API Collection

Core Video interoperability

Pass image data between Core Video and vImage.

Overview

The vImage library provides two approaches for working with Core Video pixel buffers:

- Use the `vImageBuffer_InitWithCVPixelBuffer(_:_:_:_:_:_:)` and `vImageBuffer_CopyToCVPixelBuffer(_:_:_:_:_:)` to copy and convert data between vImage buffers and Core Video pixel buffers with a single function call. This approach provides a simple API if you need to convert between image formats.
- Use the `vImageBuffer_InitForCopyFromCVPixelBuffer(_:_:_:_:)` and `vImageBuffer_InitForCopyToCVPixelBuffer(_:_:_:_:)` functions to create vImage buffers that reference the data in Core Video pixel buffers. This approach allows you to work directly with the underlying data if you don't need to convert between image formats.

Copying data between the vImage library and Core Video

Use the `vImageBuffer_InitWithCVPixelBuffer(_:_:_:_:_:_:)` and `vImageBuffer_CopyToCVPixelBuffer(_:_:_:_:_:)` functions to copy and convert data between vImage and Core Video.

The `vImageBuffer_InitWithCVPixelBuffer(_:_:_:_:_:_:)` function allocates new memory and, after you finish with the buffer, call `free()` to avoid memory leaks.

The following code shows an example of a `CIImageProcessorKernel` that reflects an image vertically. The example calls `vImageBuffer_InitWithCVPixelBuffer(_:_:_:_:_:_:)` to initialize the source vImage buffer with a copy of the input CVPixelBuffer instance's data. The code calls `vImageBuffer_CopyToCVPixelBuffer(_:_:_:_:_:)` to copy the destination vImage buffer's contents to the output CVPixelBuffer instance.

The code uses a defer statement to deallocate the source and destination vImage buffers after the image-processing operation completes.

```
class VerticalReflectImageProcessorKernelCopy: CIImageProcessorKernel {

    static var cgImageFormat = vImage_CGImageFormat(
        bitsPerComponent: 8,
        bitsPerPixel: 32,
        colorSpace: CGColorSpaceCreateDeviceRGB(),
        bitmapInfo: CGBitmapInfo(rawValue: CGImageAlphaInfo.noneSkipLast.rawValue),
        renderingIntent: .defaultIntent)!

    static let cvImageFormat = vImageCVImageFormat.make(
        format: .format32BGRA,
        colorSpace: CGColorSpaceCreateDeviceRGB(),
        alphaIsOpaqueHint: true)!

    override class var outputFormat: CIFormat {
        CIFormat.BGRA8
    }

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

    override class func process(with inputs: [CIImageProcessorInput]?,
                                arguments: [String: Any]?,
                                output: CIImageProcessorOutput) throws {

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

        var sourceBuffer = vImage_Buffer()
        vImageBuffer_InitWithCVPixelBuffer(&sourceBuffer,
                                           &cgImageFormat,
                                           inputPixelBuffer,
                                           cvImageFormat,
                                           nil,
```

```
                                        vImage_Flags(kvImageNoFlags))

        var destinationBuffer = vImage_Buffer()
        vImageBuffer_Init(&destinationBuffer,
                          sourceBuffer.height,
                          sourceBuffer.width,
                          cgImageFormat.bitsPerPixel,
                          vImage_Flags(kvImageNoFlags))

        defer {
            sourceBuffer.free()
            destinationBuffer.free()
        }

        vImageVerticalReflect_ARGB8888(&sourceBuffer,
                                       &destinationBuffer,
                                       vImage_Flags(kvImageNoFlags))

        vImageBuffer_CopyToCVPixelBuffer(&destinationBuffer,
                                         &cgImageFormat,
                                         outputPixelBuffer,
                                         cvImageFormat,
                                         nil,
                                         vImage_Flags(kvImageNoFlags))
    }
}
```

Sharing data between the vImage library and Core Video

Use the `vImageBuffer_InitForCopyFromCVPixelBuffer(_:_:_:_:)` and `vImageBuffer_InitForCopyToCVPixelBuffer(_:_:_:_:)` functions to share data between vImage and Core Video. Both of these functions require a `vImageConverter` instance that defines the vImage buffer's Core Graphics image format and the `CVPixelBuffer` instance's Core Video format.

Because the vImage functions don't allocate any additional memory, you don't need to deallocate the vImage buffer memory. However, you need to lock and unlock the `CVPixelBuffer` instances during the image-processing operation using `CVPixelBufferLockBaseAddress(_:_:)` and `CVPixelBufferUnlockBaseAddress(_:_:)`, respectively.

The following code shows an example of a `CIImageProcessorKernel` that reflects an image vertically. In this example, the base address of the `CVPixelBuffer` instances and the `data` property of their corresponding vImage buffer point to the same memory. The image data in the

CIImageProcessorInput and CIImageProcessorOutput parameters don't require conversion, and the code works directly on the pixel buffers.

The code calls vImageBuffer_InitForCopyFromCVPixelBuffer(_:_:_:_:) with a Core-Video-to-Core-Graphics converter to initialize the source vImage buffer.

The code calls vImageBuffer_InitForCopyToCVPixelBuffer(_:_:_:_:) with a Core-Graphics-to-Core-Video converter to initialize the destination vImage buffer.

```
class VerticalReflectImageProcessorKernelNoCopy: CIImageProcessorKernel {

    static var cgImageFormat = vImage_CGImageFormat(
        bitsPerComponent: 8,
        bitsPerPixel: 32,
        colorSpace: CGColorSpaceCreateDeviceRGB(),
        bitmapInfo: CGBitmapInfo(rawValue: CGImageAlphaInfo.noneSkipLast.rawValue),
        renderingIntent: .defaultIntent)!

    static let cvImageFormat = vImageCVImageFormat.make(
        format: .format32BGRA,
        colorSpace: CGColorSpaceCreateDeviceRGB(),
        alphaIsOpaqueHint: true)!

    static let converterCVtoCG = try! vImageConverter
        .make(sourceFormat: cvImageFormat,
              destinationFormat: cgImageFormat)

    static let converterCGtoCV = try! vImageConverter
        .make(sourceFormat: cgImageFormat,
              destinationFormat: cvImageFormat)

    override class var outputFormat: CIFormat {
        CIFormat.BGRA8
    }

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

    override class func process(with inputs: [CIImageProcessorInput]?,
                                arguments: [String: Any]?,
                                output: CIImageProcessorOutput) throws {
```

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

        CVPixelBufferLockBaseAddress(inputPixelBuffer,
                                     CVPixelBufferLockFlags.readOnly)
        CVPixelBufferLockBaseAddress(outputPixelBuffer,
                                     CVPixelBufferLockFlags(rawValue: 0))

        defer {
            CVPixelBufferUnlockBaseAddress(inputPixelBuffer,
                                           CVPixelBufferLockFlags.readOnly)
            CVPixelBufferUnlockBaseAddress(outputPixelBuffer,
                                           CVPixelBufferLockFlags(rawValue: 0))
        }

        var sourceBuffer = vImage_Buffer()
        vImageBuffer_InitForCopyFromCVPixelBuffer(
            &sourceBuffer,
            converterCVtoCG,
            inputPixelBuffer,
            vImage_Flags(kvImageNoAllocate))

        var destinationBuffer = vImage_Buffer()
        vImageBuffer_InitForCopyToCVPixelBuffer(
            &destinationBuffer,
            converterCGtoCV,
            outputPixelBuffer,
            vImage_Flags(kvImageNoAllocate))

        // `CVPixelBufferGetBaseAddress(inputPixelBuffer) == sourceBuffer.data`.
        // `CVPixelBufferGetBaseAddress(outputPixelBuffer) == destinationBuffer.data

        vImageVerticalReflect_ARGB8888(&sourceBuffer,
                                       &destinationBuffer,
                                       vImage_Flags(kvImageNoFlags))
    }
}
```

Topics

Copying Core Video pixel buffer data to vImage buffers

`func vImageBuffer_InitWithCVPixelBuffer(UnsafeMutablePointer<vImage_Buffer>, UnsafeMutablePointer<vImage_CGImageFormat>, CVPixelBuffer, vImageCVImageFormat!, UnsafePointer<CGFloat>!, vImage_Flags) -> vImage_Error`

Initializes a vImage buffer with a copy of the contents of a Core Video pixel buffer.

Copying and converting data between vImage buffers and Core Video pixel buffers

`func vImageBuffer_CopyToCVPixelBuffer(UnsafePointer<vImage_Buffer>, UnsafePointer<vImage_CGImageFormat>, CVPixelBuffer, vImageCVImageFormat!, UnsafePointer<CGFloat>!, vImage_Flags) -> vImage_Error`

Copies the contents of a vImage buffer to a Core Video pixel buffer.

Initializing vImage buffers that reference Core Video pixel buffer data

`func vImageBuffer_InitForCopyFromCVPixelBuffer(UnsafeMutablePointer<vImage_Buffer>, vImageConverter, CVPixelBuffer, vImage_Flags) -> vImage_Error`

Initializes an array of vImage buffers in the order necessary to copy from a Core Video pixel buffer.

`func vImageBuffer_InitForCopyToCVPixelBuffer(UnsafeMutablePointer<vImage_Buffer>, vImageConverter, CVPixelBuffer, vImage_Flags) -> vImage_Error`

Initializes an array of vImage buffers in the order necessary to copy to a Core Video pixel buffer.

Managing Core Video image formats

Core Video image format utilities

Create, copy, and query Core Video image format descriptions.

See Also

Core Video Interoperation

{} Using vImage pixel buffers to generate video effects

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

{} 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.

{} 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.