Sample Code

# Improving the quality of quantized images with dithering

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

Download

macOS 13.3+ | Xcode 14.3+

## Overview

When you convert images to lower bit depths, some colors may be unavailable in the destination bit depth. As a solution, the vImage library provides options to apply dithering, a process that uses a pattern of random pixels to simulate unavailable colors. For example, a mid-gray color from an 8-bit grayscale image that's quantized to 1 bit returns data that contains 50% white pixels and 50% black pixels.

This sample code app converts an 8-bit grayscale image to a 1-bit dithered image and provides a user interface to select between different dithering types.

The example below shows an image with continuous tones (left) and the same image with dithering applied (right):

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

## Define the source and destination Core Graphics image formats

The sample code defines two `vImage_CGImageFormat` structures that represent the source and destination image formats. The `sourceFormat` structure is an 8-bit grayscale format that supports 256 levels of gray. The `destinationFormat` structure is a 1-bit format with pixels that are either black or white.

```
let sourceFormat = vImage_CGImageFormat(
    bitsPerComponent: 8,
    bitsPerPixel: 8,
    colorSpace: CGColorSpaceCreateDeviceGray(),
    bitmapInfo: .init(rawValue: CGImageAlphaInfo.none.rawValue))!

let destinationFormat = vImage_CGImageFormat(
    bitsPerComponent: 1,
    bitsPerPixel: 1,
    colorSpace: CGColorSpaceCreateDeviceGray(),
    bitmapInfo: .init(rawValue: CGImageAlphaInfo.none.rawValue))!
```

## Allocate the source and destination image buffers

The code populates the contents of the source `vImage_Buffer` structure with a grayscale version of the source image. Because the code passes a populated `vImage_CGImageFormat` structure to the `init(cgImage:format:flags:)` initializer, vImage converts the source image to an 8-bit grayscale format.

The call to `init(size:bitsPerPixel:)` creates the destination buffer, which is the same size as the source buffer but with only 1 bit per pixel.

```
sourceBuffer = try vImage_Buffer(
    cgImage: sourceImage,
    format: sourceFormat)

destinationBuffer = try vImage_Buffer(
    size: sourceBuffer.size,
    bitsPerPixel: destinationFormat.bitsPerPixel)
```

# Create a dither-type enumeration

To support dither-type selection in the user interface, the sample code includes an enumeration that wraps the available vImage dithering algorithms.

```
enum DitheringType: String, CaseIterable {
    case none = "None"
    case orderedGaussian = "Ordered Gaussian"
    case orderedUniform = "Ordered Uniform"
    case floydSteinberg = "Floyd Steinberg"
    case atkinson = "Atkinson"

    var dither: Int32 {
        switch self {
            case .none:
                return Int32(kvImageConvert_DitherNone)
            case .orderedGaussian:
                return Int32(kvImageConvert_DitherOrdered | kvImageConvert_OrderedGa
            case .orderedUniform:
                return Int32(kvImageConvert_DitherOrdered | kvImageConvert_OrderedUr
            case .floydSteinberg:
                return Int32(kvImageConvert_DitherFloydSteinberg)
            case .atkinson:
                return Int32(kvImageConvert_DitherAtkinson)
        }
    }
}
```

The sample code app supports the following dithering types:

- `kvImageConvert_DitherNone`: Doesn't apply any dithering. This algorithm rounds the input values to the nearest representable value in the destination format.
- `kvImageConvert_DitherOrdered`: Adds precomputed blue noise to the source image before it rounds the input values to the nearest representable value in the destination format. The vImage conversion functions support uniform and Gaussian noise by including `kvImageConvert_OrderedUniformBlue` and `kvImageConvert_OrderedGaussianBlue`, respectively.
- `kvImageConvert_DitherFloydSteinberg`: Applies Floyd-Steinberg dithering to the image.
- `kvImageConvert_DitherAtkinson` Applies Atkinson dithering to the image.

The vImage library also includes `kvImageConvert_DitherOrderedReproducible`, which returns the same result as `kvImageConvert_DitherOrdered` but uses the same offset into the blue noise for each call.

## Apply dithering to the image

The `vImageConvert_Planar8toPlanar1(_:_:_:_:)` function converts the 8-bit grayscale to a 1-bit image using the dithering type that the user interface defines.

```
withUnsafePointer(to: sourceBuffer) { src in
    withUnsafePointer(to: destinationBuffer) { dest in
        _ = vImageConvert_Planar8toPlanar1(
            src, dest,
            nil,
            ditheringType.dither,
            vImage_Flags(kvImageNoFlags))
    }
}
```

On return, the destination buffer contains the 1-bit dithered version of the source image.

The vImage library provides dithering options for many conversion functions, such as `vImageConvert_ARGBFFFFtoARGB8888_dithered(_:_:_:_:_:_:)`, which converts a 32-bit-per-pixel ARGB image to an 8-bit-per-pixel ARGB image. Refer to Conversion for more details.

# 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.

Core Video interoperability

Pass image data between Core Video and vImage.