Documentation

Accelerate / Integrating vImage pixel buffers into a Core Image workflow

Sample Code

Integrating vImage pixel buffers into a Core Image workflow

Share image data between Core Video pixel buffers and vImage buffers to integrate vImage operations into a Core Image workflow.

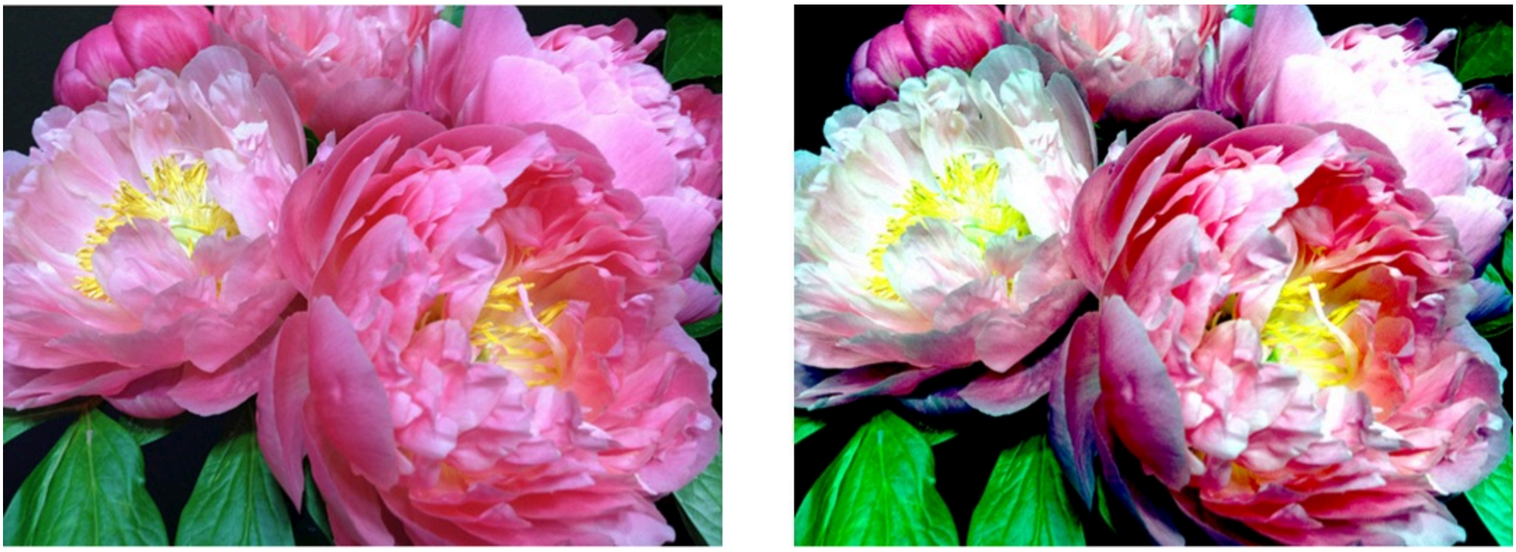
Download

macOS 13.0+ | Xcode 14.3+

Overview

vImage supports reading from and writing to Core Video pixel buffers. This sample implements ends-in contrast stretching using vImage and makes that operation available to Core Image workflows by subclassing `CIImageProcessorKernel`. An image processor kernel uses Core Video pixel buffers for input and output, so the app creates vImage pixel buffers that share data with `CVPixelBuffer` instances.

The example below shows a photograph before (left) and after (right) the app has applied ends-in contrast stretching:



To learn more about ends-in contrast stretching, see Enhancing image contrast with histogram manipulation.

Before exploring the code, try building and running the app to familiarize yourself with the effect of the different parameters on the image.

Define an ends-in contrast-stretch image processor kernel

The `ContrastStretchImageProcessorKernel` inherits from the Core Image `CIImageProcessorKernel` class.

The sample code defines a `vImage_CGImageFormat` structure that represents a four-channel, 8-bit-per-channel interleaved image format. The image processor kernel supports `kCIFormatR8`, `kCIFormatBGRA8`, `kCIFormatRGBAh`, and `kCIFormatRGBAf` input and output formats. For this sample project, the code overrides `outputFormat` and `formatForInput(at:)` to return a BGRA8 that's the same as the `bitmapInfo` property of the `vImage_CGImageFormat` structure.

```
static var cgImageFormat = vImage_CGImageFormat(
    bitsPerComponent: 8,
    bitsPerPixel: 32,
    colorSpace: nil,
    bitmapInfo: CGBitmapInfo(rawValue: CGImageAlphaInfo.last.rawValue),
    version: 0,
    decode: nil,
    renderingIntent: .defaultIntent)

override class var outputFormat: CIFormat {
    return CIFormat.BGRA8
}

override class func formatForInput(at input: Int32) -> CIFormat {
    return CIFormat.BGRA8
}
```

Create the source pixel buffer

When the app applies ends-in contrast stretching, Core Image calls the processor kernel's `process(with:arguments:output:)` function. The following code ensures that the input and output `CVPixelBuffer` instances are available:

```
guard
    let input = inputs?.first,
    let inputPixelBuffer = input.pixelBuffer,
    let outputPixelBuffer = output.pixelBuffer else {
        return
}
```

The source `vImage.PixelBuffer` shares its memory with the input `CVPixelBuffer`. The following code creates a `vImageConverter` that allows the pixel buffer to reference the Core Video buffer's memory:

```
CVPixelBufferLockBaseAddress(inputPixelBuffer,
                             CVPixelBufferLockFlags.readOnly)
defer {
    CVPixelBufferUnlockBaseAddress(inputPixelBuffer,
                                   CVPixelBufferLockFlags.readOnly)
}

guard let cvImageFormat = vImageCVImageFormat.make(buffer: inputPixelBuffer) else {
    throw ContrastStretchImageProcessorKernelError.unableToDeriveImageFormat
}

if cvImageFormat.colorSpace == nil {
    cvImageFormat.colorSpace = CGColorSpaceCreateDeviceRGB()
}

guard let converter = try? vImageConverter.make(
    sourceFormat: cvImageFormat,
    destinationFormat: cgImageFormat) else {
    throw ContrastStretchImageProcessorKernelError.vImageConverterCreationFailed
}

let sourcePixelBuffer = vImage.PixelBuffer<vImage.Interleaved8x4>(
    referencing: inputPixelBuffer,
    converter: converter)
```

Create the destination pixel buffer

The sample code app uses the same `vImageConverter` to create the destination pixel buffer, which shares memory with the output Core Video buffer's memory.

```
CVPixelBufferLockBaseAddress(outputPixelBuffer,
                             CVPixelBufferLockFlags.readOnly)
defer {
    CVPixelBufferUnlockBaseAddress(outputPixelBuffer,
                                   CVPixelBufferLockFlags.readOnly)
}

let destinationPixelBuffer = vImage.PixelBuffer<vImage.Interleaved8x4>(
    referencing: outputPixelBuffer,
    converter: converter)
```

Apply ends-in contrast stretching

The `vImageEndsInContrastStretch_ARGB8888(_:_:_:_:_:)` function applies an ends-in contrast-stretch operation to the source pixel buffer and writes the result to the destination pixel buffer. This function works equally well on all channel orderings; for example, RGBA or BGRA.

```
let error = sourcePixelBuffer.withUnsafePointerToVImageBuffer { src in
    destinationPixelBuffer.withUnsafePointerToVImageBuffer { dst in
        
        return vImageEndsInContrastStretch_ARGB8888(
            src,
            dst,
            [UInt32](repeating: UInt32(percentLow), count: 4),
            [UInt32](repeating: UInt32(percentHigh), count: 4),
            vImage_Flags(kvImageNoFlags))
        
    }
}
```

Because the destination pixel buffer shares memory with the output Core Video pixel buffer, the operation is complete after the `vImageEndsInContrastStretch_ARGB8888(_:_:_:_:_:)` returns.

Apply the ends-in contrast stretching operation to an image

The `apply(withExtent:inputs:arguments:)` method generates a `CIImage` instance based on the output of the processor's `process(with:arguments:output:)` function.

```
let ciResult = try? ContrastStretchImageProcessorKernel.apply(
    withExtent: ciImage.extent,
    inputs: [ciImage],
    arguments: ["percentLow": Int(percentLow),
                "percentHigh": Int(percentHigh)])
```

See Also

Core Video Interoperation

Using vImage pixel buffers to generate video effects

Render real-time video effects with the vImage Pixel Buffer.

Applying vImage operations to video sample buffers

Use the vImage convert-any-to-any functionality to perform real-time image processing of video frames streamed from your device’s camera.

Improving the quality of quantized images with dithering

Apply dithering to simulate colors that are unavailable in reduced bit depths.

Core Video interoperability

Pass image data between Core Video and vImage.