Article

Creating and Populating Buffers from Core Graphics Images

Initialize vImage buffers from Core Graphics images.

Overview

`vImage_Buffer` structures are the basic data structures that vImage uses for working with images. They describe an image's dimensions and contain the pixel data that vImage routines operate on.

Typically, you'll initialize a source buffer from an image and initialize and allocate a destination buffer to receive the result of a vImage operation.

Initialize a Source Buffer from a Core Graphics Image

The vImage functions that initialize a buffer's size and data require an instantiated `vImage_Buffer` structure. Typically, you declare a buffer as a variable because these functions mutate the buffer.

You can initialize a vImage buffer from a `CGImage` instance that's acquired from the `cgImage` property of an image. In the following example, the image is named `Flowers_2.jpg`. The `init(cgImage:format:flags:)` function initializes a `vImage_Buffer` structure with the image data using the format discussed in Converting bitmap data between Core Graphics images and vImage buffers.

```
guard
    let cgImage = UIImage(named: "Flowers_2.jpg")?.cgImage,
    var sourceBuffer = try? vImage_Buffer(cgImage: cgImage,
                                          format: format) else {
                                            return nil
```

```
}
```

Initialize and Allocate a Destination Buffer

Typically, in addition to creating a buffer to represent your source image, you create a destination buffer to receive the result of the vImage operation. In this case, you use the `init(width:height:bitsPerPixel:)` function to initialize a buffer of a specified size and the correct memory allocation for the bit-depth of the image:

```
guard var destinationBuffer = try? vImage_Buffer(width: Int(sourceBuffer.width),
                                                 height: Int(sourceBuffer.height),
                                                 bitsPerPixel: format.bitsPerPixel)
                                                    return nil
}
```

Free the Buffer Memory

After you're finished with buffers that have have their own memory allocation, it's important that you free the memory allocated to them:

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

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 a Core Graphics Image from a vImage Buffer
  Create displayable representations of vImage buffers.
- Building a Basic Image-Processing Workflow
  Resize an image with vImage.
- 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.