Article

Building a Basic Image-Processing Workflow

Resize an image with vImage.

Overview

vImage provides fast and accurate high-level functions for image manipulation; for example, compositing, convolution, and histogram operations. It operates on common image formats through `vImage_Buffer` structures. vImage buffers describe the size of an image and the number of bytes in each row, and point to the image pixel data. Buffers are initialized from Core Graphics images, Core Video pixel buffers, or raw pixel data. The pixel data a buffer points to can be used to create a new Core Graphics image or can be copied into a Core Video pixel buffer.

In the simplest workflow, you convert an image to a vImage buffer, apply an operation to the buffer, and convert the buffer back to an image. In this example, the width and height of the result are one-third of the original:

Initialize an Image Format and vImage Buffers

To learn about initializing the buffers you'll need to perform a scaling operation, see Converting bitmap data between Core Graphics images and vImage buffers and Creating and Populating Buffers from Core Graphics Images. In this example, you'll need the image format and buffers discussed in Creating and Populating Buffers from Core Graphics Images. However, you'll use the following code to initialize a destination buffer with a height and width that are a quarter of the source dimensions.

```
let destinationHeight = Int(CGFloat(sourceBuffer.height) * 0.25)
let destinationWidth = Int(CGFloat(sourceBuffer.width) * 0.25)

guard var destinationBuffer = try? vImage_Buffer(width: destinationWidth,
                                                 height: destinationHeight,
                                                 bitsPerPixel: format.bitsPerPixel) else {
                                                    return nil
}

defer {
    destinationBuffer.free()
}
```

Apply the Scale Operation

If you're rescaling an image with premultiplied alpha (that is, with a `bitmapInfo` value with `CGImageAlphaInfo.premultipliedFirst` or `CGImageAlphaInfo.premultiplied Last`), before you apply the scale operation, see Building a Basic Image-Processing Workflow.

Otherwise, with the source and destination buffers properly initialized, you're ready to perform the scaling operation. Because your format contains four 8-bit channels, you use the `vImageScale _ARGB8888(_:_:_:_:)` function. This function works equally well on all channel orderings; for example, RGBA or BGRA.

```
error = vImageScale_ARGB8888(&sourceBuffer,
                             &destinationBuffer,
                             nil,
                             vImage_Flags(kvImageHighQualityResampling))

guard error == kvImageNoError else {
    fatalError("Error in vImageScale_ARGB8888: \(error)")
}
```

The `kvImageHighQualityResampling` flag uses a high-quality Lanczos 5 x 5 resampling filter. If you require faster scaling, pass `kvImageNoFlags`.

`destinationBuffer` now contains the scaled version of `sourceBuffer`. To learn how to display the scaled result to your user, see Creating a Core Graphics Image from a vImage Buffer.

After you've finished with the source and destination buffers, it's important that you free the memory allocated to them:

```
sourceBuffer.free()
destinationBuffer.free()
```

Avoid Artifacts by Unpremultiplying

If you're rescaling an image with premultiplied alpha, you may see artifacts in high-frequency regions of the image. To avoid this situation, unpremultiply the image data — that is, remove the premultiplied alpha value from the image data — before the scaling operation, and premultiply the scaled result.

This code shows the additional operations required, with error handling removed for brevity:

```
error = vImageUnpremultiplyData_ARGB8888(&sourceBuffer,
                                         &sourceBuffer,
                                         vImage_Flags(kvImageNoFlags))

error = vImageScale_ARGB8888(&sourceBuffer,
                             &destinationBuffer,
                             nil,
                             vImage_Flags(kvImageHighQualityResampling))

error = vImagePremultiplyData_ARGB8888(&destinationBuffer,
                                       &destinationBuffer,
                                       vImage_Flags(kvImageNoFlags))
```

See Also

Image Processing Essentials

Converting bitmap data between Core Graphics images and vImage buffers

Pass image data between Core Graphics and vImage to create and manipulate images.

Creating and Populating Buffers from Core Graphics Images

Initialize vImage buffers from Core Graphics images.

Creating a Core Graphics Image from a vImage Buffer

Create displayable representations of vImage buffers.

Applying geometric transforms to images

Reflect, shear, rotate, and scale image buffers using vImage.

Compositing images with alpha blending

Combine two images by using alpha blending to create a single output.

Compositing images with vImage blend modes

Combine two images by using blend modes to create a single output.

Applying vImage operations to regions of interest

Limit the effect of vImage operations to rectangular regions of interest.

Optimizing image-processing performance

Improve your app’s performance by converting image buffer formats from interleaved to planar.

vImage

Manipulate large images using the CPU’s vector processor.