vImage Transform Reference

Performance > Graphics & Imaging

2007-07-12

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

Simultaneously published in the United States and Canada.

Contents

vImage Transform Reference

Framework:	Accelerate/vImage
Companion guide	vImage Programming Guide
Declared in	Transform.h

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 11)
Uses a lookup table to transform an image in Planar8 format to an image in PlanarF format.

vImageLookupTable_PlanarFtoPlanar8 (page 12)
: Uses a lookup table to transform an image in PlanarF format to an image in Planar8 format.

vImageInterpolatedLookupTable_PlanarF (page 10)
: Uses a lookup table to transform an image in PlanarF format to an image in PlanarF format.

Applying a Polynomial

vImagePiecewisePolynomial_PlanarF (page 19)
: Applies a set of piecewise polynomials to an image in PlanarF format.

vImagePiecewisePolynomial_Planar8toPlanarF (page 17)
: Applies a set of piecewise polynomials to transform an image in Planar8 format to an image in PlanarF format.

vImagePiecewisePolynomial_PlanarFtoPlanar8 (page 20)
: Applies a set of piecewise polynomials to transform an image in PlanarF format to an image in Planar8 format.

vImagePiecewiseRational_PlanarF (page 21)
: Applies a piecewise rational expression to an image in PlanarF format.

Multiplying Pixels by a Matrix

vImageMatrixMultiply_Planar8 (page 15)
: 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 16)
: 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 13)
: 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 14)
: 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 7)
: Returns a gamma function object.

vImageDestroyGammaFunction (page 8)
: Destroys a gamma function object created.

vImageGamma_Planar8toPlanarF (page 8)
: Applies a gamma function to a Planar8 image to produce a PlanarF image.

vImageGamma_PlanarFtoPlanar8 (page 9)
: Applies a gamma function to an image in PlanarF format to an image in Planar8 format.

`vImageGamma_PlanarF` (page 9)
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 24).

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 8), `vImageGamma_PlanarFtoPlanar8` (page 9), `vImageGamma_PlanarF` (page 9).

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 7).

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 7).

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 7).

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 7).

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 13) 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 15) 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 19)

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 19)

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 19) 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 Transform Reference*.

Date	Notes
2007-07-12	Updated for Mac OS X v10.4.
	The content in this document was formerly part of *Optimizing Image Processing With vImage*.
	Added `vImagePiecewiseRational_PlanarF` (page 21).

Document Revision History

Index