# vImage Reference Collection

**Mathematical Computation**

**2007-07-12**

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010



Simultaneously published in the United States and Canada.

# Contents

# Figures

# Introduction

| | |
|---|---|
| **Framework** | /System/Library/Frameworks/Accelerate/ |
| **Header file directories** | /System/Library/Frameworks/Accelerate/vImage.framework |
| **Declared in** | Alpha.h<br>BasicImageTypes.h<br>Conversion.h<br>Convolution.h<br>Geometry.h<br>Histogram.h<br>Morphology.h<br>Transform.h<br>vImage_Types.h |

The vImage framework is a high-performance image processing framework. It includes high-level functions for image manipulation—convolutions, geometric transformations, histogram operations, morphological transformations, and alpha compositing—as well as utility functions for format conversions and other operations.

The framework uses vectorized code that makes use of Single Instruction Multiple Data (SIMD) vector units, when available. It uses the best code for the hardware it is running on, in a manner completely transparent to the calling application.

Introduction

# Data Types

PART I

Data Types

# vImage Data Types and Constants Reference

| | |
|---|---|
| **Framework:** | Accelerate/vImage |
| **Declared in** | vImage_Types.h |
| **Companion guide** | vImage Programming Guide |

## Overview

The data types and constants defined in this document are used by vImage functions. The primary vImage data type is the vImage buffer, which contains a pointer to image data along with other image data information. The vImage framework also defines data types for planar and interleaved pixel types, a resampling callback filter, and an affine transform. It provides constants that specify errors that can be returned by vImage functions and flags that you can pass to a function to specify a variety of processing options.

## Data Types

### vImagePixelCount

A type for the number of pixels.

```
typedef unsigned long  vImagePixelCount;
```

**Discussion**
For LP64 (ppc64) this is a 64-bit quantity.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`vImage_Types.h`

### vImage_Buffer

The basic data structure used by vImage functions for passing image data.

```
typedef struct vImage_Buffer
{
    void        *data;
    vImagePixelCount    height;
    vImagePixelCount    width;
    size_t    rowBytes;
}vImage_Buffer;
```

**Fields**

`data`

A pointer to memory for image data. The image data can be in planar (Planar8, PlanarF) or interleaved (ARGB8888, ARGBFFFF, RGBA8888, or RGBAFFFF) formats. If you are using the vImage buffer to provide an image, then the pointer should point to the top left pixel of the image. If you are providing the vImage buffer to a function that fills the memory with image data (that is, as a destination buffer), the pointer must point to an area of memory that is an appropriate size for the destination buffer. Specifically, size of the memory, in bytes, must be at least the height of the image data multiplied by the number of row bytes.

`height`

The number of pixels in one column of the image.

`width`

The number of pixels in one row of the image.

`rowBytes`

The number of bytes in a pixel row. This is the distance, in bytes, between the start of one row of the image and the start of the next. This quantity must be at least the width times the pixel size, where pixel size depends on the image format. You can provide a larger value, in which case the extra bytes will extend beyond the end of each row of pixels.

You may want to provide a larger value for one of two reasons: to improve performance, or to describe an image within a larger image without copying the data. The extra bytes are not considered part of the image represented by the vImage buffer.

**Discussion**

vImage functions will not attempt to read pixel data outside the area described by the height and width fields of the vImage buffer. The function will also not write data outside that area.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`vImage_Types.h`

## vImage_AffineTransform

A structure for values that represent an affine transformation.

```
typedef struct vImage_AffineTransform
{
    float a, b, c, d;
    float tx, ty;
}vImage_AffineTransform;
```

**Discussion**

This structure represents the 3x2 matrix :

$$\begin{pmatrix} a & b \\ c & d \\ tx & ty \end{pmatrix}$$

The vImage affine transform structure is just like the Quartz `CGAffineTransform` data structure. *CGAffineTransform Reference* describes functions for creating and manipulating matrixes of this form.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`vImage_Types.h`

## vImage_Error

A type for image errors.

```
typedef  ssize_t  vImage_Error;
```

**Discussion**
"Error Codes" (page 15) describes the constants that use this type.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`vImage_Types.h`

## vImage_Flags

A type for processing options.

```
typedef uint32_t vImage_Flags;
```

**Discussion**
"Processing Flags" (page 17) describes the constants that use this type.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`vImage_Types.h`

## Pixel_8

A type for an 8-bit planar pixel value

```
typedef uint8_t Pixel_8;
```

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`vImage_Types.h`

## Pixel_F

A type for a floating-point planar pixel value

```
typedef float Pixel_F;
```

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`vImage_Types.h`

## Pixel_8888

A type for an interleaved, 8 bits per channel pixel value.

```
typedef uint8_t Pixel_8888[4];
```

**Discussion**
For example, `uint8_t[4] = { alpha, red, green, blue }` for ARGB data.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`vImage_Types.h`

## Pixel_FFFF

A type for an interleaved, floating-point pixel value.

```
typedef float Pixel_FFFF[4];
```

**Discussion**
For example, `float[4] = { alpha, red, green, blue }` for ARGB data.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`vImage_Types.h`

## GammaFunction

A type for a gamma function.

```
typedef void * GammaFunction;
```

**Discussion**
You use this data type when you create a gamma function. See `vImageCreateGammaFunction` (page 247).

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`vImage_Types.h`

### ResamplingFilter

A pointer to a resampling filter callback function.

```
typedef void * ResamplingFilter;
```

**Discussion**
You pass a resampling filter callback to a shear function. The resampling filter pointer can point to a structure that contains a function, rows of precalculated values, flag settings, and so on. The shear function requires that the structure contains a scale factor.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`vImage_Types.h`

## Constants

### Error Codes

Error codes returned by vImage functions.

```
enum
{
    kvImageNoError                  =   0,
    kvImageRoiLargerThanInputBuffer =   -21766,
    kvImageInvalidKernelSize        =   -21767,
    kvImageNoEdgeStyleSpecified     =   -21768,
    kvImageInvalidOffset_X          =   -21769,
    kvImageInvalidOffset_Y          =   -21770,
    kvImageMemoryAllocationError    =   -21771,
    kvImageNullPointerArgument      =   -21772,
    kvImageInvalidParameter         =   -21773,
    kvImageBufferSizeMismatch       =   -21774,
    kvImageUnknownFlagsBit           =    -21775
};
```

**Constants**

`kvImageNoError`

The vImage function completed without error.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageRoiLargerThanInputBuffer`

The region of interest, as specified by the `srcOffsetToROI_X` and `srcOffsetToROI_Y` parameters and the height and width of the destination buffer, extends beyond the bottom edge or right edge of the source buffer.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageInvalidKernelSize`

Either the kernel height, the kernel width, or both, are even.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageNoEdgeStyleSpecified`

The function requires you to set at least one edge option flags: `kvImageCopyInPlace`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`, but none is set. See "Processing Flags" (page 17).

`kvImageInvalidOffset_X`

The `srcOffsetToROI_X` parameter that specifies the left edge of the region of interest is greater than the width of the source image.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageInvalidOffset_Y`

The `srcOffsetToROI_Y` parameter that specifies the top edge of the region of interest is greater than the height of the source image.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageMemoryAllocationError`

An attempt to allocate memory failed.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageNullPointerArgument`
A pointer parameter is `NULL` and it must not be.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageInvalidParameter`
Invalid parameter.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageBufferSizeMismatch`
The function requires the source and destination buffers to have the same height and the same width, but they do not.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageUnknownFlagsBit`
The flag is not recognized.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

**Declared In**
`vImage_Types.h`

## Processing Flags

Flags that specify options for vImage functions.

```
enum
{
    kvImageNoFlags                  =   0,
    kvImageLeaveAlphaUnchanged      =   1,
    kvImageCopyInPlace              =   2,
    kvImageBackgroundColorFill      =   4,
    kvImageEdgeExtend               =   8,
    kvImageDoNotTile                =   16,
    kvImageHighQualityResampling    =   32,
    kvImageTruncateKernel           =   64,
    kvImageGetTempBufferSize        =   128
};
```

**Constants**
`kvImageNoFlags`
Do not set any flags.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageLeaveAlphaUnchanged`
Operate on red, green, and blue channels only. When you set this flag, the alpha value is copied from source to destination. You can set this flag only for interleaved image formats.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageCopyInPlace`

Copy the value of the edge pixel in the source to the destination. When you set this flag, and a convolution function is processing an image pixel for which some of the kernel extends beyond the image boundaries, vImage does not computer the convolution. Instead, the value of the particular pixel unchanged; it's simply copied to the destination image. This flag is valid only for convolution operations. The morphology functions do not use this flag because they do not use pixels outside the image in any of their calculations.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageBackgroundColorFill`

A background color fill. The associated value is a background color (that is, a pixel value). When you set this flag, vImage assigns the pixel value to all pixels outside the image. You can set this flag for convolution and geometry functions. The morphology functions do not use this flag because they do not use pixels outside the image in any of their calculations.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageEdgeExtend`

Extend the edges of the image infinitely. When you set this flag, vImage replicates the edges of the image outward. It repeats the top row of the image infinitely above the image, the bottom row infinitely below the image, the first column infinitely to the left of the image, and the last column infinitely to the right. For spaces that are diagonal to the image, vImage uses the value of the corresponding corner pixel. For example, for all pixels that are both above and to the left of the image, the upper-leftmost pixel of the image (the pixel at row 0, column 0) supplies the value. In this way, vImage assigns every pixel location outside the image some value. You can set this flag for convolution and geometry functions. The morphology functions do not use this flag because they do not use pixels outside the image in any of their calculations.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageDoNotTile`

Do not use vImage internal tiling routines. When you set this flag, vImage turns off internal tiling. Set this flag if you want to perform your own tiling or your own multithreading, or to use the minimum or maximum filters in place.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageHighQualityResampling`

Use a higher quality, slower resampling filter for for geometry operations—shear, scale, rotate, affine transform, and so forth.

Available in Mac OS X v10.3 and later.

Declared in `vImage_Types.h`.

`kvImageTruncateKernel`

Use the part of the kernel that overlaps the image. This flag is valid only for convolution operations. When you set this flag, vImage restricts calculations to the portion of the kernel overlapping the image. It corrects the calculated pixel by first multiplying by the sum of all the kernel elements, then dividing by the sum of the kernel elements that are actually used. This preserves image brightness at the edges.

For integer kernels:

```
real_divisor = divisor * (sum of used kernel elements) / (sum of kernel
elements)
```

The morphology functions do not use this flag because they do not use pixels outside the image in any of their calculations.

Kernel truncation is not robust for certain kernels. It can ail when any rectangular segment of the kernel that includes the center, and at least one of the corners, sums to zero. You typically see this with emboss or edge detection filters, or other filters that are designed to find the slope of a signal. For those kinds of filters, you should use the `kvImageEdgeExtend` option instead.

Available in Mac OS X v10.4 and later.

Declared in `vImage_Types.h`.

`kvImageGetTempBufferSize`

Get the minimum temporary buffer size for the operation, given the parameters provided. When you set this flag, the function returns the number of bytes required for the temporary buffer. A negative value specifies an error.

Available in Mac OS X v10.4 and later.

Declared in `vImage_Types.h`.

**Discussion**

You can pass multiple flags to a function by adding the flag values together. For example, to leave alpha unchanged and turn off tiling, you can pass:

```
kvImageLeaveAlphaUnchanged + kvImageDoNotTile
```

Three of the flags are mutually exclusive: `kvImageCopyInPlace`, `kvImageBackgroundColorFill`, and `kvImageEdgeExtend`. Never pass more than one of these flag values in the same flag parameter.

When passing flags to a function, do not set values for flags that are not used by the function. If the function requires you to set certain flag values, do so. For example, for the convolution function, you must set exactly one of `kvImageCopyInPlace`, `kvImageBackgroundColorFill`, and `kvImageEdgeExtend`. Otherwise the function may return an error. If you don't want to set flag values, pass `kvImageNoFlags`.

**Declared In**

`vImage_Types.h`

# Other References

# vImage Alpha Compositing Reference

| | |
|---|---|
| **Framework:** | Accelerate/vImage |
| **Declared in** | Alpha.h |
| **Companion guide** | vImage Programming Guide |

## Overview

Alpha compositing (also known as alpha blending) is the process of layering multiple images, with the alpha value for a pixel in a given layer indicating what fraction of the colors from lower layers are seen through the color at the given level. The functions described in this reference operate on the alpha values of pixels either by blending alpha values or clipping them.

Most of the alpha compositing functions blend two input images—a top image and a bottom image—to create a composite image. The vImage framework computes the alpha values of the composite image from the alpha values of the input images. Some functions operate on interleaved formats (ARGB8888, ARGBFFFF, RGBA8888, RGBAFFFF) while others operate on planar formats. Interleaved formats contain an alpha value for each pixel, but planar formats do not. To perform alpha compositing with planar images, you need to supply the alpha information separately.

Alpha compositing functions by default perform tiling internally and may multithread internally as well. If you plan to perform your own tiling or multithreading, you must turn off vImage internal tiling and multithreading by supplying the `kvImageDoNotTile` flag as an option to the functions you use.

The vImage framework provides functions for alpha compositing for both the premultiplied alpha case and the nonpremultiplied alpha case. Mac OS X v10.4 adds some alpha compositing functions for common mixed cases. Premultiplying pixel color values by the associated alpha value results in greater computational efficiency than providing nonpremultiplied data, especially when you composite more than two images. When you use premultiplied alpha, you still need to maintain the original alpha information, so that you can retrieve the original, nonpremultiplied values of the pixels when you need them. You also need to supply the original alpha value for the bottom layer in a compositing operation.

For floating-point formats, you can multiply the color value by the alpha value directly. For integer formats in which both values are in the range of 0 to 255, you multiply the color and alpha values, then you must scale the result so that it is in the 0 to 255 range. The scaling calculation is:

```
scaledColor = (alpha * color + 127) / 255
```

Alpha compositing functions use a vImage buffer structure (`vImage_Buffer`—see *vImage Data Types and Constants Reference*) to receive and supply image data. This buffer contains a pointer to image data, the height and width (in pixels) of the image data, and the number of row bytes. You actually pass a pointer to

a vImage buffer structure. You can provide a pointer to the same vImage buffer structure for one of the source images and the destination image because alpha compositing functions "work in place". That is , the source and destination images can occupy the same memory if the they are strictly aligned pixel for pixel.

# Functions by Task

## Performing Nonpremultiplied Alpha Compositing

`vImageAlphaBlend_ARGBFFFF` (page 27)
: Performs nonpremultiplied alpha compositing of two ARGBFFFF images, placing the result in a destination buffer.

`vImageAlphaBlend_ARGB8888` (page 26)
: Performs nonpremultiplied alpha compositing of two ARGB8888 images, placing the result in a destination buffer.

`vImageAlphaBlend_PlanarF` (page 33)
: Performs nonpremultiplied alpha compositing of two PlanarF images, placing the result in a destination buffer.

`vImageAlphaBlend_Planar8` (page 32)
: Performs nonpremultiplied alpha compositing of two Planar8 images, placing the result in a destination buffer.

## Performing Premultiplied Alpha Compositing

`vImagePremultipliedAlphaBlend_ARGBFFFF` (page 38)
: Performs premultiplied alpha compositing of two ARGBFFFF images, placing the result in a destination buffer.

`vImagePremultipliedAlphaBlend_ARGB8888` (page 38)
: Performs premultiplied alpha compositing of two ARGB8888 images, placing the result in a destination buffer.

`vImagePremultipliedAlphaBlend_PlanarF` (page 40)
: Performs premultiplied alpha compositing of two PlanarF images, placing the result in a destination buffer.

`vImagePremultipliedAlphaBlend_Planar8` (page 39)
: Performs premultiplied alpha compositing of two Planar8 images, placing the result in a destination buffer.

## Performing Nonpremultiplied Alpha Compositing With a Single Alpha Value

`vImagePremultipliedConstAlphaBlend_ARGBFFFF` (page 41)
: Performs premultiplied alpha compositing of two ARGBFFFF images, using a single alpha value for the whole image and placing the result in a destination buffer.

vImagePremultipliedConstAlphaBlend_ARGB8888 (page 41)
Performs premultiplied alpha compositing of two ARGB8888 images, using a single alpha value for the whole image and placing the result in a destination buffer.

vImagePremultipliedConstAlphaBlend_PlanarF (page 43)
Performs premultiplied alpha compositing of a two PlanarF images, using a single alpha value for the whole image and placing the result in a destination buffer.

vImagePremultipliedConstAlphaBlend_Planar8 (page 42)
Performs premultiplied alpha compositing of two Planar8 images, using a single alpha value for the entire image and placing the result in a destination buffer.

## Performing Nonpremultiplied to Premultiplied Alpha Compositing

vImageAlphaBlend_NonpremultipliedToPremultiplied_ARGBFFFF (page 29)
Performs mixed alpha compositing of a nonpremultiplied ARGBFFFF image over a premultiplied ARGBFFFF image, placing the premultiplied result in a destination buffer.

vImageAlphaBlend_NonpremultipliedToPremultiplied_ARGB8888 (page 28)
Performs mixed alpha compositing of a nonpremultiplied ARGB8888 image over a premultiplied ARGB8888 image, placing the premultiplied result in a destination buffer.

vImageAlphaBlend_NonpremultipliedToPremultiplied_PlanarF (page 31)
Performs mixed alpha compositing of a nonpremultiplied PlanarF image over a premultiplied PlanarF image, placing the premultiplied result in a destination buffer.

vImageAlphaBlend_NonpremultipliedToPremultiplied_Planar8 (page 30)
Performs mixed alpha compositing of a nonpremultiplied Planar8 image over a premultiplied Planar8 image, placing the premultiplied result in a destination buffer.

## Converting from Unpremultiplied to Premultiplied Format

vImagePremultiplyData_ARGBFFFF (page 45)
Takes an ARGBFFFF image in nonpremultiplied alpha format and transforms it into an image in premultiplied alpha format.

vImagePremultiplyData_RGBAFFFF (page 48)
Takes an RGBAFFFF image in nonpremultiplied alpha format and transforms it into an image in premultiplied alpha format.

vImagePremultiplyData_ARGB8888 (page 44)
Takes an ARGB8888 image in nonpremultiplied alpha format and transforms it into an image in premultiplied alpha format.

vImagePremultiplyData_RGBA8888 (page 47)
Takes an RGBA8888 image in nonpremultiplied alpha format and transforms it into an image in premultiplied alpha format.

vImagePremultiplyData_PlanarF (page 46)
Takes a PlanarF image in nonpremultiplied alpha format, along with alpha information, and transforms it into an image in premultiplied alpha format.

vImagePremultiplyData_Planar8 (page 46)
Takes a Planar8 image in nonpremultiplied alpha format, along with alpha information, and transforms it into an image in premultiplied alpha format.

### Converting from Premultiplied to Unpremultiplied Format

`vImageUnpremultiplyData_ARGBFFFF` (page 49)
: Takes an ARGBFFFF image in premultiplied alpha format and transforms it into an image in nonpremultiplied alpha format.

`vImageUnpremultiplyData_RGBAFFFF` (page 52)
: Takes an RGBAFFFF image in premultiplied alpha format and transforms it into an image in nonpremultiplied alpha format.

`vImageUnpremultiplyData_ARGB8888` (page 48)
: Takes an ARGB8888 image in premultiplied alpha format and transforms it into an image in nonpremultiplied alpha format.

`vImageUnpremultiplyData_RGBA8888` (page 51)
: Takes an RGBA8888 image in premultiplied alpha format and transforms it into an image in nonpremultiplied alpha format.

`vImageUnpremultiplyData_PlanarF` (page 51)
: Takes a PlanarF image in premultiplied alpha format, along with alpha information, and transforms it into an image in nonpremultiplied alpha format.

`vImageUnpremultiplyData_Planar8` (page 50)
: Takes a Planar8 image in premultiplied alpha format, along with alpha information, and transforms it into an image in nonpremultiplied alpha format.

### Clipping Color Values to Alpha

`vImageClipToAlpha_Planar8` (page 36)
: Sets the color channel of each pixel in a Planar8 image to the smaller of two values—either the color channel or the alpha value for that pixel.

`vImageClipToAlpha_ARGB8888` (page 34)
: Sets the color channel of each pixel in an ARGB8888 image to the smaller of two values—either the color channel or the alpha value for that pixel.

`vImageClipToAlpha_PlanarF` (page 37)
: Sets the color channel of each pixel in a PlanarF image to the smaller of two values—either the color channel or the alpha value for that pixel.

`vImageClipToAlpha_ARGBFFFF` (page 35)
: Sets the color channel of each pixel in an ARGBFFFF image to the smaller of two values—either the color channel or the alpha value for that pixel.

# Functions

## vImageAlphaBlend_ARGB8888

Performs nonpremultiplied alpha compositing of two ARGB8888 images, placing the result in a destination buffer.

```
vImage_Error vImageAlphaBlend_ARGB8888 (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*

A pointer to a vImage buffer structure that contains data for the top source image.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The vImage buffer structures for the source and destination images must use the same height and width.

The calculation for each color channel is:

```
resultAlpha = (topAlpha * 255 + (255 - topAlpha)
                * bottomAlpha + 127) / 255
resultColor = (topAlpha * topColor + (((255 - topAlpha)
                * bottomAlpha + 127) / 255) * bottomColor +  127)
                   / resultAlpha
```

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Alpha.h`

## vImageAlphaBlend_ARGBFFFF

Performs nonpremultiplied alpha compositing of two ARGBFFFF images, placing the result in a destination buffer.

```
vImage_Error vImageAlphaBlend_ARGBFFFF (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*

A pointer to a vImage buffer structure that contains data for the top source image.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The vImage buffer structures for the source and destination images must use the same height and width.

The calculation for each color channel is:

```
resultAlpha = topAlpha + (1.0f - topAlpha) * bottomAlpha
resultColor = (topAlpha * topColor + (1.0f - topAlpha)
                  * bottomAlpha * bottomColor) / resultAlpha
```

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Alpha.h`

## vImageAlphaBlend_NonpremultipliedToPremultiplied_ARGB8888

Performs mixed alpha compositing of a nonpremultiplied ARGB8888 image over a premultiplied ARGB8888 image, placing the premultiplied result in a destination buffer.

```
vImage_Error vImageAlphaBlend_NonpremultipliedToPremultiplied_ARGB8888 (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The vImage buffer structures for the source images must be at least as wide and at least as high as the destination buffer.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Alpha.h`

## vImageAlphaBlend_NonpremultipliedToPremultiplied_ARGBFFFF

Performs mixed alpha compositing of a nonpremultiplied ARGBFFFF image over a premultiplied ARGBFFFF image, placing the premultiplied result in a destination buffer.

```
vImage_Error vImageAlphaBlend_NonpremultipliedToPremultiplied_ARGBFFFF (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The vImage buffer structures for the source images must be at least as wide and at least as high as the destination buffer.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImageAlphaBlend_NonpremultipliedToPremultiplied_Planar8

Performs mixed alpha compositing of a nonpremultiplied Planar8 image over a premultiplied Planar8 image, placing the premultiplied result in a destination buffer.

```
vImage_Error vImageAlphaBlend_NonpremultipliedToPremultiplied_Planar8 (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcTopAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*srcTopAlpha*

A pointer to a vImage buffer structure that contains data for the alpha values of the top source image.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The vImage buffer structures for the source images must be at least as wide and at least as high as the destination buffer.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImageAlphaBlend_NonpremultipliedToPremultiplied_PlanarF

Performs mixed alpha compositing of a nonpremultiplied PlanarF image over a premultiplied PlanarF image, placing the premultiplied result in a destination buffer.

```
vImage_Error vImageAlphaBlend_NonpremultipliedToPremultiplied_PlanarF (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcTopAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*
: A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*srcTopAlpha*
: A pointer to a vImage buffer structure that contains data for the alpha values of the top source image.

*srcBottom*
: A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*
: A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*
: The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The vImage buffer structures for the source images must be at least as wide and at least as high as the destination buffer.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImageAlphaBlend_Planar8

Performs nonpremultiplied alpha compositing of two Planar8 images, placing the result in a destination buffer.

```
vImage_Error vImageAlphaBlend_Planar8 (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcTopAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *srcBottomAlpha,
   const vImage_Buffer *alpha,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*
: A pointer to a vImage buffer structure that contains data for the top source image.

*srcTopAlpha*
: A pointer to a vImage buffer structure that contains data for the alpha values of the top source image.

*srcBottom*
: A pointer to a vImage buffer structure that contains data for the bottom source image.

*srcBottomAlpha*
: A pointer to a vImage buffer structure that contains data for the alpha values of the bottom source image.

*alpha*
: A pointer to a vImage buffer structure that contains data for the precalculated alpha values of the composite image. You must precalculate these values by calling the function `vPremultipliedAlphaBlend_PlanarF`. See the Discussion for details on using this function.

*dest*
: A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*
: The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The vImage buffer structures for the source, alpha values, destination, and composite alpha values must contain the same height and width.

For performance reasons, this function does not calculate alpha values for the composite image; you must provide them. You'll typically call this function three times, once for each color channel (red, green, blue). Because each call uses the same alpha value, it is much more efficient for you to precalculate the alpha values

using the function `vImagePremultipliedAlphaBlend_Planar8`, rather than have the calculation repeated three times by the`vImageAlphaBlend_Planar8` function. Call the function `vPremultipliedAlphaBlend_Planar8` using the parameters shown:

```
vImagePremultipliedAlphaBlend_Planar8(srcTopAlpha,
             srcTopAlpha, // Yes, use this twice
             srcBottomAlpha,
             alpha, //On return, contains the composite alpha values
             kvImageNoFlags );
```

After calling the `vPremultipliedAlphaBlend_Planar8` function, the resulting values for each color channel are:

```
resultAlpha = topAlpha + (1.0f - topAlpha) * bottomAlpha
resultColor = (topAlpha * topColor + (1.0f - topAlpha)
                  * bottomAlpha * bottomColor) / resultAlpha
```

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImageAlphaBlend_PlanarF

Performs nonpremultiplied alpha compositing of two PlanarF images, placing the result in a destination buffer.

```
vImage_Error vImageAlphaBlend_PlanarF (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcTopAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *srcBottomAlpha,
   const vImage_Buffer *alpha,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*
: A pointer to a vImage buffer structure that contains data for the top source image.

*srcTopAlpha*
: A pointer to a vImage buffer structure that contains data for the alpha values of the top source image.

*srcBottom*
: A pointer to a vImage buffer structure that contains data for the bottom source image.

*srcBottomAlpha*
: A pointer to a vImage buffer structure that contains data for the alpha values of the bottom source image.

*alpha*
: A pointer to a vImage buffer structure that contains data for the precalculated alpha values of the composite image. You must precalculate these values by calling the function `vPremultipliedAlphaBlend_PlanarF`. See the Discussion for details on using this function.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The vImage buffer structures for the source, alpha values, destination, and composite alpha values must contain the same height and width.

For performance reasons, this function does not calculate alpha values for the composite image; you must provide them. You'll typically call this function three times, once for each color channel (red, green, blue). Because each call uses the same alpha value, it is much more efficient for you to precalculate the alpha values using the function `vImagePremultipliedAlphaBlend_PlanarF`, rather than to have the calculation repeated three times by the`vImageAlphaBlend_PlanarF` function. Call the function `vPremultipliedAlphaBlend_PlanarF` using the parameters shown:

```
vImagePremultipliedAlphaBlend_PlanarF(srcTopAlpha,
             srcTopAlpha, // Yes, use this twice
             srcBottomAlpha,
             alpha, //On return, contains the composite alpha values
             kvImageNoFlags );
```

After calling the `vPremultipliedAlphaBlend_PlanarF` function, the resulting values for each color channel are:

```
resultAlpha = topAlpha + (1.0f - topAlpha) * bottomAlpha
resultColor = (topAlpha * topColor + (1.0f - topAlpha)
                  * bottomAlpha * bottomColor) / resultAlpha
```

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImageClipToAlpha_ARGB8888

Sets the color channel of each pixel in an ARGB8888 image to the smaller of two values—either the color channel or the alpha value for that pixel.

```
vImage_Error vImageClipToAlpha_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

For each pixel:

```
alpha_result = sourceAlpha;
color_result = MIN(sourceColor, sourceAlpha);
```

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Alpha.h`

## vImageClipToAlpha_ARGBFFFF

Sets the color channel of each pixel in an ARGBFFFF image to the smaller of two values—either the color channel or the alpha value for that pixel.

```
vImage_Error vImageClipToAlpha_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
For each pixel:

```
alpha_result = sourceAlpha;
color_result = MIN(sourceColor, sourceAlpha);
```

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImageClipToAlpha_Planar8

Sets the color channel of each pixel in a Planar8 image to the smaller of two values—either the color channel or the alpha value for that pixel.

```
vImage_Error vImageClipToAlpha_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *alpha,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the top source image.

*alpha*

A pointer to a vImage buffer structure that contains data for alpha values of the source image. The planar source image does not contain its own alpha information, so you must supply the alpha information.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
For each pixel:

```
alpha_result = sourceAlpha;
```

```
color_result = MIN(sourceColor, sourceAlpha);
```

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImageClipToAlpha_PlanarF

Sets the color channel of each pixel in a PlanarF image to the smaller of two values—either the color channel or the alpha value for that pixel.

```
vImage_Error vImageClipToAlpha_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *alpha,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*alpha*

A pointer to a vImage buffer structure that contains data for alpha values of the source image. The planar source image does not contain its own alpha information, so you must supply the alpha information.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
For each pixel:

```
alpha_result = sourceAlpha;
color_result = MIN(sourceColor, sourceAlpha);
```

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImagePremultipliedAlphaBlend_ARGB8888

Performs premultiplied alpha compositing of two ARGB8888 images, placing the result in a destination buffer.

```
vImage_Error vImagePremultipliedAlphaBlend_ARGB8888 (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*srcTop*

A pointer to a vImage buffer structure that contains data for the top source image.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

### Return Value

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

### Discussion

The vImage buffer structures for the source and destination images must use the same height and width.

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

`Alpha.h`

## vImagePremultipliedAlphaBlend_ARGBFFFF

Performs premultiplied alpha compositing of two ARGBFFFF images, placing the result in a destination buffer.

```
vImage_Error vImagePremultipliedAlphaBlend_ARGBFFFF (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*srcTop*

A pointer to a vImage buffer structure that contains data for the top source image.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The vImage buffer structures for the source and destination images must use the same height and width.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Alpha.h`

## vImagePremultipliedAlphaBlend_Planar8

Performs premultiplied alpha compositing of two Planar8 images, placing the result in a destination buffer.

```
vImage_Error vImagePremultipliedAlphaBlend_Planar8 (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcTopAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*

A pointer to a vImage buffer structure that contains data for the top source image.

*srcTopAlpha*

A pointer to a vImage buffer structure that contains data for the alpha values of the top source image. Even though the alpha values are already premultiplied into the pixel values, the function also requires the original alpha information for the top image to do its calculations. There is no way to extract this information from the premultiplied planar values, so you must provide it.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The vImage buffer structures for the source and destination images must use the same height and width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImagePremultipliedAlphaBlend_PlanarF

Performs premultiplied alpha compositing of two PlanarF images, placing the result in a destination buffer.

```
vImage_Error vImagePremultipliedAlphaBlend_PlanarF (
   const vImage_Buffer *srcTop,
   const vImage_Buffer *srcTopAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*
A pointer to a vImage buffer structure that contains data for the top source image.

*srcTopAlpha*
A pointer to a vImage buffer structure that contains data for the alpha values of the top source image. Even though the alpha values are already premultiplied into the pixel values, the function also requires the original alpha information for the top image to do its calculations. There is no way to extract this information from the premultiplied planar values, so you must provide it.

*srcBottom*
A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*
The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImagePremultipliedConstAlphaBlend_ARGB8888

Performs premultiplied alpha compositing of two ARGB8888 images, using a single alpha value for the whole image and placing the result in a destination buffer.

```
vImage_Error vImagePremultipliedConstAlphaBlend_ARGB8888 (
   const vImage_Buffer *srcTop,
   Pixel_8 constAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*srcTop*

A pointer to a vImage buffer structure that contains data for the top source image.

*constAlpha*

The alpha value you want to apply to the image.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

### Return Value

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

### Discussion

The vImage buffer structures for the source and destination images must use the same height and width.

### Availability

Available in Mac OS X v10.4 and later.

### Declared In

`Alpha.h`

## vImagePremultipliedConstAlphaBlend_ARGBFFFF

Performs premultiplied alpha compositing of two ARGBFFFF images, using a single alpha value for the whole image and placing the result in a destination buffer.

```
vImage_Error vImagePremultipliedConstAlphaBlend_ARGBFFFF (
   const vImage_Buffer *srcTop,
   Pixel_F constAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*

A pointer to a vImage buffer structure that contains data for the top source image.

*constAlpha*

The alpha value you want to apply to the image.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The vImage buffer structures for the source and destination images must use the same height and width.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Alpha.h`

## vImagePremultipliedConstAlphaBlend_Planar8

Performs premultiplied alpha compositing of two Planar8 images, using a single alpha value for the entire image and placing the result in a destination buffer.

```
vImage_Error vImagePremultipliedConstAlphaBlend_Planar8 (
   const vImage_Buffer *srcTop,
   Pixel_8 constAlpha,
   const vImage_Buffer *srcTopAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*

A pointer to a vImage buffer structure that contains data for the top source image.

*constAlpha*
The alpha value you want to apply to the image.

*srcTopAlpha*
A pointer to a vImage buffer structure that contains data for the alpha values of the top source image. Even though the alpha values are already premultiplied into the pixel values, the function also requires the original alpha information for the top image to do its calculations. There is no way to extract this information from the premultiplied planar values, so you must provide it.

*srcBottom*
A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*
The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The vImage buffer structures for the source and destination images must use the same height and width.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImagePremultipliedConstAlphaBlend_PlanarF

Performs premultiplied alpha compositing of a two PlanarF images, using a single alpha value for the whole image and placing the result in a destination buffer.

```
vImage_Error vImagePremultipliedConstAlphaBlend_PlanarF (
   const vImage_Buffer *srcTop,
   Pixel_F constAlpha,
   const vImage_Buffer *srcTopAlpha,
   const vImage_Buffer *srcBottom,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcTop*
A pointer to a vImage buffer structure that contains data for the top source image.

*constAlpha*
The alpha value you want to apply to the image.

*srcTopAlpha*

A pointer to a vImage buffer structure that contains data for the alpha values of the top source image. Even though the alpha values are already premultiplied into the pixel values, the function also requires the original alpha information for the top image to do its calculations. There is no way to extract this information from the premultiplied planar values, so you must provide it.

*srcBottom*

A pointer to a vImage buffer structure that contains data for the bottom source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The vImage buffer structures for the source and destination images must use the same height and width.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Alpha.h`

## vImagePremultiplyData_ARGB8888

Takes an ARGB8888 image in nonpremultiplied alpha format and transforms it into an image in premultiplied alpha format.

```
vImage_Error vImagePremultiplyData_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function gets the required alpha information from the alpha channel of the original image. The alpha channel is copied over unchanged to the destination image.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImagePremultiplyData_ARGBFFFF

Takes an ARGBFFFF image in nonpremultiplied alpha format and transforms it into an image in premultiplied alpha format.

```
vImage_Error vImagePremultiplyData_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function gets the required alpha information from the alpha channel of the original image. The alpha channel is copied over unchanged to the destination image.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImagePremultiplyData_Planar8

Takes a Planar8 image in nonpremultiplied alpha format, along with alpha information, and transforms it into an image in premultiplied alpha format.

```
vImage_Error vImagePremultiplyData_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *alpha,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*alpha*

A pointer to a vImage buffer structure that contains data for alpha values of the source image. The planar source image does not contain its own alpha information, so you must supply the alpha information.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

### Return Value

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

`Alpha.h`

## vImagePremultiplyData_PlanarF

Takes a PlanarF image in nonpremultiplied alpha format, along with alpha information, and transforms it into an image in premultiplied alpha format.

```
vImage_Error vImagePremultiplyData_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *alpha,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*alpha*

A pointer to a vImage buffer structure that contains data for alpha values of the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImagePremultiplyData_RGBA8888

Takes an RGBA8888 image in nonpremultiplied alpha format and transforms it into an image in premultiplied alpha format.

```
vImage_Error vImagePremultiplyData_RGBA8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function gets the required alpha information from the alpha channel of the original image. The alpha channel is copied over unchanged to the destination image.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImagePremultiplyData_RGBAFFFF

Takes an RGBAFFFF image in nonpremultiplied alpha format and transforms it into an image in premultiplied alpha format.

```
vImage_Error vImagePremultiplyData_RGBAFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function gets the required alpha information from the alpha channel of the original image. The alpha channel is copied over unchanged to the destination image.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImageUnpremultiplyData_ARGB8888

Takes an ARGB8888 image in premultiplied alpha format and transforms it into an image in nonpremultiplied alpha format.

```
vImage_Error vImageUnpremultiplyData_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function gets the required alpha information from the alpha channel of the original image. The alpha channel is copied over unchanged to the destination image.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImageUnpremultiplyData_ARGBFFFF

Takes an ARGBFFFF image in premultiplied alpha format and transforms it into an image in nonpremultiplied alpha format.

```
vImage_Error vImageUnpremultiplyData_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function gets the required alpha information from the alpha channel of the original image. The alpha channel is copied over unchanged to the destination image.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImageUnpremultiplyData_Planar8

Takes a Planar8 image in premultiplied alpha format, along with alpha information, and transforms it into an image in nonpremultiplied alpha format.

```
vImage_Error vImageUnpremultiplyData_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *alpha,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the premultiplied data for the source image.]

*alpha*

A pointer to a vImage buffer structure that contains data for alpha values of the source image. The planar source image does not contain its own alpha information, so you must supply the alpha information.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Alpha.h`

## vImageUnpremultiplyData_PlanarF

Takes a PlanarF image in premultiplied alpha format, along with alpha information, and transforms it into an image in nonpremultiplied alpha format.

```
vImage_Error vImageUnpremultiplyData_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *alpha,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the premultiplied data for the source image.]

*alpha*

A pointer to a vImage buffer structure that contains data for alpha values of the source image. The planar source image does not contain its own alpha information, so you must supply the alpha information.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

### Return Value

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

`Alpha.h`

## vImageUnpremultiplyData_RGBA8888

Takes an RGBA8888 image in premultiplied alpha format and transforms it into an image in nonpremultiplied alpha format.

```
vImage_Error vImageUnpremultiplyData_RGBA8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the premultiplied data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function gets the required alpha information from the alpha channel of the original image. The alpha channel is copied over unchanged to the destination image.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

## vImageUnpremultiplyData_RGBAFFFF

Takes an RGBAFFFF image in premultiplied alpha format and transforms it into an image in nonpremultiplied alpha format.

```
vImage_Error vImageUnpremultiplyData_RGBAFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the nonpremultiplied data for the top source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the compositing. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function gets the required alpha information from the alpha channel of the original image. The alpha channel is copied over unchanged to the destination image.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Alpha.h`

# vImage Conversion Reference

| | |
|---|---|
| **Framework:** | Accelerate/vImage |
| **Declared in** | Conversion.h |
| **Companion guide** | vImage Programming Guide |

## Overview

Conversion functions change an image from one image format into another. These functions work with the formats supported by vImage (Planar8, PlanarF, ARGB8888, ARGBFFFF, RGBA8888, and RGBAFFFF) but they can also change between a supported format to one that's not supported by vImage (such as RGB565). Conversion functions can also fill buffers with a color, overwrite channels, permute channels, flatten data, and clip data.

Conversion functions use a vImage buffer structure (`vImage_Buffer`—see *vImage Data Types and Constants Reference*) to receive and supply image data. This buffer contains a pointer to image data, the height and width (in pixels) of the image data, and the number of row bytes. You actually pass a pointer to a vImage buffer structure. For some functions, you can provide a pointer to the same vImage buffer structure for the source images and the destination image because the function "works in place". That is , the source and destination images can occupy the same memory if the they are strictly aligned pixel for pixel.

## Functions by Task

### Filling Buffers

`vImageBufferFill_ARGB8888` (page 59)
Fills an ARGB8888 buffer with a specified color.

`vImageBufferFill_ARGBFFFF` (page 60)
Fills an ARGBFFFF buffer with a specified color.

### Permuting Channels

`vImagePermuteChannels_ARGB8888` (page 100)
Reorders the channels in an ARGB8888 image.

`vImagePermuteChannels_ARGBFFFF` (page 102)
Reorders the channels in an ARGBFFFF image.

## Overwriting Channels

`vImageSelectChannels_ARGB8888` (page 102)
Overwrites the specified channels in an ARGB8888 image buffer with the provided channels from an ARGB8888 image buffer.

`vImageSelectChannels_ARGBFFFF` (page 103)
Overwrites the specified channels in an ARGBFFFF image buffer with the provided channels in an ARGBFFFF image buffer.

`vImageOverwriteChannels_ARGB8888` (page 98)
Overwrites one or more planes of an ARGB8888 image buffer with the provided planar buffer.

`vImageOverwriteChannels_ARGBFFFF` (page 99)
Overwrites one or more planes of an ARGBFFFF image buffer with the provided planar buffer.

`vImageOverwriteChannelsWithScalar_ARGB8888` (page 95)
Overwrites the pixels of one or more planes of an ARGB8888 image buffer with the provided scalar value.

`vImageOverwriteChannelsWithScalar_ARGBFFFF` (page 96)
Overwrites the pixels of one or more planes of an ARGBFFFF image buffer with the provided scalar value.

`vImageOverwriteChannelsWithScalar_Planar8` (page 97)
Overwrites a Planar8 image buffer with the provided value.

`vImageOverwriteChannelsWithScalar_PlanarF` (page 97)
Overwrites a PlanarF image buffer with the provided value.

`vImageOverwriteChannelsWithPixel_ARGB8888` (page 93)
Overwrites an ARGB8888 image buffer with the provided pixel value.

`vImageOverwriteChannelsWithPixel_ARGBFFFF` (page 94)
Overwrites an ARGBFFFF image buffer with the provided pixel value.

## Converting From 16 Bit

`vImageConvert_16SToF` (page 61)
Converts an image in a special planar format—in which each pixel value is a 16-bit signed integer—to a PlanarF format.

`vImageConvert_16UToF` (page 62)
Converts an image in a special planar format—in which each pixel value is a 16-bit unsigned integer—to a PlanarF format.

`vImageConvert_16UToPlanar8` (page 63)
Converts an image in a special planar format—in which each pixel value is a 16-bit unsigned integer—to a Planar8 image.

## Transforming Using Table Lookups

`vImageTableLookUp_ARGB8888` (page 104)
Transforms an ARGB8888 image by substituting pixel values with pixel values provided by four lookup tables.

`vImageTableLookUp_Planar8` (page 106)
Transforms an Planar8 image by substituting pixel values with pixel values provided by four lookup tables.

## Flattening Data

`vImageFlatten_ARGB8888ToRGB888` (page 91)
Transforms an ARGB8888 image to an RGB888 image against an opaque background of the provided color.

`vImageFlatten_ARGBFFFFToRGBFFF` (page 92)
Transforms an ARGBFFFF image to an RGBFFF image against an opaque background of the provided color.

## Clipping Data

`vImageClip_PlanarF` (page 60)
Clips the pixel values of an image in PlanarF format, using the provided minimum and maximum values.

## Converting Between Chunky and Planar

These convenience functions allow you to convert between various interleaved (or *chunky*) formats that vImage does not explicitly support (and that may have less than or more than four channels) and the formats that vImage supports explicitly. You can represent some non-interleaved formats as well. The functions are not fast or vectorized.

`vImageConvert_PlanarToChunky8` (page 84)
Combines a collection of planar source images into a single interleaved destination image, with one 8-bit channel for each planar image.

`vImageConvert_PlanarToChunkyF` (page 85)
Combines a collection of planar source images into a single interleaved destination image, with one floating-point channel for each planar image.

`vImageConvert_ChunkyToPlanar8` (page 70)
Separates a source image into a collection of corresponding planar destination images, one for each 8-bit channel of the original image.

`vImageConvert_ChunkyToPlanarF` (page 71)
Separates a source image into a collection of corresponding planar destination images, one for each floating-point channel of the original image.

## Converting From Planar Formats

`vImageConvert_Planar8To16U` (page 75)
Converts a Planar8 image to a 16U image .

`vImageConvert_Planar8toARGB1555` (page 76)
Combines four Planar8 images into one ARGB1555 image.

`vImageConvert_Planar8toARGB8888` (page 77)
Combines four Planar8 images into one ARGB8888 image.

`vImageConvert_Planar8toPlanarF` (page 78)
Converts a Planar8 image to a PlanarF image.

`vImageConvert_Planar8toRGB565` (page 79)
Combines three Planar8 images into one RGB565 image.

`vImageConvert_Planar8toRGB888` (page 80)
Combines three Planar8 images into one RGB888 image.

`vImageConvert_PlanarFtoRGBFFF` (page 83)
Combines three PlanarF images into one RGBFFF image.

`vImageConvert_PlanarFtoARGBFFFF` (page 80)
Combines four PlanarF images into one ARGBFFFF image.

`vImageConvert_PlanarFtoPlanar16F` (page 81)
Converts a PlanarF image to a Planar16F image.

`vImageConvert_PlanarFtoPlanar8` (page 82)
Converts a PlanarF image to a Planar8 image, clipping values to the provided minimum and maximum values.

`vImageConvert_Planar16FtoPlanarF` (page 74)
Converts a Planar16F image to a PlanarF image.

`vImageConvert_FTo16S` (page 73)
Converts a PlanarF image into a special format in which each pixel is a 16-bit signed integer.

`vImageConvert_FTo16U` (page 73)
Converts a PlanarF image into a special format in which each pixel is a 16-bit unsigned integer.

## Converting From ARGB Formats

`vImageConvert_ARGB1555toARGB8888` (page 64)
Converts an ARGB1555 image to an ARGB8888 image.

`vImageConvert_ARGB1555toPlanar8` (page 64)
Separates an ARGB1555 image into four Planar8 images.

`vImageConvert_ARGB8888toARGB1555` (page 66)
Converts an ARGB8888 image into an ARGB1555 image.

`vImageConvert_ARGB8888toPlanar8` (page 66)
Separates an ARGB8888 image into four Planar8 images.

`vImageConvert_ARGB8888toRGB565` (page 68)
Converts an ARGB8888 image into an RGB565 image.

`vImageConvert_ARGB8888toRGB888` (page 68)
Converts an ARGB8888 image into an RGB888 image.

`vImageConvert_ARGBFFFFtoPlanarF` (page 69)
Separates an ARGBFFFF image into four PlanarF images.

### Converting From RGB Formats

`vImageConvert_RGB565toPlanar8` (page 87)
Separates an RGB565 image into three Planar8 images.

`vImageConvert_RGB565toARGB8888` (page 86)
Converts an RGB565 image into an ARGB8888 image, using the provided 8-bit alpha value.

`vImageConvert_RGB888toARGB8888` (page 88)
Converts an RGB888 image into an ARGB8888 image, using the provided alpha value (either as planar or pixel data).

`vImageConvert_RGB888toPlanar8` (page 89)
Separates an RGB888 image into three Planar8 images.

`vImageConvert_RGBFFFtoPlanarF` (page 90)
Separates an RGBFFF image into three PlanarF images.

## Functions

### vImageBufferFill_ARGB8888

Fills an ARGB8888 buffer with a specified color.

```
vImage_Error vImageBufferFill_ARGB8888 (
   const vImage_Buffer *dest,
   const Pixel_8888 color,
   vImage_Flags flags
);
```

**Parameters**

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data created with the fill color. When you no longer need the data buffer, you must deallocate the memory.

*color*
The color to fill the buffer with.

*flags*
Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageOverwriteChannelsWithScalar_Planar8` (page 97)

**Declared In**
`Conversion.h`

## vImageBufferFill_ARGBFFFF

Fills an ARGBFFFF buffer with a specified color.

```
vImage_Error vImageBufferFill_ARGBFFFF (
   const vImage_Buffer *dest,
   const Pixel_FFFF color,
   vImage_Flags flags
);
```

### Parameters

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data created with the fill color. When you no longer need the data buffer, you must deallocate the memory.

*color*

The color to fill the buffer with.

*flags*

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

### Return Value

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

### Availability

Available in Mac OS X v10.4 and later.

### See Also

`vImageOverwriteChannelsWithScalar_PlanarF` (page 97)

### Declared In

`Conversion.h`

## vImageClip_PlanarF

Clips the pixel values of an image in PlanarF format, using the provided minimum and maximum values.

```
vImage_Error vImageClip_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   Pixel_F maxFloat,
   Pixel_F minFloat,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to clip.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*maxFloat*
A maximum pixel value. The function clips larger values to this value in the destination image.

*minFloat*
A maximum pixel value. The function clips smaller values to this value in the destination image.

*flags*
The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Conversion.h`

## vImageConvert_16SToF

Converts an image in a special planar format—in which each pixel value is a 16-bit signed integer— to a PlanarF format.

```
vImage_Error vImageConvert_16SToF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   float offset,
   float scale,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image (for which each pixel value is a 16-bit signed integer) whose data you want to overwrite.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data in PlanarF format. When you no longer need the data buffer, you must deallocate the memory.

*offset*
The offset value to add to each pixel.

*scale*
The value to multiply each pixel by.

*flags*
The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The function changes each pixel value to a floating-point value, scales each value and then adds the offset value. The calculation is

```
resultPixel = (float) sourcePixel * scale + offset
```

The functions `vImageConvert_16SToF` and `vImageConvert_FTo16S` are inverse transformations when you use the same offset and scale values for each. (The inversion is not precise due to round-off error.) This requires the two functions to use these values differently (and in a different order).

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Conversion.h`

## vImageConvert_16UToF

Converts an image in a special planar format—in which each pixel value is a 16-bit unsigned integer— to a PlanarF format.

```
vImage_Error vImageConvert_16UToF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   float offset,
   float scale,
   vImage_Flags flags
);
```

**Parameters**

*src*
: A pointer to a vImage buffer structure that contains the source image (for which each pixel value is a 16-bit unsigned integer) whose data you want to overwrite.

*dest*
: A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data in PlanarF format. When you no longer need the data buffer, you must deallocate the memory.

*offset*
: The offset value to add to each pixel.

*scale*
: The value to multiply each pixel by.

*flags*
: The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
Each pixel value is changed to a floating-point value, then scaled and offset by the provided values. The calculation is

```
resultPixel =   SATURATED_CLIP_SHRT_MIN_to_SHRT_MAX( (sourcePixel
                - offset) / scale  + 0.5f)
```

The functions `vImageConvert_16SToF` and `vImageConvert_FTo16S` are inverse transformations when you use the same offset and scale values for each. (The inversion is not precise due to round-off error.) This requires the two functions to use these values differently (and in a different order).

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Conversion.h`

## vImageConvert_16UToPlanar8

Converts an image in a special planar format—in which each pixel value is a 16-bit unsigned integer—to a Planar8 image.

```
vImage_Error vImageConvert_16UToPlanar8 (
const vImage_Buffer *src,
const vImage_Buffer *dest,
vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The conversion from 16-bit to 8-bit values is:

```
uint8_t result = (srcPixel * 255 + 32767) / 65535
```

You can also use this function to convert a 4-channel interleaved 16U image to an ARGB8888. image. Simply multiply the width of the destination buffer by four.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageConvert_ARGB1555toARGB8888

Converts an ARGB1555 image to an ARGB8888 image.

```
vImage_Error vImageConvert_ARGB1555toARGB8888 (
const vImage_Buffer *src,
const vImage_Buffer *dest,
vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the converted data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

### Return Value

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

### Discussion

The ARGB1555 format has 16-bit pixels with 1 bit for alpha and 5 bits each for red, green, and blue. The function calculates the 8-bit pixels in the destination image as follows:

```
Pixel8 alpha =  1bitAlphaChannel * 255
Pixel8 red   = (5bitRedChannel   * 255 + 15) / 31
Pixel8 green = (5bitGreenChannel * 255 + 15) / 31
Pixel8 blue  = (5bitBlueChannel  * 255 + 15) / 31
```

### Availability

Available in Mac OS X v10.4 and later.

### Declared In

`Conversion.h`

## vImageConvert_ARGB1555toPlanar8

Separates an ARGB1555 image into four Planar8 images.

```
vImage_Error vImageConvert_ARGB1555toPlanar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *destA,
   const vImage_Buffer *destR,
   const vImage_Buffer *destG,
   const vImage_Buffer *destB,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to separate.

*destA*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the alpha channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destR*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the red channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destG*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the green channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destB*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the blue channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The ARGB1555 format has 16-bit pixels with 1 bit for alpha and 5 bits each for red, green, and blue. The function calculates the 8-bit pixels in the destination image as follows:

```
Pixel8 alpha =  1bitAlphaChannel * 255
Pixel8 red   = (5bitRedChannel   * 255 + 15) / 31
Pixel8 green = (5bitGreenChannel * 255 + 15) / 31
Pixel8 blue  = (5bitBlueChannel  * 255 + 15) / 31
```

This function works in place for one destination buffer; the others must be allocated separately.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageConvert_ARGB8888toARGB1555

Converts an ARGB8888 image into an ARGB1555 image.

```
vImage_Error vImageConvert_ARGB8888toARGB1555 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the converted image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The ARGB1555 format has 16-bit pixels with 1 bit for alpha and 5 bits each for red, green, and blue. The function calculates the 16-bit pixels in the destination image as follows:

```
uint32_t alpha = (8bitAlphaChannel      + 127) / 255
uint32_t red   = (8bitRedChannel   * 31 + 127) / 255
uint32_t green = (8bitGreenChannel * 31 + 127) / 255
uint32_t blue  = (8bitBlueChannel  * 31 + 127) / 255
uint16_t ARGB1555pixel = (alpha << 15) | (red <<  10) | (green << 5) | blue
```

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageConvert_ARGB8888toPlanar8

Separates an ARGB8888 image into four Planar8 images.

```
vImage_Error vImageConvert_ARGB8888toPlanar8 (
   const vImage_Buffer *srcARGB,
   const vImage_Buffer *destA,
   const vImage_Buffer *destR,
   const vImage_Buffer *destG,
   const vImage_Buffer *destB,
   vImage_Flags flags
);
```

**Parameters**

*srcARGB*

A pointer to a vImage buffer structure that contains the source image whose data you want to separate.

*destA*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the alpha channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destR*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the red channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destG*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the green channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destB*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the blue channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The source image, and the *destA*, *destR*, *destG*, and *destB* destination buffers, must have the same height and the same width. This function works in place for one destination buffer. The others must be allocated separately.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Conversion.h`

## vImageConvert_ARGB8888toRGB565

Converts an ARGB8888 image into an RGB565 image.

```
vImage_Error vImageConvert_ARGB8888toRGB565 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the data in RGB565 format. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

### Return Value

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

### Discussion

The alpha channel in the ARGB8888 image is ignored. (The RGB565 format has 16-bit pixels with 5 bits for red, 6 for green, and 5 for blue.) The function calculates the pixels in the destination image as follows:

```
uint32_t red   = (8bitRedChannel   * (31*2) + 255) / (255*2)
uint32_t green = (8bitGreenChannel * 63 + 127) / 255
uint32_t blue  = (8bitBlueChannel  * 31 + 127) / 255
uint16_t RGB565pixel =  (red << 11) | (green <<  5) | blue
```

### Availability

Available in Mac OS X v10.4 and later.

### Declared In

`Conversion.h`

## vImageConvert_ARGB8888toRGB888

Converts an ARGB8888 image into an RGB888 image.

```
vImage_Error vImageConvert_ARGB8888toRGB888 (
   const vImage_Buffer *argbSrc,
   const vImage_Buffer *rgbDest,
   vImage_Flags flags
);
```

### Parameters

*argbSrc*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*rgbDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `argbSrc` data buffer. On return, the data buffer pointed to by this structure contains the data in RGB888 format. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The red, green, and blue channels are simply copied.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Conversion.h`

## vImageConvert_ARGBFFFFtoPlanarF

Separates an ARGBFFFF image into four PlanarF images.

```
vImage_Error vImageConvert_ARGBFFFFtoPlanarF (
   const vImage_Buffer *srcARGB,
   const vImage_Buffer *destA,
   const vImage_Buffer *destR,
   const vImage_Buffer *destG,
   const vImage_Buffer *destB,
   vImage_Flags flags
);
```

**Parameters**

*srcARGB*

A pointer to a vImage buffer structure that contains the source image whose data you want to separate.

*destA*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a PlanarF image equivalent to the alpha channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destR*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a PlanarF image equivalent to the red channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destG*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a PlanarF image equivalent to the green channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destB*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a PlanarF image equivalent to the blue channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The source image, and the *destA*, *destR*, *destG*, and *destB* destination buffers, must have the same height and the same width. This function works in place for one destination buffer. The others must be allocated separately.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Conversion.h`

## vImageConvert_ChunkyToPlanar8

Separates a source image into a collection of corresponding planar destination images, one for each 8-bit channel of the original image.

```
vImage_Error vImageConvert_ChunkyToPlanar8 (
   const void *srcChannels[],
   const vImage_Buffer *destPlanarBuffers[],
   unsigned int channelCount,
   size_t srcStrideBytes,
   vImagePixelCount srcWidth,
   vImagePixelCount srcHeight,
   size_t srcRowBytes,
   vImage_Flags flags
);
```

**Parameters**

*srcChannels*

An array of pointers to channels of the source image. Each pointer points to the start of the data for one source channel.

*destPlanarBuffers*

An array of vImage buffer structures, each of which contains image data in Planar8 format. Each structure must have the same width and height values, but may have different row byte values. On return, the data buffer in each vImage buffer structure contains a planar image equivalent to the corresponding channel of the source image.

*channelCount*

The number of channels in the source image.

*srcStrideBytes*

The number of bytes from one pixel value in a given channel to the next pixel of that channel (within a row). This value must be the same for all channels.

*srcWidth*

The number of pixels in a row. This value must be the same for all channels in the source image, and for all the destination buffers.

*srcHeight*

The number of rows. This value must be the same for all channels in the source image, and for all the destination buffers.

*srcRowBytes*

The number of bytes from the beginning of a channel row to the beginning of the next row of the channel. This value must be the same for all channels of the source image. (It does not have the be the same as the *rowBytes* values of the destination buffers. Each destination buffer can have its own *rowBytes* value.)

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Conversion.h`

## vImageConvert_ChunkyToPlanarF

Separates a source image into a collection of corresponding planar destination images, one for each floating-point channel of the original image.

```
vImage_Error vImageConvert_ChunkyToPlanarF (
   const void *srcChannels[],
   const vImage_Buffer *destPlanarBuffers[],
   unsigned int channelCount,
   size_t srcStrideBytes,
   vImagePixelCount srcWidth,
   vImagePixelCount srcHeight,
   size_t srcRowBytes,
   vImage_Flags flags
);
```

**Parameters**

*srcChannels*

An array of pointers to channels of the source image. Each pointer points to the start of the data for one source channel.

*destPlanarBuffers*

An array of vImage buffer structures, each of which contains image data in PlanarF format. Each structure must have the same width and height values, but may have different row byte values. On return, the data buffer in each each vImage buffer structure contains a planar image equivalent to the corresponding channel of the source image.

*channelCount*

The number of channels in the source image.

*srcStrideBytes*

The number of bytes from one pixel value in a given channel to the next pixel of that channel (within a row). This value must be the same for all channels.

*srcWidth*

The number of pixels in a row. This value must be the same for all channels in the source image, and for all the destination buffers.

*srcHeight*

The number of rows. This value must be the same for all channels in the source image, and for all the destination buffers.

*srcRowBytes*

The number of bytes from the beginning of a channel row to the beginning of the next row of the channel. This value must be the same for all channels of the source image. (It does not have the be the same as the *rowBytes* values of the destination buffers. Each destination buffer can have its own *rowBytes* value.)

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Conversion.h`

## vImageConvert_FTo16S

Converts a PlanarF image into a special format in which each pixel is a 16-bit signed integer.

```
vImage_Error vImageConvert_FTo16S (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   float offset,
   float scale,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the destination image converted to 16-bit signed integer format. When you no longer need the data buffer, you must deallocate the memory.

*offset*

The offset value to subtract from every pixel.

*scale*

The scale value to divide each pixel by.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

### Return Value

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

### Discussion

Each pixel value is first offset and scaled by the provided values, and then changed to a 16-bit signed integer (rounded and clipped as necessary). The calculation is as follows:

```
resultPixel =   SATURATED_CLIP_0_to_USHRT_MAX( (srcPixel - offset)
                / scale  + 0.5f)
```

The functions `vImageConvert_16SToF` and `vImageConvert_FTo16S` are inverse transformations when you use the same offset and scale values for each. (The inversion is not precise due to round-off error.) This requires the two functions to use these values differently (and in a different order).

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

`Conversion.h`

## vImageConvert_FTo16U

Converts a PlanarF image into a special format in which each pixel is a 16-bit unsigned integer.

```
vImage_Error vImageConvert_FTo16U (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   float offset,
   float scale,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the destination image converted to 16-bit unsigned integer format. When you no longer need the data buffer, you must deallocate the memory.

*offset*

The offset value to subtract from every pixel.

*scale*

The scale value to divide each pixel by.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

Each pixel value is first offset and scaled by user-supplied values, and then changed to a 16-bit unsigned integer (rounded and clipped as necessary). The calculation is as follows:

```
resultPixel =   SATURATED_CLIP_0_to_USHRT_MAX( (sourcePixel  - offset)
                / scale  + 0.5f)
```

The functions `vImageConvert_16UToF` and `vImageConvert_FTo16U` are inverse transformations when you use the same offset and scale values for each. (The inversion is not precise due to round-off error.) This requires the two functions to use these values differently (and in a different order).

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Conversion.h`

## vImageConvert_Planar16FtoPlanarF

Converts a Planar16F image to a PlanarF image.

```
vImage_Error vImageConvert_Planar16FtoPlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image converted to PlanarF format. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The Planar16F format is identical to the OpenEXR format; it uses 16-bit floating-point numbers. In conformance with IEEE-754, the function quiets signaling NaNs during the conversion. (OpenEXR-1.2.1 does not do this.)

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Conversion.h`

## vImageConvert_Planar8To16U

Converts a Planar8 image to a 16U image .

```
vImage_Error vImageConvert_Planar8To16U (
const vImage_Buffer *src,
const vImage_Buffer *dest,
vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The function converts from 8-bit to 16-bit values as follows:

```
uint16_t result = (srcPixel * 65535 + 127 ) / 255
```

You can also use this function to convert an ARGB8888 image to a 4-channel interleaved 16U image. Simply multiply the width of the destination buffer by four.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageConvert_Planar8toARGB1555

Combines four Planar8 images into one ARGB1555 image.

```
vImage_Error vImageConvert_Planar8toARGB1555 (
   const vImage_Buffer *srcA,
   const vImage_Buffer *srcR,
   const vImage_Buffer *srcG,
   const vImage_Buffer *srcB,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcA*
A pointer to vImage buffer structure that contains the Planar8 image to use as the alpha channel of the destination image.

*srcR*
A pointer to vImage buffer structure that contains the Planar8 image to use as the red channel of the destination image.

*srcG*
A pointer to vImage buffer structure that contains the Planar8 image to use as the green channel of the destination image.

*srcB*
A pointer to vImage buffer structure that contains the Planar8 image to use as the blue channel of the destination image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data in ARGB1555 format (16-bit pixels with 1 bit for alpha and 5 bits each for red, green, and blue). When you no longer need the data buffer, you must deallocate the memory. The destination buffer can be the same as the source buffer.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function calculates the 8-bit pixels in the destination image as follows:

```
Pixel8 alpha =  1bitAlphaChannel * 255
Pixel8 red   = (5bitRedChannel   * 255 + 15) / 31
Pixel8 green = (5bitGreenChannel * 255 + 15) / 31
Pixel8 blue  = (5bitBlueChannel  * 255 + 15) / 31
```

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageConvert_Planar8toARGB8888

Combines four Planar8 images into one ARGB8888 image.

```
vImage_Error vImageConvert_Planar8toARGB8888 (
   const vImage_Buffer *srcA,
   const vImage_Buffer *srcR,
   const vImage_Buffer *srcG,
   const vImage_Buffer *srcB,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcA*

A pointer to vImage buffer structure that contains the Planar8 image to use as the alpha channel of the destination image.

*srcR*

A pointer to vImage buffer structure that contains the Planar8 image to use as the red channel of the destination image.

*srcG*

A pointer to vImage buffer structure that contains the Planar8 image to use as the green channel of the destination image.

*srcB*

A pointer to vImage buffer structure that contains the Planar8 image to use as the blue channel of the destination image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image in ARGB8888 format. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The source and destination buffers must have the same height and width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Conversion.h`

## vImageConvert_Planar8toPlanarF

Converts a Planar8 image to a PlanarF image.

```
vImage_Error vImageConvert_Planar8toPlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   Pixel_F maxFloat,
   Pixel_F minFloat,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image converted to PlanarF format. When you no longer need the data buffer, you must deallocate the memory.

*maxFloat*

The maximum pixel value for the destination image.

*minFloat*

The minimum pixel value for the destination image.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function transforms a Planar8 image to a PlanarF image, using a *minFloat* value and a *maxFloat* value to specify the range of values for the PlanarF image. The function maps each source pixel value(which can be in the range of 0 to 255 inclusive) linearly into the range *minFloat* to *maxFloat*, using the following mapping (where `i` is the old pixel value):

```
new pixel value = i* (maxFloat - minFloat)/255.0f + minFloat
```

The two buffers must have the same number of rows and the same number of columns.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Conversion.h`

## vImageConvert_Planar8toRGB565

Combines three Planar8 images into one RGB565 image.

```
vImage_Error vImageConvert_Planar8toRGB565 (
   const vImage_Buffer *srcR,
   const vImage_Buffer *srcG,
   const vImage_Buffer *srcB,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcR*

A pointer to vImage buffer structure that contains the Planar8 image to use as the red channel of the destination image.

*srcG*

A pointer to vImage buffer structure that contains the Planar8 image to use as the green channel of the destination image.

*srcB*

A pointer to vImage buffer structure that contains the Planar8 image to use as the blue channel of the destination image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image in RGB565 format (16-bit pixels with 5 bits for red, 6 for green, and 5 for blue). When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The function calculates the pixels in the destination image as follows:

```
uint32_t red   = (8bitRedChannel   * (31*2) + 255) / (255*2)
uint32_t green = (8bitGreenChannel * 63 + 127) / 255
uint32_t blue  = (8bitBlueChannel  * 31 + 127) / 255
uint16_t RGB565pixel =  (red << 11) | (green <<  5) | blue
```

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageConvert_Planar8toRGB888

Combines three Planar8 images into one RGB888 image.

```
vImage_Error vImageConvert_Planar8toRGB888 (
   const vImage_Buffer *planarRed,
   const vImage_Buffer *planarGreen,
   const vImage_Buffer *planarBlue,
   const vImage_Buffer *rgbDest,
   vImage_Flags flags
);
```

**Parameters**

*planarRed*

A pointer to vImage buffer structure that contains the Planar8 image to use as the red channel of the destination image.

*planarGreen*

A pointer to vImage buffer structure that contains the Planar8 image to use as the green channel of the destination image.

*planarBlue*

A pointer to vImage buffer structure that contains the Planar8 image to use as the blue channel of the destination image.

*rgbDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image in RGB888 format. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The source and destination buffers must have the same height and width.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageConvert_PlanarFtoARGBFFFF

Combines four PlanarF images into one ARGBFFFF image.

```
vImage_Error vImageConvert_PlanarFtoARGBFFFF (
   const vImage_Buffer *srcA,
   const vImage_Buffer *srcR,
   const vImage_Buffer *srcG,
   const vImage_Buffer *srcB,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*srcA*

A pointer to vImage buffer structure that contains the PlanarF image to use as the alpha channel of the destination image.

*srcR*

A pointer to vImage buffer structure that contains the PlanarF image to use as the red channel of the destination image.

*srcG*

A pointer to vImage buffer structure that contains the PlanarF image to use as the green channel of the destination image.

*srcB*

A pointer to vImage buffer structure that contains the PlanarF image to use as the blue channel of the destination image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data in ARGBFFFF format. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The source and destination buffers must have the same height and width.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Conversion.h`

## vImageConvert_PlanarFtoPlanar16F

Converts a PlanarF image to a Planar16F image.

```
vImage_Error vImageConvert_PlanarFtoPlanar16F (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the destination image converted to Planar16F format. The destination buffer can be the same as the source buffer. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

Planar16F pixels are 16-bit floating-point numbers, conforming to the OpenEXR standard. Denormals, NaNs. and +/– Infinity are supported. In conformance with IEEE-754, all signaling NaNs are quieted during the conversion (OpenEXR-1.2.1 does not do this.)

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Conversion.h`

## vImageConvert_PlanarFtoPlanar8

Converts a PlanarF image to a Planar8 image, clipping values to the provided minimum and maximum values.

```
vImage_Error vImageConvert_PlanarFtoPlanar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   Pixel_F maxFloat,
   Pixel_F minFloat,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the destination image converted to Planar8 format. When you no longer need the data buffer, you must deallocate the memory.

*maxFloat*

A maximum pixel value. The function clips larger values to this value in the destination image.

*maxFloat*

A minimum pixel value. The function clips smaller values to this value in the destination image.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The minimum and maximum value determine the mapping of intensity values to the destination image. The mapping is:

```
if oldPixel < minFloat
newPixel = 0

if minfloat <= oldPixel <= maxFloat
newPixel = (oldPixel - minFloat) * 255.0f / (maxFloat - minFloat)

if oldPixel > maxFloat
newPixel = 255
```

The source and destination buffers must have the same height and width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Conversion.h`

## vImageConvert_PlanarFtoRGBFFF

Combines three PlanarF images into one RGBFFF image.

```
vImage_Error vImageConvert_PlanarFtoRGBFFF (
   const vImage_Buffer *planarRed,
   const vImage_Buffer *planarGreen,
   const vImage_Buffer *planarBlue,
   const vImage_Buffer *rgbDest,
   vImage_Flags flags
);
```

**Parameters**

*planarRed*

A pointer to vImage buffer structure that contains the PlanarF image to use as the red channel of the destination image.

*planarGreen*

A pointer to vImage buffer structure that contains the PlanarF image to use as the green channel of the destination image.

*planarBlue*

A pointer to vImage buffer structure that contains the PlanarF image to use as the blue channel of the destination image.

*rgbDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image in RGBFFF format. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The source and destination buffers must have the same height and width.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageConvert_PlanarToChunky8

Combines a collection of planar source images into a single interleaved destination image, with one 8-bit channel for each planar image.

```
vImage_Error vImageConvert_PlanarToChunky8 (
   const vImage_Buffer *srcPlanarBuffers[],
   void *destChannels[],
   unsigned int channelCount,
   size_t destStrideBytes,
   vImagePixelCount destWidth,
   vImagePixelCount destHeight,
   size_t destRowBytes,
   vImage_Flags flags
);
```

**Parameters**

*srcPlanarBuffers*

An array of vImage buffer structures, each of which contains image data in Planar8 format. Each structure must have the same width and height values, but may have different row byte values.

*destChannels*

An array of pointers to channels of the destination image. Each pointer points to the start of the data for one destination channel. The function fills the pixel values of each channel, using the corresponding source image for each channel.

*channelCount*

The number of vImage buffer structures in the *srcPlanarBuffers* array and the number of channels in the destination image.

*destStrideBytes*

The number of bytes from one pixel value in a given channel to the next pixel of that channel (within a row). This value is used for all channels.

*destWidth*

The number of pixels in a row. This value is used for all channels. It must be the same as the width of each of the planar source images.

*destHeight*

The number of rows. This value will be used for all channels. It must be the same as the height of each of the planar source images.

*destRowBytes*

The number of bytes from the beginning of a channel row to the beginning of the next row in that channel. This value is used for all channels. (It does not have to be the same as the *rowBytes* values of the source buffers. Each source buffer can have its own *rowBytes* value.)

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Conversion.h`

## vImageConvert_PlanarToChunkyF

Combines a collection of planar source images into a single interleaved destination image, with one floating-point channel for each planar image.

```
vImage_Error vImageConvert_PlanarToChunkyF (
   const vImage_Buffer *srcPlanarBuffers[],
   void *destChannels[],
   unsigned int channelCount,
   size_t destStrideBytes,
   vImagePixelCount destWidth,
   vImagePixelCount destHeight,
   size_t destRowBytes,
   vImage_Flags flags
);
```

**Parameters**

*srcPlanarBuffers*

An array of vImage buffer structures, each of which contains image data in PlanarF format. Each structure must have the same width and height values, but may have different row byte values.

*destChannels*

An array of pointers to channels of the destination image. Each pointer points to the start of the data for one destination channel. The function fills the pixel values of each channel, using the corresponding source image for each channel.

*channelCount*

The number of vImage buffer structures in the *srcPlanarBuffers* array and the number of channels in the destination image.

*destStrideBytes*

The number of bytes from one pixel value in a given channel to the next pixel of that channel (within a row). This value is used for all channels.

*destWidth*

The number of pixels in a row. This value is used for all channels. It must be the same as the width of each of the planar source images.

*destHeight*

The number of rows. This value is used for all channels. It must be the same as the height of each of the planar source images.

*destRowBytes*

The number of bytes from the beginning of a channel row to the beginning of the next row in that channel. This value is used for all channels. (It does not have to be the same as the *rowBytes* values of the source buffers. Each source buffer can have its own *rowBytes* value.)

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Conversion.h`

## vImageConvert_RGB565toARGB8888

Converts an RGB565 image into an ARGB8888 image, using the provided 8-bit alpha value.

```
vImage_Error vImageConvert_RGB565toARGB8888 (
   Pixel_8 alpha,
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*alpha*

A value of type `Pixel_8` to be used as the alpha value for all pixels in the destination image.

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the destination data converted to ARGB8888 format.. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The RGB565 format has 16-bit pixels with 5 bits for red, 6 for green, and 5 for blue. The function calculates the pixels in the destination image as follows:

```
Pixel8 alpha = alpha
Pixel8 red   = (5bitRedChannel   * 255 + 15) / 31
Pixel8 green = (6bitGreenChannel * 255 + 31) / 63
Pixel8 blue  = (5bitBlueChannel  * 255 + 15) / 31
```

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Conversion.h`

## vImageConvert_RGB565toPlanar8

Separates an RGB565 image into three Planar8 images.

```
vImage_Error vImageConvert_RGB565toPlanar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *destR,
   const vImage_Buffer *destG,
   const vImage_Buffer *destB,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to separate.

*destR*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the red channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destG*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the green channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*destB*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the blue channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The RGB565 format has 16-bit pixels with 5 bits for red, 6 for green, and 5 for blue. The function calculates the pixels in the destination image as follows:

```
Pixel8 red   = (5bitRedChannel   * 255 + 15) / 31
Pixel8 green = (6bitGreenChannel * 255 + 31) / 63
Pixel8 blue  = (5bitBlueChannel  * 255 + 15) / 31
```

This function works in place for one destination buffer. You must allocate the others separately.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Conversion.h`

## vImageConvert_RGB888toARGB8888

Converts an RGB888 image into an ARGB8888 image, using the provided alpha value (either as planar or pixel data).

```
vImage_Error vImageConvert_RGB888toARGB8888 (
   const vImage_Buffer *rgbSrc,
   const vImage_Buffer *aSrc,
   Pixel_8 alpha,
   const vImage_Buffer *argbDest,
   bool premultiply,
   vImage_Flags flags
);
```

**Parameters**

*rgbSrc*

A pointer to a vImage buffer structure that contains the source image whose data you want to convert.

*aSrc*

A pointer to a vImage buffer structure that contains a Planar8 alpha plane to use as the alpha values for in the destination image. If you pass `NULL`, the function assigns the value of the `alpha` parameter for all pixels in the destination image.

*alpha*

An alpha value for all pixels in the destination image. The function ignores this value if the `aSrc` parameter is not `NULL`.

*argbDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the converted image data. When you no longer need the data buffer, you must deallocate the memory.

*premultiply*

Pass `YES` if the data is premultiplied by the alpha value; `NO` otherwise.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If you specify premultiplied data, the function calculates each channel in the destination image as follows:

```
(alpha * sourceValue + 127) / 255
```

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Conversion.h`

## vImageConvert_RGB888toPlanar8

Separates an RGB888 image into three Planar8 images.

```
vImage_Error vImageConvert_RGB888toPlanar8 (
   const vImage_Buffer *rgbSrc,
   const vImage_Buffer *redDest,
   const vImage_Buffer *greenDest,
   const vImage_Buffer *blueDest,
   vImage_Flags flags
);
```

**Parameters**

*rgbSrc*

A pointer to a vImage buffer structure that contains the source image whose data you want to separate.

*redDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the red channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*greenDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the green channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*blueDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a Planar8 image equivalent to the blue channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The source image and the destination buffers, must all have the same height and the same width. This function works in place for one destination buffer. You must allocate the others separately.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Conversion.h`

## vImageConvert_RGBFFFtoPlanarF

Separates an RGBFFF image into three PlanarF images.

```
vImage_Error vImageConvert_RGBFFFtoPlanarF (
   const vImage_Buffer *rgbSrc,
   const vImage_Buffer *redDest,
   const vImage_Buffer *greenDest,
   const vImage_Buffer *blueDest,
   vImage_Flags flags
);
```

**Parameters**

*rgbSrc*

A pointer to a vImage buffer structure that contains the source image whose data you want to separate.

*redDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a PlanarF image equivalent to the red channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*greenDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a PlanarF image equivalent to the green channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*blueDest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains data for a PlanarF image equivalent to the blue channel of the source image. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The source image and the destination buffers, must all have the same height and the same width. This function works in place for one destination buffer. You must allocated the others separately.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Conversion.h`

## vImageFlatten_ARGB8888ToRGB888

Transforms an ARGB8888 image to an RGB888 image against an opaque background of the provided color.

```
vImage_Error vImageFlatten_ARGB8888ToRGB888 (
const vImage_Buffer *argb8888Src,
const vImage_Buffer *rgb888dest,
Pixel_8888 backgroundColor,
bool isImagePremultiplied,
vImage_Flags flags
);
```

**Parameters**

*argb8888Src*

A pointer to a vImage buffer structure that contains the source image whose data you want to flatten.

*rgb888dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `argb8888Src` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The destination buffer can be the same as the source buffer.

*backgroundColor*

An 8-bit interleaved pixel value.

*isImagePremultiplied*

`TRUE` if the source image uses premultiplied data , `FALSE` otherwise.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If the source image uses premultiplied data, the function calculates each channel value for a pixel in the destination image as follows (where i is the source value for the channel):

```
new value = (i * 255 + (255 - alpha) * backgroundColor + 127)  / 255
```

If the source image does not use premultiplied data, the function calculates each channel value for a pixel in the destination image as follows (where i is the source value for the channel):

```
new value = (i * alpha + (255 - alpha) * backgroundColor + 127)  / 255
```

The source and destinations buffer must have the same height and the same width.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Conversion.h`

## vImageFlatten_ARGBFFFFToRGBFFF

Transforms an ARGBFFFF image to an RGBFFF image against an opaque background of the provided color.

```
vImage_Error vImageFlatten_ARGBFFFFToRGBFFF (
const vImage_Buffer *argbFFFFSrc,
const vImage_Buffer *rgbFFFdest,
Pixel_FFFF backgroundColor,
bool isImagePremultiplied,
vImage_Flags flags
);
```

**Parameters**

*argbFFFFSrc*

A pointer to a vImage buffer structure that contains the source image whose data you want to flatten.

*rgbFFFdest*
: A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `argbFFFFSrc` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The destination buffer can be the same as the source buffer.

*backgroundColor*
: A floating-point interleaved pixel value.

*isImagePremultiplied*
: True if the source image is premultiplied, false otherwise.

*flags*
: The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If the source image uses premultiplied data, the function calculates each channel value for a pixel in the destination image as follows (where i is the source value for the channel):

```
newcolor = i + (1.0f - alpha) * backgroundColor
```

If the source image does not use premultiplied data, the function calculates each channel value for a pixel in the destination image as follows (where i is the source value for the channel):

```
newcolor = i * alpha + (1.0f - alpha) * backgroundColor
```

The source and destinations buffer must have the same height and the same width.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageOverwriteChannelsWithPixel_ARGB8888

Overwrites an ARGB8888 image buffer with the provided pixel value.

```
vImage_Error vImageOverwriteChannelsWithPixel_ARGB8888 (
   const Pixel_8888 the_pixel,
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   uint8_t copyMask,
   vImage_Flags flags
);
```

**Parameters**

*the_pixel*
: An ARGB pixel value.

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to overwrite.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `origSrc` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*copyMask*

An output value that selects the plane (or planes) from the ARGB8888 source buffer that you want replaced with the pixel value. The value `0x8` selects the alpha channel, `0x4` the red channel, `0x2` the green channel, and `0x1` the blue channel. You can add these values together to select multiple channels.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

`Conversion.h`

## vImageOverwriteChannelsWithPixel_ARGBFFFF

Overwrites an ARGBFFFF image buffer with the provided pixel value.

```
vImage_Error vImageOverwriteChannelsWithPixel_ARGBFFFF (
   const Pixel_FFFF the_pixel,
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   uint8_t copyMask,
   vImage_Flags flags
);
```

**Parameters**

*the_pixel*

An ARGB pixel value.

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to overwrite.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `origSrc` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*copyMask*

An output value that selects the plane (or planes) from the ARGBFFFF source buffer that you want replaced with the pixel value. The value `0x8` selects the alpha channel, `0x4` the red channel, `0x2` the green channel, and `0x1` the blue channel. You can add these values together to select multiple channels.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

`Conversion.h`

## vImageOverwriteChannelsWithScalar_ARGB8888

Overwrites the pixels of one or more planes of an ARGB8888 image buffer with the provided scalar value.

```
vImage_Error vImageOverwriteChannelsWithScalar_ARGB8888 (
   Pixel_8 scalar,
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   uint8_t copyMask,
   vImage_Flags flags
);
```

**Parameters**

*scalar*

An 8-bit pixel value.

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to overwrite.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `origSrc` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*copyMask*

An output value that selects the plane (or planes) from the ARGB8888 source buffer that you want replaced with the scalar value. The value `0x8` selects the alpha channel, `0x4` the red channel, `0x2` the green channel, and `0x1` the blue channel. You can add these values together to select multiple channels.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageOverwriteChannelsWithScalar_ARGBFFFF

Overwrites the pixels of one or more planes of an ARGBFFFF image buffer with the provided scalar value.

```
vImage_Error vImageOverwriteChannelsWithScalar_ARGBFFFF (
   Pixel_F scalar,
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   uint8_t copyMask,
   vImage_Flags flags
);
```

**Parameters**

*scalar*

A floating-point pixel value.

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to overwrite.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `origSrc` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*copyMask*

An output value that selects the plane (or planes) from the ARGBFFFF source buffer that you want replaced with the scalar value. The value `0x8` selects the alpha channel, `0x4` the red channel, `0x2` the green channel, and `0x1` the blue channel. You can add these values together to select multiple channels.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageOverwriteChannelsWithScalar_Planar8

Overwrites a Planar8 image buffer with the provided value.

```
vImage_Error vImageOverwriteChannelsWithScalar_Planar8 (
   Pixel_8 scalar,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*scalar*

An 8-bit pixel value.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

Set the `kvImageDoNotTile` field in the flags parameter to prevent vImage from using tiling internally. (This is appropriate if you are doing tiling yourself.)

### Return Value

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

### Availability

Available in Mac OS X v10.4 and later.

### Declared In

`Conversion.h`

## vImageOverwriteChannelsWithScalar_PlanarF

Overwrites a PlanarF image buffer with the provided value.

```
vImage_Error vImageOverwriteChannelsWithScalar_PlanarF (
   Pixel_F scalar,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*scalar*

A floating-point pixel value.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageOverwriteChannels_ARGB8888

Overwrites one or more planes of an ARGB8888 image buffer with the provided planar buffer.

```
vImage_Error vImageOverwriteChannels_ARGB8888 (
   const vImage_Buffer *newSrc,
   const vImage_Buffer *origSrc,
   const vImage_Buffer *dest,
   uint8_t copyMask,
   vImage_Flags flags
);
```

**Parameters**

*newSrc*

A pointer to a vImage buffer structure that contains the data, in Planar8 format, for overwriting the `origSrc` image data.

*origSrc*

A pointer to a vImage buffer structure that contains the source image whose data you want to overwrite.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `origSrc` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*copyMask*

An output value that selects the plane (or planes) from the ARGB8888 source buffer that you want replaced with data from the Planar8 source buffer. The value `0x8` selects the alpha channel, `0x4` the red channel, `0x2` the green channel, and `0x1` the blue channel. You can add these values together to select multiple channels.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The function overwrites pixel values in the `origSrc` image buffer using the corresponding pixel value from the `newSrc` image buffer.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageOverwriteChannels_ARGBFFFF

Overwrites one or more planes of an ARGBFFFF image buffer with the provided planar buffer.

```
vImage_Error vImageOverwriteChannels_ARGBFFFF (
   const vImage_Buffer *newSrc,
   const vImage_Buffer *origSrc,
   const vImage_Buffer *dest,
   uint8_t copyMask,
   vImage_Flags flags
);
```

**Parameters**

*newSrc*

A pointer to a vImage buffer structure that contains the data, in PlanarF format, for overwriting the `origSrc` image data.

*origSrc*

A pointer to a vImage buffer structure that contains the source image whose data you want to overwrite.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `origSrc` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*copyMask*

An output value that selects the plane (or planes) from the ARGBFFFF source buffer that you want replaced with data from the Planar8 source buffer. The value `0x8` selects the alpha channel, `0x4` the red channel, `0x2` the green channel, and `0x1` the blue channel. You can add these values together to select multiple channels.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The function overwrites pixel values in the `origSrc` image buffer using the corresponding pixel value from the `newSrc` image buffer.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImagePermuteChannels_ARGB8888

Reorders the channels in an ARGB8888 image.

```
vImage_Error vImagePermuteChannels_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const uint8_t permuteMap[4],
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to permute.

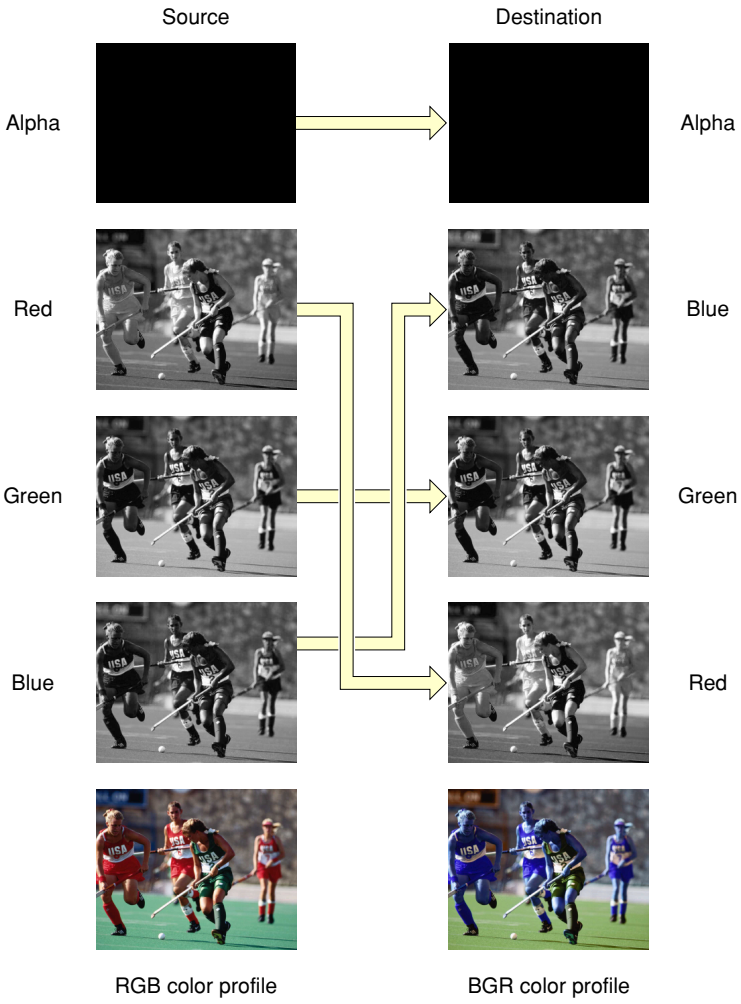
*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*permuteMap*

An array of four 8-bit integers with the values 0, 1, 2, and 3, in some order. The ith value specifies the plane from the source image that you want copied to the ith plane of the destination image. 0 denotes the alpha channel, 1 the red channel, 2 the green channel, and 3 the blue channel. The following figure shows the result of using a permute map shows values are (0, 3, 2, 1). The data in the alpha and green channels remain the same, but the data in the source red channel maps to the destination blue channel while the data in the source blue channel maps to the destination red channel.

**Figure 3-1** Permuting the red and blue channels




*flags*

The options to use when performing the permutation operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImagePermuteChannels_ARGBFFFF

Reorders the channels in an ARGBFFFF image.

```
vImage_Error vImagePermuteChannels_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const uint8_t permuteMap[4],
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to permute.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*permuteMap*

An array of four 8-bit integers with the values 0, 1, 2, and 3, in some order. The ith value specifies the plane from the source image that will be copied to the ith plane of the destination image. 0 denotes the alpha channel, 1 the red channel, 2 the green channel, and 3 the blue channel. Figure 3-1 (page 101) shows the result of using a permute map shows values are (0, 3, 2, 1). The data in the alpha and green channels remain the same, but the data in the source red channel maps to the destination blue channel while the data in the source blue channel maps to the destination red channel.

*flags*

The options to use when performing the permutation operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Conversion.h`

## vImageSelectChannels_ARGB8888

Overwrites the specified channels in an ARGB8888 image buffer with the provided channels from an ARGB8888 image buffer.

```
vImage_Error vImageSelectChannels_ARGB8888 (
   const vImage_Buffer *newSrc,
   const vImage_Buffer *origSrc,
   const vImage_Buffer *dest,
   uint8_t copyMask,
   vImage_Flags flags
);
```

**Parameters**

*newSrc*

A pointer to a vImage buffer structure that contains the data, in ARGB8888 format, for overwriting the `origSrc` image data.

*origSrc*

A pointer to a vImage buffer structure that contains the source image whose data you want to overwrite.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `origSrc` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*copyMask*

An output value that selects the plane (or planes) from the ARGB8888 source buffer that you want replaced with the corresponding plane from the `newSrc` image buffer. The value `0x8` selects the alpha channel, `0x4` the red channel, `0x2` the green channel, and `0x1` the blue channel. You can add these values together to select multiple channels.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.5 and later.

**See Also**

`vImageOverwriteChannels_ARGB8888` (page 98)

**Declared In**

`Conversion.h`

## vImageSelectChannels_ARGBFFFF

Overwrites the specified channels in an ARGBFFFF image buffer with the provided channels in an ARGBFFFF image buffer.

```
vImage_Error vImageSelectChannels_ARGBFFFF (
   const vImage_Buffer *newSrc,
   const vImage_Buffer *origSrc,
   const vImage_Buffer *dest,
   uint8_t copyMask,
   vImage_Flags flags
);
```

**Parameters**

*newSrc*

A pointer to a vImage buffer structure that contains the data, in ARGBFFFF format, for overwriting the `origSrc` image data.

*origSrc*

A pointer to a vImage buffer structure that contains the source image whose data you want to overwrite.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `origSrc` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*copyMask*

An output value that selects the plane (or planes) from the ARGBFFFF source buffer that you want replaced with the corresponding plane from the `newSrc` image buffer. The value `0x8` selects the alpha channel, `0x4` the red channel, `0x2` the green channel, and `0x1` the blue channel. You can add these values together to select multiple channels.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

`Conversion.h`

## vImageTableLookUp_ARGB8888

Transforms an ARGB8888 image by substituting pixel values with pixel values provided by four lookup tables.

```
vImage_Error vImageTableLookUp_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const Pixel_8 alphaTable[256],
   const Pixel_8 redTable[256],
   const Pixel_8 greenTable[256],
   const Pixel_8 blueTable[256],
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to transform.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*alphaTable*

The lookup table to use for the alpha channel of the source image. If you pass `NULL` for this table, the function copies the alpha channel unchanged to the destination buffer.

*redTable*

The lookup table to use for the red channel of the source image. If you pass `NULL` for this table, the function copies the red channel unchanged to the destination buffer.

*greenTable*

The lookup table to use for the green channel of the source image. If you pass `NULL` for this table, the function copies the green channel unchanged to the destination buffer.

*blueTable*

The lookup table to use for the blue channel of the source image. If you pass `NULL` for this table, the function copies the blue channel unchanged to the destination buffer.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The function separates each pixel into four channels—alpha, red, green, and blue. It substitutes each channel separately, using the appropriate table. Then the function recombines each pixel into a single ARGB8888 value, placing the transformed image into the destination buffer.

The source and destinations buffer must have the same height and the same width.

You cannot use this function to perform an arbitrary color mapping. For example, if two different source colors have the same green component, you must map them to two destination colors whose green components are equal. You can't map them to two arbitrary colors.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**
`Conversion.h`

## vImageTableLookUp_Planar8

Transforms an Planar8 image by substituting pixel values with pixel values provided by four lookup tables.

```
vImage_Error vImageTableLookUp_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const Pixel_8 table[256],
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to transform.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. The destination data buffer can be the same as the `src` data buffer. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*table*

The lookup table to use.

*flags*

The options to use when performing this operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function transforms a Planar8 image by replacing all pixels of a given intensity value with pixels of a new intensity value. It maps old values to new values using the provided 256-element lookup table (LUT).

The source and destinations buffer must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Conversion.h`

# vImage Convolution Reference

| | |
|---|---|
| **Framework:** | Accelerate/vImage |
| **Declared in** | Convolution.h |
| **Companion guide** | vImage Programming Guide |

## Overview

Convolution functions implement various techniques for smoothing or sharpening an image by replacing a pixel with a weighted sum of itself and nearby pixels. Image convolution does not alter the size of an image.

Each convolution function requires that you pass it a convolution kernel, which determines how the values of neighboring pixels are used to compute the value of a destination pixel. A kernel is a packed array, without padding at the ends of the rows. The elements of the array must be of type `uint8_t` (for the Planar8 and ARGB8888 formats) or of type `float` (for the PlanarF and ARGBFFFF formats). The height and the width of the array must both be odd numbers.

For example, a 3 x 3 convolution kernel for a Planar8 image consist of nine 8-bit (1-byte) values, arranged consecutively. The first three values represent the first row of the kernel, the next three values the second row, and the last three values the third row.

Typically, you use normalized values for the convolution kernel. For floating-point formats, this means the sum of the elements of the kernel is 1.0. For integer formats, the sum of the elements of the kernel, divided by the given divisor, is 1. A non-normalized kernel either lightens or darkens the image.

For integer formats, the sum of any subset of elements of the kernel must be in the range $-2^{24}$ to $2^{24} - 1$, inclusive to prevent integer overflow. If your kernel does not meet this restriction, either use a floating-point format or scale the kernel to use smaller values.

A convolution function transforms a source image as follows:

1. Places the kernel over the image so that the center element of the kernel lies over the source pixel.
2. For floating-point formats, performs this calculation:

$$\sum kernel_{(x,y)} * pixel_{(x,y)}$$

For integer formats, performs this calculation:

$$\frac{\sum kernel_{(x,y)} * pixel_{(x,y)}}{M * N}$$

3. Assigns the result to the destination pixel.

If the image is in a planar format, the convolution operation uses the single-channel values of the pixels directly. If the image is in an interleaved format, the convolution operation processes each channel (alpha, red, green, and blue) separately. In both the planar and interleaved format, the kernel itself is always planar.

When the pixel to be transformed is near the edge of the image—not merely the region of interest, but the entire image of which it is a part—the kernel may extend beyond the edge of the image, so that there are no existing pixels beneath some of the kernel elements. In these cases you must pass a flag that specifies a technique for the convolution function to use: `kvImageCopyInPlace`, `kvImageBackgroundColorFill`, `kvImageEdgeExtend`, and `kvImageTruncateKernel`. For a discussion of these options, see *vImage Data Types and Constants Reference*.

# Functions by Task

## Deconvolving

`vImageRichardsonLucyDeConvolve_ARGBFFFF` (page 133)
Sharpens an ARGBFFFF image by undoing a previous convolution that blurred the image, such as diffraction effects in a camera lens.

`vImageRichardsonLucyDeConvolve_ARGB8888` (page 131)
Sharpens an ARGB8888 image by undoing a previous convolution that blurred the image, such as diffraction effects in a camera lens.

`vImageRichardsonLucyDeConvolve_PlanarF` (page 137)
Sharpens a PlanarF image by undoing a previous convolution that blurred the image, such as diffraction effects in a camera lens.

`vImageRichardsonLucyDeConvolve_Planar8` (page 135)
Sharpens a Planar8 image by undoing a previous convolution that blurred the image, such as diffraction effects in a camera lens.

## Convolving Without Bias

`vImageConvolve_ARGBFFFF` (page 125)
Convolves a region of interest within an ARGBFFFF source image by an M x N kernel.

`vImageConvolve_ARGB8888` (page 124)
Convolves a region of interest within a source image by an M x N kernel, then divides the pixel values by a divisor.

vImageConvolve_PlanarF (page 129)
Convolves a region of interest within a source image by an M x N kernel.

vImageConvolve_Planar8 (page 127)
Convolves a region of interest within a source image by an M x N kernel, then divides the pixel values by a divisor.

## Convolving With a Bias

vImageConvolveWithBias_ARGB8888 (page 117)
Convolves a region of interest within an ARGB8888 source image by an M x N kernel, then normalizes the pixel values.

vImageConvolveWithBias_PlanarF (page 122)
Convolves a region of interest within a PlanarF source image by an M x N kernel.

vImageConvolveWithBias_Planar8 (page 120)
Convolves a region of interest within a Planar8 source image by an M x N kernel, then normalizes the pixel values.

vImageConvolveWithBias_ARGBFFFF (page 118)
Convolves a region of interest within an ARGBFFFF source image by an M x N kernel.

## Convolving With Multiple Kernels

vImageConvolveMultiKernel_ARGBFFFF (page 115)
Convolves each channel of a region of interest within an ARGBFFFF source image by one of the four M x N kernels.

vImageConvolveMultiKernel_ARGB8888 (page 113)
Convolves each channel of a region of interest within an ARGB8888 source image by one of the four M x N kernels, then divides the pixel values by one of the four divisors.

## Convolving With High-Speed Box and Tent Filters

vImageBoxConvolve_Planar8 (page 111)
Convolves a region of interest within a Planar8 source image by an implicit M x N kernel that has the effect of a box filter.

vImageBoxConvolve_ARGB8888 (page 110)
Convolves a region of interest within an ARGB8888 source image by an implicit M x N kernel that has the effect of a box filter.

vImageTentConvolve_Planar8 (page 141)
Convolves a region of interest within a Planar8 source image by an implicit M x N kernel that has the effect of a tent filter.

vImageTentConvolve_ARGB8888 (page 139)
Convolves a region of interest within an ARGB8888 source image by an implicit M x N kernel that has the effect of a tent filter.

### Getting the Minimum Buffer Size

`vImageGetMinimumTempBufferSizeForConvolution` (page 130) Deprecated in Mac OS X v10.4

Returns the minimum size, in bytes, for the temporary buffer that the caller supplies to any of the convolution functions. (Deprecated. Use the `kvImageGetTempBufferSize` flag with the appropriate convolution function instead of calling this function.)

# Functions

## vImageBoxConvolve_ARGB8888

Convolves a region of interest within an ARGB8888 source image by an implicit M x N kernel that has the effect of a box filter.

```
vImage_Error vImageBoxConvolve_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   uint32_t kernel_height,
   uint32_t kernel_width,
   Pixel_8888 backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*backgroundColor*
A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*
The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function uses an implicit divisor and an implicit kernel of specified size instead of a kernel provided by the caller.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageConvolve_ARGB8888` (page 124)
`vImageTentConvolve_ARGB8888` (page 139)

**Declared In**
`Convolution.h`

## vImageBoxConvolve_Planar8

Convolves a region of interest within a Planar8 source image by an implicit M x N kernel that has the effect of a box filter.

```
vImage_Error vImageBoxConvolve_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   uint32_t kernel_height,
   uint32_t kernel_width,
   Pixel_8 backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function uses an implicit divisor and an implicit kernel of specified size instead of a kernel provided by the caller.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageConvolve_Planar8` (page 127)
`vImageTentConvolve_Planar8` (page 141)

**Declared In**
`Convolution.h`

## vImageConvolveMultiKernel_ARGB8888

Convolves each channel of a region of interest within an ARGB8888 source image by one of the four M x N kernels, then divides the pixel values by one of the four divisors.

```
vImage_Error vImageConvolveMultiKernel_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const int16_t *kernels[4],
   uint32_t kernel_height,
   uint32_t kernel_width,
   const int32_t divisors[4],
   const int32_t biases[4],
   Pixel_8888 backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernels*

An array of pointers to the data for four kernels. The first kernel is for the alpha channel, the second for red, the third for green, and the fourth for blue. The data for each kernel is a packed array of integer values.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*divisors*

An array of values, for normalization purposes, to divide into the convolution results. Supply one value for each channel.

*biases*

An array of four values to be added to each element of the convolution result for one channel, before clipping. The first value is for the alpha channel, the second for red, the third for green, and the fourth for blue

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageConvolveWithBias_ARGB8888` (page 117)

**Declared In**
`Convolution.h`

## vImageConvolveMultiKernel_ARGBFFFF

Convolves each channel of a region of interest within an ARGBFFFF source image by one of the four M x N kernels.

```
vImage_Error vImageConvolveMultiKernel_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernels[4],
   uint32_t kernel_height,
   uint32_t kernel_width,
   const float biases[4],
   Pixel_FFFF backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernels*

An array of pointers to the data for four kernels. The first kernel is for the alpha channel, the second for red, the third for green, and the fourth for blue. The data for each kernel is a packed array of floating-point values.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*biases*

An array of four values to be added to each element of the convolution result for one channel, before clipping. The first value is for the alpha channel, the second for red, the third for green, and the fourth for blue

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
vImageConvolveWithBias_ARGBFFFF (page 118)

**Declared In**
Convolution.h

## vImageConvolveWithBias_ARGB8888

Convolves a region of interest within an ARGB8888 source image by an M x N kernel, then normalizes the pixel values.

```
vImage_Error vImageConvolveWithBias_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const int16_t *kernel,
   uint32_t kernel_height,
   uint32_t kernel_width,
   int32_t divisor,
   int32_t bias,
   Pixel_8888 backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the convolution kernel data, which must be a packed array without any padding.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*divisor*
The value, for normalization purposes, to divide into the convolution results.

*bias*
The value to add to each element in the convolution result, before applying the divisor or performing any clipping.

*backgroundColor*
A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*
The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageConvolve_ARGB8888` (page 124)

**Declared In**
`Convolution.h`

## vImageConvolveWithBias_ARGBFFFF

Convolves a region of interest within an ARGBFFFF source image by an M x N kernel.

```
vImage_Error vImageConvolveWithBias_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   uint32_t kernel_height,
   uint32_t kernel_width,
   float bias,
   Pixel_FFFF backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the convolution kernel data, which must be a packed array without any padding.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*bias*

The value to add to each element in the convolution result, before performing any clipping.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageConvolve_ARGBFFFF` (page 125)

**Declared In**
`Convolution.h`

## vImageConvolveWithBias_Planar8

Convolves a region of interest within a Planar8 source image by an M x N kernel, then normalizes the pixel values.

```
vImage_Error vImageConvolveWithBias_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const int16_t *kernel,
   uint32_t kernel_height,
   uint32_t kernel_width,
   int32_t divisor,
   int32_t bias,
   Pixel_8 backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the convolution kernel data, which must be a packed array without any padding.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*divisor*

The value, for normalization purposes, to divide into the convolution results.

*bias*

The value to add to each element in the convolution result, before applying the divisor or performing any clipping.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageConvolve_Planar8` (page 127)

**Declared In**
`Convolution.h`

## vImageConvolveWithBias_PlanarF

Convolves a region of interest within a PlanarF source image by an M x N kernel.

```
vImage_Error vImageConvolveWithBias_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   uint32_t kernel_height,
   uint32_t kernel_width,
   float bias,
   Pixel_F backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the convolution kernel data, which must be a packed array without any padding.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*bias*

The value to add to each element in the convolution result, before performing any clipping.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageConvolve_PlanarF` (page 129)

**Declared In**
`Convolution.h`

## vImageConvolve_ARGB8888

Convolves a region of interest within a source image by an M x N kernel, then divides the pixel values by a divisor.

```
vImage_Error vImageConvolve_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const int16_t *kernel,
   uint32_t kernel_height,
   uint32_t kernel_width,
   int32_t divisor,
   Pixel_8888 backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*
A pointer to the convolution kernel data, which must be a packed array without any padding.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*divisor*
A value to divide the results of the convolution by. This is commonly used for normalization.

*backgroundColor*
A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*
The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Convolution.h`

## vImageConvolve_ARGBFFFF

Convolves a region of interest within an ARGBFFFF source image by an M x N kernel.

```
vImage_Error vImageConvolve_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   uint32_t kernel_height,
   uint32_t kernel_width,
   Pixel_FFFF backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image. offsets to a point within the source image to define the upper left-hand point of the region of interest.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the convolution kernel data, which must be a packed array without any padding.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Convolution.h`

## vImageConvolve_Planar8

Convolves a region of interest within a source image by an M x N kernel, then divides the pixel values by a divisor.

```
vImage_Error vImageConvolve_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const int16_t *kernel,
   uint32_t kernel_height,
   uint32_t kernel_width,
   int32_t divisor,
   Pixel_8 backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the convolution kernel data, which must be a packed array without any padding.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*divisor*

A value to divide the results of the convolution with. This is commonly used for normalization.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Convolution.h`

## vImageConvolve_PlanarF

Convolves a region of interest within a source image by an M x N kernel.

```
vImage_Error vImageConvolve_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   uint32_t kernel_height,
   uint32_t kernel_width,
   Pixel_F backgroundColor,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the convolution kernel data, which must be a packed array without any padding.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Convolution.h`

## vImageGetMinimumTempBufferSizeForConvolution

Returns the minimum size, in bytes, for the temporary buffer that the caller supplies to any of the convolution functions. (Deprecated in Mac OS X v10.4. Use the `kvImageGetTempBufferSize` flag with the appropriate convolution function instead of calling this function.)

```
size_t vImageGetMinimumTempBufferSizeForConvolution (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   uint32_t kernel_height,
   uint32_t kernel_width,
   vImage_Flags flags,
   size_t bytesPerPixel
);
```

**Parameters**

*src*

A pointer to the vImage buffer structure that you plan to pass to the convolution function.

*dest*

A pointer to the vImage buffer structure that you plan to pass to the convolution function.

*kernel_height*

The height, in pixels, of the kernel that you plan to use in the convolution function.

*kernel_width*
The width, in pixels, of the kernel that you plan to use in the convolution function.

*flags*
The flags that you plan to pass to the convolution function.

*bytesPerPixel*
The number of bytes in a pixel. Make sure to pass the value appropriate for the format of the pixel.

**Return Value**
The minimum size, in bytes, of the temporary buffer.

**Discussion**
This function does not depend on the *data* or *rowBytes* fields of the *src* or *dest* parameters; it only uses the *height* and *width* fields from those parameters. If the size of the images you are processing stay the same, then the required size of the buffer also stays the same. More specifically, if, between two calls to `vImageGetMinimumTempBufferSizeForConvolution`, the `height` and `width` of the *src* and *dest* parameters do not increase, and the other parameters remain the same, then the result of the `vImageGetMinimumTempBufferSizeForConvolution` does not increase. This makes it easy to reuse the same temporary buffer when you are processing a number of images of the same size, as in tiling.

**Availability**
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.

**Declared In**
`Convolution.h`

## vImageRichardsonLucyDeConvolve_ARGB8888

Sharpens an ARGB8888 image by undoing a previous convolution that blurred the image, such as diffraction effects in a camera lens.

```
vImage_Error vImageRichardsonLucyDeConvolve_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const int16_t *kernel,
   const int16_t *kernel2,
   uint32_t kernel_height,
   uint32_t kernel_width,
   uint32_t kernel_height2,
   uint32_t kernel_width2,
   int32_t divisor,
   int32_t divisor2,
   Pixel_8888 backgroundColor,
   uint32_t iterationCount,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the deconvolution kernel data, which must be a packed array without any padding. The kernel expresses a blurring convolution or point-spread function.

*kernel2*

A pointer to the data of a second kernel, which must be a packed array without any padding. Supply this kernel only if the first kernel is asymmetrical; otherwise pass `NULL`.

*kernel_height*

The height of the first kernel in pixels. This value must be odd.

*kernel_width*

The width of the first kernel in pixels. This value must be odd.

*kernel_height2*

The height of the second kernel in pixels (ignored if *kernel2* is `NULL`). This value must be odd.

*kernel_width2*

The width of the second kernel in pixels (ignored if *kernel2* is `NULL`). This value must be odd.

*divisor*

The divisor to be used in convolutions with the first kernel.

*divisor2*

The divisor to be used in convolutions with the second kernel.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*iterationCount*

The number of times to iterate the deconvolution algorithm.

*flags*

The options to use when performing the deconvolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function performs a Richardson-Lucy deconvolution of a region of interest within a source image by an M x N kernel, performing a specified number of iterations and placing the result in a destination buffer.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageRichardsonLucyDeConvolve_Planar8` (page 135)

**Declared In**
`Convolution.h`

## vImageRichardsonLucyDeConvolve_ARGBFFFF

Sharpens an ARGBFFFF image by undoing a previous convolution that blurred the image, such as diffraction effects in a camera lens.

```
vImage_Error vImageRichardsonLucyDeConvolve_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   const float *kernel2,
   uint32_t kernel_height,
   uint32_t kernel_width,
   uint32_t kernel_height2,
   uint32_t kernel_width2,
   Pixel_FFFF backgroundColor,
   uint32_t iterationCount,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the deconvolution kernel data, which must be a packed array without any padding. The kernel expresses a blurring convolution or point-spread function.

*kernel2*

A pointer to the data of a second kernel, which must be a packed array without any padding. Supply this kernel only if the first kernel is asymmetrical; otherwise pass `NULL`.

*kernel_height*

The height of the first kernel in pixels. This value must be odd.

*kernel_width*

The width of the first kernel in pixels. This value must be odd.

*kernel_height2*

The height of the second kernel in pixels (ignored if *kernel2* is `NULL`). This value must be odd.

*kernel_width2*

The width of the second kernel in pixels (ignored if *kernel2* is `NULL`). This value must be odd.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*iterationCount*

The number of times to iterate the deconvolution algorithm.

*flags*

The options to use when performing the deconvolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function performs a Richardson-Lucy deconvolution of a region of interest within a source image by an M x N kernel, performing a specified number of iterations and placing the result in a destination buffer.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageRichardsonLucyDeConvolve_PlanarF` (page 137)

**Declared In**
`Convolution.h`

## vImageRichardsonLucyDeConvolve_Planar8

Sharpens a Planar8 image by undoing a previous convolution that blurred the image, such as diffraction effects in a camera lens.

```
vImage_Error vImageRichardsonLucyDeConvolve_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const int16_t *kernel,
   const int16_t *kernel2,
   uint32_t kernel_height,
   uint32_t kernel_width,
   uint32_t kernel_height2,
   uint32_t kernel_width2,
   int32_t divisor,
   int32_t divisor2,
   Pixel_8 backgroundColor,
   uint32_t iterationCount,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the deconvolution kernel data, which must be a packed array without any padding. The kernel expresses a blurring convolution or point-spread function.

*kernel2*

A pointer to the data of a second kernel, which must be a packed array without any padding. Supply this kernel only if the first kernel is asymmetrical; otherwise pass `NULL`.

*kernel_height*

The height of the first kernel in pixels. This value must be odd.

*kernel_width*

The width of the first kernel in pixels. This value must be odd.

*kernel_height2*

The height of the second kernel in pixels (ignored if *kernel2* is `NULL`). This value must be odd.

*kernel_width2*

The width of the second kernel in pixels (ignored if *kernel2* is `NULL`). This value must be odd.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*iterationCount*

The number of times to iterate the deconvolution algorithm.

*flags*

The options to use when performing the deconvolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function performs a Richardson-Lucy deconvolution of a region of interest within a source image by an M x N kernel, performing a specified number of iterations and placing the result in a destination buffer.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageRichardsonLucyDeConvolve_ARGB8888` (page 131)

**Declared In**
`Convolution.h`

## vImageRichardsonLucyDeConvolve_PlanarF

Sharpens a PlanarF image by undoing a previous convolution that blurred the image, such as diffraction effects in a camera lens.

```
vImage_Error vImageRichardsonLucyDeConvolve_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   const float *kernel2,
   uint32_t kernel_height,
   uint32_t kernel_width,
   uint32_t kernel_height2,
   uint32_t kernel_width2,
   Pixel_F backgroundColor,
   uint32_t iterationCount,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the deconvolution kernel data, which must be a packed array without any padding. The kernel expresses a blurring convolution or point-spread function.

*kernel2*

A pointer to the data of a second kernel, which must be a packed array without any padding. Supply this kernel only if the first kernel is asymmetrical; otherwise pass `NULL`.

*kernel_height*

The height of the first kernel in pixels. This value must be odd.

*kernel_width*

The width of the first kernel in pixels. This value must be odd.

*kernel_height2*

The height of the second kernel in pixels (ignored if *kernel2* is `NULL`). This value must be odd.

*kernel_width2*

The width of the second kernel in pixels (ignored if *kernel2* is `NULL`). This value must be odd.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*iterationCount*

The number of times to iterate the deconvolution algorithm.

*flags*

The options to use when performing the deconvolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The function performs a Richardson-Lucy deconvolution of a region of interest within a source image by an M x N kernel, performing a specified number of iterations and placing the result in a destination buffer.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageRichardsonLucyDeConvolve_ARGBFFFF` (page 133)

**Declared In**
`Convolution.h`

## vImageTentConvolve_ARGB8888

Convolves a region of interest within an ARGB8888 source image by an implicit M x N kernel that has the effect of a tent filter.

```
vImage_Error vImageTentConvolve_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   uint32_t kernel_height,
   uint32_t kernel_width,
   Pixel_8888 backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

This function uses an implicit divisor and an implicit kernel of specified size instead of a kernel provided by the caller.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**

Available in Mac OS X v10.4 and later.

**See Also**

`vImageConvolve_ARGB8888` (page 124)
`vImageBoxConvolve_ARGB8888` (page 110)

**Declared In**
`Convolution.h`

## vImageTentConvolve_Planar8

Convolves a region of interest within a Planar8 source image by an implicit M x N kernel that has the effect of a tent filter.

```
vImage_Error vImageTentConvolve_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   uint32_t kernel_height,
   uint32_t kernel_width,
   Pixel_8 backgroundColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory. . The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*backgroundColor*

A background color. If you supply a color, you must also set the `kvImageBackgroundColorFill` flag, otherwise the function ignores the color.

*flags*

The options to use when performing the convolution operation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageCopyInPlace`, `kvImageTruncateKernel`, `kvImageBackgroundColorFill`, or `kvImageEdgeExtend`.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function uses an implicit divisor and an implicit kernel of specified size instead of a kernel provided by the caller.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImageConvolve_Planar8` (page 127)
`vImageBoxConvolve_Planar8` (page 111)

**Declared In**
`Convolution.h`

# vImage Decompression Filtering Reference

**Framework:** Accelerate/vImage
**Declared in** vImage_BasicImageTypes.h

**Companion guide** vImage Programming Guide

## Overview

The vImage framework provides one function for filtering data prior to decompression.

## Functions

### vImagePNGDecompressionFilter

Performs PNG decompression filtering.

```
vImage_Error  vImagePNGDecompressionFilter( const vImage_Buffer *buffer,
   vImagePixelCount    startScanline,
   vImagePixelCount    scanlineCount,
   uint32_t   bitsPerPixel,
   uint32_t  filterMethodNumber,
   uint32_t  filterType,
   vImage_Flags flags)
```

**Parameters**

*buffer*
On input, the image data to filter. On output, the filtered data. The filtering is always applied in place.

*startScanline*
The starting scanline.

*scanlineCount*
The number of scanlines in the buffer.

*bitsPerPixel*
The bits per pixel.

*filterMethodNumber*
The filter method number. You must pass `0`, because this is the only filtering method offered by this function.

*filterType*
The filtering algorithm to apply to the image data. For filter method 0, you can pass any of the constants described in "PNG Filter Types" (page 144).

*flags*
The options to use when performing this operation. Set the kvImageDoNotTile flag if you plan to perform your own tiling or use multithreading. See *vImage Data Types and Constants Reference* for a complete description of vImage processing flags.

**Discussion**
This function implements PNG decompression filtering for filter method 0 of the PNG standard, section 9.2, as described in: http://www.w3.org/TR/PNG-Filters.html. When a pixel that is needed for a filtering calculation falls outside the source buffer, its value is presumed to be 0.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
BasicImageTypes.h

# Constants

## PNG Filter Types

Filtering algorithms to apply to image data before compressing the data.

```
enum
{

    kvImage_PNG_FILTER_VALUE_NONE  = 0,
    kvImage_PNG_FILTER_VALUE_SUB   = 1,
    kvImage_PNG_FILTER_VALUE_UP    = 2,
    kvImage_PNG_FILTER_VALUE_AVG   = 3,
    kvImage_PNG_FILTER_VALUE_PAETH = 4
};
```

**Constants**
kvImage_PNG_FILTER_VALUE_NONE
No filtering.

Available in Mac OS X v10.4 and later.

Declared in BasicImageTypes.h.

kvImage_PNG_FILTER_VALUE_SUB
A filter that computes the difference between each byte of a pixel and the value of the corresponding byte of the pixel located to the left.

Available in Mac OS X v10.4 and later.

Declared in BasicImageTypes.h.

`kvImage_PNG_FILTER_VALUE_UP`

A filter that computes the difference between each byte of a pixel and the value of the corresponding byte of the pixel located above.

Available in Mac OS X v10.4 and later.

Declared in `BasicImageTypes.h`.

`kvImage_PNG_FILTER_VALUE_AVG`

A filter that predicts a pixel value from the average of the pixels to the left and above the predicted pixel location.

Available in Mac OS X v10.4 and later.

Declared in `BasicImageTypes.h`.

`kvImage_PNG_FILTER_VALUE_PAETH`

A filter that predicts a pixel value by applying a linear function to the pixels located to the left, above, and to the upper left of the predicted pixel location.

Available in Mac OS X v10.4 and later.

Declared in `BasicImageTypes.h`.

**Declared In**

`BasicImageTypes.h`

# vImage Geometry Reference

| | |
|---|---|
| **Framework:** | Accelerate/vImage |
| **Declared in** | Geometry.h |
| **Companion guide** | vImage Programming Guide |

## Overview

Geometric functions rotate, resize, and distort the geometry of images. vImage provides both high-level (rotation, scaling, and warping) and low-level geometric functions (reflection, shearing, and low-level rotation).

Most vImage geometric functions resample image data to avoid creating artifacts, such as interference patterns, in the destination image. vImage uses resampling kernels, which combine data from a target pixel and other nearby pixels to calculate a value for the destination pixel, a procedure somewhat similar to that used for convolution. However, for geometric operations, the resampling kernel itself is resampled during the process of pairing kernel values against the sampled pixel data. The kernel is evaluated at both fractional and integral pixel locations. This has implications for the nature of the kernel—which must be supplied as a function rather than as an M by N matrix. A resampling kernel function is also called a resampling filter, or simply a filter.

For almost all geometric operations, vImage supplies a default resampling filter unless you set the flag `kvImageHighQualityResampling`, in which case vImage uses a higher-quality filter, but that filter may be slower to use.

The reflection and high-level rotation functions don't resample. The shear functions can either use a default resampling filter or, if you require more control, a custom filter that you provide.

## Functions by Task

### Applying Affine Transforms

`vImageAffineWarp_ARGBFFFF` (page 151)
Applies an affine transform to an ARGBFFFF source image.

`vImageAffineWarp_ARGB8888` (page 150)
Applies an affine transform to an ARGB8888 source image.

`vImageAffineWarp_PlanarF` (page 154)
Applies an affine transform to a PlanarF source image.

`vImageAffineWarp_Planar8` (page 153)
Applies an affine transform to a Planar8 source image.

## Reflecting

`vImageHorizontalReflect_ARGBFFFF` (page 159)
Reflects an ARGBFFFF source image left to right across the center vertical line of the image.

`vImageHorizontalReflect_PlanarF` (page 160)
Reflects a PlanarF source image left to right across the center vertical line of the image, placing the result in a destination buffer.

`vImageHorizontalReflect_Planar8` (page 160)
Reflects a Planar9 source image left to right across the center vertical line of the image.

`vImageHorizontalReflect_ARGB8888` (page 158)
Reflects an ARGB8888 source image left to right across the center vertical line of the image.

`vImageVerticalReflect_ARGBFFFF` (page 184)
Reflects an ARGBFFFF source image top to bottom across the center vertical line of the image.

`vImageVerticalReflect_ARGB8888` (page 184)
Reflects an ARGBFFFF source image top to bottom across the center vertical line of the image.

`vImageVerticalReflect_PlanarF` (page 186)
Reflects a PlanarF source image top to bottom across the center vertical line of the image.

`vImageVerticalReflect_Planar8` (page 185)
Reflects a Planar 8 source image top to bottom across the center vertical line of the image.

## Shearing

`vImageHorizontalShear_ARGBFFFF` (page 162)
Performs a horizontal shear operation on a region of interest of an ARGBFFFF source image.

`vImageHorizontalShear_ARGB8888` (page 161)
Performs a horizontal shear operation on a region of interest of an ARGB8888 source image.

`vImageHorizontalShear_PlanarF` (page 165)
Performs a horizontal shear operation on a region of interest of a PlanarF source image.

`vImageHorizontalShear_Planar8` (page 164)
Performs a horizontal shear operation on a region of interest of a Planar8 source image.

`vImageVerticalShear_ARGBFFFF` (page 188)
Performs a vertical shear operation on a region of interest of an ARGBFFFF source image.

`vImageVerticalShear_ARGB8888` (page 187)
Performs a vertical shear operation on a region of interest of an ARGB8888 source image.

`vImageVerticalShear_PlanarF` (page 191)
Performs a vertical shear operation on a region of interest of a PlanarF source image.

`vImageVerticalShear_Planar8` (page 189)
Performs a vertical shear operation on a region of interest of a Planar8 source image.

## Rotating

`vImageRotate90_ARGBFFFF` (page 170)
Rotates an ARGBFFFF source image by the provided factor of 90.

`vImageRotate90_ARGB8888` (page 169)
Rotates an ARGB8888 source image by the provided factor of 90.

`vImageRotate90_PlanarF` (page 172)
Rotates a PlanarF source image by the provided factor of 90.

`vImageRotate90_Planar8` (page 171)
Rotates a Planar8 source image by the provided factor of 90.

`vImageRotate_ARGBFFFF` (page 175)
Rotates an ARGBFFFF source image by the provided angle.

`vImageRotate_ARGB8888` (page 173)
Rotates an ARGB8888 source image by the provided angle.

`vImageRotate_PlanarF` (page 178)
Rotates a PlanarF source image by the provided angle.

`vImageRotate_Planar8` (page 176)
Rotates a Planar8 source image by the provided angle.

## Scaling

`vImageScale_ARGBFFFF` (page 180)
Scales an ARGBFFFF source image to fit a destination buffer.

`vImageScale_ARGB8888` (page 179)
Scales an ARGB8888 source image to fit a destination buffer.

`vImageScale_PlanarF` (page 182)
Scales a PlanarF source image to fit a destination buffer.

`vImageScale_Planar8` (page 181)
Scales a Planar8 source image to fit a destination buffer.

## Resampling

`vImageDestroyResamplingFilter` (page 156)
Disposes of a resampling filter object.

`vImageGetResamplingFilterSize` (page 157)
Returns the minimum size, in bytes, for the buffer needed by the function `vImageNewResamplingFilterForFunctionUsingBuffer`.

`vImageNewResamplingFilter` (page 167)
Creates a resampling filter object that corresponds to the default kernel supplied by the vImage framework.

`vImageNewResamplingFilterForFunctionUsingBuffer` (page 167)
Creates a resampling filter object that encapsulates a resampling kernel function that you provide.

### Getting the Buffer Size

`vImageGetMinimumGeometryTempBufferSize` (page 156) Deprecated in Mac OS X v10.4

Returns the minimum size, in bytes, for the temporary buffer needed by a high-level geometry function. (Deprecated. Use the `kvImageGetTempBufferSize` flag with the appropriate geometry function instead of calling this function.)

# Functions

### vImageAffineWarp_ARGB8888

Applies an affine transform to an ARGB8888 source image.

```
vImage_Error vImageAffineWarp_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   const vImage_AffineTransform *transform,
   Pixel_8888 backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to transform.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*transform*

The affine transformation matrix to apply to the source image.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when applying the transform. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function maps each pixel in the source image `[x, y]` to a new position `[x', y']` in the destination image by the formula:

```
(x', y') = (x, y) * transform
```

where `transform` is the 3x3 affine transformation matrix.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageAffineWarp_ARGBFFFF

Applies an affine transform to an ARGBFFFF source image.

```
vImage_Error vImageAffineWarp_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   const vImage_AffineTransform *transform,
   Pixel_FFFF backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to transform.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*transform*

The affine transformation matrix to apply to the source image.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when applying the transform. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

This function maps each pixel in the source image `[x, y]` to a new position `[x', y']` in the destination image by the formula:

```
(x', y') = (x, y) * transform
```

where `transform` is the 3x3 affine transformation matrix.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageAffineWarp_Planar8

Applies an affine transform to a Planar8 source image.

```
vImage_Error vImageAffineWarp_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   const vImage_AffineTransform *transform,
   Pixel_8 backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to transform.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*transform*

The affine transformation matrix to apply to the source image.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when applying the transform. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function maps each pixel in the source image `[x, y]` to a new position `[x', y']` in the destination image by the formula:

```
(x', y') = (x, y) * transform
```

where `transform` is the 3x3 affine transformation matrix.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageAffineWarp_PlanarF

Applies an affine transform to a PlanarF source image.

```
vImage_Error vImageAffineWarp_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   const vImage_AffineTransform *transform,
   Pixel_F backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to transform.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*transform*

The affine transformation matrix to apply to the source image.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when applying the transform. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

This function maps each pixel in the source image `[x, y]` to a new position `[x', y']` in the destination image by the formula:

```
(x', y') = (x, y) * transform
```

where `transform` is the 3x3 affine transformation matrix.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageDestroyResamplingFilter

Disposes of a resampling filter object.

```
void vImageDestroyResamplingFilter (
   ResamplingFilter filter
);
```

**Parameters**

*filter*

The resampling filter object to dispose of.

**Discussion**
This function deallocates the memory associated with a resampling filter object that was created by calling the function `vImageNewResamplingFilter` (page 167). Do not directly deallocate this memory yourself.

Do not pass this function a resampling filter object created by the function `vImageNewResamplingFilterForFunctionUsingBuffer` (page 167). You are responsible for deallocating the memory associated with resampling filter objects created by that call yourself.

**Availability**
Available in Mac OS X v10.3 and later.

**See Also**
`vImageNewResamplingFilterForFunctionUsingBuffer` (page 167)

**Declared In**
`Geometry.h`

## vImageGetMinimumGeometryTempBufferSize

Returns the minimum size, in bytes, for the temporary buffer needed by a high-level geometry function. (Deprecated in Mac OS X v10.4. Use the `kvImageGetTempBufferSize` flag with the appropriate geometry function instead of calling this function.)

```
size_t vImageGetMinimumGeometryTempBufferSize (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags,
   size_t bytesPerPixel
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that you plan to pass to the geometry function.

*dest*

A pointer to a vImage buffer structure that you plan to pass to the geometry function. You must set the fields of this structure yourself, and allocate memory for its data. When you are done with the buffer structure, you must deallocate the memory.

*flags*

The flags that you plan to pass to the geometry function.

*bytesPerPixel*

The number of bytes in a pixel. Make sure to pass the value appropriate for the format of the pixel.

**Return Value**

The minimum size, in bytes, of the temporary buffer.

**Discussion**

This function does not depend on the *data* or *rowBytes* fields of the *src* or *dest* parameters; it only uses the *height* and *width* fields from those parameters. If the size of the images you are processing stay the same, then the required size of the buffer will also stay the same. More specifically, if, between two calls to `vImageGetMinimumGeometryTempBufferSize`, the `height` and `width` of the *src* and *dest* parameters do not increase, and the and the other parameters remain the same, then the result of the `vImageGetMinimumGeometryTempBufferSize` will not increase. This makes it easy to reuse the same temporary buffer when you are processing a number of images of the same size, as in tiling.

**Availability**

Available in Mac OS X v10.3 and later.

Deprecated in Mac OS X v10.4.

**Declared In**

`Geometry.h`

## vImageGetResamplingFilterSize

Returns the minimum size, in bytes, for the buffer needed by the function `vImageNewResamplingFilterForFunctionUsingBuffer`.

```
size_t vImageGetResamplingFilterSize (
   const float *xArray,
   float *yArray,
   unsigned long count,
   void *userData
);
```

**Parameters**

*scale*

The scale factor that you plan to pass to the function `vImageNewResamplingFilterForFunctionUsingBuffer`.

*kernelFunc*

The function pointer that you plan to pass to the function `vImageNewResamplingFilterForFunctionUsingBuffer`.

*userData*

The user data pointer that you plan to pass to the function `vImageNewResamplingFilterForFunctionUsingBuffer`.

*kernelWidth*

The kernel width that you plan to pass to the function `vImageNewResamplingFilterForFunctionUsingBuffer`.

*flags*

The flags that you plan to pass to the function `vImageNewResamplingFilterForFunctionUsingBuffer`.

**Return Value**
The minimum size, in bytes, of the buffer.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageHorizontalReflect_ARGB8888

Reflects an ARGB8888 source image left to right across the center vertical line of the image.

```
vImage_Error vImageHorizontalReflect_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to reflect.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the reflection. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function does not scale or resample. The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageHorizontalReflect_ARGBFFFF

Reflects an ARGBFFFF source image left to right across the center vertical line of the image.

```
vImage_Error vImageHorizontalReflect_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to reflect.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the reflection. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function does not scale or resample. The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageHorizontalReflect_Planar8

Reflects a Planar9 source image left to right across the center vertical line of the image.

```
vImage_Error vImageHorizontalReflect_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to reflect.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the reflection. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

### Return Value

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

### Discussion

This function does not scale or resample. The source and destination buffers must have the same height and the same width.

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

`Geometry.h`

## vImageHorizontalReflect_PlanarF

Reflects a PlanarF source image left to right across the center vertical line of the image, placing the result in a destination buffer.

```
vImage_Error vImageHorizontalReflect_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to reflect.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the reflection. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function does not scale or resample. The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageHorizontalShear_ARGB8888

Performs a horizontal shear operation on a region of interest of an ARGB8888 source image.

```
vImage_Error vImageHorizontalShear_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   float xTranslate,
   float shearSlope,
   ResamplingFilter filter,
   Pixel_8888 backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to shear.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

This parameter also specifies the size of the region of interest in within the source image. The region of interest has the same height and width as the destination image buffer.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*xTranslate*
A translation value for the horizontal direction.

*shearSlope*
The slope of the front edge of the sheared image, measured in a clockwise direction.

*filter*
The resampling filter to be used with this function. You create this object by calling `vImageNewResamplingFilter` (to use a default resampling filter supplied by vImage) or `vImageNewResamplingFilterForFunctionUsingBuffer` (to use a custom resampling filter that you supply). When the resampling filter is created, you can also set a scale factor that will be used in the horizontal shear operation.

*backgroundColor*
A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*
The options to use when performing the shear. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

The `kvImageBackgroundColorFill` and `kvImageEdgeExtend` flags are mutually exclusive. You must set exactly one of these flags.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function also translates and scales the image, both in the horizontal direction. The function transforms as much of the source image as it needs in order to attempt to fill the destination buffer, which means it can transform pixels outside the region of interest.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageHorizontalShear_ARGBFFFF

Performs a horizontal shear operation on a region of interest of an ARGBFFFF source image.

```
vImage_Error vImageHorizontalShear_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   float xTranslate,
   float shearSlope,
   ResamplingFilter filter,
   Pixel_FFFF backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to shear.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

This parameter also specifies the size of the region of interest in within the source image. The region of interest has the same height and width as the destination image buffer.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*xTranslate*

A translation value for the horizontal direction.

*shearSlope*

The slope of the front edge of the sheared image, measured in a clockwise direction.

*filter*

The resampling filter to be used with this function. You create this object by calling `vImageNewResamplingFilter` (to use a default resampling filter supplied by vImage) or `vImageNewResamplingFilterForFunctionUsingBuffer` (to use a custom resampling filter that you supply). When the resampling filter is created, you can also set a scale factor that will be used in the horizontal shear operation.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when performing the shear. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

The `kvImageBackgroundColorFill` and `kvImageEdgeExtend` flags are mutually exclusive. You must set exactly one of these flags.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function also translates and scales the image, both in the horizontal direction. The function transforms as much of the source image as it needs in order to attempt to fill the destination buffer, which means it can transform pixels outside the region of interest.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageHorizontalShear_Planar8

Performs a horizontal shear operation on a region of interest of a Planar8 source image.

```
vImage_Error vImageHorizontalShear_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   float xTranslate,
   float shearSlope,
   ResamplingFilter filter,
   Pixel_8 backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to shear.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

This parameter also specifies the size of the region of interest in within the source image. The region of interest has the same height and width as the destination image buffer.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*xTranslate*

A translation value for the horizontal direction.

*shearSlope*

The slope of the front edge of the sheared image, measured in a clockwise direction.

*filter*

The resampling filter to be used with this function. You create this object by calling `vImageNewResamplingFilter` (to use a default resampling filter supplied by vImage) or `vImageNewResamplingFilterForFunctionUsingBuffer` (to use a custom resampling filter that you supply). When the resampling filter is created, you can also set a scale factor that will be used in the horizontal shear operation.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when performing the shear. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

The `kvImageBackgroundColorFill` and `kvImageEdgeExtend` flags are mutually exclusive. You must set exactly one of these flags.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

This function also translates and scales the image, both in the horizontal direction. The function transforms as much of the source image as it needs in order to attempt to fill the destination buffer, which means it can transform pixels outside the region of interest.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Geometry.h`

## vImageHorizontalShear_PlanarF

Performs a horizontal shear operation on a region of interest of a PlanarF source image.

```
vImage_Error vImageHorizontalShear_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   float xTranslate,
   float shearSlope,
   ResamplingFilter filter,
   Pixel_F backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to shear.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

This parameter also specifies the size of the region of interest in within the source image. The region of interest has the same height and width as the destination image buffer.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*xTranslate*

A translation value for the horizontal direction.

*shearSlope*

The slope of the front edge of the sheared image, measured in a clockwise direction.

*filter*

The resampling filter to be used with this function. You create this object by calling `vImageNewResamplingFilter` (to use a default resampling filter supplied by vImage) or `vImageNewResamplingFilterForFunctionUsingBuffer` (to use a custom resampling filter that you supply). When the resampling filter is created, you can also set a scale factor that will be used in the horizontal shear operation.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when performing the shear. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

The `kvImageBackgroundColorFill` and `kvImageEdgeExtend` flags are mutually exclusive. You must set exactly one of these flags.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function also translates and scales the image, both in the horizontal direction. The function transforms as much of the source image as it needs in order to attempt to fill the destination buffer, which means it can transform pixels outside the region of interest.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageNewResamplingFilter

Creates a resampling filter object that corresponds to the default kernel supplied by the vImage framework.

```
ResamplingFilter vImageNewResamplingFilter (
   float scale,
   vImage_Flags flags
);
```

### Parameters

*scale*

A scale factor to associated with the resampling filter object. Shear functions to which you pass the resampling filter object use this factor when performing a shear operation. The shear function applies the scale factor to the entire image, in a direction appropriate to the shear function, either horizontal or vertical.

*flags*

The options to use when creating the resampling filter object. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

### Return Value

A pointer to a newly created resampling filter object; otherwise `NULL`.

### Discussion

This function creates a reusable resampling filter object that you can pass to a shear function. The resampling filter encapsulated by the object is the default kernel for vImage This function allocates the memory needed for the resampling filter object. To deallocate this memory, call the function `vImageDestroyResamplingFilter` (page 156). Do not attempt to deallocate the memory yourself.

### Availability

Available in Mac OS X v10.3 and later.

### See Also

`vImageNewResamplingFilterForFunctionUsingBuffer` (page 167)

### Declared In

`Geometry.h`

## vImageNewResamplingFilterForFunctionUsingBuffer

Creates a resampling filter object that encapsulates a resampling kernel function that you provide.

```
vImage_Error vImageNewResamplingFilterForFunctionUsingBuffer (
   const float *xArray,
   float *yArray,
   unsigned long count,
   void *userData
);
```

**Parameters**

*filter*

A pointer to a buffer. On return, the buffer contains a resampling filter object. You must allocate the memory for this buffer yourself. Call the function `vImageGetResamplingFilterSize` (page 157) to obtain the size of this buffer.

*scale*

A scale factor to associated with the resampling filter object. Shear functions to which you pass the resampling filter object use this factor when performing a shear operation. The shear function applies the scale factor to the entire image, in a direction appropriate to the shear function, either horizontal or vertical.

*kernelFunc*

A pointer to your custom resampling kernel function. If this pointer is `NULL`, vImage creates a resampling filter object that corresponds to its default kernel. The kernel function must remain valid for the life of the resampling filter object.

If you need precise control over the memory used for a resampling filter object but want to use the default, pass `NULL` as the `kernelFunc` parameter. This causes vImage to create a resampling filter object that corresponds to its default kernel. You can then determine where in memory the resampling filter object is located. You can reuse the buffer to reduce memory allocation, and so on. Note that a resampling filter object created in this way is not necessarily the same as one created by the function `vImageNewResamplingFilter` (page 167). You must still deallocate the object yourself; you can not pass it to the function `vImageDestroyResamplingFilter` (page 156).

*kernelWidth*

A bounding value for the domain of your resampling kernel function. When your function is called, the x-values it will be passed will lie between –*kernelWidth* and +*kernelWidth*, inclusive.

*userData*

A pointer to custom data that you want to use when calculating your resampling kernel function. When vImage invokes your resampling kernel function, this pointer is passed to the function. The data can be anything you want to use in calculating your resampling kernel function—a table, a list of pointers to related functions, or so on. The data must remain valid for the life of the resampling kernel object.

If your resampling kernel function does nor require user data, pass `NULL`.

*flags*

The options to use when creating the resampling filter object. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function creates a reusable resampling filter object that you can pass to a shear function. Before calling this function, you must allocate a buffer to contain the resampling filter object returned by this function. You can get the necessary size by calling the function `vImageGetResamplingFilterSize` (page 157). When you no longer need this object, you are responsible for deallocating its memory. (Do not use the function `vImageDestroyResamplingFilter` (page 156) to deallocate custom resampling filter objects; it is not designed to deallocate them.)

If you need precise control over the memory used for a resampling filter object but want to use the default, pass `NULL` as the `kernelFunc` parameter. This causes vImage to create a resampling filter object that corresponds to its default kernel. You can then determine where in memory the resampling filter object is located. You can reuse the buffer to reduce memory allocation, and so on. Note that a resampling filter object created in this way is not necessarily the same as one created by the function `vImageNewResamplingFilter` (page 167). You must still deallocate the object yourself; you can not pass it to the function `vImageDestroyResamplingFilter` (page 156).

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageRotate90_ARGB8888

Rotates an ARGB8888 source image by the provided factor of 90.

```
vImage_Error vImageRotate90_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   uint8_t rotationConstant,
   Pixel_8888 backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image whose data you want to rotate.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*rotationConstant*
A value specifying the angle of rotation.

*backgroundColor*
A background color.

*flags*
The options to use when performing the rotation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function maps the center point of the source image to the center point of the destination image. It does not scale or resample, instead, the function copies individual pixels unchanged to new locations.

This function places certain restrictions on the pixel height and widths of the source and destination buffers, so that it can map the center of the source to the center of the destination precisely. The restrictions are:

- If you are rotating the image 90 or 270 degrees, the height of the source image and the width of the destination image must both be even or both be odd; and the width of the source image and the height of the destination image must both be even or both be odd.

If your images do not meet these restrictions, you can use the general (high-level) Rotate function instead, with an angle of 90 or 270 degrees.

Depending on the relative sizes of the source image and the destination buffer, parts of the source image may be clipped. Areas outside the source image may appear in the destination image the background color passed to the function.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageRotate90_ARGBFFFF

Rotates an ARGBFFFF source image by the provided factor of 90.

```
vImage_Error vImageRotate90_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   uint8_t rotationConstant,
   Pixel_FFFF backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image whose data you want to rotate.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*rotationConstant*
A value specifying the angle of rotation.

*backgroundColor*
A background color.

*flags*

The options to use when performing the rotation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function maps the center point of the source image to the center point of the destination image. It does not scale or resample, instead, the function copies individual pixels unchanged to new locations.

This function places certain restrictions on the pixel height and widths of the source and destination buffers, so that it can map the center of the source to the center of the destination precisely. The restrictions are:

- If you are rotating the image 90 or 270 degrees, the height of the source image and the width of the destination image must both be even or both be odd; and the width of the source image and the height of the destination image must both be even or both be odd.

If your images do not meet these restrictions, you can use the general (high-level) Rotate function instead, with an angle of 90 or 270 degrees.

Depending on the relative sizes of the source image and the destination buffer, parts of the source image may be clipped. Areas outside the source image may appear in the destination image the background color passed to the function.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageRotate90_Planar8

Rotates a Planar8 source image by the provided factor of 90.

```
vImage_Error vImageRotate90_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   uint8_t rotationConstant,
   Pixel_8 backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to rotate.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*rotationConstant*
A value specifying the angle of rotation.

*backgroundColor*
A background color.

*flags*
The options to use when performing the rotation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function maps the center point of the source image to the center point of the destination image. It does not scale or resample, instead, the function copies individual pixels unchanged to new locations.

This function places certain restrictions on the pixel height and widths of the source and destination buffers, so that it can map the center of the source to the center of the destination precisely. The restrictions are:

- If you are rotating the image 90 or 270 degrees, the height of the source image and the width of the destination image must both be even or both be odd; and the width of the source image and the height of the destination image must both be even or both be odd.

If your images do not meet these restrictions, you can use the general (high-level) Rotate function instead, with an angle of 90 or 270 degrees.

Depending on the relative sizes of the source image and the destination buffer, parts of the source image may be clipped. Areas outside the source image may appear in the destination image the background color passed to the function.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageRotate90_PlanarF

Rotates a PlanarF source image by the provided factor of 90.

```
vImage_Error vImageRotate90_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   uint8_t rotationConstant,
   Pixel_F backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image whose data you want to rotate.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*rotationConstant*

A value specifying the angle of rotation.

*backgroundColor*

A background color.

*flags*

The options to use when performing the rotation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function maps the center point of the source image to the center point of the destination image. It does not scale or resample, instead, the function copies individual pixels unchanged to new locations.

This function places certain restrictions on the pixel height and widths of the source and destination buffers, so that it can map the center of the source to the center of the destination precisely. The restrictions are:

- If you are rotating the image 90 or 270 degrees, the height of the source image and the width of the destination image must both be even or both be odd; and the width of the source image and the height of the destination image must both be even or both be odd.

If your images do not meet these restrictions, you can use the general (high-level) Rotate function instead, with an angle of 90 or 270 degrees.

Depending on the relative sizes of the source image and the destination buffer, parts of the source image may be clipped. Areas outside the source image may appear in the destination image the background color passed to the function.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageRotate_ARGB8888

Rotates an ARGB8888 source image by the provided angle.

```
vImage_Error vImageRotate_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   float angleInRadians,
   Pixel_8888 backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to rotate.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*angleInRadians*

The angle of rotation, in radians.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when applying the rotation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

This function maps the center point of the source image to the center point of the destination image. It does not scale, but it resamples. Depending on the relative sizes of the source image and the destination buffer, parts of the source image may be clipped. Areas outside the source image may appear in the destination image the background color passed to the function.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageRotate_ARGBFFFF

Rotates an ARGBFFFF source image by the provided angle.

```
vImage_Error vImageRotate_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   float angleInRadians,
   Pixel_FFFF backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to rotate.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*angleInRadians*

The angle of rotation, in radians.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when applying the rotation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function maps the center point of the source image to the center point of the destination image. It does not scale, but it resamples. Depending on the relative sizes of the source image and the destination buffer, parts of the source image may be clipped. Areas outside the source image may appear in the destination image the background color passed to the function.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageRotate_Planar8

Rotates a Planar8 source image by the provided angle.

```
vImage_Error vImageRotate_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   float angleInRadians,
   Pixel_8 backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to rotate.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*angleInRadians*

The angle of rotation, in radians.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when applying the rotation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function maps the center point of the source image to the center point of the destination image. It does not scale, but it resamples. Depending on the relative sizes of the source image and the destination buffer, parts of the source image may be clipped. Areas outside the source image may appear in the destination image the background color passed to the function.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageRotate_PlanarF

Rotates a PlanarF source image by the provided angle.

```
vImage_Error vImageRotate_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   float angleInRadians,
   Pixel_F backColor,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to rotate.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*angleInRadians*

The angle of rotation, in radians.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when applying the rotation. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

### Return Value

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

### Discussion

This function maps the center point of the source image to the center point of the destination image. It does not scale, but it resamples. Depending on the relative sizes of the source image and the destination buffer, parts of the source image may be clipped. Areas outside the source image may appear in the destination image the background color passed to the function.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageScale_ARGB8888

Scales an ARGB8888 source image to fit a destination buffer.

```
vImage_Error vImageScale_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to scale.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*flags*

The options to use when applying the scaling. Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag. The function uses edge extend (see `kvImageEdgeExtend`) to assign values to pixels that are outside the source buffer. Edge extend prevents a background color from impinging on the edges of the scaled image.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The relative size of the source image and the destination buffer determine the scaling factors, which may be different in the X and Y directions.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageScale_ARGBFFFF

Scales an ARGBFFFF source image to fit a destination buffer.

```
vImage_Error vImageScale_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to scale.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*flags*

The options to use when applying the scaling. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The relative size of the source image and the destination buffer determine the scaling factors, which may be different in the X and Y directions.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageScale_Planar8

Scales a Planar8 source image to fit a destination buffer.

```
vImage_Error vImageScale_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to scale.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*flags*

The options to use when applying the scaling. Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag. The function uses edge extend (see `kvImageEdgeExtend`) to assign values to pixels that are outside the source buffer. Edge extend prevents a background color from impinging on the edges of the scaled image.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The relative size of the source image and the destination buffer determine the scaling factors, which may be different in the X and Y directions.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageScale_PlanarF

Scales a PlanarF source image to fit a destination buffer.

```
vImage_Error vImageScale_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to scale.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*flags*

The options to use when applying the scaling. Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

This function ignores the `kvImageLeaveAlphaUnchanged` flag. The function uses edge extend (see `kvImageEdgeExtend`) to assign values to pixels that are outside the source buffer. Edge extend prevents a background color from impinging on the edges of the scaled image.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The relative size of the source image and the destination buffer determine the scaling factors, which may be different in the X and Y directions.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageVerticalReflect_ARGB8888

Reflects an ARGBFFFF source image top to bottom across the center vertical line of the image.

```
vImage_Error vImageVerticalReflect_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to reflect.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the reflection. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The resulting image appears upside down, as if seen from the back of the image.

This function does not scale or resample. The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Related Sample Code**
Denoise
From A View to A Movie
From A View to A Picture
GLSL Showpiece Lite
QTCoreVideo202

**Declared In**
`Geometry.h`

## vImageVerticalReflect_ARGBFFFF

Reflects an ARGBFFFF source image top to bottom across the center vertical line of the image.

```
vImage_Error vImageVerticalReflect_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to reflect.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the reflection. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The resulting image appears upside down, as if seen from the back of the image.

This function does not scale or resample. The source and destination buffers must have the same height and the same width.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Geometry.h`

## vImageVerticalReflect_Planar8

Reflects a Planar 8 source image top to bottom across the center vertical line of the image.

```
vImage_Error vImageVerticalReflect_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to reflect.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the reflection. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The resulting image appears upside down, as if seen from the back of the image.

This function does not scale or resample. The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageVerticalReflect_PlanarF

Reflects a PlanarF source image top to bottom across the center vertical line of the image.

```
vImage_Error vImageVerticalReflect_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a structure of type `vImage_Buffer` containing the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use when performing the reflection. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The resulting image appears upside down, as if seen from the back of the image.

This function does not scale or resample. The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageVerticalShear_ARGB8888

Performs a vertical shear operation on a region of interest of an ARGB8888 source image.

```
vImage_Error vImageVerticalShear_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   float yTranslate,
   float shearSlope,
   ResamplingFilter filter,
   Pixel_8888 backColor,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to shear.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

This parameter also specifies the size of the region of interest in within the source image. The region of interest has the same height and width as the destination image buffer.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*yTranslate*

A translation value for the vertical direction.

*shearSlope*

The slope of the top edge of the sheared image, measured in a clockwise direction.

*filter*

The resampling filter to be used with this function. You create this object by calling `vImageNewResamplingFilter` (to use a default resampling filter supplied by vImage) or `vImageNewResamplingFilterForFunctionUsingBuffer` (to use a custom resampling filter that you supply). When the resampling filter is created, you can also set a scale factor that will be used in the horizontal shear operation.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when performing the shear. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

The `kvImageBackgroundColorFill` and `kvImageEdgeExtend` flags are mutually exclusive. You must set exactly one of these flags.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function also translates and scales the image, both in the vertical direction. The function transforms as much of the source image as it needs in order to attempt to fill the destination buffer, which means it can transform pixels outside the region of interest.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageVerticalShear_ARGBFFFF

Performs a vertical shear operation on a region of interest of an ARGBFFFF source image.

```
vImage_Error vImageVerticalShear_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   float yTranslate,
   float shearSlope,
   ResamplingFilter filter,
   Pixel_FFFF backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to shear.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

This parameter also specifies the size of the region of interest in within the source image. The region of interest has the same height and width as the destination image buffer.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*yTranslate*
A translation value for the vertical direction.

*shearSlope*
The slope of the top edge of the sheared image, measured in a clockwise direction.

*filter*
The resampling filter to be used with this function. You create this object by calling `vImageNewResamplingFilter` (to use a default resampling filter supplied by vImage) or `vImageNewResamplingFilterForFunctionUsingBuffer` (to use a custom resampling filter that you supply). When the resampling filter is created, you can also set a scale factor that will be used in the horizontal shear operation.

*backgroundColor*
A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*
The options to use when performing the shear. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

The `kvImageBackgroundColorFill` and `kvImageEdgeExtend` flags are mutually exclusive. You must set exactly one of these flags.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function also translates and scales the image, both in the vertical direction. The function transforms as much of the source image as it needs in order to attempt to fill the destination buffer, which means it can transform pixels outside the region of interest.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageVerticalShear_Planar8

Performs a vertical shear operation on a region of interest of a Planar8 source image.

```
vImage_Error vImageVerticalShear_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   float yTranslate,
   float shearSlope,
   ResamplingFilter filter,
   Pixel_8 backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to shear.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

This parameter also specifies the size of the region of interest in within the source image. The region of interest has the same height and width as the destination image buffer.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*yTranslate*

A translation value for the vertical direction.

*shearSlope*

The slope of the top edge of the sheared image, measured in a clockwise direction.

*filter*

The resampling filter to be used with this function. You create this object by calling `vImageNewResamplingFilter` (to use a default resampling filter supplied by vImage) or `vImageNewResamplingFilterForFunctionUsingBuffer` (to use a custom resampling filter that you supply). When the resampling filter is created, you can also set a scale factor that will be used in the horizontal shear operation.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when performing the shear. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

The `kvImageBackgroundColorFill` and `kvImageEdgeExtend` flags are mutually exclusive. You must set exactly one of these flags.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function also translates and scales the image, both in the vertical direction. The function transforms as much of the source image as it needs in order to attempt to fill the destination buffer, which means it can transform pixels outside the region of interest.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

## vImageVerticalShear_PlanarF

Performs a vertical shear operation on a region of interest of a PlanarF source image.

```
vImage_Error vImageVerticalShear_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   float yTranslate,
   float shearSlope,
   ResamplingFilter filter,
   Pixel_F backColor,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image whose data you want to shear.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

This parameter also specifies the size of the region of interest in within the source image. The region of interest has the same height and width as the destination image buffer.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*yTranslate*

A translation value for the vertical direction.

*shearSlope*

The slope of the top edge of the sheared image, measured in a clockwise direction.

*filter*

The resampling filter to be used with this function. You create this object by calling `vImageNewResamplingFilter` (to use a default resampling filter supplied by vImage) or `vImageNewResamplingFilterForFunctionUsingBuffer` (to use a custom resampling filter that you supply). When the resampling filter is created, you can also set a scale factor that will be used in the horizontal shear operation.

*backgroundColor*

A background color. Pass a pixel value only if you also set the `kvImageBackgroundColorFill` flag.

*flags*

The options to use when performing the shear. You must set exactly one of the following flags to specify how vImage handles pixel locations beyond the edge of the source image: `kvImageBackgroundColorFill` or `kvImageEdgeExtend`.

Set the `kvImageHighQualityResampling` flag if you want vImage to use a higher quality, but slower, resampling filter.

Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

The `kvImageBackgroundColorFill` and `kvImageEdgeExtend` flags are mutually exclusive. You must set exactly one of these flags.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function also translates and scales the image, both in the vertical direction. The function transforms as much of the source image as it needs in order to attempt to fill the destination buffer, which means it can transform pixels outside the region of interest.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Geometry.h`

# Constants

## Rotation Constants

The number of degrees in the clockwise direction.

```
enum
{
    kRotate0DegreesClockwise        = 0,
    kRotate90DegreesClockwise       = 3,
    kRotate180DegreesClockwise      = 2,
    kRotate270DegreesClockwise      = 1
    kRotate0DegreesCounterClockwise = 0,
    kRotate90DegreesCounterClockwise= 1,
    kRotate180DegreesCounterClockwise= 2,
    kRotate270DegreesCounterClockwise= 3
};
```

**Constants**

`kRotate0DegreesClockwise`
Rotate 0 degrees (that is, copy without rotating).
Available in Mac OS X v10.3 and later.
Declared in `Geometry.h`.

`kRotate90DegreesClockwise`
Rotate 90 degrees clockwise.
Available in Mac OS X v10.3 and later.
Declared in `Geometry.h`.

`kRotate180DegreesClockwise`
Rotate 180 degrees clockwise.
Available in Mac OS X v10.3 and later.
Declared in `Geometry.h`.

`kRotate270DegreesClockwise`
Rotate 270 degrees clockwise.
Available in Mac OS X v10.3 and later.
Declared in `Geometry.h`.

`kRotate0DegreesCounterClockwise`
Rotate 0 degrees (that is, copy without rotating).
Available in Mac OS X v10.3 and later.
Declared in `Geometry.h`.

`kRotate90DegreesCounterClockwise`
Rotate 90 degrees counter-clockwise.
Available in Mac OS X v10.3 and later.
Declared in `Geometry.h`.

`kRotate180DegreesCounterClockwise`
Rotate 180 degrees counter-clockwise.
Available in Mac OS X v10.3 and later.
Declared in `Geometry.h`.

`kRotate270DegreesCounterClockwise`
Rotate 270 degrees counter-clockwise.
Available in Mac OS X v10.3 and later.
Declared in `Geometry.h`.

**Declared In**
`Geometry.h`

# vImage Histogram Reference

| | |
|---|---|
| **Framework:** | Accelerate/vImage |
| **Declared in** | Histogram.h |
| **Companion guide** | vImage Programming Guide |

## Overview

Histogram functions calculate image histograms or manipulate a histogram to modify an image. There are a number of reasons to apply histogram operations to an image. An image may not make full use of the possible range of intensity values—for example, most of its pixels may be fairly dark, making details difficult to see. Changing the image so that it has a more uniform histogram can improve contrast. Also, it may be easier to compare two images (with respect to texture or other aspects) if you change each histogram to match some standard histogram.

Histogram operations are **point operations**: that is, the intensity of a destination pixel depends only on the intensity of the source pixel, modified by values that are the same over the entire image. Two pixels of the same intensity always map to two pixels of the same (but presumably altered) intensity. If the original image has N different intensity values, the transformed image will have at most N different intensity levels represented.

The vImage histogram functions either calculate histograms or perform one of these point operations:

- Contrast stretch transforms an image so that its intensity values stretch out along the full range of intensity values. It is best used on images in which all the pixels are concentrated in one area of the intensity spectrum, and intensity values outside that area are not represented.
- Ends-in contrast stretch is a more complex version of the contrast stretch operation. These types of functions are best used on images that have some pixels at or near the lowest and highest values of the intensity spectrum, but whose histogram is still mainly concentrated in one area. The ends-in contrast stretch functions map all intensities less than or equal to a certain level to 0; all intensities greater than or equal to a certain level to 255; and perform a contrast stretch on all the values in between. The low and high levels are not defined directly by two given intensity values, but by percentages: the ends-in contrast stretch operation must find intensity levels such that a certain percent of pixels are below one of the intensity values, and a certain percent are above the other intensity value
- Equalization transforms an image so that it has a more uniform histogram. A truly uniform histogram is one in which each intensity level occurs with equal frequency. These functions approximate that histogram.
- Histogram specification transforms an image so that its histogram more closely resembles a given histogram.

# Functions by Task

## Stretching the Contrast

`vImageContrastStretch_ARGBFFFF` (page 198)
Stretches the contrast of an ARGBFFFF source image.

`vImageContrastStretch_ARGB8888` (page 197)
Stretches the contrast of an ARGB8888 source image.

`vImageContrastStretch_PlanarF` (page 200)
Stretches the contrast of a PlanarF source image.

`vImageContrastStretch_Planar8` (page 199)
Stretches the contrast of a Planar8 source image.

`vImageEndsInContrastStretch_ARGBFFFF` (page 203)
Performs an ends-in contrast stretch operation on an ARGBFFFF source image.

`vImageEndsInContrastStretch_ARGB8888` (page 202)
Performs an ends-in contrast stretch operation on an ARGB8888 source image.

`vImageEndsInContrastStretch_PlanarF` (page 205)
Performs an ends-in contrast stretch operation on a PlanarF source image.

`vImageEndsInContrastStretch_Planar8` (page 204)
Performs an ends-in contrast stretch operation on a Planar8 source image.

## Equalizing a Histogram

`vImageEqualization_ARGBFFFF` (page 208)
Equalizes the histogram of an ARGBFFFF source image.

`vImageEqualization_ARGB8888` (page 207)
Equalizes the histogram of an ARGB8888 source image.

`vImageEqualization_PlanarF` (page 210)
Equalizes the histogram of a PlanarF source image.

`vImageEqualization_Planar8` (page 209)
Equalizes the histogram of an ARGB8888 source image.

## Specifying a Histogram

`vImageHistogramSpecification_ARGBFFFF` (page 216)
Performs a histogram specification operation on an ARGBFFFF source image.

`vImageHistogramSpecification_ARGB8888` (page 215)
Performs a histogram specification operation on an ARGB8888 source image.

`vImageHistogramSpecification_PlanarF` (page 218)
Performs a histogram specification operation on a PlanarF source image.

`vImageHistogramSpecification_Planar8` (page 218)
Performs a histogram specification operation on a Planar8 source image.

### Calculating a Histogram

`vImageHistogramCalculation_ARGBFFFF` (page 213)
Calculates histograms for each channel of an ARGBFFFF image.

`vImageHistogramCalculation_ARGB8888` (page 212)
Calculates histograms for each channel of an ARGB8888 image.

`vImageHistogramCalculation_PlanarF` (page 214)
Calculates the histogram a PlanarF image.

`vImageHistogramCalculation_Planar8` (page 214)
Calculates a histogram for a Planar8 image.

### Getting the Minimum Buffer Size.

`vImageGetMinimumTempBufferSizeForHistogram` (page 211) Deprecated in Mac OS X v10.4
Returns the minimum size, in bytes, for the temporary buffer needed by a histogram function. (Deprecated. Use the `kvImageGetTempBufferSize` flag with the appropriate histogram function instead of calling this function.)

## Functions

### vImageContrastStretch_ARGB8888

Stretches the contrast of an ARGB8888 source image.

```
vImage_Error vImageContrastStretch_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*
The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The contrast stretch operation alters the image histogram so that pixel values can be found at both the lowest and highest end of the histogram, with values in between "stretched" in a linear fashion. The contrast stretch operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms.

The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageContrastStretch_ARGBFFFF

Stretches the contrast of an ARGBFFFF source image.

```
vImage_Error vImageContrastStretch_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*histogram_entries*

The number of histogram entries, or bins, to use in histograms for this operation.

*minVal*

A minimum pixel value, the low end of the histogram. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry. This minimum value is applied to each of the four channels separately.

*maxVal*

A maximum pixel value, the high end of the histogram. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry. This maximum value is applied to each of the four channels separately.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The contrast stretch operation alters the image histogram so that pixel values can be found at both the lowest and highest end of the histogram, with values in between "stretched" in a linear fashion. The contrast stretch operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms.

The source and destination buffers must have the same height and the same width.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageContrastStretch_Planar8

Stretches the contrast of a Planar8 source image.

```
vImage_Error vImageContrastStretch_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The contrast stretch operation alters the image histogram so that pixel values can be found at both the lowest and highest end of the histogram, with values in between "stretched" in a linear fashion. The contrast stretch operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms.

The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageContrastStretch_PlanarF

Stretches the contrast of a PlanarF source image.

```
vImage_Error vImageContrastStretch_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*histogram_entries*

The number of histogram entries, or "bins," to be used in histograms for this operation.

*minVal*

A minimum pixel value, the low end of the histogram. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry.

*maxVal*

A maximum pixel value, the high end of the histogram. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The contrast stretch operation alters the image histogram so that pixel values can be found at both the lowest and highest end of the histogram, with values in between "stretched" in a linear fashion. The contrast stretch operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms.

The source and destination buffers must have the same height and the same width.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Histogram.h`

## vImageEndsInContrastStretch_ARGB8888

Performs an ends-in contrast stretch operation on an ARGB8888 source image.

```
vImage_Error vImageEndsInContrastStretch_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const unsigned int percent_low[4],
   const unsigned int percent_high[4],
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*percent_low*

A percentage value. The number of pixels that map to the lowest end of the histogram of the transformed image should represent this percentage of the total pixels.

*percent_high*

A percentage value. The number of pixels that map to the highest end of the histogram of the transformed image should represent this percentage of the total pixels.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

### Return Value

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

### Discussion

The ends-in contrast stretch operation alters the image histogram so that a certain percentage of pixels are mapped to the lowest end of the histogram, a certain percentage are mapped to the highest end, and the values in between "stretched" between the lowest and the highest. The ends-in contrast stretch operation is done separately for each of the four channels—alpha, red, green, and blue. However the size and range values are the same for each of the four histograms. In general, it is not possible to get exactly the desired percentage of pixels at the low end and the high end of the histogram of the transformed image. This operation only approximates the given values.

The source and destination buffers must have the same height and the same width.

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

`Histogram.h`

## vImageEndsInContrastStretch_ARGBFFFF

Performs an ends-in contrast stretch operation on an ARGBFFFF source image.

```
vImage_Error vImageEndsInContrastStretch_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   const unsigned int percent_low[4],
   const unsigned int percent_high[4],
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*percent_low*

A percentage value. The number of pixels that map to the lowest end of the histogram of the transformed image should represent this percentage of the total pixels.

*percent_high*

A percentage value. The number of pixels that map to the highest end of the histogram of the transformed image should represent this percentage of the total pixels.

*histogram_entries*

The number of histogram entries, or “bins,” to be used in histograms for this operation.

*minVal*

A minimum pixel value, the low end of the histogram. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry. This minimum value is applied to each of the four channels separately.

*maxVal*

A maximum pixel value, the high end of the histogram. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry. This maximum value is applied to each of the four channels separately.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The ends-in contrast stretch operation alters the image histogram so that a certain percentage of pixels are mapped to the lowest end of the histogram, a certain percentage are mapped to the highest end, and the values in between "stretched" between the lowest and the highest. The ends-in contrast stretch operation is done separately for each of the four channels—alpha, red, green, and blue. However the size and range values are the same for each of the four histograms. In general, it is not possible to get exactly the desired percentage of pixels at the low end and the high end of the histogram of the transformed image. This operation only approximates the given values.

The source and destination buffers must have the same height and the same width.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageEndsInContrastStretch_Planar8

Performs an ends-in contrast stretch operation on a Planar8 source image.

```
vImage_Error vImageEndsInContrastStretch_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   unsigned int percent_low,
   unsigned int percent_high,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*percent_low*

A percentage value. The number of pixels that map to the lowest end of the histogram of the transformed image should represent this percentage of the total pixels.

*percent_high*

A percentage value. The number of pixels that map to the highest end of the histogram of the transformed image should represent this percentage of the total pixels.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The ends-in contrast stretch operation alters the image histogram so that a certain percentage of pixels are mapped to the lowest end of the histogram, a certain percentage are mapped to the highest end, and the values in between "stretched" between the lowest and the highest. The ends-in contrast stretch operation is done separately for each of the four channels—alpha, red, green, and blue. However the size and range values are the same for each of the four histograms. In general, it is not possible to get exactly the desired percentage of pixels at the low end and the high end of the histogram of the transformed image. This operation only approximates the given values.

The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageEndsInContrastStretch_PlanarF

Performs an ends-in contrast stretch operation on a PlanarF source image.

```
vImage_Error vImageEndsInContrastStretch_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   unsigned int percent_low,
   unsigned int percent_high,
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*percent_low*

A percentage value. The number of pixels that map to the lowest end of the histogram of the transformed image should represent this percentage of the total pixels.

*percent_high*

A percentage value. The number of pixels that map to the highest end of the histogram of the transformed image should represent this percentage of the total pixels.

*histogram_entries*

The number of histogram entries, or "bins," to be used in histograms for this operation.

*minVal*

A minimum pixel value, the low end of the histogram. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry.

*maxVal*

A maximum pixel value, the high end of the histogram. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The ends-in contrast stretch operation alters the image histogram so that a certain percentage of pixels are mapped to the lowest end of the histogram, a certain percentage are mapped to the highest end, and the values in between "stretched" between the lowest and the highest. The ends-in contrast stretch operation is done separately for each of the four channels—alpha, red, green, and blue. However the size and range values are the same for each of the four histograms. In general, it is not possible to get exactly the desired percentage of pixels at the low end and the high end of the histogram of the transformed image. This operation only approximates the given values.

The source and destination buffers must have the same height and the same width.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageEqualization_ARGB8888

Equalizes the histogram of an ARGB8888 source image.

```
vImage_Error vImageEqualization_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The equalization operation alters the image histogram so that it is closer to a uniform intensity distribution. The histogram equalization operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms.

The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageEqualization_ARGBFFFF

Equalizes the histogram of an ARGBFFFF source image.

```
vImage_Error vImageEqualization_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*histogram_entries*

The number of histogram entries, or "bins," to be used in histograms for this operation.

*minVal*

A minimum pixel value. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry. This minimum value is applied to each of the four channels separately.

*maxVal*

A maximum pixel value. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry. This maximum value is applied to each of the four channels separately.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The equalization operation alters the image histogram so that it is closer to a uniform intensity distribution. The histogram equalization operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms.

The source and destination buffers must have the same height and the same width.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageEqualization_Planar8

Equalizes the histogram of an ARGB8888 source image.

```
vImage_Error vImageEqualization_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The equalization operation alters the image histogram so that it is closer to a uniform intensity distribution. The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageEqualization_PlanarF

Equalizes the histogram of a PlanarF source image.

```
vImage_Error vImageEqualization_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*histogram_entries*

The number of histogram entries, or "bins," to be used in histograms for this operation.

*minVal*

A minimum pixel value. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry. This minimum value is applied to each of the four channels separately.

*maxVal*

A maximum pixel value. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry. This maximum value is applied to each of the four channels separately.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The equalization operation alters the image histogram so that it is closer to a uniform intensity distribution. The histogram equalization operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms.

The source and destination buffers must have the same height and the same width.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageGetMinimumTempBufferSizeForHistogram

Returns the minimum size, in bytes, for the temporary buffer needed by a histogram function. (Deprecated in Mac OS X v10.4. Use the `kvImageGetTempBufferSize` flag with the appropriate histogram function instead of calling this function.)

```
size_t vImageGetMinimumTempBufferSizeForHistogram (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   unsigned int histogram_entries,
   vImage_Flags flags,
   size_t bytesPerPixel
);
```

**Parameters**

*src*

A pointer to the vImage buffer structure that you plan to pass to the histogram function.

*dest*
A pointer to a vImage buffer structure that you plan to pass to the histogram function.

*histogram_entries*
The number of histogram that you plan to pass to the histogram function.

*flags*
The flags that you plan to pass to the histogram function.

*pixelBytes*
The number of bytes in a pixel. Make sure to pass the value appropriate for the format of the pixel.

**Return Value**
The minimum size, in bytes, of the temporary buffer.

**Discussion**
This function does not depend on the *data* or *rowBytes* fields of the *src* or *dest* parameters; it only uses the *height* and *width* fields from those parameters. If the size of the images you are processing stay the same, then the required size of the buffer will also stay the same. More specifically, if, between two calls to `vImageGetMinimumTempBufferSizeForHistogram`, the `height` and `width` of the *src* and *dest* parameters do not increase, and the other parameters remain the same, then the result of the `vImageGetMinimumTempBufferSizeForHistogram` will not increase. This makes it easy to reuse the same temporary buffer when you are processing a number of images of the same size, as in tiling.

**Availability**
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.

**Declared In**
`Histogram.h`

## vImageHistogramCalculation_ARGB8888

Calculates histograms for each channel of an ARGB8888 image.

```
vImage_Error vImageHistogramCalculation_ARGB8888 (
   const vImage_Buffer *src,
   vImagePixelCount *histogram[4],
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image.

*histogram*
A pointer to a histograms, one each for alpha, red, green, and blue (in that order). On return, this array will contain the four histograms for the corresponding channels. Each of the four histograms will be an array with 256 elements.

*flags*
The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The function calculates the histogram for each channel completely separately from the others. However, size and range values are the same for each of the four histograms.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageHistogramCalculation_ARGBFFFF

Calculates histograms for each channel of an ARGBFFFF image.

```
vImage_Error vImageHistogramCalculation_ARGBFFFF (
   const vImage_Buffer *src,
   vImagePixelCount *histogram[4],
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*histogram*

A pointer to an array of four histograms, one each for alpha, red, green, and blue (in that order). On return, this array will contain the four histograms for the corresponding channels. Each of the four histograms will be an array with histogram_entries elements.

*histogram_entries*

The number of histogram entries, or "bins." Each of the four calculated histograms will be an array with histogram_entries elements.

*minVal*

A minimum pixel value. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry. This minimum value is applied to each of the four channels separately.

*maxVal*

A maximum pixel value. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry. This maximum value is applied to each of the four channels separately.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The function calculates the histogram for each channel completely separately from the others. However, size and range values are the same for each of the four histograms.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageHistogramCalculation_Planar8

Calculates a histogram for a Planar8 image.

```
vImage_Error vImageHistogramCalculation_Planar8 (
   const vImage_Buffer *src,
   vImagePixelCount *histogram,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*histogram*

A pointer to a histogram. On return, this array will contain the histogram for the source image. The histogram will be an array with 256 elements.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageHistogramCalculation_PlanarF

Calculates the histogram a PlanarF image.

```
vImage_Error vImageHistogramCalculation_PlanarF (
   const vImage_Buffer *src,
   vImagePixelCount *histogram,
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*histogram*

A pointer to a histogram. On return, this array will contain the histogram for the source image. The histogram will have histogram_entries elements.

*histogram_entries*

The number of histogram entries, or "bins." The histogram will be an array with histogram_entries elements.

*minVal*

A minimum pixel value. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry.

*maxVal*

A maximum pixel value. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Histogram.h`

## vImageHistogramSpecification_ARGB8888

Performs a histogram specification operation on an ARGB8888 source image.

```
vImage_Error vImageHistogramSpecification_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const vImagePixelCount *desired_histogram[4],
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*desiredHistogram*

A pointer to an array of four histograms, one each for alpha, red, green, and blue (in that order). These are the desired histograms for the transformed image. Each histogram should be an array with 256 elements.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The specification operation alters the image histogram so that it more closely resembles a given histogram. The histogram specification operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms, and for each of the four desired histograms.

The source and destination buffers must have the same height and the same width.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

## vImageHistogramSpecification_ARGBFFFF

Performs a histogram specification operation on an ARGBFFFF source image.

```
vImage_Error vImageHistogramSpecification_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   const vImagePixelCount *desired_histogram[4],
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*desiredHistogram*

A pointer to an array of four histograms, one each for alpha, red, green, and blue (in that order). These are the desired histograms for the transformed image. Each histogram should be an array with histogram_entries elements.

*histogram_entries*

The number of histogram entries, or "bins," to be used in histograms for this operation.

*minVal*

A minimum pixel value. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry. This minimum value is applied to each of the four channels separately.

*maxVal*

A maximum pixel value. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry. This maximum value is applied to each of the four channels separately.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

The specification operation alters the image histogram so that it more closely resembles a given histogram. The histogram specification operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms, and for each of the four desired histograms.

The source and destination buffers must have the same height and the same width.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Histogram.h`

## vImageHistogramSpecification_Planar8

Performs a histogram specification operation on a Planar8 source image.

```
vImage_Error vImageHistogramSpecification_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const vImagePixelCount *desired_histogram,
   vImage_Flags flags
);
```

### Parameters

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*desiredHistogram*

A pointer the desired histogram for the transformed image. The histogram should be an array with 256 elements.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

### Return Value

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

### Discussion

The specification operation alters the image histogram so that it more closely resembles a given histogram. The histogram specification operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms, and for each of the four desired histograms.

The source and destination buffers must have the same height and the same width.

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

`Histogram.h`

## vImageHistogramSpecification_PlanarF

Performs a histogram specification operation on a PlanarF source image.

```
vImage_Error vImageHistogramSpecification_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   const vImagePixelCount *desired_histogram,
   unsigned int histogram_entries,
   Pixel_F minVal,
   Pixel_F maxVal,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*desiredHistogram*

A pointer the desired histogram for the transformed image. The histogram should be an array with *histogram_entries* elements.

*histogram_entries*

The number of histogram entries, or "bins," to be used in histograms for this operation.

*minVal*

A minimum pixel value. Any pixel value less than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the first histogram entry. This minimum value is applied to each of the four channels separately.

*maxVal*

A maximum pixel value. Any pixel value greater than this will be clipped to this value (for the purposes of histogram calculation), and assigned to the last histogram entry. This maximum value is applied to each of the four channels separately.

*flags*

The options to use. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading.

Set the `kvImageLeaveAlphaUnchanged` flag to copy the alpha channel to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The specification operation alters the image histogram so that it more closely resembles a given histogram. The histogram specification operation is done separately for each of the four channels—alpha, red, green, and blue. However, the size and range values are the same for each of the four histograms, and for each of the four desired histograms.

The source and destination buffers must have the same height and the same width.

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Histogram.h`

# vImage Morphology Reference

| | |
|---|---|
| **Framework:** | Accelerate/vImage |
| **Declared in** | Morphology.h |
| **Companion guide** | vImage Programming Guide |

## Overview

Morphological functions change the shape of an object by performing dilatation, erosion, maximum, and minimum operations. Dilation expands objects. Erosion contracts them. Maximum is a special case of dilation, while minimum is a special case of erosion. The precise nature of the expanding or shrinking is determined by a kernel (also known as a *structure element*) provided by the caller. The number of rows and number of columns of the image does not change after applying a morphological operation.

You can use morphological functions on grayscale images, where the source image is planar (single-channel) or on full-color images. The kernel itself is always planar.

vImage applies morphological operations to an **object**, which is not the same as the entire image. An object is either comprised of the brightest pixels in an image or the darkest pixels in the image, where brightness is defined relative to the particular image. When you define bright pixels as the object, dark pixels become the background. In this case dilation expands objects with erosion contracts them. When you define dark pixels as the object, bright pixels become the background. In this case, dilation contracts objects and erosion expands them.

Each morphological function requires that you pass it a convolution kernel that determines how the values of neighboring pixels are used to compute the value of a destination pixel. A kernel is a packed array, without padding at the ends of the rows. The elements of the array must be of type `uint8_t` (for the Planar8 and ARGB8888 formats) or of type `float` (for the PlanarF and ARGBFFFF formats). The height and the width of the array must both be odd numbers.

For example, a 3 x 3 convolution kernel for a Planar8 image consists of nine 8-bit (1-byte) values, arranged consecutively. The first three values represent the first row of the kernel, the next three values the second row, and the last three values the third row.

Morphology functions perform clipping to prevent overflow for the Planar8 and ARGB8888 formats. Saturated clipping maps all intensity levels above 255, to 255, all intensity levels below 0, to 0, and leaves intensity levels between 0 and 255, inclusive, unchanged.

When the pixel to be transformed is near the edge of the image—not merely the region of interest, but the entire image of which it is a part—the kernel may extend beyond the edge of the image, so that there are no existing pixels beneath some of the kernel elements. In this case the morphology functions only make use of that part of the kernel which overlaps the source buffer. The other kernel elements are ignored.

# Functions by Task

## Dilating an Object

`vImageDilate_ARGBFFFF` (page 224)
Dilates a region of interest within an ARGBFFFF source image using an M x N kernel.

`vImageDilate_ARGB8888` (page 223)
Dilates a region of interest within an ARGB8888 source image using an M x N kernel.

`vImageDilate_PlanarF` (page 226)
Dilates a region of interest within a PlanarF source image using an M x N kernel.

`vImageDilate_Planar8` (page 225)
Dilates a region of interest within a Planar8 source image using an M x N kernel.

## Eroding an Object

`vImageErode_ARGBFFFF` (page 228)
Erodes a region of interest within an ARGBFFFF source image using an M x N kernel.

`vImageErode_ARGB8888` (page 227)
Erodes a region of interest within an ARGB8888 source image using an M x N kernel.

`vImageErode_PlanarF` (page 230)
Erodes a region of interest within a PlanarF source image using an M x N kernel.

`vImageErode_Planar8` (page 229)
Erodes a region of interest within a Planar8 source image using an M x N kernel.

## Maximizing an Object

`vImageMax_ARGBFFFF` (page 233)
Maximizes a region of interest within an ARGBFFFF source image using an M x N kernel.

`vImageMax_ARGB8888` (page 232)
Maximizes a region of interest within an ARGB8888 source image using an M x N kernel.

`vImageMax_PlanarF` (page 236)
Maximizes with a region of interest within a PlanarF source image using an M x N kernel.

`vImageMax_Planar8` (page 235)
Maximizes a region of interest within a Planar8 source image using an M x N kernel.

## Minimizing an Object

`vImageMin_ARGBFFFF` (page 239)
Minimizes a region of interest within an ARGBFFFF source image using an M x N kernel.

`vImageMin_ARGB8888` (page 238)
Minimizes a region of interest within an ARGB8888 source image using an M x N kernel.

`vImageMin_PlanarF` (page 242)
Minimizes a region of interest within a PlanarF source image using an M x N kernel.

`vImageMin_Planar8` (page 241)
Minimizes a region of interest within a Planar8 source image using an M x N kernel.

### Getting the Buffer Size

`vImageGetMinimumTempBufferSizeForMinMax` (page 231) Deprecated in Mac OS X v10.4
Returns the minimum size, in bytes, for the temporary buffer that the caller supplies to any of the Min or Max morphological functions. (Deprecated. Use the `kvImageGetTempBufferSize` flag with the appropriate morphological function instead of calling this function.)

## Functions

### vImageDilate_ARGB8888

Dilates a region of interest within an ARGB8888 source image using an M x N kernel.

```
vImage_Error vImageDilate_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const unsigned char *kernel,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

#### Parameters

*src*
A pointer to a vImage buffer structure that contains data for the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*
A pointer to the kernel data, which must be a packed array without any padding. The function uses the same kernel for all channels.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*flags*
The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageDilate_ARGBFFFF

Dilates a region of interest within an ARGBFFFF source image using an M x N kernel.

```
vImage_Error vImageDilate_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains data for the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*
A pointer to the kernel data, which must be a packed array without any padding.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*flags*
The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function uses the same kernel is for all channels. In addition to supplying space for the destination image, the `dest` parameter also specifies the size of the region of interest in within the source image. The region of interest has the same height and width (in pixels) as the destination buffer pointed to by dest.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageDilate_Planar8

Dilates a region of interest within a Planar8 source image using an M x N kernel.

```
vImage_Error vImageDilate_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const unsigned char *kernel,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains data for the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*
A pointer to the kernel data, which must be a packed array without any padding.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*flags*
The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageDilate_PlanarF

Dilates a region of interest within a PlanarF source image using an M x N kernel.

```
vImage_Error vImageDilate_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains data for the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*
A pointer to the kernel data, which must be a packed array without any padding.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*flags*
The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageErode_ARGB8888

Erodes a region of interest within an ARGB8888 source image using an M x N kernel.

```
vImage_Error vImageErode_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const unsigned char *kernel,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains data for the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the kernel data, which must be a packed array without any padding. The function uses the same kernel for all channels.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*flags*

The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Morphology.h`

## vImageErode_ARGBFFFF

Erodes a region of interest within an ARGBFFFF source image using an M x N kernel.

```
vImage_Error vImageErode_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*

A pointer to the kernel data, which must be a packed array without any padding. The function uses the same kernel for all channels.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*flags*

The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`Morphology.h`

## vImageErode_Planar8

Erodes a region of interest within a Planar8 source image using an M x N kernel.

```
vImage_Error vImageErode_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const unsigned char *kernel,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*
A pointer to the kernel data, which must be a packed array without any padding.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*flags*
The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageErode_PlanarF

Erodes a region of interest within a PlanarF source image using an M x N kernel.

```
vImage_Error vImageErode_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   const float *kernel,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains data for the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel*
A pointer to the kernel data, which must be a packed array without any padding.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*flags*
The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageGetMinimumTempBufferSizeForMinMax

Returns the minimum size, in bytes, for the temporary buffer that the caller supplies to any of the Min or Max morphological functions. (Deprecated in Mac OS X v10.4. Use the `kvImageGetTempBufferSize` flag with the appropriate morphological function instead of calling this function.)

```
size_t vImageGetMinimumTempBufferSizeForMinMax (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags,
   size_t bytesPerPixel
);
```

**Parameters**

*src*
A pointer to the vImage buffer structure that you plan to pass to the Min or Max function.

*dest*
A pointer to the vImage buffer structure that you plan to pass to the Min or Max function.

*kernel_height*
The height, in pixels, of the kernel that you plan to pass to the Min or Max function.

*kernel_width*
The width, in pixels, of the kernel that you plan to pass to the Min or Max function.

*flags*
The flags that you plan to pass to the Min or Max function.

*pixelBytes*

The number of bytes in a pixel. Make sure to pass the value appropriate for the format of the pixel.

**Return Value**
The minimum size, in bytes, of the temporary buffer.

**Discussion**
This function uses the *height* and *width* fields from the *src* or *dest* parameters; it does not use the *data* or *rowBytes* fields. If the size of the images you are processing remain the same, then the required size of the buffer also remains the same. More specifically, if, between two calls to `vImageGetMinimumTempBufferSizeForMinMax`, the `height` and `width` of the *src* and *dest* parameters do not increase, and the other parameters remain the same, then the result of the `vImageGetMinimumTempBufferSizeForMinMax` does not increase. This makes it easy to reuse the same temporary buffer when you process a number of images of the same size, as you would when tiling.

**Availability**
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.

**Declared In**
`Morphology.h`

## vImageMax_ARGB8888

Maximizes a region of interest within an ARGB8888 source image using an M x N kernel.

```
vImage_Error vImageMax_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*flags*
The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged. Set the `kvImageGetTempBufferSize` flag if you want to determine the minimum size to allocate for the `tempBuffer` parameter.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

The morphological operation Max is a special case of the dilation operation. In the Max operation, all the elements of the kernel have the same value. The actual value does not matter; only the size of the kernel is significant. vImage optimizes this special case, so the Max function is considerably faster than the Dilate function called with a uniform kernel.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageMax_ARGBFFFF

Maximizes a region of interest within an ARGBFFFF source image using an M x N kernel.

```
vImage_Error vImageMax_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*flags*

The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

The morphological operation Max is a special case of the dilation operation. In the Max operation, all the elements of the kernel have the same value. The actual value does not matter; only the size of the kernel is significant. vImage optimizes this special case, so the Max function is considerably faster than the Dilate function called with a uniform kernel.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageMax_Planar8

Maximizes a region of interest within a Planar8 source image using an M x N kernel.

```
vImage_Error vImageMax_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*
A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*flags*
The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

The morphological operation Max is a special case of the dilation operation. In the Max operation, all the elements of the kernel have the same value. The actual value does not matter; only the size of the kernel is significant. vImage optimizes this special case, so the Max function is considerably faster than the Dilate function called with a uniform kernel.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageMax_PlanarF

Maximizes with a region of interest within a PlanarF source image using an M x N kernel.

```
vImage_Error vImageMax_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*flags*

The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

The morphological operation Max is a special case of the dilation operation. In the Max operation, all the elements of the kernel have the same value. The actual value does not matter; only the size of the kernel is significant. vImage optimizes this special case, so the Max function is considerably faster than the Dilate function called with a uniform kernel.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageMin_ARGB8888

Minimizes a region of interest within an ARGB8888 source image using an M x N kernel.

```
vImage_Error vImageMin_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*
A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*
The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*
The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*
The height of the kernel in pixels. This value must be odd.

*kernel_width*
The width of the kernel in pixels. This value must be odd.

*flags*
The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

The morphological operation Min is a special case of the erosion operation. In the Min operation, all the elements of the kernel have the same value. The actual value does not matter; only the size of the kernel is significant. vImage optimizes this special case, so the Min function is considerably faster than the Erode function called with a uniform kernel.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageMin_ARGBFFFF

Minimizes a region of interest within an ARGBFFFF source image using an M x N kernel.

```
vImage_Error vImageMin_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*flags*

The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

The morphological operation Min is a special case of the erosion operation. In the Min operation, all the elements of the kernel have the same value. The actual value does not matter; only the size of the kernel is significant. vImage optimizes this special case, so the Min function is considerably faster than the Erode function called with a uniform kernel.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageMin_Planar8

Minimizes a region of interest within a Planar8 source image using an M x N kernel.

```
vImage_Error vImageMin_Planar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*flags*

The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**
`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

The morphological operation Min is a special case of the erosion operation. In the Min operation, all the elements of the kernel have the same value. The actual value does not matter; only the size of the kernel is significant. vImage optimizes this special case, so the Min function is considerably faster than the Erode function called with a uniform kernel.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

## vImageMin_PlanarF

Minimizes a region of interest within a PlanarF source image using an M x N kernel.

```
vImage_Error vImageMin_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   void *tempBuffer,
   vImagePixelCount srcOffsetToROI_X,
   vImagePixelCount srcOffsetToROI_Y,
   vImagePixelCount kernel_height,
   vImagePixelCount kernel_width,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains data for the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

The size (number of rows and number of columns) of the destination buffer also specifies the size of the region of interest in the source buffer.

*tempBuffer*

A pointer to a temporary buffer. If you pass `NULL`, the function allocates the buffer, then deallocates it before returning. Alternatively, you can allocate the buffer yourself, in which case you are responsible for deallocating it when you is no longer need it.

If you want to allocate the buffer yourself, see the Discussion for information on how to determine the minimum size that you must allocate.

*srcOffsetToROI_X*

The horizontal offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*srcOffsetToROI_Y*

The vertical offset, in pixels, to the upper-left pixel of the region of interest within the source image.

*kernel_height*

The height of the kernel in pixels. This value must be odd.

*kernel_width*

The width of the kernel in pixels. This value must be odd.

*flags*

The options to use when performing the morphological operation. Set the `kvImageDoNotTile` flag if you plan to perform your own tiling or use multithreading. Set the `kvImageLeaveAlphaUnchanged` flag to specify that the alpha channel should be copied to the destination image unchanged.

**Return Value**

`kvImageNoError`, otherwise it returns one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

If you want to allocate the memory for the `tempBuffer` parameter yourself, call this function twice, as follows:

1. To determine the minimum size for the temporary buffer, the first time you call this function pass the `kvImageGetTempBufferSize` flag. Pass the same values for all other parameters that you intend to use in for the second call. The function returns the required minimum size, which should be a positive value. (A negative returned value indicates an error.) The `kvImageGetTempBufferSize` flag prevents the function from performing any processing other than to determine the minimum buffer size.

2. After you allocate enough space for a buffer of the returned size, call the function a second time, passing a valid pointer in the `tempBuffer` parameter. This time, do not pass the `kvImageGetTempBufferSize` flag.

The morphological operation Min is a special case of the erosion operation. In the Min operation, all the elements of the kernel have the same value. The actual value does not matter; only the size of the kernel is significant. vImage optimizes this special case, so the Min function is considerably faster than the Erode function called with a uniform kernel.

**Availability**
Available in Mac OS X v10.3 and later.

**Declared In**
`Morphology.h`

# vImage Transform Reference

| | |
|---|---|
| **Framework:** | Accelerate/vImage |
| **Declared in** | Transform.h |
| **Companion guide** | vImage Programming Guide |

## Overview

Transformation functions alter the values of pixels in the image. Unlike convolutions, transformation functions do not depend on the values of nearby pixels. The vImage transformation functions fall into four broad categories:

- Gamma correction functions correct the brightness profile of an image by multiplying each pixel by the value of the function. Gamma correction prepares an image for display or printing on a particular device.
- Lookup table functions are like the piecewise polynomial functions, but instead of applying a polynomial they use a lookup table that you supply.
- Matrix multiplication functions have a variety of uses, such as to convert between color spaces (RGB and YUV, for example), change a color image to a grayscale one, and for "color twisting."
- Piecewise functions are similar to the gamma correction functions, but instead of applying a predefined gamma function they apply one or more polynomials that you supply. The number of polynomials must be an integer power of 2, and they must all be of the same order.

Transformation functions use a vImage buffer structure (`vImage_Buffer`—see *vImage Data Types and Constants Reference*) to receive and supply image data. This buffer contains a pointer to image data, the height and width (in pixels) of the image data, and the number of row bytes. You actually pass a pointer to a vImage buffer structure.

Some transformation functions "work in place". That is, the source and destination images can occupy the same memory if the they are strictly aligned pixel for pixel. For these, you can can provide a pointer to the same vImage buffer structure for one of the source images and the destination image.

## Functions by Task

### Transforming with a Lookup Table

`vImageLookupTable_Planar8toPlanarF` (page 251)
Uses a lookup table to transform an image in Planar8 format to an image in PlanarF format.

`vImageLookupTable_PlanarFtoPlanar8` (page 252)
Uses a lookup table to transform an image in PlanarF format to an image in Planar8 format.

`vImageInterpolatedLookupTable_PlanarF` (page 250)
Uses a lookup table to transform an image in PlanarF format to an image in PlanarF format.

## Applying a Polynomial

`vImagePiecewisePolynomial_PlanarF` (page 259)
Applies a set of piecewise polynomials to an image in PlanarF format.

`vImagePiecewisePolynomial_Planar8toPlanarF` (page 257)
Applies a set of piecewise polynomials to transform an image in Planar8 format to an image in PlanarF format.

`vImagePiecewisePolynomial_PlanarFtoPlanar8` (page 260)
Applies a set of piecewise polynomials to transform an image in PlanarF format to an image in Planar8 format.

`vImagePiecewiseRational_PlanarF` (page 261)
Applies a piecewise rational expression to an image in PlanarF format.

## Multiplying Pixels by a Matrix

`vImageMatrixMultiply_Planar8` (page 255)
Operates on a set of 8-bit source image planes, multiplying each pixel by the provided matrix to produce a set of 8-bit destination image planes.

`vImageMatrixMultiply_PlanarF` (page 256)
Operates upon a set of floating-point source image planes, multiplying each pixel by the provided matrix to produce a set of floating-point destination image planes.

`vImageMatrixMultiply_ARGB8888` (page 253)
Operates upon an interleaved 8-bit source image, multiplying each pixel by the provided matrix to produce an interleaved 8-bit destination image.

`vImageMatrixMultiply_ARGBFFFF` (page 254)
Operates upon an interleaved floating-point source image, multiplying each pixel by the provided matrix to produce an interleaved floating-point destination image.

## Correcting Gamma

`vImageCreateGammaFunction` (page 247)
Returns a gamma function object.

`vImageDestroyGammaFunction` (page 248)
Destroys a gamma function object created.

`vImageGamma_Planar8toPlanarF` (page 248)
Applies a gamma function to a Planar8 image to produce a PlanarF image.

`vImageGamma_PlanarFtoPlanar8` (page 249)
Applies a gamma function to an image in PlanarF format to an image in Planar8 format.

`vImageGamma_PlanarF` (page 249)
Applies a gamma function to a PlanarF image.

# Functions

## vImageCreateGammaFunction

Returns a gamma function object.

```
GammaFunction vImageCreateGammaFunction (
   float gamma,
   int gamma_type,
   vImage_Flags flags
);
```

### Parameters

*gamma*
The exponent of a power function for calculating full-precision gamma correction.

*gamma-type*
A selector for the type of gamma correction to use. Pass one of the full- or half-precision type constants defined in "Gamma Function Types" (page 264).

*flags*
Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

### Return Value

A gamma function object that encapsulates a gamma value, a gamma function type, and option flags.

### Discussion

You can pass a gamma function object to any of the three gamma correction functions: `vImageGamma_Planar8toPlanarF` (page 248), `vImageGamma_PlanarFtoPlanar8` (page 249), `vImageGamma_PlanarF` (page 249).

The gamma-type parameter determines the type of calculation to be used. The simplest calculation is:

```
if (value == 0) result = 0;

else {
    if (value < 0)
        sign = -1.0f;
    else
        sign = 1.0f;
    result = pow( fabs( value ), gamma) * sign;
}
```

This calculation results in symmetric gamma curves about 0, and makes sure that only well-behaved values are used in `pow()`.

You can use an equivalent calculation that uses a more efficient method, depending on the desired precision.

In addition to the full-precision gamma correction, there is a faster half-precision option that provides 12-bit precision.

f your data will ultimately be converted to 8-bit integer data, consider using half-precision. The half-precision variants work correctly only for floating-point input values in the range 0.0 ... 1.0, though out-of-range values produce results that clamp appropriately to 0 or 255 on conversion back to 8-bit. In addition, there are restrictions on the range of the exponent: it must be positive, in the range 0.1 to 10.0.

Finally, there is a set of still faster half-precision options that use predefined gamma values, ignoring the value set in `vImageCreateGammaFunction`. These options have the same restrictions on input values as stated previously.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

## vImageDestroyGammaFunction

Destroys a gamma function object created.

```
void vImageDestroyGammaFunction (
   GammaFunction f
);
```

**Parameters**

*f*

A gamma function object created with the function `vImageCreateGammaFunction` (page 247).

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

## vImageGamma_Planar8toPlanarF

Applies a gamma function to a Planar8 image to produce a PlanarF image.

```
vImage_Error vImageGamma_Planar8toPlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const GammaFunction gamma,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*gamma*

A gamma function object, created with by calling the function `vImageCreateGammaFunction` (page 247).

*flags*

Reserved for future use; pass 0.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Transform.h`

## vImageGamma_PlanarF

Applies a gamma function to a PlanarF image.

```
vImage_Error vImageGamma_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const GammaFunction gamma,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*gamma*

A gamma function object, created with by calling the function `vImageCreateGammaFunction` (page 247).

*flags*

Reserved for future use; pass 0.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Transform.h`

## vImageGamma_PlanarFtoPlanar8

Applies a gamma function to an image in PlanarF format to an image in Planar8 format.

```
vImage_Error vImageGamma_PlanarFtoPlanar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const GammaFunction gamma,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*gamma*

A gamma function object, created with by calling the function `vImageCreateGammaFunction` (page 247).

*flags*

Reserved for future use; pass 0.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Transform.h`

## vImageInterpolatedLookupTable_PlanarF

Uses a lookup table to transform an image in PlanarF format to an image in PlanarF format.

```
vImage_Error vImageInterpolatedLookupTable_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const Pixel_F *table,
   vImagePixelCount tableEntries,
   float maxFloat,
   float minFloat,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*table*
A lookup table of floating-point values.

*tableEntries*
A value of type `vImagePixelCount`, giving the number of values in the array.

*maxFloat*
A value of type `float`.

*minFloat*
A value of type `float`.

*flags*
The options to use when performing the transformation. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
It will work in place. The table contains an arbitrary number of values; it is entered with an index interpolated from a value from the source image, to look up a floating-point value for the destination image.

The input pixel is first clipped to the range *minFloat* ... *maxFloat*. The result is then calculated as

```
float clippedPixel =MAX(MIN(src_pixel, maxFloat), minFloat);
float fIndex =  (float) (tableEntries - 1) * (clippedPixel -  minFloat)
               / (maxFloat - minFloat);
float fract = fIndex - floor(fIndex);
unsigned long i = fIndex;
float result = table[i] * (1.0f - fract) + table[i + 1] * fract;
```

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

## vImageLookupTable_Planar8toPlanarF

Uses a lookup table to transform an image in Planar8 format to an image in PlanarF format.

```
vImage_Error vImageLookupTable_Planar8toPlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const Pixel_F table[256],
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*table*
A lookup table that contains 256 values.

*flags*
Reserved for future use; pass 0.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
For each pixel, the 8-bit value from the source Planar8 image is used as an index to get a floating-point value from the table. This value is used as the corresponding pixel in the PlanarF result image.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

## vImageLookupTable_PlanarFtoPlanar8

Uses a lookup table to transform an image in PlanarF format to an image in Planar8 format.

```
vImage_Error vImageLookupTable_PlanarFtoPlanar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const Pixel_8 table[4096],
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*table*
A lookup table that contains 4096 values.

*flags*
Reserved for future use; pass 0.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The table contains 4096 values; it is entered with an integer index derived from a pixel value in the source image, to look up an 8-bit value for the destination image.

For each pixel, the floating-point value from the source PlanarF image is first clipped to the range 0.0 ... 1.0, and then converted to an integer in the range 0 ... 4095. The conversion calculation is equivalent to

```
    if (realValue < 0.0f) realValue = 0.0f;
    if (realValue > 1.0f) realValue = 1.0f;
```

```
intValue = (int)(realValue * 4095.0f + 0.5f);
```

This integer is used as an index to get an 8-bit value from the table. This value is used as the corresponding pixel in the Planar8 result image.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

## vImageMatrixMultiply_ARGB8888

Operates upon an interleaved 8-bit source image, multiplying each pixel by the provided matrix to produce an interleaved 8-bit destination image.

```
vImage_Error vImageMatrixMultiply_ARGB8888 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const int16_t matrix[4 *4],
   int32_t divisor,
   const int16_t *pre_bias,
   const int32_t *post_bias,
   vImage_Flags flags
);
```

**Parameters**

*src*
: A pointer to a vImage buffer structure that contains the source image.

*dest*
: A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*matrix*
: A 1-dimensional array whose values represent a 4x4 matrix. vImage multiplies each source pixel by this matrix to produce a destination pixel.

*divisor*
: A divisor for normalization after performing the matrix multiplication.

*pre_bias*
: A packed array of bias values, one for each source plane. vImage adds the appropriate bias value to each source value prior to matrix multiplication. Pass `NULL` if you do not want to apply a preprocessing bias value.

*post_bias*
: A packed array of bias values, one for each destination plane. vImage adds the appropriate bias value to each destination value after matrix multiplication. Pass `NULL` if you do not want to apply a preprocessing bias value.

*flags*
: Reserved for future use; pass 0.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
Be aware that 32-bit signed accumulators are used. If the sum over any matrix column is greater than ±223, overflow may occur. Generally speaking this will not happen because the matrix elements are 16-bit integers, so it would take more than 256 source planes before trouble could arise.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

## vImageMatrixMultiply_ARGBFFFF

Operates upon an interleaved floating-point source image, multiplying each pixel by the provided matrix to produce an interleaved floating-point destination image.

```
vImage_Error vImageMatrixMultiply_ARGBFFFF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const float matrix[4 *4],
   const float *pre_bias,
   const float *post_bias,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*matrix*
A 1-dimensional array whose values represent a 4x4 matrix. vImage multiplies each source pixel by this matrix to produce a destination pixel.

*pre_bias*
A packed array of bias values, one for each source plane. vImage adds the appropriate bias value to each source value prior to matrix multiplication. Pass `NULL` if you do not want to apply a preprocessing bias value.

*post_bias*
A packed array of bias values, one for each destination plane. vImage adds the appropriate bias value to each destination value after matrix multiplication. Pass `NULL` if you do not want to apply a preprocessing bias value.

*flags*
Reserved for future use; pass 0.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The operation is the same as `vImageMatrixMultiply_ARGB8888` (page 253) except that floating-point values are used and there is no divisor.

Be aware that 32-bit signed accumulators are used. If the sum over any matrix column is greater than ±223, overflow may occur. Generally speaking this will not happen because the matrix elements are 16-bit integers, so it would take more than 256 source planes before trouble could arise.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

## vImageMatrixMultiply_Planar8

Operates on a set of 8-bit source image planes, multiplying each pixel by the provided matrix to produce a set of 8-bit destination image planes.

```
vImage_Error vImageMatrixMultiply_Planar8 (
   const vImage_Buffer *srcs[],
   const vImage_Buffer *dests[],
   uint32_t src_planes,
   uint32_t dest_planes,
   const int16_t matrix[],
   int32_t divisor,
   const int16_t *pre_bias,
   const int32_t *post_bias,
   vImage_Flags flags
);
```

**Parameters**

*srcs*
A pointer to an array of vImage buffer structures, one buffer for each source plane.

*dests*
A pointer to an array of pointers to vImage buffer data structures, one buffer structure for each destination plane. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of these structures, and for allocating data buffers of the appropriate size. On return, the data buffers in these structures contains the destination image data for each plane. When you no longer need the data buffers, you must deallocate the memory.

*src_planes*
The number of source planes.

*dest_planes*
The number of destination planes.

*matrix*
A 1-dimensional array whose values represent a matrix with dimensions *dest_planes*x*src_planes*. vImage multiplies each source pixel by this matrix to produce a destination pixel.

*divisor*
A divisor for normalization after performing the matrix multiplication.

*pre_bias*

A packed array of bias values, one for each source plane. vImage adds the appropriate bias value to each source value prior to matrix multiplication. Pass `NULL` if you do not want to apply a preprocessing bias value.

*post_bias*

A packed array of bias values, one for each destination plane. vImage adds the appropriate bias value to each destination value after matrix multiplication. Pass `NULL` if you do not want to apply a preprocessing bias value.

*flags*

The options to use when performing the transformation. Pass `kvImageDoNotTile` if you plan to perform your own tiling or use multithreading.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

Be aware that 32-bit signed accumulators are used. If the sum over any matrix column is greater than ±223, overflow may occur. Generally speaking this will not happen because the matrix elements are 16-bit integers, so it would take more than 256 source planes before trouble could arise.

**Availability**

Available in Mac OS X v10.4 and later.

**Declared In**

`Transform.h`

## vImageMatrixMultiply_PlanarF

Operates upon a set of floating-point source image planes, multiplying each pixel by the provided matrix to produce a set of floating-point destination image planes.

```
vImage_Error vImageMatrixMultiply_PlanarF (
   const vImage_Buffer *srcs[],
   const vImage_Buffer *dests[],
   uint32_t src_planes,
   uint32_t dest_planes,
   const float matrix[],
   const float *pre_bias,
   const float *post_bias,
   vImage_Flags flags
);
```

**Parameters**

*srcs*

A pointer to an array of vImage buffer structures, one buffer for each source plane.

*dests*

A pointer to an array of pointers to vImage buffer data structures, one buffer structure for each destination plane. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of these structures, and for allocating data buffers of the appropriate size. On return, the data buffers in these structures contains the destination image data for each plane. When you no longer need the data buffers, you must deallocate the memory.

*src_planes*

The number of source planes.

*dest_planes*
The number of destination planes.

*matrix*
A 1-dimensional array whose values represent a matrix with dimensions *dest_planes*x*src_planes*. vImage multiplies each source pixel by this matrix to produce a destination pixel.

*pre_bias*
A packed array of bias values, one for each source plane. vImage adds the appropriate bias value to each source value prior to matrix multiplication. Pass `NULL` if you do not want to apply a preprocessing bias value.

*post_bias*
A packed array of bias values, one for each destination plane. vImage adds the appropriate bias value to each destination value after matrix multiplication. Pass `NULL` if you do not want to apply a preprocessing bias value.

*flags*
Reserved for future use; pass 0.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
The operation is the same as `vImageMatrixMultiply_Planar8` (page 255) except that floating-point values are used and there is no divisor.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

## vImagePiecewisePolynomial_Planar8toPlanarF

Applies a set of piecewise polynomials to transform an image in Planar8 format to an image in PlanarF format.

```
vImage_Error vImagePiecewisePolynomial_Planar8toPlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const float **coefficients,
   const float *boundaries,
   uint32_t order,
   uint32_t log2segments,
   vImage_Flags flags
);
```

**Parameters**

*src*
A pointer to a vImage buffer structure that contains the source image.

*dest*
A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*coefficients*

A pointer to an array of polynomial coefficient arrays. Each polynomial coefficient array contains the coefficients for one polynomial. Note that a polynomial of order R has R+1 coefficients. All the polynomial coefficient arrays must be the same size, R+1, and in each array the coefficients must be ordered from the 0th-order term to the highest-order term.

*boundaries*

A pointer to an array of boundary values, in increasing order, for separating adjacent ranges of pixel values. The first boundary value is the lowest in the range; input values lower than this are clipped to this value. The last boundary value is the highest in the range; input values higher than this are clipped to this value. The boundary values between the first and last separate the subranges from each other.

*log2segments*

The number of polynomials represented as a base-2 logarithm. If you pass a non-integer power-of-two number of polynomials (for example, 5), you must round up to the next integer power of 2 (for the example of 5, that would be 8), and simply repeat the last polynomial the appropriate number of times.

*flags*

Reserved for future use; pass 0.

**Return Value**

`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**

You can approximate many different correction functions by carefully choosing the polynomials and the ranges of input values they operate on. The number of polynomials must be a non-negative integer power of 2.

Suppose that you want to use N polynomials of order R to process N contiguous ranges of pixel values. For each pixel in the image, the range of usable values is divided into segments by the values passed in the *boundaries* array. Each segment is processed by the corresponding polynomial. Since there are N polynomials, then there must be N segments, so you must supply N+1 boundaries.

You must order the boundaries by increasing value. The ith segment is the set of pixel values that fall in the range:

```
boundary[i] <= value < boundary{i+1}
```

where i ranges from 0 to N. Values in this segment are processed by the i-th polynomial.

From a performance standpoint, it costs much more to resolve additional polynomials than to work with higher-order polynomials. You typically achieve better performance with one 9th-order polynomial that covers the whole range of values you are interested in than with many lower-order polynomials covering the range piecewise.

This function uses single-precision floating-point arithmetic. As a result, polynomials with large high-order coefficients may cause significant rounding error.

**Availability**

Available in Mac OS X v10.4 and later.

**See Also**

`vImagePiecewisePolynomial_PlanarF` (page 259)

**Declared In**
`Transform.h`

## vImagePiecewisePolynomial_PlanarF

Applies a set of piecewise polynomials to an image in PlanarF format.

```
vImage_Error vImagePiecewisePolynomial_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const float **coefficients,
   const float *boundaries,
   uint32_t order,
   uint32_t log2segments,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*coefficients*

A pointer to an array of polynomial coefficient arrays. Each polynomial coefficient array contains the coefficients for one polynomial. Note that a polynomial of order R has R+1 coefficients. All the polynomial coefficient arrays must be the same size, R+1, and in each array the coefficients must be ordered from the 0th-order term to the highest-order term.

*boundaries*

A pointer to an array of boundary values, in increasing order, for separating adjacent ranges of pixel values. The first boundary value is the lowest in the range; input values lower than this are clipped to this value. The last boundary value is the highest in the range; input values higher than this are clipped to this value. The boundary values between the first and last separate the subranges from each other.

*log2segments*

The number of polynomials represented as a base-2 logarithm. If you pass a non-integer power-of-two number of polynomials (for example, 5), you must round up to the next integer power of 2 (for the example of 5, that would be 8), and simply repeat the last polynomial the appropriate number of times.

*flags*

Reserved for future use; pass 0.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
You can approximate many different correction functions by carefully choosing the polynomials and the ranges of input values they operate on. The number of polynomials must be a non-negative integer power of 2.

Suppose that you want to use N polynomials of order R to process N contiguous ranges of pixel values. For each pixel in the image, the range of usable values is divided into segments by the values passed in the *boundaries* array. Each segment is processed by the corresponding polynomial. Since there are N polynomials, then there must be N segments, so you must supply N+1 boundaries.

You must order the boundaries by increasing value. The ith segment is the set of pixel values that fall in the range:

```
boundary[i] <= value < boundary{i+1}
```

where i ranges from 0 to N. Values in this segment are processed by the i-th polynomial.

From a performance standpoint, it costs much more to resolve additional polynomials than to work with higher-order polynomials. You typically achieve better performance with one 9th-order polynomial that covers the whole range of values you are interested in than with many lower-order polynomials covering the range piecewise.

This function uses single-precision floating-point arithmetic. As a result, polynomials with large high-order coefficients may cause significant rounding error.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

## vImagePiecewisePolynomial_PlanarFtoPlanar8

Applies a set of piecewise polynomials to transform an image in PlanarF format to an image in Planar8 format.

```
vImage_Error vImagePiecewisePolynomial_PlanarFtoPlanar8 (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const float **coefficients,
   const float *boundaries,
   uint32_t order,
   uint32_t log2segments,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*coefficients*

A pointer to an array of polynomial coefficient arrays. Each polynomial coefficient array contains the coefficients for one polynomial. Note that a polynomial of order R has R+1 coefficients. All the polynomial coefficient arrays must be the same size, R+1, and in each array the coefficients must be ordered from the 0th-order term to the highest-order term.

*boundaries*
: A pointer to an array of boundary values, in increasing order, for separating adjacent ranges of pixel values. The first boundary value is the lowest in the range; input values lower than this are clipped to this value. The last boundary value is the highest in the range; input values higher than this are clipped to this value. The boundary values between the first and last separate the subranges from each other.

*log2segments*
: The number of polynomials represented as a base-2 logarithm. If you pass a non-integer power-of-two number of polynomials (for example, 5), you must round up to the next integer power of 2 (for the example of 5, that would be 8), and simply repeat the last polynomial the appropriate number of times.

*flags*
: Reserved for future use; pass 0.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
You can approximate many different correction functions by carefully choosing the polynomials and the ranges of input values they operate on. The number of polynomials must be a non-negative integer power of 2.

Suppose that you want to use N polynomials of order R to process N contiguous ranges of pixel values. For each pixel in the image, the range of usable values is divided into segments by the values passed in the *boundaries* array. Each segment is processed by the corresponding polynomial. Since there are N polynomials, then there must be N segments, so you must supply N+1 boundaries.

You must order the boundaries by increasing value. The ith segment is the set of pixel values that fall in the range:

```
boundary[i] <= value < boundary{i+1}
```

where i ranges from 0 to N. Values in this segment are processed by the i-th polynomial.

From a performance standpoint, it costs much more to resolve additional polynomials than to work with higher-order polynomials. You typically achieve better performance with one 9th-order polynomial that covers the whole range of values you are interested in than with many lower-order polynomials covering the range piecewise.

This function uses single-precision floating-point arithmetic. As a result, polynomials with large high-order coefficients may cause significant rounding error.

**Availability**
Available in Mac OS X v10.4 and later.

**See Also**
`vImagePiecewisePolynomial_PlanarF` (page 259)

**Declared In**
`Transform.h`

## vImagePiecewiseRational_PlanarF

Applies a piecewise rational expression to an image in PlanarF format.

```
vImage_Error vImagePiecewiseRational_PlanarF (
   const vImage_Buffer *src,
   const vImage_Buffer *dest,
   const float **topCoefficients,
   const float **bottomCoefficients,
   const float *boundaries,
   uint32_t topOrder,
   uint32_t bottomOrder,
   uint32_t log2segments,
   vImage_Flags flags
);
```

**Parameters**

*src*

A pointer to a vImage buffer structure that contains the source image.

*dest*

A pointer to a vImage buffer data structure. You are responsible for filling out the `height`, `width`, and `rowBytes` fields of this structure, and for allocating a data buffer of the appropriate size. On return, the data buffer pointed to by this structure contains the destination image data. When you no longer need the data buffer, you must deallocate the memory.

*topCoefficients*

An array of pointers to polynomial coefficient arrays The array of pointers has length $2^{log2segments}$. Each array pointed to has length `topOrder`+1.

Each polynomial coefficient array contains the coefficients for one polynomial. Note that a polynomial of order R has R+1 coefficients. All the polynomial coefficient arrays must be the same size, R+1, and in each array the coefficients must be ordered from the 0th-order term to the highest-order term.

*bottomCoefficients*

An array of pointers to polynomial coefficient arrays The array of pointers has length $2^{log2segments}$. Each array pointed to has length `bottomOrder`+1.

Each polynomial coefficient array contains the coefficients for one polynomial. Note that a polynomial of order R has R+1 coefficients. All the polynomial coefficient arrays must be the same size, R+1, and in each array the coefficients must be ordered from the 0th-order term to the highest-order term. These do not need to be the same order as the top polynomials.

*boundaries*

An array of floating-point values with size $(2^{log2segments})+1$, in increasing order, for separating adjacent ranges of pixel values. The first boundary value is the lowest in the range; input values lower than this are clipped to this value. The last boundary value is the highest in the range; input values higher than this are clipped to this value. The boundary values between the first and last separate the subranges from each other. The boundaries must be the same for both the top and bottom polynomials.

*topOrder*

The order of the top polynomial. Make sure you pass the *order* (that is, the highest power of x), not the number of coefiicients.

*bottomOrder*

The order of the bottom polynomial. Make sure you pass the *order* (that is, the highest power of x), not the number of coefiicients.

*log2segments*

The number of rationals represented as a base-2 logarithm. If you pass a non-integer power-of-two number of rational (for example, 5), you must round up to the next integer power of 2 (for the example of 5, that would be 8), and simply repeat the last rational the appropriate number of times.

*flags*

Reserved for future use; pass 0.

**Return Value**
`kvImageNoError`, otherwise one of the error codes described in *vImage Data Types and Constants Reference*.

**Discussion**
This function is similar to `vImagePiecewisePolynomial_PlanarF` (page 259) except that it evaluates a piecewise rational expression in the form of:

$$result = \frac{c0 + c1 * x + c2 * x^2 + c3 * x^3 \ldots}{d0 + d1 * x + d2 * x^2 + d3 * x^3 \ldots}$$

Each polynomial has its own set of coefficients and its own polynomial order. The two polynomials share the same set of segment boundaries. If the polynomials are split then all the top polynomials must be of the same order, and all the bottom polynomials must be of the same order. However, regardless of whether the polynomial is split or not, the top polynomials do not need to be the same order as the bottom polynomials.

This function does not deliver IEEE-754 correct division. The divide does not round per the IEEE-754 current rounding mode. It incurs up to 2 ULPs (Units in the Last Place) of error. Edge cases involving denormals, infinities, NaNs and division by zero return undefined results. (They will not crash, but NaN is a likely result in such cases.) Denormals can be rescued on AltiVec enabled machines by turning off the Non-Java bit in the VSCR, at the expense of taking a many-thousand cycle kernel exception every time a denormal number is encountered. Since you can predict ahead of time whether a given set of bounded polynomials is going to encounter these conditions, this problem should be avoidable by wise choice of polynomials. Developers who require IEEE-754 correct results should call the polynomial evaluator above twice and do the division themselves.

The approximate cost of evaluating a rational (in the same units as polynomial above) is:

```
time = (base cost to touch all the data) + top polynomial order
      + bottom polynomial order + 4 + 4 * log2segments
```

With data not in cache, the time may be significantly different. For sufficiently small polynomials, the cost may be a fixed cost, dependent only on how much data is touched, and not on polynomial order.

This performance behavior is provided to help you evaluate speed tradeoffs. It is not a guaranteed. It is subject to change in future operating system revisions, and may be different on different hardware within the same or different operating system revisions.

**Availability**
Available in Mac OS X v10.4 and later.

**Declared In**
`Transform.h`

# Constants

## Gamma Function Types

Types of full- or half-precision gamma functions.

```
enum
{
    kvImageGamma_UseGammaValue                  =   0,
    kvImageGamma_UseGammaValue_half_precision   =   1,
    kvImageGamma_5_over_9_half_precision        =   2,
    kvImageGamma_9_over_5_half_precision        =   3,
    kvImageGamma_5_over_11_half_precision       =   4,
    kvImageGamma_11_over_5_half_precision       =   5,
    kvImageGamma_sRGB_forward_half_precision    =   6,
    kvImageGamma_sRGB_reverse_half_precision    =   7,
    kvImageGamma_11_over_9_half_precision       =   8,
    kvImageGamma_9_over_11_half_precision       =   9,
    kvImageGamma_BT709_forward_half_precision   =  10,
    kvImageGamma_BT709_reverse_half_precision   =   11
};
```

**Constants**

`kvImageGamma_UseGammaValue`

Full-precision calculation using the gamma value set in `vImageCreateGammaFunction`.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_UseGammaValue_half_precision`

Half-precision calculation using the gamma value set in `vImageCreateGammaFunction`.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_5_over_9_half_precision`

Half-precision calculation using a gamma value of 5/9 or 1/1.8.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_9_over_5_half_precision`

Half-precision calculation using a gamma value of 9/5 or 1.8.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_5_over_11_half_precision`

Half-precision calculation using a gamma value of 5/11 or 1/2.2.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_11_over_5_half_precision`
Half-precision calculation using a gamma value of 11/5 or 2.2. On exit, gamma is 5/11.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_sRGB_forward_half_precision`
Half-precision calculation using the sRGB standard gamma value of 2.2.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_sRGB_reverse_half_precision`
Half-precision calculation using the sRGB standard gamma value of 1/2.2.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_11_over_9_half_precision`
Half-precision calculation using a gamma value of 11/9 or (11/5)/(9/5).

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_9_over_11_half_precision`
Half-precision calculation using a gamma value of 9/11 or (9/5)/(11/5).

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_BT709_forward_half_precision`
ITU-R BT.709 standard. This is like kvImageGamma_sRGB_forward_half_precision above but without the 1.125 viewing gamma for computer graphics: `x<0.081? x/4.5: pow((x+0.099)/1.099, 1/0.45)`.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

`kvImageGamma_BT709_reverse_half_precision`
ITU-R BT.709 standard reverse. This is like kvImageGamma_sRGB_reverse_half_precision above but without the 1.125 viewing gamma for computer graphics: `x<0.018? 4.5*x: 1.099*pow(x,0.45) - 0.099`.

Available in Mac OS X v10.4 and later.

Declared in `Transform.h`.

**Declared In**
`Transform.h`

# Document Revision History

This table describes the changes to *vImage Reference Collection*.

| Date | Notes |
| --- | --- |
| 2007-07-12 | New collection that describes the C programming interface for high-performance image processing. |

Document Revision History

# Index

## E



## G



## K



## P

## R



## V