Sample Code

Halftone descreening with 2D fast Fourier transform

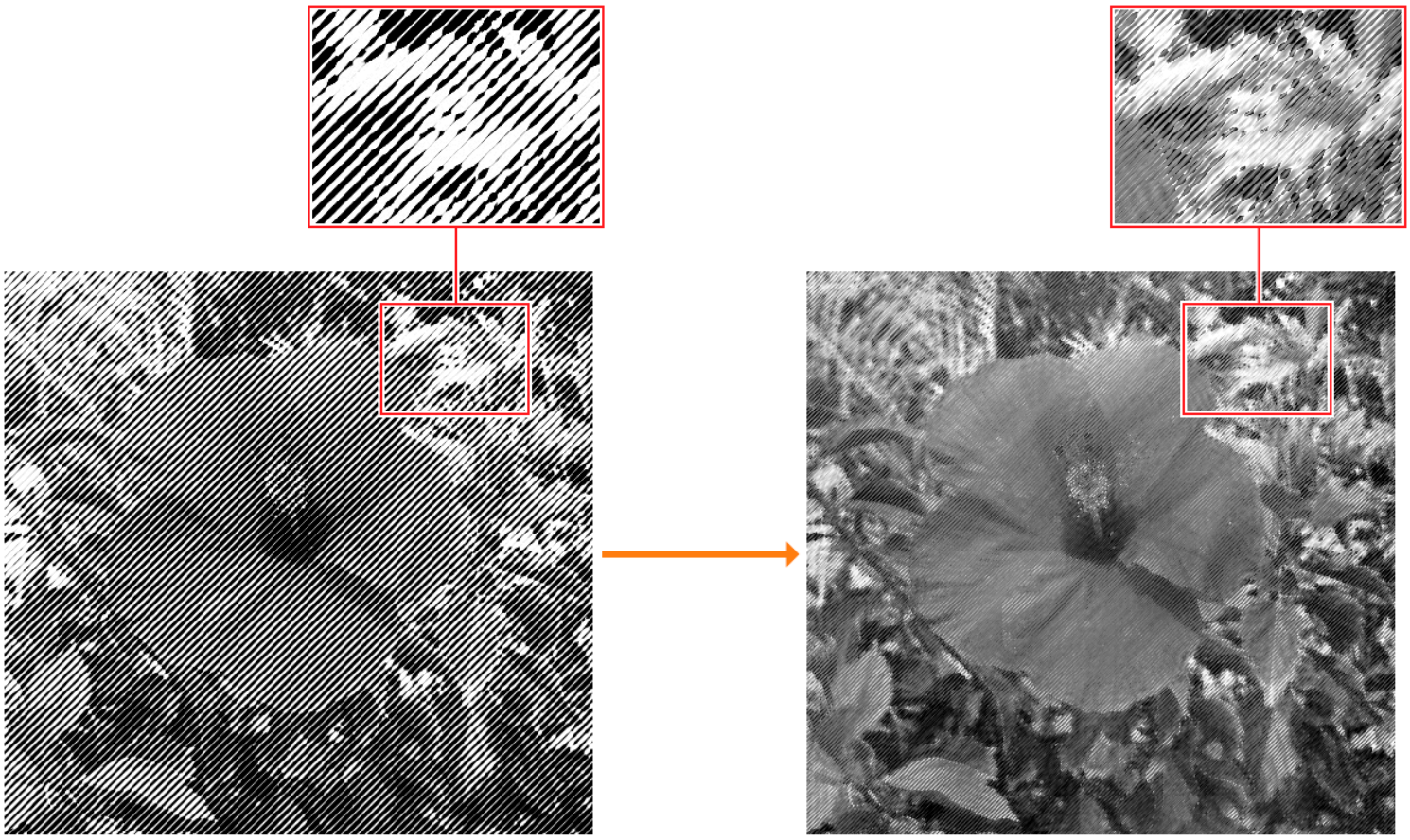
Reduce or remove periodic artifacts from images.

Download

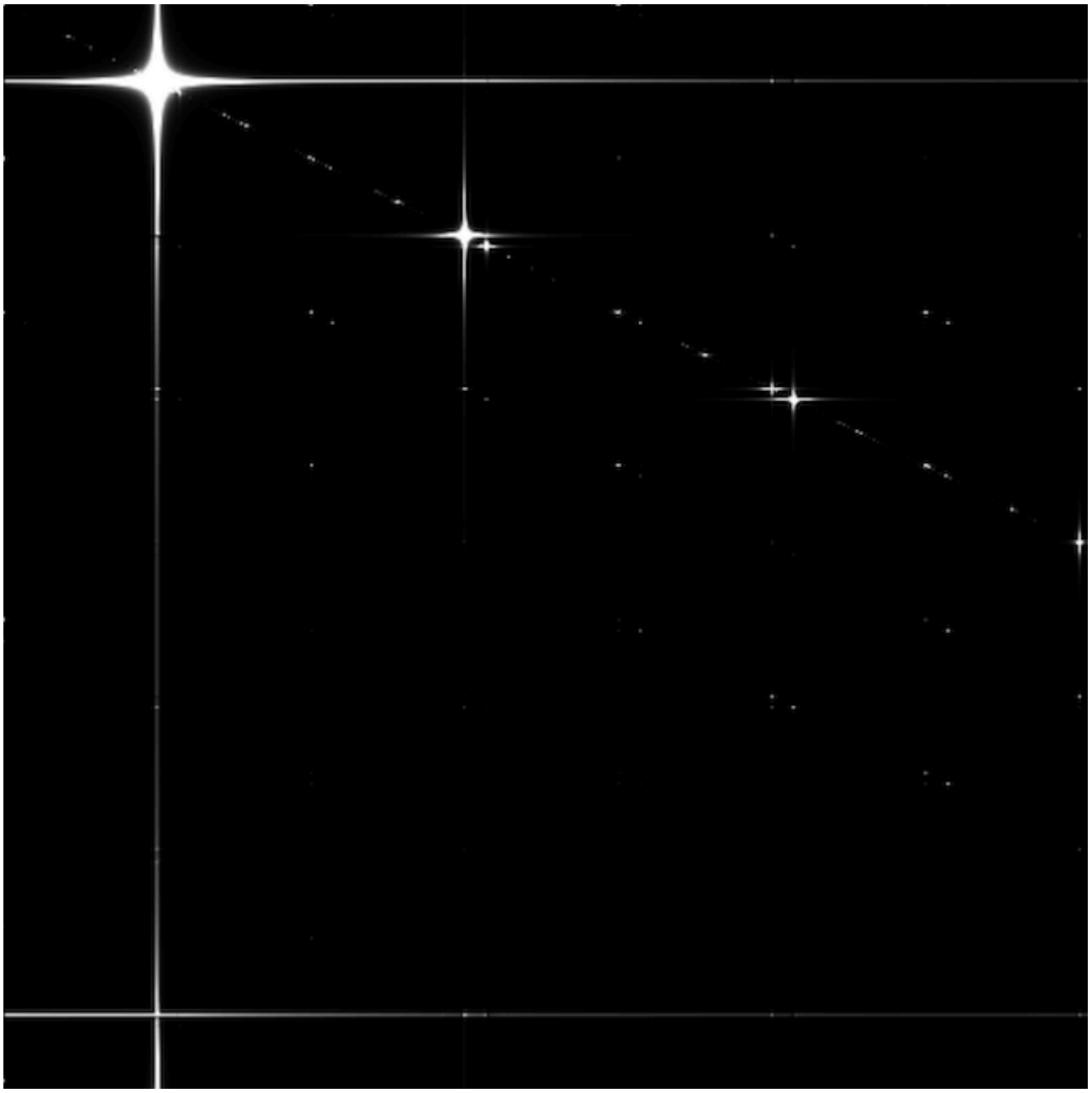
macOS 13.0+ | Xcode 14.0+

Overview

Accelerate's vDSP module provides functions to perform 2D fast Fourier transforms (FFTs) on matrices of data, such as images. You can exploit the amplitude peaks in the frequency domain of periodic patterns, such as halftone screens, to reduce or remove such artifacts from images. The example below shows an image with halftone artifacts (left) and the same image with the halftone artifacts reduced (right):



The sample app reduces halftone artifacts from a source image by first performing forward FFTs on the image and a halftone screen sample. The following image shows the positive frequencies of the halftone sample's frequency-domain representation. For more information on performing Fourier transforms on 2D data, see Understanding data packing for Fourier transforms.



The descreening operation zeroes frequency-domain values in the source image that correspond to the bright peaks in the halftone sample's frequency-domain values.

Convert the image data to a split-complex vector

The vImage `vImageBuffer_InitWithCGImage(_:_:_:_:)` function converts the source Core Graphics image to an array of single-precision values. The sample app creates its own backing storage, `pixelsStorage`, rather than using `vImageBuffer_Init(_:_:_:_:)` to ensure that the vImage buffer doesn't contain any additional padding bytes.

```
let pixelsStorage = UnsafeMutableBufferPointer<Float>.allocate(capacity: pixelCount)
defer {
    pixelsStorage.deallocate()
}
```

```
var tmpBuffer = vImage_Buffer(
    data: pixelsStorage.baseAddress,
    height: vImagePixelCount(cgImage.height),
    width: vImagePixelCount(cgImage.width),
    rowBytes: cgImage.width * MemoryLayout<Float>.stride)

vImageBuffer_InitWithCGImage(
    &tmpBuffer,
    &Self.imageFormat,
    [0, 0, 0, 0],
    cgImage,
    vImage_Flags(kvImageNoAllocate))
```

The vDSP_ctoz function converts the interleaved pixel values to split-complex format. The function copies odd-numbered pixels to the real parts and the even-numbered pixels to the imaginary parts of each complex value.

```
pixelsStorage.withMemoryRebound(to: DSPComplex.self) {

    vDSP_ctoz([DSPComplex]($0), 2,
              &self.dspSplitComplex, 1,
              vDSP_Length(complexValuesCount))
}
```

Create the FFT setup object

The sample app creates an FFT setup object that contains all the information required to perform the forward and inverse 2D FFT operations. Creating this setup object can be expensive, so the sample only performs this initialization once.

The following code creates a setup object suitable for performing forward and inverse 2D FFTs on a 1024 x 1024 pixel image:

```
static let fftSetUp = vDSP.FFT2D(width: imageWidth,
                                  height: imageHeight,
                                  ofType: DSPSplitComplex.self)!
```

Prepare arrays for transformed image data

Rather than allocating and deallocating memory with each call to the descreening code, the sample app declares several `SplitComplex` structures and an array that the descreening

operation uses.

```
/// The `SplitComplex` structure that stores the source image frequency-domain pixel
var imageFrequencyDomainPixels = SplitComplex(count: HalftoneDescreener.complexValue

/// The `SplitComplex` structure that stores the halftone sample frequency-domain pi
var halftoneFrequencyDomainPixels = SplitComplex(count: HalftoneDescreener.complexVa

/// The array that stores the square magnitudes of the halftone frequency-domain val
var halftoneSampleAmplitudes = [Float](repeating: 0,
                                       count: HalftoneDescreener.complexValuesCount)
```

Perform forward 2D FFTs on the image data

The `transform(input:output:direction:)` function performs a forward 2D FFT on the image data, and creates the frequency-domain representation of the image.

The following code performs the FFT on the source image and the halftone sample data. After the code completes the forward FFT of the halftone sample, the `squareMagnitudes(_:result:)` function computes the magnitudes of the complex values representing the halftone sample:

```
fftSetUp.transform(input: imagePixels.dspSplitComplex,
                   output: &imageFrequencyDomainPixels.dspSplitComplex,
                   direction: .forward)

fftSetUp.transform(input: halftonePixels.dspSplitComplex,
                   output: &halftoneFrequencyDomainPixels.dspSplitComplex,
                   direction: .forward)

vDSP.squareMagnitudes(halftoneFrequencyDomainPixels.dspSplitComplex,
                      result: &halftoneSampleAmplitudes)
```

Zero the peaks in the halftone sample magnitude

The sample app reduces the halftone screen artifacts by manipulating the magnitude of the frequency-domain data for the halftone sample.

The `threshold(_:to:with:)` function sets all magnitude values that are over the threshold to -1, and all magnitude values that are less than or equal to the threshold to 1:

```
let outputConstant: Float = -1

vDSP.threshold(halftoneSampleAmplitudes,
               to: threshold,
               with: .signedConstant(outputConstant),
               result: &halftoneSampleAmplitudes)
```

The `clip(_:to:result:)` function clips the values in the `halftoneSampleAmplitude`, setting all of the high-magnitude values to `0.0`, and all of the low-magnitude values to `1.0`.

```
vDSP.clip(halftoneSampleAmplitudes,
          to: 0 ... 1,
          result: &halftoneSampleAmplitudes)
```

Descreen the source image

The app multiplies the frequency-domain data of the source image by the values in `halftoneSampleAmplitude`, thus removing or reducing the halftone screen.

```
vDSP.multiply(imageFrequencyDomainPixels.dspSplitComplex,
              by: halftoneSampleAmplitudes,
              result: &imageFrequencyDomainPixels.dspSplitComplex)
```

Perform an inverse 2D FFT on the frequency domain data

The `transform(input:output:direction:)` function performs an inverse FFT on the frequency-domain image data to generate the descreened spatial-domain image.

```
fftSetUp.transform(input: imageFrequencyDomainPixels.dspSplitComplex,
                   output: &destinationSpatialDomainPixels.dspSplitComplex,
                   direction: .inverse)
```

Generate an image from the split-complex vector

Finally, the app creates a displayable image from the spatial-domain representation of the treated source image.

```
var floatPixels = [Float](fromSplitComplex: self.dspSplitComplex,
                          scale: 1 / Float(count),
                          count: count * 2)

return floatPixels.withUnsafeMutableBytes {
    let tmpBuffer = vImage_Buffer(
        data: $0.baseAddress,
        height: vImagePixelCount(HalftoneDescreener.imageHeight),
        width: vImagePixelCount(HalftoneDescreener.imageWidth),
        rowBytes: HalftoneDescreener.imageWidth * MemoryLayout<Float>.stride)

    return try! tmpBuffer.createCGImage(format: SplitComplex.imageFormat)
}
```

See Also

Fourier and Cosine Transforms

- Understanding data packing for Fourier transforms

  Format source data for the vDSP Fourier functions, and interpret the results.

- Finding the component frequencies in a composite sine wave

  Use 1D fast Fourier transform to compute the frequency components of a signal.

- Performing Fourier transforms on interleaved-complex data

  Optimize discrete Fourier transform (DFT) performance with the vDSP interleaved DFT routines.

- Reducing spectral leakage with windowing

  Multiply signal data by window sequence values when performing transforms with noninteger period signals.

- Signal extraction from noise

  Use Accelerate's discrete cosine transform to remove noise from a signal.

- Performing Fourier Transforms on Multiple Signals

  Use Accelerate's multiple-signal fast Fourier transform (FFT) functions to transform multiple signals with a single function call.

- Fast Fourier transforms

Transform vectors and matrices of temporal and spatial domain complex values to the frequency domain, and vice versa.

- Discrete Fourier transforms

  Transform vectors of temporal and spatial domain complex values to the frequency domain, and vice versa.

- Discrete Cosine transforms

  Transform vectors of temporal and spatial domain real values to the frequency domain, and vice versa.