Sample Code

Applying tone curve adjustments to images

Use the vImage library's polynomial transform to apply tone curve adjustments to images.

Download

macOS 13.3+ | Xcode 15.0+

Overview

The vImage library provides a suite of functions for applying polynomials to images. The results of these functions are similar to the Curves adjustment tool in the Mac Photos app. You can use the polynomial adjustment functions to adjust the overall look of a photo, change the look of specific colors (red, green, and blue), and adjust settings for the black point, midtones, and white point.

This sample code project calculates the polynomial coefficients from a set of values that the user defines using handles in the user interface. The vImage polynomial functions evaluate the polynomial coefficients to define the tone curve. The technique that calculates the coefficients is the Vandermonde method. To learn more about this technique, see Finding an interpolating polynomial using the Vandermonde method.

To generate the smooth curves in the user interface, the app passes the same coefficients that the vImage polynomial transform function uses to the vDSP `evaluatePolynomial(using Coefficients:withVariables:)` function.

The following image shows the sample code project's app. The circles on the curves are the handles that the user can drag vertically, and the image changes to show the effect of the polynomial transform.

Convert the interleaved source image to planar buffers

The sample code project accepts source images that it converts to RGB, 32-bit per channel format. Because the vImage polynomial transform functions work on planar buffers, the code creates a `vImage.PlanarFx3` multiple-plane `vImage.PixelBuffer` structure that contains the separate red, green, and blue channels.

After creating `vImage.PixelBuffer` structures that store the interleaved and planar representations of the source image, the `populatePlanarSourceBuffers()` function copies and deinterleaves the interleaved image to the planar buffers.

```
func populatePlanarSourceBuffers() {
    srcInterleavedBuffer.deinterleave(destination: srcPlanarBuffers)
}
```

To learn more about working with planar buffers in vImage, see Optimizing image-processing performance.

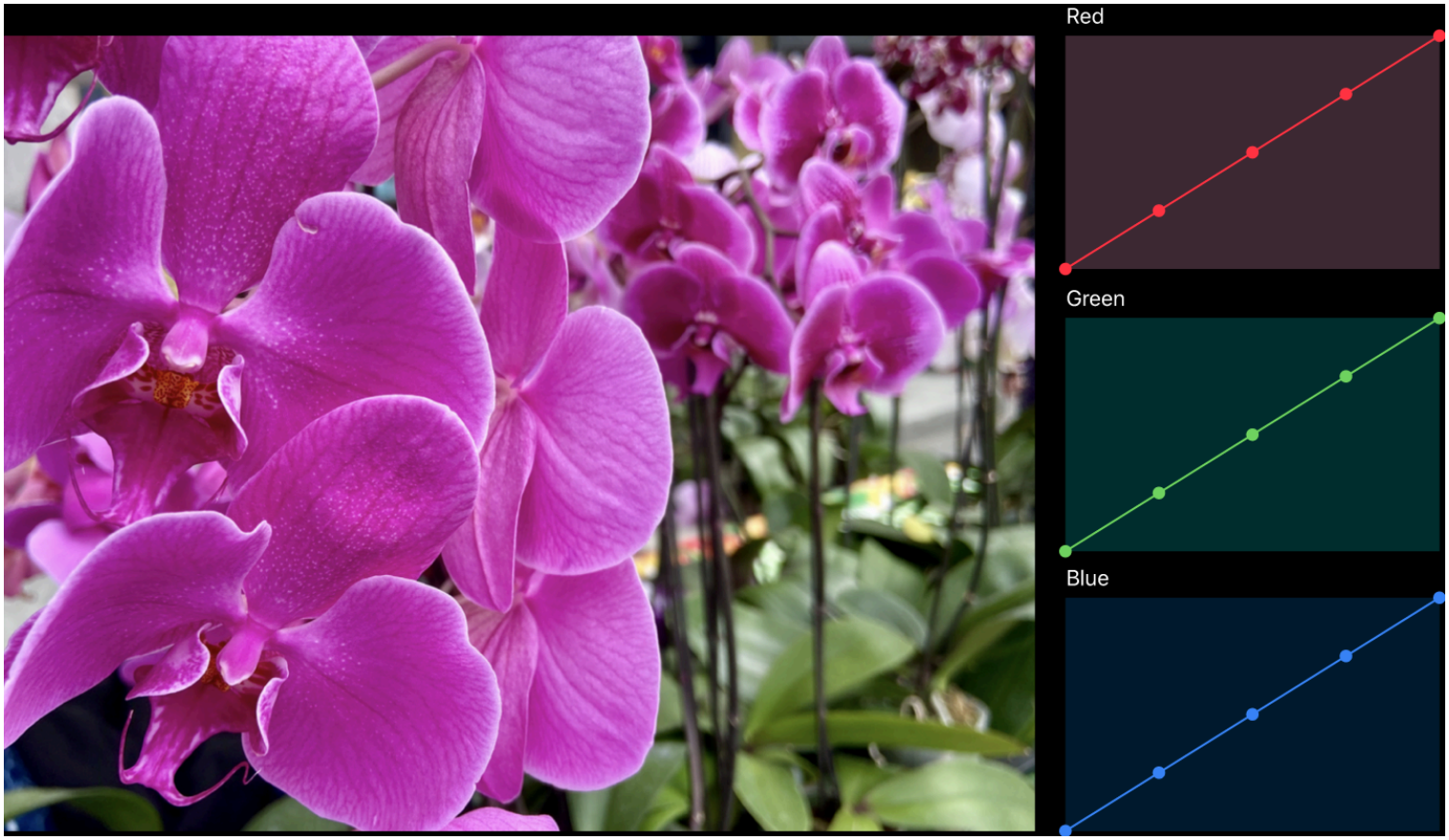
Create the default curve control points

The sample code project defines five data points for each color channel that control the tone curve. By default, these form a linear ramp from `0` to `1`. The code calls `ramp(withInitialValue:increment:count:)` to populate the `redHandleValues`, `greenHandleValues`, and

`blueHandleValues` arrays. The following shows the code that populates the `greenHandleValues` array:

```
greenHandleValues = vDSP.ramp(
    in: 0 ... 1,
    count: PolynomialTransformer.count)
```

When the sample code app first launches, the default values form a linear tone curve that renders as a stright line in the user interface.



The default control points create an output image that's identical to the input image. That is, for each pixel, the transformed output value is equal to the input value.

Apply the polynomial transform

When the user adjusts the control points in the user interface, a `didSet` property observer applies the polynomial to the corresponding planar buffer. For example, the following shows the code for the green values:

```
@Published var greenHandleValues: [Double]! {
    didSet {
        greenCoefficients = calculateAndApplyPolynomial(
```

```
            forHandleValues: greenHandleValues,
            at: 1,
            source: srcPlanarBuffers,
            destination: destPlanarBuffers)

        displayPlanarDestinationBuffers()
    }
}
```

The `calculateAndApplyPolynomial()` function calls `calculateCoefficients()` to calculate the coefficients using the Vandermonde method. It then passes the coefficients to `applyPolynomial(coefficientSegments:boundaries:destination:)`. The vImage polynomial function effectively creates a polynomial curve from the specified coefficients and uses that as the tone curve. For each point on the curve, the horizontal position represents the input value, and the vertical position represents the output value.

```
func calculateAndApplyPolynomial(
    forHandleValues values: [Double],
    at planeIndex: Int,
    source: vImage.PixelBuffer<vImage.PlanarFx4>,
    destination: vImage.PixelBuffer<vImage.PlanarFx4>) -> [Float] {

        let coefficients = calculateCoefficients(values: values.map { Float($0) })

        source.withUnsafePixelBuffer(at: planeIndex) { src in
            destination.withUnsafePixelBuffer(at: planeIndex) { dest in

                src.applyPolynomial(
                    coefficientSegments: [coefficients],
                    boundaries: [-.infinity, .infinity],
                    destination: dest)
            }
        }

        return coefficients
    }
```

After the transform, the `displayPlanarDestinationBuffers()` function calls `interleave(destination:)` to generate an interleaved image that the sample code displays in the user interface.

Display the tone curve in the user interface

The `PolynomialEditor` class uses the coefficients that the `applyPolynomial()` function computes to render a representation of the response curve.

The `updatePath()` function calls `evaluatePolynomial(usingCoefficients:withVariables:result:)` to build a `CGPath` instance that the editor uses to render a smooth curve in the user interface.

```
static func updatePath(path: inout Path,
                       size: CGSize,
                       coefficients: [Float]) {

    let polynomialResult = [Float](unsafeUninitializedCapacity: ramp.count) {
        buffer, initializedCount in

        vDSP.evaluatePolynomial(usingCoefficients: coefficients.reversed(),
                                withVariables: ramp,
                                result: &buffer)

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

        initializedCount = ramp.count
    }

    let cgPath = CGMutablePath()
    let hScale = size.width / 256
    let points: [CGPoint] = polynomialResult.enumerated().map {
        CGPoint(x: CGFloat($0.offset) * hScale,
                y: size.height - (size.height * CGFloat($0.element) ))
    }

    cgPath.addLines(between: points)

    path = Path(cgPath)
}
```

See Also

Color and Tone Adjustment

{} Adjusting the brightness and contrast of an image

Use a gamma function to apply a linear or exponential curve.

{} Adjusting saturation and applying tone mapping

Convert an RGB image to discrete luminance and chrominance channels, and apply color and contrast treatments.

{} Adjusting the hue of an image

Convert an image to L*a*b* color space and apply hue adjustment.

{} Specifying histograms with vImage

Calculate the histogram of one image, and apply it to a second image.

Enhancing image contrast with histogram manipulation

Enhance and adjust the contrast of an image with histogram equalization and contrast stretching.

Histogram

Calculate or manipulate an image's histogram.