisteq_f (
    const vsip_iview_f *src,
    const vsip_vview_i *hst_src,
    const vsip_iview_f *dest);
```

Arguments

<code>*src</code>	View of the input object
<code>*hst_src</code>	Vector view of histogram object
<code>*dest</code>	View of output object

Return Value

Void

Restrictions**Errors/Exceptions****Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

`vsip_icdfeq_p`, `vsip_ihistogram_p`

vsip_ihistogram_p**Compute Image Histogram**

Compute the histogram of an image.

Functionality

The histogram of an image is a function that measures the frequency of occurrence for every gray level in the image. Image histograms are frequently used in image segmentation and image quantization algorithms. This function computes a histogram of an image by counting the number of pixels present in a series of bins. The bins are specified by providing the lower and upper bound of the range of interest; the length of the output vector supplies the number of bins.

$$P(r) \leftarrow \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \begin{cases} 1, & \left[\frac{upper - lower}{length(dest)} r \right] \leq src(m, n) < \left[\frac{upper - lower}{length(dest)} (r+1) \right], 0 \leq r < length(dest) \\ 0, & otherwise \end{cases}$$

where P is the output histogram vector, src is the input image, r is the bin index and range, and $length(dest)$ is the number of bins. The lower bound of interest is assumed to be $lower$ and the upper bound is assumed to be $upper$.

Prototypes

```
void vsip_ihistogram_byte (
    const vsip_iview_byte *src,
    vsip_scalar_byte lower,
    vsip_scalar_byte upper,
    const vsip_vview_i *dest);

void vsip_ihistogram_i(
    const vsip_iview_i *src,
    vsip_scalar_i lower,
    vsip_scalar_i upper,
    const vsip_vview_i *dest);

void vsip_ihistogram_f(
    const vsip_iview_f *src,
    vsip_scalar_f lower,
    vsip_scalar_f upper,
    const vsip_vview_i *dest);
```

Arguments

<code>*src</code>	View of input image object
<code>lower</code>	Lower bound of first bin
<code>upper</code>	Upper bound of last bin
<code>*dest</code>	View of output histogram vector object

Return Value

Void

Restrictions

The output view must be of an integer datatype large enough to hold the maximum number of occurrences.

Errors/Exceptions**Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

vsip_ihistfunc_p**Moving Histogram Operation**

Compute a histogram from a moving window in an image.

Functionality

The moving histogram operation moves a window across an image and computes a histogram for each window, centered on every pixel. For each pixel, it calls a user-specified function with a vector containing the histogram, which returns a value to be placed in the output image at the pixel location. The output region of support is determined by the size of *dest*.

Prototypes

```
void vsip_ihistfunc_bit(
    const vsip_iview_bit *src,
    vsip_scalar_i w,
    vsip_scalar_i h,
    vsip_scalar_rowlet (*op)(
        const vsip_vview_i *h)
    const vsip_iview_bit *dest);

void vsip_ihistfunc_byte (
    const vsip_iview_byte *src,
    vsip_scalar_i w,
    vsip_scalar_i h,
    vsip_scalar_byte (*op)(
        const vsip_vview_i *h,
        vsip_scalar_i min,
        vsip_scalar_i max),
    const vsip_iview_byte *dest);

void vsip_ihistfunc_i(
    const vsip_iview_i *src,
    vsip_scalar_i w,
    vsip_scalar_i h,
    vsip_scalar_i (*op)(
        const vsip_vview_i *h
        vsip_scalar_i min,
        vsip_scalar_i max),
    const vsip_iview_i *dest);

void vsip_ihistfunc_f(
    const vsip_iview_f *src,
    vsip_scalar_i w,
    vsip_scalar_i h,
    vsip_scalar_i (*op)(
        const vsip_vview_i *h
        vsip_scalar_f min,
        vsip_scalar_f max),
    const vsip_iview_f *dest);
```

Arguments

<code>*src</code>	View of input image object
<code>w</code>	Width of the histogram window
<code>h</code>	Height of the histogram window
<code>*op</code>	Function to produce output pixel based on windowed histogram
<code>*dest</code>	View of output image object

Return Value

Void

Restrictions

The width and height of the histogram window must be odd and greater than 1. The difference between the width and height of the input and output objects must be 0, or $\pm(w-1)/2$ and $\pm(h-1)/2$, respectively.

Errors/Exceptions**Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

vsip_imedfilt_p**Median Filter**

Smooth an image with a median filter.

Functionality

A median filter is an image smoothing technique that tends to preserve edges. It also eliminates some isolated spikes and can cause destruction of fine lines. The median filter is implemented by computing the median value of all pixels in a neighborhood around each pixel. An $n \times n$ median filter replaces each pixel of the input image with the median value of the pixels within the $n \times n$ window centered on the pixel being evaluated. The output region of support is determined by the size of *dest*.

Prototypes

```
void vsip_imedfilt_bit(  
    const vsip_iview_bit *src,  
        vsip_scalar_i n,  
    const vsip_iview_bit *dest);
```

```
void (vsip_imedfilt_byte (  
    const vsip_iview_byte *src,  
        vsip_scalar_i n,  
    const vsip_iview_byte *dest)
```

```
void vsip_imedfilt_i(  
    const vsip_iview_i *src,  
        vsip_scalar_i n,  
    const vsip_iview_i *dest);
```

```
void vsip_imedfilt_f(  
    const vsip_iview_f *src,  
        vsip_scalar_i n,  
    const vsip_iview_f *dest);
```

Arguments

<i>*src</i>	View of input image object
<i>n</i>	Width and height of window.
<i>*dest</i>	View of output image object

Return Value

Void

Restrictions

The width and height of the window must be odd and greater than 1. The difference between the sizes of the input and output objects must be 0, or $\pm(n-1)/2$.

Errors/Exceptions**Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

Convolution Operators

vsip_iconv_p

Convolution

Perform a linear convolution with a $p \times p$ kernel.

Functionality

This function performs a linear convolution over the input image with a $p \times p$ kernel. This function generates an output image that is only valid over the kernel's region of support.

$$dest(i, j) \leftarrow \sum_{m=0}^{p-1} \sum_{n=0}^{p-1} k(p-1-m, p-1-n) \cdot src(i-(p-1)/2+m, j-(p-1)/2+n),$$

$$k = \begin{bmatrix} k_{0,0} & \Lambda & k_{0,p-1} \\ \mathbf{M} & \mathbf{O} & \mathbf{M} \\ k_{p-1,0} & \Lambda & k_{p-1,p-1} \end{bmatrix}$$

where $dest$ is the output image, src is the input image, and k is an $p \times p$ convolution kernel. The output region of support is determined by the size of $dest$.

Prototypes

```
void vsip_iconv_bit(
    const vsip_iview_bit *src,
    const vsip_iview_i *k,
    const vsip_iview_i *dest);

void vsip_iconv_byte(
    const vsip_iview_byte *src,
    const vsip_iview_byte *k,
    const vsip_iview_byte *dest);

void vsip_iconv_i(
    const vsip_iview_i *src,
    const vsip_iview_i *k,
    const vsip_iview_i *dest);

void vsip_ciconv_i(
    const vsip_ciview_i *src,
    const vsip_ciview_i *k,
    const vsip_ciview_i *dest);

void vsip_iconv_f(
    const vsip_iview_f *src,
    const vsip_iview_f *k,
    const vsip_iview_f *dest);
```

```
void vsip_ciconv_f(  
    const vsip_ciview_f *src,  
    const vsip_ciview_f *k,  
    const vsip_ciview_f *dest);
```

Arguments

*src View of input image object
*k View of the *pxp* kernel object
*dest View of output image object

Return Value

Void

Restrictions

The kernel must be either 3x3, 5x5, or 7x7. The difference between the sizes of the input and output objects must be 0 or $\pm(p-1)/2$. TBD if strided data is supported.

Errors/Exceptions

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

Discrete Difference Operators

`vsip_isobel_p`

Sobel Gradient Edge Detection

Apply the Sobel gradient edge detection.

Functionality

Gradient operators are a collection of functions that measure the gradient of the image in two orthogonal directions. This function computes the magnitude of the gradient at each pixel using the Sobel operator.

$$A_{i,j} = src(i+1, j-1) + 2src(i+1, j) + src(i+1, j+1) - src(i-1, j-1) - 2src(i-1, j) - src(i-1, j+1)$$

$$B_{i,j} = src(i-1, j-1) + 2src(i, j-1) + src(i+1, j-1) - src(i-1, j+1) - 2src(i, j+1) - src(i+1, j+1)$$

$$dest_{i,j} \leftarrow \sqrt{A * A + B * B}$$

where * denotes complex conjugate. The output region of support is determined by the size of *dest*.

Prototypes

```
void vsip_isobel_bit(
    const vsip_iview_bit *src,
    const vsip_iview_byte *dest);

void vsip_isobel_byte(
    const vsip_iview_byte *src,
    const vsip_iview_byte *dest);

void vsip_isobel_i (
    const vsip_iview_i *src,
    const vsip_iview_i *dest);

void vsip_cisobel_i (
    const vsip_ciview_i *src,
    const vsip_ciview_i *dest);

void vsip_isobel_f (
    const vsip_iview_f *src,
    const vsip_iview_f *dest);

void vsip_cisobel_f (
    const vsip_ciview_f *src,
    const vsip_ciview_f *dest);
```

Arguments

*src View of input image object
*dest View of output image object

Return Value

Void

Restrictions

The difference between the sizes of the input and output objects must be 0 or ± 2 . TBD if strided data is supported.

Errors/Exceptions**Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

`vsip_iroberts_p`, `vsip_iprewitt_p`

vsip_iroberts_p**Roberts Gradient Edge Detection**

Apply the Roberts gradient edge detection.

Functionality

Gradient operators are a collection of functions that measure the gradient of the image in two orthogonal directions. This function computes the magnitude of the gradient at each pixel using the Roberts operator.

$$A_{i,j} = src(i, j) + src(i+1, j+1)$$

$$B_{i,j} = src(i, j) + src(i-1, j+1)$$

$$dest_{i,j} \leftarrow \sqrt{A * A + B * B}$$

where * denotes complex conjugate. The output region of support is determined by the size of *dest*.

Prototypes

```
void vsip_iroberts_bit (
    const vsip_iview_bit *src,
    const vsip_iview_byte *dest);
```

```
void vsip_iroberts_byte (
    const vsip_iview_byte *src,
    const vsip_iview_byte *dest);
```

```
void vsip_iroberts_i (
    const vsip_iview_i *src,
    const vsip_iview_i *dest);
```

```
void vsip_ciroberts_i (
    const vsip_ciview_i *src,
    const vsip_ciview_i *dest);
```

```
void vsip_iroberts_f (
    const vsip_iview_f *src,
    const vsip_iview_f *dest);
```

```
void vsip_ciroberts_f (
    const vsip_ciview_f *src,
    const vsip_ciview_f *dest);
```

Arguments

*src View of input image object
*dest View of output image object

Return Value

Void

Restrictions

The difference between the sizes of the input and output objects must be 0 or ± 1 . TBD if strided data is supported.

Errors/Exceptions**Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

`vsip_isobel_p`, `vsip_iprewitt_p`

vsip_iprewitt_p**Prewitt Gradient Edge Detection**

Apply the Prewitt gradient edge detection.

Functionality

Gradient operators are a collection of functions that measure the gradient of the image in two orthogonal directions. This function computes the magnitude of the gradient at each pixel using the Prewitt operator.

$$A_{i,j} = src(i+1, j-1) + src(i+1, j) + src(i+1, j+1) - src(i-1, j-1) - src(i-1, j) - src(i-1, j+1)$$

$$B_{i,j} = src(i-1, j-1) + src(i, j-1) + src(i+1, j-1) - src(i-1, j+1) - src(i, j+1) - src(i+1, j+1)$$

$$dest_{i,j} \leftarrow \sqrt{A * A + B * B}$$

where * denotes complex conjugate. The output region of support is determined by the size of *dest*.

Prototypes

```
void vsip_iprewitt_bit (
    const vsip_iview_bit *src,
    const vsip_iview_i  *dest);

void vsip_iprewitt_byte (
    const vsip_iview_byte *src,
    const vsip_iview_byte *dest);

void vsip_iprewitt_i (
    const vsip_iview_i *src,
    const vsip_iview_i *dest);

void vsip_ciprewitt_i (
    const vsip_ciview_i *src,
    const vsip_ciview_i *dest);

void vsip_iprewitt_f (
    const vsip_iview_f *src,
    const vsip_iview_f *dest);

void vsip_ciprewitt_f (
    const vsip_ciview_f *src,
    const vsip_ciview_f *dest);
```

Arguments

*src View of input image object
*dest View of output image object

Return Value

Void

Restrictions

The difference between the sizes of the input and output objects must be 0 or ± 2 . TBD if strided data is supported.

Errors/Exceptions**Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

vsip_isobel_p, *vsip_iroberts_p*

Morphological Operators

vsip_idilate_p

Grayscale Morphological Dilation

Morphological dilation of a grayscale image.

Functionality

Morphological dilation of an image src by a $p \times p$ structuring element k involves reflecting src about its origin and shifting this reflected image over k . The value of the operation at the pixel given by the position of k 's origin in src is given by the maximum value of the sum of src and k within the overlapping region of the shifted and reflected image src and the structuring element k . The dilation of an image tends to enhance brightness of an image and reduce or eliminate darker image objects.

$$dest(i, j) \leftarrow src(i - (p-1)/2 + m, j - (p-1)/2 + n) \\ \exists \max_{i, j} \{src(i - (p-1)/2 + m, j - (p-1)/2 + n) + k(m, n), k(m, n) \neq 0\}$$

where D_i is the region of support of src , and D_k is the region of support of k . The output region of support is determined by $dest$.

Prototypes

```
void vsip_idilate_bit(
    const vsip_iview_bit *src,
    const vsip_iview_bit *k,
    const vsip_iview_bit *dest);

void vsip_idilate_byte(
    const vsip_iview_byte *src,
    const vsip_iview_byte *k,
    const vsip_iview_byte *dest);

void vsip_idilate_i(
    const vsip_iview_i *src,
    const vsip_iview_i *k,
    const vsip_iview_i *dest);

void vsip_cidilate_i(
    const vsip_ciview_i *src,
    const vsip_ciview_i *k,
    const vsip_ciview_i *dest);

void vsip_idilate_f(
    const vsip_iview_f *src,
    const vsip_iview_f *k,
    const vsip_iview_f *dest);
```

```
void vsip_cidilate_f(  
    const vsip_ciview_f *src,  
    const vsip_ciview_f *k,  
    const vsip_ciview_f *dest);
```

Arguments

*src	View of input image object
*k	View of input structuring element object.
*dest	View of output image object

Return Value

Void

Restrictions

The width and height of k must be odd and greater than 1. The difference between the sizes of the input and output objects must be 0 or $(p-1)/2$. Some value of the structuring element must be nonzero including boundary cases where not all the structuring element is employed. TBD if strided data is supported.

Errors/Exceptions

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

vsip_ierode_p**Grayscale Morphological Erosion**

Morphological erosion of a grayscale image.

Functionality

Morphological erosion of an image src by a $p \times p$ structuring element k involves shifting the image src over the structuring element k . The value of the operation at the pixel location given by the position of k 's origin in src is given by the minimum value of the difference of the shifted src and k within the overlapping region. Erosion tends to darken images and reduce or eliminate bright objects within an image.

$$dest(i, j) \leftarrow src(i - (p-1)/2 + m, j - (p-1)/2 + n)$$

$$\exists \min_{i, j} \{src(i - (p-1)/2 + m, j - (p-1)/2 + n) - k(m, n), k(m, n) \neq 0\}$$

where D_i is the domain of i and D_k is the domain of k . The output region of support is determined by $dest$.

Prototypes

```
void vsip_ierode_bit(
    const vsip_iview_bit *src,
    const vsip_iview_bit *k,
    const vsip_iview_bit *dest);
```

```
void vsip_ierode_byte(
    const vsip_iview_byte *src,
    const vsip_iview_byte *k,
    const vsip_iview_byte *dest);
```

```
void vsip_ierode_i(
    const vsip_iview_i *src,
    const vsip_iview_i *k,
    const vsip_iview_i *dest);
```

```
void vsip_cierode_i(
    const vsip_ciview_i *src,
    const vsip_ciview_i *k,
    const vsip_ciview_i *dest);
```

```
void vsip_ierode_f(
    const vsip_iview_f *src,
    const vsip_iview_f *k,
    const vsip_iview_f *dest);
```

```
void vsip_cierode_f(
    const vsip_ciview_f *src,
    const vsip_ciview_f *k,
    const vsip_ciview_f *dest);
```

Arguments

*src View of input image object
*k View of input structuring element object.
*dest View of output image object

Return Value

Void

Restrictions

The width and height of k must be odd and greater than 1. The difference between the sizes of the input and output objects must be 0 or $(p-1)/2$. Some value of the structuring element must be nonzero including boundary cases where not all the structuring element is employed. TBD if strided data is supported.

Errors/Exceptions**Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

vsip_iopen_p

Grayscale Morphological Opening

Morphological opening of a grayscale image.

Functionality

Morphological opening of an image is conceptually a composition of erosion and dilation. Opening is geometrically a contour smoothing operation where small isolated bright objects are reduced or eliminated while larger bright objects and the overall brightness of the image is preserved.

$$dest = dilate(erode(src, k), k)$$

The output region of support is determined by *dest*.

Prototypes

```
void vsip_iopen_bit(  
    const vsip_iview_bit *src,  
    const vsip_iview_bit *k,  
    const vsip_iview_bit *dest);
```

```
void vsip_iopen_byte(  
    const vsip_iview_byte *src,  
    const vsip_iview_byte *k,  
    const vsip_iview_byte *dest);
```

```
void vsip_iopen_i(  
    const vsip_iview_i *src,  
    const vsip_iview_i *k,  
    const vsip_iview_i *dest);
```

```
void vsip_ciopen_i(  
    const vsip_ciview_i *src,  
    const vsip_ciview_i *k,  
    const vsip_ciview_i *dest);
```

```
void vsip_iopen_f(  
    const vsip_iview_f *src,  
    const vsip_iview_f *k,  
    const vsip_iview_f *dest);
```

```
void vsip_ciopen_f(  
    const vsip_ciview_f *src,  
    const vsip_ciview_f *k,  
    const vsip_ciview_f *dest);
```

Arguments

**src* View of input image object

*k View of input structuring element object.
*dest View of output image object

Return Value

Void

Restrictions

The width and height of k must be odd and greater than 1. The difference between the sizes of the input and the output objects must be 0 or $(p-1)/2$, where p is the size of k . Some value of the structuring element must be nonzero including boundary cases where not all the structuring element is employed. TBD if strided data is supported.

Errors/Exceptions

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

vsip_iclose_p**Grayscale Morphological Closing**

Morphological closing of a grayscale image.

Functionality

Morphological closing of an image is conceptually a composition of dilation and erosion. Closing tends to eliminate dark features of an image, preserve bright objects, and retain the overall brightness of the image.

$$dest = erode(dilate(src, k), k)$$

The output region of support is determined by *dest*.

Prototypes

```
void vsip_iclose_bit(  
    const vsip_iview_bit *src,  
    const vsip_iview_bit *k,  
    const vsip_iview_bit *dest);
```

```
void vsip_iclose_byte(  
    const vsip_iview_byte *src,  
    const vsip_iview_byte *k,  
    const vsip_iview_byte *dest);
```

```
void vsip_iclose_i(  
    const vsip_iview_i *src,  
    const vsip_iview_i *k,  
    const vsip_iview_i *dest);
```

```
void vsip_ciclose_i(  
    const vsip_ciview_i *src,  
    const vsip_ciview_i *k,  
    const vsip_ciview_i *dest);
```

```
void vsip_iclose_f(  
    const vsip_iview_f *src,  
    const vsip_iview_f *k,  
    const vsip_iview_f *dest);
```

```
void vsip_ciclose_f(  
    const vsip_ciview_f *src,  
    const vsip_ciview_f *k,  
    const vsip_ciview_f *dest);
```

Arguments

*src View of input image object
*k View of input structuring element object.
*dest View of output image object

Return Value

Void

Restrictions

The width and height of k must be odd and greater than 1. The difference between the sizes of the input and the output objects must be 0 or $(p-1)/2$, where p is the size of k . Some value of the structuring element must be nonzero including boundary cases where not all the structuring element is employed. TBD if strided data is supported.

Errors/Exceptions**Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

Resize Operators

vsip_iinterp_p

Expand Image

Increase the size of an image.

Functionality

This function expands an image by using bilinear interpolation to obtain the new image values

$$\begin{aligned} dest(i, j) \leftarrow & \frac{1}{\Delta m \Delta n} \left[src(m, n) \left(\frac{m+1}{M} - \frac{i}{I} \right) \left(\frac{n+1}{N} - \frac{j}{J} \right) \right. \\ & + src(m+1, n) \left(\frac{i}{I} - \frac{m}{M} \right) \left(\frac{n+1}{N} - \frac{j}{J} \right) \\ & + src(m+1, n+1) \left(\frac{i}{I} - \frac{m}{M} \right) \left(\frac{j}{J} - \frac{n}{N} \right) \\ & \left. + src(m, n+1) \left(\frac{m+1}{M} - \frac{i}{I} \right) \left(\frac{j}{J} - \frac{n}{N} \right) \right], \\ & m \leq i \leq m+1, n \leq j \leq n+1 \end{aligned}$$

where $\Delta m = 1/m$ and $\Delta n = 1/n$.

Prototypes

```
Void vsip_iinterp_bit(
    const vsip_iview_bit *src,
    const vsip_iview_bit *dest);
```

```
Void vsip_iinterp_byte(
    const vsip_iview_byte *src,
    const vsip_iview_byte *dest);
```

```
Void vsip_iinterp_i(
    const vsip_iview_i *src,
    const vsip_iview_i *dest);
```

```
Void vsip_ciinterp_i(
    const vsip_ciview_i *src,
    const vsip_ciview_i *dest);
```

```
Void vsip_iinterp_f(  
    const vsip_iview_f *src,  
    const vsip_iview_f *dest);  
  
Void vsip_ciinterp_f(  
    const vsip_ciview_f *src,  
    const vsip_ciview_f *dest);
```

Arguments

*src View of input image object
*dest View of output image object

Return Value

Void

Restrictions**Errors/Exceptions****Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

vsip_iresample_p

Shrink Image

Reduce the size of an image.

Functionality

This function reduces the size of an image

$$dest(i, j) \leftarrow resample_method(src, (m, n), i-1 \leq m \leq i+1, j-1 \leq n \leq j+1)$$

where *resample_method* can be a maximum, minimum, average or median.

Prototypes

```
Void vsip_iresample_bit(  
    const vsip_iview_bit *src,  
    vsip_resample method,  
    const vsip_iview_bit *dest);
```

```
Void vsip_iresample_byte(  
    const vsip_iview_byte *src,  
    vsip_resample method,  
    const vsip_iview_byte *dest);
```

```
Void vsip_iresample_i(  
    const vsip_iview_i *src,  
    vsip_resample method,  
    const vsip_iview_i *dest);
```

```
Void vsip_ciresample_i(  
    const vsip_ciview_i *src,  
    vsip_resample method,  
    const vsip_ciview_i *dest);
```

```
Void vsip_iresample_f(  
    const vsip_iview_f *src,  
    vsip_resample method,  
    const vsip_iview_f *dest);
```

```
Void vsip_ciresample_f(  
    const vsip_ciview_f *src,  
    vsip_resample method,  
    const vsip_ciview_f *dest);
```

Arguments

*src View of input image object

method Resample method

```
typedef enum {  
    VSIP_RESAMPLE_MIN=0,  
    VSIP_RESAMPLE_MAX=1,  
    VSIP_RESAMPLE_AVERAGE=2,  
    VSIP_RESAMPLE_MEDIAN=3  
} vsip_resample;
```

*dest View of output image object

Return Value

Void

Restrictions

Errors/Exceptions

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

Pad Operators

`vsip_ipadconst_p`

Pad Image with Constant

Pad an image with a constant.

Functionality

This function is used in conjunction with the convolution operators to create a border around an input image to produce the desired statistics for the convolution. The kernel object's size is used to determine the size of the border of the image.

$$dest(m,n) \leftarrow \begin{cases} src(mn)(m,n) \in D_i \\ c & (m,n) \notin D_i \end{cases}$$

where D_i is the domain of i , c is a scalar constant.

Prototypes

```
void vsip_ipadconst_bit(
    const vsip_iview_bit *src,
    const vsip_iview_bit *k,
    vsip_scalar_bit c,
    const vsip_iview_bit *dest);
```

```
void vsip_ipadconst_byte(
    const vsip_iview_byte *src,
    const vsip_iview_byte *k,
    vsip_scalar_byte c,
    const vsip_iview_byte *dest);
```

```
void vsip_ipadconst_i(
    const vsip_iview_i *src,
    const vsip_iview_i *k,
    vsip_scalar_i c,
    const vsip_iview_i *dest);
```

```
void vsip_cipadconst_i(
    const vsip_ciview_i *src,
    const vsip_ciview_i *k,
    vsip_cscalar_i c,
    const vsip_ciview_i *dest);
```

```
void vsip_ipadconst_f(  
    const vsip_iview_f *src,  
    const vsip_iview_f *k,  
    vsip_scalar_f c,  
    const vsip_iview_f *dest);  
  
void vsip_cipadconst_f(  
    const vsip_ciview_f *src,  
    const vsip_ciview_f *k,  
    vsip_cscalar_f c,  
    const vsip_ciview_f *dest);
```

Arguments

<code>*src</code>	View of input image object
<code>*k</code>	View of input kernel object
<code>c</code>	Constant value of pad
<code>*dest</code>	View of output image object

Return Value

Void

Restrictions

Errors/Exceptions

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

`vsip_ipadreflect_p`, `vsip_ipadextend_p`, `vsip_iconv_p`

vsip_ipadreflect_p**Pad Image by Reflection**

Pad an image by reflection.

Functionality

This function is used in conjunction with the convolution operators. It borders an input image to produce the desired statistics for the convolution. The kernel object's size is used to determine the size of the border of the image. The boundary of the original image is reflected along the edge of the image to the width indicated by the kernel.

$$dest(i, j) \leftarrow src(m, n)$$

where:

$$m = \begin{cases} I - |i| \bmod I, & \left\lfloor \frac{i}{I} \right\rfloor \bmod 2 = 0 \\ |i| \bmod I, & \left\lfloor \frac{i}{I} \right\rfloor \bmod 2 = 1 \end{cases}$$

$$n = \begin{cases} J - |j| \bmod J, & \left\lfloor \frac{j}{J} \right\rfloor \bmod 2 = 0 \\ |j| \bmod J, & \left\lfloor \frac{j}{J} \right\rfloor \bmod 2 = 1 \end{cases}$$

Prototypes

```
void vsip_ipadreflect_bit(
    const vsip_iview_bit *src,
    const vsip_iview_bit *k,
    const vsip_iview_bit *dest);

void vsip_ipadreflect_byte(
    const vsip_iview_byte *src,
    const vsip_iview_byte *k,
    const vsip_iview_byte *dest);

void vsip_ipadreflect_i(
    const vsip_iview_i *src,
    const vsip_iview_i *k,
    const vsip_iview_i *dest);
```

```
void vsip_cipadreflect_i(
    const vsip_ciview_i *src,
    const vsip_ciview_i *k,
    const vsip_ciview_i *dest);

void vsip_ipadreflect_f(
    const vsip_iview_f *src,
    const vsip_iview_f *k,
    const vsip_iview_f *dest);

void vsip_cipadreflect_f(
    const vsip_ciview_f *src,
    const vsip_ciview_f *k,
    const vsip_ciview_f *dest);
```

Arguments

*src	View of input image object
*k	View of input kernel object
*dest	View of output image object

Return Value

Void

Restrictions

Errors/Exceptions

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

`vsip_ipadconst_p`, `vsip_ipadextend_p`, `vsip_iconv_p`

vsip_ipadextend_p

Pad Image by Extension

Pad an image by extension.

Functionality

This function is used in conjunction with the convolution operators to create a border around an input image to produce the desired statistics for the convolution. The kernel object's size is used to determine the size of the border of the image. The boundary of the image is extended around the edge of the image by taking the edge pixel value and repeating as indicated by the kernel object.

Prototypes

```
void vsip_ipadextend_bit(  
    const vsip_iview_bit *src,  
    const vsip_iview_bit *k,  
    const vsip_iview_bit *dest);  
  
void vsip_ipadextend_byte(  
    const vsip_iview_byte *src,  
    const vsip_iview_byte *k,  
    const vsip_iview_byte *dest);  
  
void vsip_ipadextend_i(  
    const vsip_iview_i *src,  
    const vsip_iview_i *k,  
    const vsip_iview_i *dest);  
  
void vsip_cipadextend_i(  
    const vsip_ciview_i *src,  
    const vsip_ciview_i *k,  
    const vsip_ciview_i *dest);  
  
void vsip_ipadextend_f(  
    const vsip_iview_f *src,  
    const vsip_iview_f *k,  
    const vsip_iview_f *dest);  
  
void vsip_cipadextend_f(  
    const vsip_ciview_f *src,  
    const vsip_ciview_f *k,  
    const vsip_ciview_f *dest);
```

Arguments

*src View of input image object
*k View of input kernel object
*dest View of output image object

Return Value

Void

Restrictions**Errors/Exceptions****Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

`vsip_ipadconst_p`, `vsip_ipadreflect_p`, `vsip_iconv_p`

Other Functions

`vsip_istats_p`

Compute Image Statistics

Compute the statistics for an image.

Functionality

Image statistics are used in a wide range of image processing operations such as image enhancement and feature extraction. This function computes many of the image statistics that are commonly used in image processing applications. The statistics that are calculated can be controlled by ORing a value associated with each statistic into the mask argument to this function. The mask values are as follows:

The resulting vector will contain all statistics that have been indicated by the mask argument. Regardless of which statistics are chosen, the output vector will be ordered as indicated in the table.

Mean

$$\mu \leftarrow \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} src(m, n)$$

Variance

$$\sigma \leftarrow \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} [src(m, n) - \mu]^2$$

Root Mean Square

$$rms \leftarrow \sqrt{\frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} src(m, n)^2}$$

Skewness

$$skew \leftarrow \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \left[\frac{src(m, n) - \mu}{\sigma} \right]^3$$

Kurtosis

$$kurt \leftarrow \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \left[\frac{src(m, n) - \mu}{\sigma} \right]^4 - 3$$

Integral

$$I \leftarrow \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} src(m, n)$$

Positive Integral

$$P \leftarrow \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} src(m, n), src(m, n) > 0$$

Negative Integral	$Q \leftarrow \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} src(m,n), src(m,n) < 0$
Minimum	$\min\{src(m,n), 0 \leq m < M, 0 \leq n < N\}$
Maximum	$\max\{src(m,n), 0 \leq m < M, 0 \leq n < N\}$
Median Value	$median\{src(m,n), 0 \leq m < M, 0 \leq n < N\}$
Number of Zeros	$n_{zeros} \leftarrow \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} 1, src(m,n) = 0$

Prototypes

```

void vsip_istats_bit(
    const vsip_iview_bit    *src,
    vsip_image_stats        stats,
    const vsip_vview_f      *v);

void vsip_istats_byte(
    const vsip_iview_byte   *src,
    vsip_image_stats        stats,
    const vsip_vview_f      *v);

void vsip_istats_i(
    const vsip_iview_i      *src,
    vsip_image_stats        stats,
    const vsip_vview_f      *v);

void vsip_cistats_i(
    const vsip_ciview_i     *src,
    vsip_image_stats        stats,
    const vsip_cvview_f     *v);

void vsip_istats_f(
    const vsip_iview_f      *src,
    vsip_image_stats        stats,
    const vsip_vview_f      *v);

void vsip_cistats_f(
    const vsip_ciview_f     *src,
    vsip_image_stats        stats,
    const vsip_cvview_f     *v);

```

Arguments

`*src` View of input image object
`stats` The mask of desired statistics

```
typedef enum{  
    /* mean 0x0001 */  
    VSIP_MEAN = 1<<0,  
    /* variance 0x0002 */  
    VSIP_VAR = 1<<1,  
    /* root mean square 0x0004 */  
    VSIP_RMS = 1<<2,  
    /* skewness 0x0008 */  
    VSIP_SKEW = 1<<3,  
    /* kurtosis 0x0010 */  
    VSIP_KURT = 1<<4,  
    /* integral 0x0020 */  
    VSIP_INTGRL = 1<<5,  
    /* positive integral 0x0040 */  
    VSIP_PINTG = 1<<6,  
    /* negative integral 0x0080 */  
    VSIP_NINTG = 1<<7,  
    /* minimum 0x0100 */  
    VSIP_MIN = 1<<8,  
    /* maximum 0x0200 */  
    VSIP_MAX = 1<<9,  
    /* median value 0x0400 */  
    VSIP_MED = 1<<10,  
    /* number of zeros 0x0800 */  
    VSIP_NZERO = 1<<11,  
} vsip_image_stats;
```

`*v` View of output vector object containing desired statistics

Return Value

Void

Restrictions

Errors/Exceptions

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

vsip_idct8x8_p**8x8 Discrete Cosine Transform**

Transform an 8x8 cosine transform.

Functionality

The Discrete Cosine Transform (DCT) is a linear transform that has near optimal energy compaction characteristics. For this reason it is frequently used as part of lossy compression algorithms. This function partitions the input image into 8×8 tiles and performs an 8×8 DCT on each tile and returns the resulting image. The result image can then be post processed by selecting a subset of the coefficients in each tile to implement lossy compression.

Prototypes

```
void vsip_idct8x8_bit(
    const vsip_iview_bit *src,
    const vsip_iview_bit *dest);

void vsip_idct8x8_byte(
    const vsip_iview_byte *src,
    const vsip_iview_byte *dest);

void vsip_idct8x8_i(
    const vsip_iview_i *src,
    const vsip_iview_i *dest);

void vsip_cidct8x8_i(
    const vsip_ciview_i *src,
    const vsip_ciview_i *dest);

void vsip_idct8x8_f(
    const vsip_iview_f *src,
    const vsip_iview_f *dest);

void vsip_cidct8x8_f(
    const vsip_ciview_f *src,
    const vsip_ciview_f *dest);
```

Arguments

<i>*src</i>	View of input image object
<i>*dest</i>	View of output image object

Return Value

Void

Restrictions

Errors/Exceptions

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

vsip_ilut_p**Compute Image using Look Up Table****Functionality****Prototypes**

```
void vsip_ilut_byte(  
    const vsip_iview_byte *src,  
    const vsip_vview_byte *l,  
    const vsip_iview_byte *dest);
```

```
void vsip_ilut_i(  
    const vsip_iview_i *src,  
    const vsip_vview_i *l,  
    const vsip_iview_i *dest);
```

Arguments

*src	View of input image object
*l	View of input lookup table object
*dest	View of output image object

Return Value

Void

Restrictions**Errors/Exceptions****Notes/References**

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples**See Also**

vsip_ilabel_p**Compute Connected Components Image**

Compute the connected components of an image.

Functionality

Connected components labeling is an image segmentation technique in which a binary image is replaced with an image in which each object indicated by a connected cluster of pixels is labeled with a unique pixel value.

Two connectivity rules are allowed: 4-neighbor and 8-neighbor. The four neighbor rule defines cluster membership as all pixels with the same value above, below, to the left, and to the right of the current pixel. The eight neighbor rule adds the four diagonal pixels.

Prototypes

```
void vsip_ilabel_bit(  
    const vsip_iview_bit    *src,  
    vsip_neighborhood      n,  
    const vsip_iview_i      *dest);
```

```
void vsip_ilabel_byte(  
    const vsip_iview_byte    *i,  
    vsip_neighborhood      n,  
    const vsip_iview_byte    *o);
```

Arguments

`*src` View of input image object

`n` Neighborhood

```
typedef enum {  
    VSIP_4NEIGHBOR = 0,  
    VSIP_8NEIGHBOR = 1,  
} vsip_neighborhood;
```

`*dest` 32-bit integer view of output image object

Return Value

Void

Restrictions

This size of the input image is limited to that which the output can handle with a 32-bit integer.

Errors/Exceptions

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

vsip_ibind_p

Create and Bind an Image View

Create an image view object and bind it to a block object.

Functionality

This function attempts to create an image view object and returns NULL if it fails. Otherwise, it binds the image view object to the block object; sets the *offset* from the beginning of the data array to the beginning of the image; the number, *col_length*, of scalar elements in a column; the stride, *col_stride*, between scalar elements in a column; the number, *row_length*, of scalar elements in a row; and the stride, *row_stride*, between scalar elements in a row and returns a pointer to the created image view object.

Prototypes

```
vsip_iview_bit *vsip_ibind_bit(  
    const vsip_block_bit *block,  
    vsip_offset          offset,  
    vsip_stride          col_stride,  
    vsip_length          col_length,  
    vsip_stride          row_stride,  
    vsip_length          row_length);
```

```
vsip_iview_byte *vsip_ibind_byte(  
    const vsip_block_byte *block,  
    vsip_offset          offset,  
    vsip_stride          col_stride,  
    vsip_length          col_length,  
    vsip_stride          row_stride,  
    vsip_length          row_length);
```

```
vsip_iview_i *vsip_ibind_i(  
    const vsip_block_i *block,  
    vsip_offset          offset,  
    vsip_stride          col_stride,  
    vsip_length          col_length,  
    vsip_stride          row_stride,  
    vsip_length          row_length);
```

```
vsip_ciview_i *vsip_cibind_i(  
    const vsip_block_i *block,  
    vsip_offset          offset,  
    vsip_stride          col_stride,  
    vsip_length          col_length,  
    vsip_stride          row_stride,  
    vsip_length          row_length);
```

```

vsip_iview_f *vsip_ibind_f(
    const vsip_block_f  *block,
    vsip_offset         offset,
    vsip_stride         col_stride,
    vsip_length         col_length,
    vsip_stride         row_stride,
    vsip_length         row_length);

vsip_ciview_f *vsip_cibind_f(
    const vsip_block_f  *block,
    vsip_offset         offset,
    vsip_stride         col_stride,
    vsip_length         col_length,
    vsip_stride         row_stride,
    vsip_length         row_length);

```

Arguments

`*block` Block object to which to bind the image object

`offset` Image view offset relative to base of block object

`col_stride` Image view `col_stride` relative to base of block object

`col_length` Image view `col_length` relative to base of block object

`row_stride` Image view `row_stride` relative to base of block object

`row_length` Image view `row_length` relative to base of block object

Return Value

Returns a pointer to the created image view object, or NULL if the memory allocation for the new object fails.

Restrictions

Errors/Exceptions

Note: It is important for the application to check the return value for a memory allocation failure. In *development mode*, an informative error message should be issued to the application programmer: 1) if the block is invalid, or 2) unless

$$0 \leq \textit{offset} < N$$

$$0 \leq \textit{offset} < (\textit{row_length} - 1) \cdot \textit{row_stride} < N$$

$$0 \leq \textit{offset} < (\textit{col_length} - 1) \cdot \textit{col_stride} < N, \textit{ and}$$

$$0 \leq \textit{offset} < (\textit{row_length} - 1) \cdot \textit{row_stride} + (\textit{col_length} - 1) \cdot \textit{col_stride} < N$$

where N is the number of element of the bound block object.

Notes/References

R.C. Gonzalez and R. C. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

Examples

See Also

vsip_ikcreate_p**Create Image**

Creates a block object and an associated bound simple m by n image view object.

Functionality

The function `vsip_iview_p *vsip_ikcreate_p(m,n,VSIP_ROW,hint)`; returns the same result as

```
vsip_ibind(vsip_blockcreate(m*n, hint),
           (vsip_offset) 0,
           (vsip_stride) n,
           (vsip_length) m,
           (vsip_stride) 1,
           (vsip_length) n);
```

or `vsip_iview_p *vsip_ikcreate_p(m,n,VSIP_COL,hint)`; returns the same result as

```
vsip_ibind(vsip_blockcreate(m*n, hint),
           (vsip_offset) 0,
           (vsip_stride) 1,
           (vsip_length) m,
           (vsip_stride) m,
           (vsip_length) n);
```

except that `vsip_iview_p` returns NULL if the `vsip_block_create(m*n, hint)` returns NULL.

Prototypes

```
vsip_iview_bit *vsip_ikcreate_bit(
    vsip_length    m,
    vsip_length    n,
    vsip_major     major,
    vsip_memory_hint hint);
```

```
vsip_iview_byte *vsip_ikcreate_byte(
    vsip_length    m,
    vsip_length    n,
    vsip_major     major,
    vsip_memory_hint hint);
```

```
vsip_iview_i *vsip_ikcreate_i(
    vsip_length    m,
    vsip_length    n,
    vsip_major     major,
    vsip_memory_hint hint);
```

```
vsip_ciview_i *vsip_cikcreate_i(
    vsip_length    m,
    vsip_length    n,
```

```
        vsip_major      major,  
        vsip_memory_hint hint);  
  
vsip_ciview_f *vsip_cicreate_f(  
    vsip_length      m,  
    vsip_length      n,  
    vsip_major      major,  
    vsip_memory_hint hint);
```

Arguments

`m` Number of elements (`col_length`) per column image

`n` Number of elements (`row_length`) per row of image

`major` Row or column major

```
typedef enum {  
    VSIP_ROW=0,  
    VSIP_COL=1  
} vsip_major;
```

`hint` TBD memory type hint

Return Value

Returns a pointer to the created image view object, or NULL if it fails.

Restrictions

Errors/Exceptions

In *development mode*, the function should issue an error message to the application programmer if it 1) fails to allocate memory for the private array, or 2) if $m = 0$ or $n = 0$. In *development mode*, the function should issue a warning message to the application programmer if the hint is invalid.

Notes/References

In *development mode*, the function should increment the number of bindings (reference count) recorded in the block object.

Examples

See Also

vsip_idestroy_p**Destroy Image View**

Destroy (free) an image view object and return a pointer to the associated block.

Functionality

This function frees an image view object, the block object to which it was bound, destroys the image view object, and then returns a pointer to the block object. If the image view argument is NULL, it returns NULL.

Prototypes

```
vsip_block_bit *vsip_idestroy_bit(  
    vsip_iview_bit *image);  
  
vsip_block_byte *vsip_idestroy_byte(  
    vsip_iview_byte *image);  
  
vsip_block_i *vsip_idestroy_i(  
    vsip_iview_i *image);  
  
vsip_block_i *vsip_cidestroy_i(  
    vsip_ciview_i *image);  
  
vsip_block_f *vsip_idestroy_f(  
    vsip_iview_f *image);  
  
vsip_block_f *vsip_cidestroy_f(  
    vsip_ciview_f *image);
```

Arguments

*image Image view object

Return Value

(Reference to) block object to which the image view was bound, or NULL if the calling argument was NULL.

Restrictions**Errors/Exceptions**

In *development mode*, the function should issue an informative error message to the application programmer if the image view object is invalid. An argument of NULL is not an error.

Notes/References

In *development mode*, the function should increment the number of bindings recorded in the block object.

Examples**See Also**

vsip_igetoffset_p

Image Get Offset

Get the *offset* attribute of an image view object.

Functionality

This function returns the value of the *offset* attribute of the image view object. (Offset is in simple scalar elements relative to the start of the data array of the block object.)

Prototypes

```
vsip_offset vsip_igetoffset_bit(  
    const vsip_iview_bit *image);,  
  
vsip_offset vsip_igetoffset_byte(  
    const vsip_iview_byte *image);,  
  
vsip_offset vsip_igetoffset_i(  
    const vsip_iview_i *image);,  
  
vsip_offset vsip_cigetoffset_i(  
    const vsip_ciview_i *image);,  
  
vsip_offset vsip_igetoffset_f(  
    const vsip_iview_f *image);,  
  
vsip_offset vsip_cigetoffset_f(  
    const vsip_ciview_f *image);,
```

Arguments

*image Image view object

Return Value

Returns the value of the *offset* attribute of the image view object.

Restrictions

Errors/Exceptions

In *development mode*, the function should issue an informative error message to the application programmer if the image view object is invalid.

Notes/References

Examples

See Also

vsip_igetrowstride_p**Image Get Row_Stride**

Get the *row_stride* attribute of an image view object.

Functionality

This function returns the stride between scalar elements in a row of an image view object.

Prototypes

```
vsip_stride vsip_igetrowstride_bit(  
    const vsip_iview_bit *image);,  
  
vsip_stride vsip_igetrowstride_byte(  
    const vsip_iview_byte *image);,  
  
vsip_stride vsip_igetrowstride_i(  
    const vsip_iview_i *image);,  
  
vsip_stride vsip_cigetrowstride_i(  
    const vsip_ciview_i *image);,  
  
vsip_stride vsip_igetrowstride_f(  
    const vsip_iview_f *image);,  
  
vsip_stride vsip_cigetrowstride_f(  
    const vsip_ciview_f *image);,
```

Arguments

*image Image view object

Return Value

Returns the value of the *row_stride* attribute of the image view object.

Restrictions**Errors/Exceptions**

In *development mode*, the function should issue an informative error message to the application programmer if the image view object is invalid.

Notes/References

Examples

See Also

vsip_igetcolstride_p

Image Get Col_Stride

Get the *col_stride* attribute of an image view object.

Functionality

This function returns the stride between scalar elements in a column of an image view object.

Prototypes

```
vsip_stride vsip_igetcolstride_bit(  
    const vsip_iview_bit *image);,  
  
vsip_stride vsip_igetcolstride_byte(  
    const vsip_iview_byte *image);,  
  
vsip_stride vsip_igetcolstride_i(  
    const vsip_iview_i *image);,  
  
vsip_stride vsip_cigetcolstride_i(  
    const vsip_ciview_i *image);,  
  
vsip_stride vsip_igetcolstride_f(  
    const vsip_iview_f *image);,  
  
vsip_stride vsip_cigetcolstride_f(  
    const vsip_ciview_f *image);,
```

Arguments

*image Image view object

Return Value

Returns the value of the *col_stride* attribute of the image view object.

Restrictions

Errors/Exceptions

In *development mode*, the function should issue an informative error message to the application programmer if the image view object is invalid.

Notes/References

Examples

See Also

vsip_igetrowlength_p**Image Get Row_Length**

Get the *row_length* attribute of an image view object.

Functionality

This function returns the number of columns (width) of an image view object.

Prototypes

```
vsip_length vsip_igetrowlength_bit(  
    const vsip_iview_bit *image);,  
  
vsip_length vsip_igetrowlength_byte(  
    const vsip_iview_byte *image);,  
  
vsip_length vsip_igetrowlength_i(  
    const vsip_iview_i *image);,  
  
vsip_length vsip_cigetrowlength_i(  
    const vsip_ciview_i *image);,  
  
vsip_length vsip_igetrowlength_f(  
    const vsip_iview_f *image);,  
  
vsip_length vsip_cigetrowlength_f(  
    const vsip_ciview_f *image);,
```

Arguments

*image Image view object

Return Value

Returns the value of the *row_length* attribute of the image view object.

Restrictions**Errors/Exceptions**

In *development mode*, the function should issue an informative error message to the application programmer if the image view object is invalid.

Notes/References

Examples

See Also

vsip_igetcollength_p**Image Get Col_Length**

Get the *col_length* attribute of an image view object.

Functionality

This function returns the number of rows (height) of an image view object.

Prototypes

```
vsip_length vsip_igetcollength_bit(  
    const vsip_iview_bit *image);,  
  
vsip_length vsip_igetcollength_byte(  
    const vsip_iview_byte *image);,  
  
vsip_length vsip_igetcollength_i(  
    const vsip_iview_i *image);,  
  
vsip_length vsip_cigetcollength_i(  
    const vsip_ciview_i *image);,  
  
vsip_length vsip_igetcollength_f(  
    const vsip_iview_f *image);,  
  
vsip_length vsip_cigetcollength_f(  
    const vsip_ciview_f *image);,
```

Arguments

*image Image view object

Return Value

Returns the value of the *col_length* attribute of the image view object.

Restrictions**Errors/Exceptions**

In *development mode*, the function should issue an informative error message to the application programmer if the image view object is invalid.

Notes/References

Examples

See Also