Article

Optimizing image-processing performance

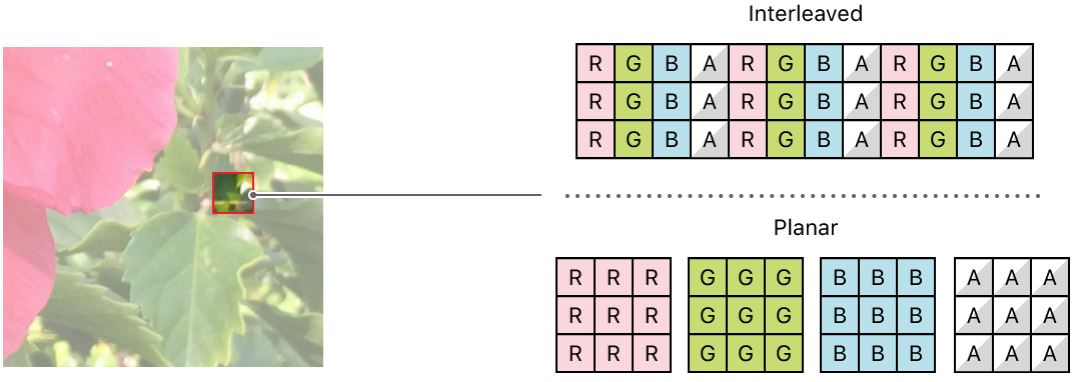
Improve your app's performance by converting image buffer formats from interleaved to planar.

Overview

The vImage library operates on image data with two memory layouts:

Interleaved stores each pixel's color data consecutively in a single buffer. For example, the data that describes a 4-channel image (red, green, blue, and alpha) would be stored as RGBARGBARGBA...

Planar stores each color channel in separate buffers. For example, a 4-channel image would be stored as four individual buffers containing red, green, blue, and alpha data.



Because many vImage functions operate on a single color channel at a time — by converting an interleaved buffer to planar buffers — you can often improve your app's performance by doing this conversion manually. However, most vImage functions are available in both the interleaved and planar variants, so before you do the conversion, try both to see which works better in your context.

In some cases, you may not want to apply a vImage operation to all four channels of an image. For example, you may know beforehand that the alpha channel is irrelevant in the images that you're dealing with, or perhaps all of your images are grayscale and you need to operate on only one channel. Using planar formats makes it possible to isolate and work with only the channels you need.

Review interleaved performance

Typically, your source imagery is in interleaved format, and your default option will be to use the interleaved variant of a vImage function. For example, the following code scales a Core Graphics image to one tenth of its original size. Note that the 4-channel, 8-bit-per-channel interleaved pixel buffer `scale(destination:)` function calls `vImageScale_ARGB8888(_:_:_:_:)`.

```
var cgImageFormat = vImage_CGImageFormat(
    bitsPerComponent: 8,
    bitsPerPixel: 8 * 4,
    colorSpace: CGColorSpaceCreateDeviceRGB(),
    bitmapInfo: CGBitmapInfo(rawValue:
                             CGImageAlphaInfo.noneSkipLast.rawValue))!

let sourceBuffer = try vImage.PixelBuffer(cgImage: sourceImage,
                                          cgImageFormat: &cgImageFormat,
                                          pixelFormat: vImage.Interleaved8x4.self)

let destinationBuffer = vImage.PixelBuffer(size: .init(width: sourceBuffer.width / 1
                                                       height: sourceBuffer.height /
                                           pixelFormat: vImage.Interleaved8x4.self)

let time = ContinuousClock().measure {
    sourceBuffer.scale(destination: destinationBuffer)
}
```

You can use `ContinuousClock` to measure the execution time.

Convert an interleaved source buffer to planar buffers

The pixel buffer `init(cgImage:cgImageFormat:pixelFormat:)` initializer and the vImage buffer `vImageBuffer_InitWithCGImage(_:_:_:_:_:)` function both populate a buffer based on the properties of a `vImage_CGImageFormat` structure.

For example, the following code creates an interleaved 3-channel, 8-bit-per-channel `vImage.PixelBuffer` structure from the source Core Graphics image. The code calls `deinterleave(destination:)` to deinterleave the image data and populate the individual red, green, and blue planar pixel buffers.

```
var cgImageFormat = vImage_CGImageFormat(
    bitsPerComponent: 8,
    bitsPerPixel: 8 * 3,
    colorSpace: CGColorSpaceCreateDeviceRGB(),
    bitmapInfo: CGBitmapInfo(rawValue:
                                CGImageAlphaInfo.none.rawValue))!

let sourceBuffer = try vImage.PixelBuffer(cgImage: sourceImage,
                                          cgImageFormat: &cgImageFormat,
                                          pixelFormat: vImage.Interleaved8x3.self)
let sourceRedBuffer = vImage.PixelBuffer(size: sourceBuffer.size,
                                         pixelFormat: vImage.Planar8.self)
let sourceGreenBuffer = vImage.PixelBuffer(size: sourceBuffer.size,
                                           pixelFormat: vImage.Planar8.self)
let sourceBlueBuffer = vImage.PixelBuffer(size: sourceBuffer.size,
                                          pixelFormat: vImage.Planar8.self)

sourceBuffer.deinterleave(planarDestinationBuffers: [sourceRedBuffer,
                                                     sourceGreenBuffer,
                                                     sourceBlueBuffer])
```

Initialize the destination buffers

Create an interleaved 3-channel, 8-bit-per-channel destination buffer and three planar destination buffers:

```
let destinationBuffer = vImage.PixelBuffer(size: .init(width: sourceBuffer.width / 1
                                                       height: sourceBuffer.height /
                                           pixelFormat: vImage.Interleaved8x3.self)

let destinationRedBuffer = vImage.PixelBuffer(size: destinationBuffer.size,
                                              pixelFormat: vImage.Planar8.self)
let destinationGreenBuffer = vImage.PixelBuffer(size: destinationBuffer.size,
                                                pixelFormat: vImage.Planar8.self)
let destinationBlueBuffer = vImage.PixelBuffer(size: destinationBuffer.size,
                                               pixelFormat: vImage.Planar8.self)
```

Apply the scale operation to the planar buffers

Use the `withTaskGroup(of:returning:body:)` function to start a new scope that contains the three planar scale operations. Note that the 8-bit planar `scale(destination:)` function calls `vImageScale_Planar8(_:_:_:_:)`.

In the code below, the `interleave(destination:)` function interleaves the three planar buffers and populates the interleaved destination buffer with the scaled image:

```
let time = await ContinuousClock().measure {

    await withTaskGroup(of: Void.self) { group in

        group.addTask(priority: .userInitiated) {
            sourceRedBuffer.scale(destination: destinationRedBuffer)
        }

        group.addTask(priority: .userInitiated) {
            sourceGreenBuffer.scale(destination: destinationGreenBuffer)
        }

        group.addTask(priority: .userInitiated) {
            sourceBlueBuffer.scale(destination: destinationBlueBuffer)
        }
    }

    destinationBuffer.interleave(planarSourceBuffers: [destinationRedBuffer,
                                                       destinationGreenBuffer,
                                                       destinationBlueBuffer])
}
```

The following code calls `makeCGImage(cgImageFormat:)` to create a Core Graphics image from the result of the scale operation:

```
let scaledImage = destinationBuffer.makeCGImage(cgImageFormat: cgImageFormat)
```

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.
- 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.

vImage

Manipulate large images using the CPU's vector processor.