Sample Code

Adjusting the brightness and contrast of an image

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

Download

macOS 13.0+ | Xcode 14.3+

Overview

This sample code project uses the vImage piecewise gamma function to adjust the response curve (that is, the value of an output pixel based on the value of the corresponding input pixel) of an 8-bit RGB image. Changing the shape of the response curve changes the brightness and contrast of an image.

You can use a piecewise gamma function to apply either a linear or an exponential response curve to pixels in an image based on their value.

This app displays a sample image and uses a SwiftUI `Picker` control to apply different preset linear (labeled L1 to L4) and exponential (labeled E1 to E3) response curves. This sample code project demonstrates how different response curves affect an image by changing its brightness and contrast.

Define response curve presets

The sample app defines a structure, `ResponseCurvePreset`, that contains the coefficients the linear function uses, the gamma the exponential function uses, and the boundary between the linear and exponential functions.

```
struct ResponseCurvePreset: Hashable, Identifiable {
    let id: String
    let boundary: Pixel_8
    let linearScale: Float
    let linearBias: Float
    let gamma: Float
}
```

The `presets` array contains sample presets that apply different adjustments to the sample image. When the user changes the selected value of the `Picker` control, the app passes the appropriate `preset` structure to the `getGammaCorrectedImage(preset:source:destination:imageFormat:)` function. This function applies the adjustment to the image and returns the result.

Define the adjustment parameters

The sample app specifies the division between linear and gamma adjustments by passing a boundary parameter to the piecewise gamma function, `applyGamma(linearParameters:exponentialParameters:boundary:destination:)`. The function uses the exponential curve to calculate the output value when the input value is greater than or equal to the boundary value. Otherwise, the function uses the linear curve.

For 8-bit images, the boundary is a `Pixel_8` value.

- A value of `0` specifies that the gamma function applies the exponential adjustment to all pixels.
- A value of `255` specifies that the gamma function applies the linear adjustment to all pixels.
- A value of `127` specifies that the gamma function applies the linear adjustment to all pixels with a value less than one-half, and the exponential adjustment to the remaining pixels.

The sample app passes the linear and exponential coefficients (for example, the `scale` and `bias` in `(scale * inputvalue) + bias`) as tuples of floating-point values.

```
let linearCoefficients = (preset.linearScale, preset.linearBias)

let exponentialCoefficients = (Float(1), Float(0), preset.gamma, Float(0))
```

Remove the alpha channel

The `applyGamma(linearParameters:exponentialParameters:boundary:destination:)` function in the sample app treats an interleaved buffer as a single plane and

applies the same gamma adjustment to all channels — including any alpha channel. Adjusting the response curve of the alpha channel changes transparency properties. To avoid this, the sample app converts the RGBA source image to RGB and applies the adjustment to that.

The sample app creates the RGB version of the source image by creating a three-channel, 8-bit-per-channel format.

```
var imageFormat = vImage_CGImageFormat(
    bitsPerComponent: 8,
    bitsPerPixel: 8 * 3,
    colorSpace: CGColorSpaceCreateDeviceRGB(),
    bitmapInfo: CGBitmapInfo(rawValue: CGImageAlphaInfo.none.rawValue),
    renderingIntent: .defaultIntent)!
```

Then it declares three-channel source and destination buffers.

```
let sourceBuffer: vImage.PixelBuffer<vImage.Interleaved8x3>
let destinationBuffer: vImage.PixelBuffer<vImage.Interleaved8x3>
```

Finally, it creates the source buffer using the `init(cgImage:cgImageFormat:pixelFormat:)` initializer.

```
sourceBuffer = try vImage.PixelBuffer(
    cgImage: sourceImage,
    cgImageFormat: &imageFormat,
    pixelFormat: vImage.Interleaved8x3.self)
```

On return, `sourceBuffer` contains the red, green, and blue channels of `sourceImage`.

Apply the adjustment

The sample app calls `applyGamma(linearParameters:exponentialParameters:boundary:destination:)` to apply the adjustment.

```
source.applyGamma(linearParameters: linearCoefficients,
                  exponentialParameters: exponentialCoefficients,
                  boundary: preset.boundary,
                  destination: destination)
```

To create the image, the sample app passes the destination buffer and RGB format to `make CGImage(cgImageFormat:)`.

```
if let result = destination.makeCGImage(cgImageFormat: imageFormat) {
    return result
} else {
    fatalError("Unable to generate output image.")
}
```

The following sections explain the presets in more detail.

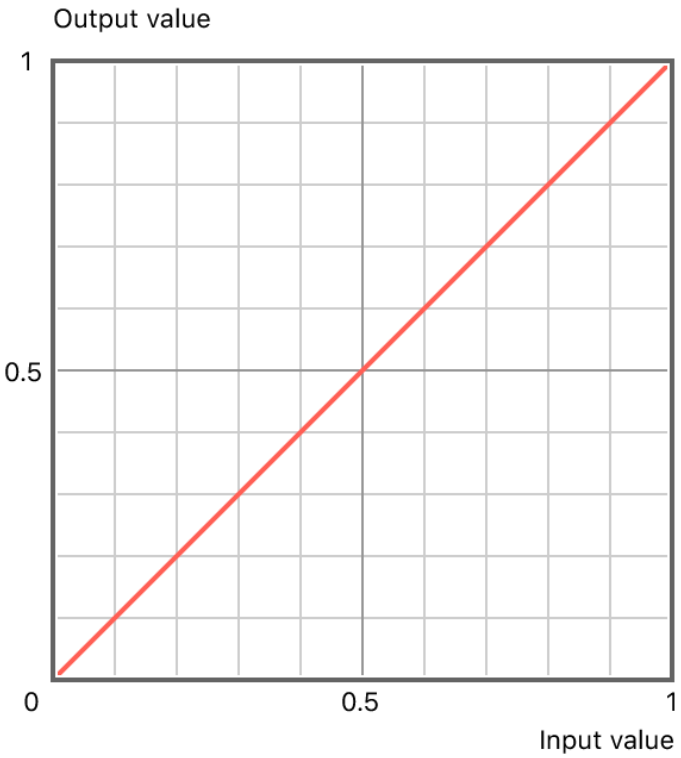
Apply linear adjustment

The following presets use the linear adjustment (that is, `boundary` is 255). The output value for each pixel is calculated as:

```
(scale * inputValue) + bias
```

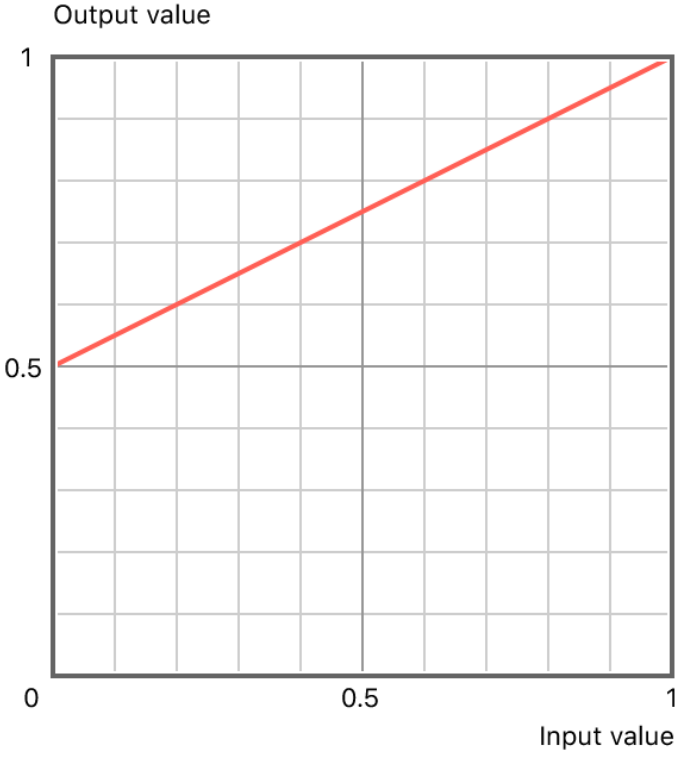
The L1 preset returns each pixel unchanged.

```
ResponseCurvePreset(id: "L1",
                    boundary: 255,
                    linearScale: 1,
                    linearBias: 0,
                    gamma: 0),
```



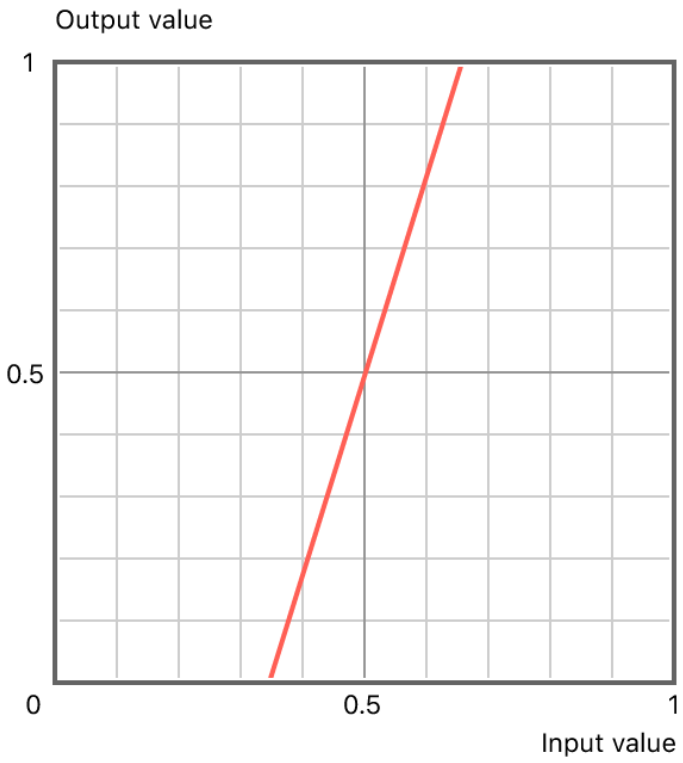
The L2 preset returns a washed-out image where blacks transform to grays. When the input value is `0`, the output value is `0.5`.

```
ResponseCurvePreset(id: "L2",
                    boundary: 255,
                    linearScale: 0.5,
                    linearBias: 0.5,
                    gamma: 0),
```



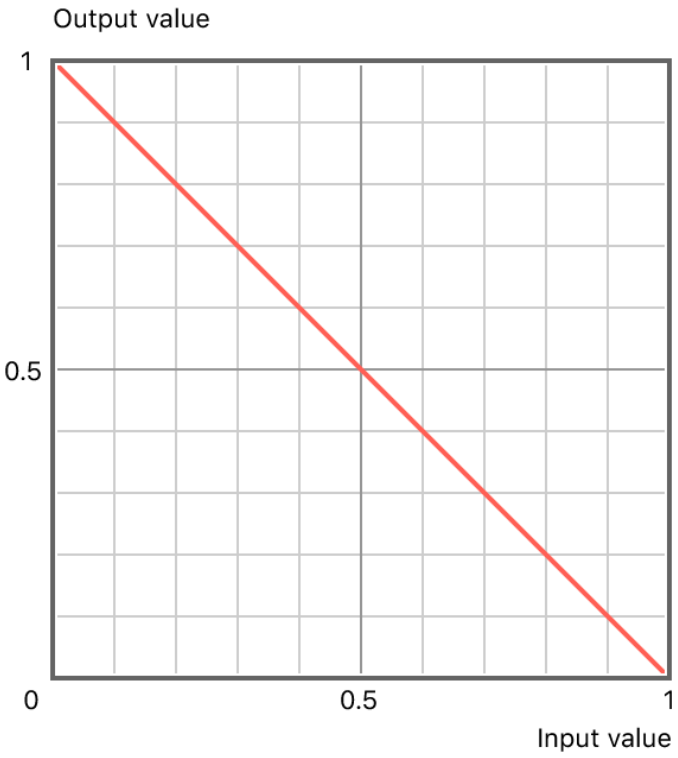
The L3 preset returns an image with a lot of contrast. When the input value is less than one-third, the output value is `0`; when the input value is greater than two-thirds, the output value is `1`. The preset transforms input values between one-third and two-thirds to the range `0 – 1`.

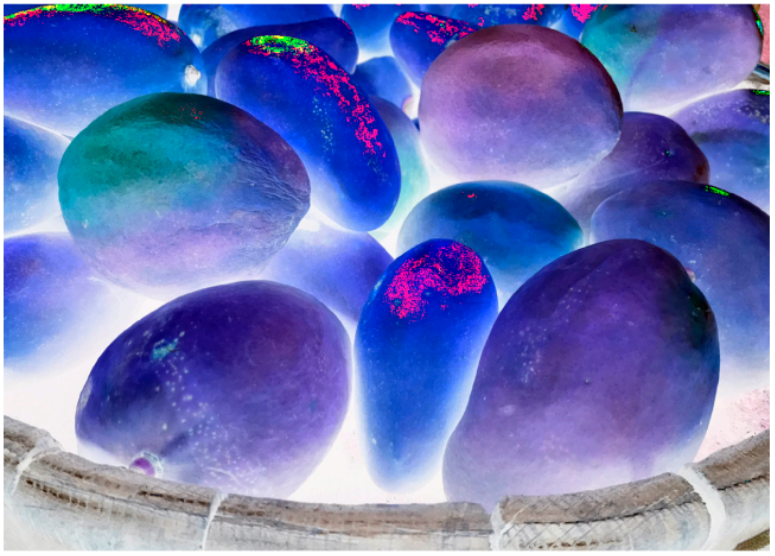
```
ResponseCurvePreset(id: "L3",
                    boundary: 255,
                    linearScale: 3,
                    linearBias: -1,
                    gamma: 0),
```



The L4 preset returns a negative version of the image. When the input value is `1`, the output value is `0`; when the input value is `0`, the output value is `1`.

```
ResponseCurvePreset(id: "L4",
                    boundary: 255,
                    linearScale: -1,
                    linearBias: 1,
                    gamma: 0),
```





Apply exponential adjustment

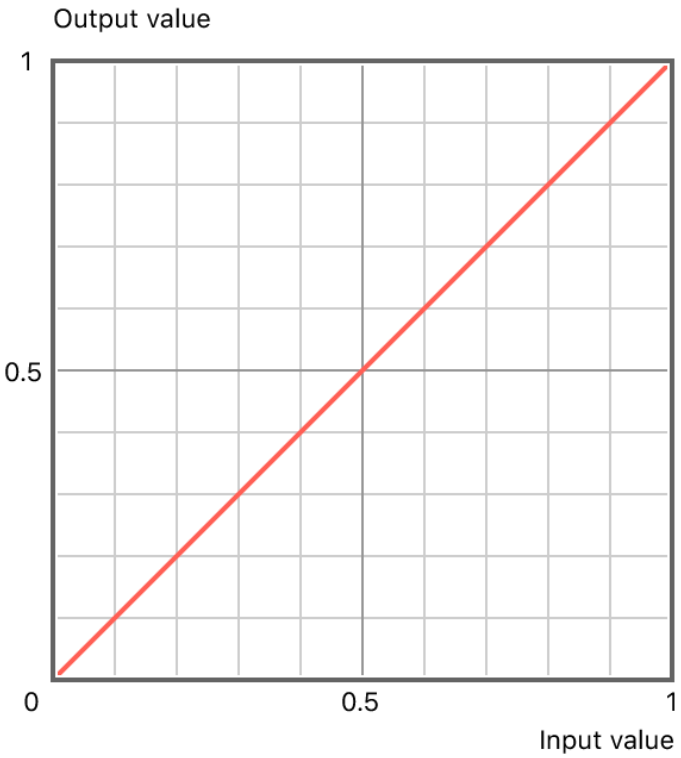
The following presets use the exponential adjustment (that is, `boundary` is 0). The output value for each pixel is calculated as:

```
pow((scale * inputValue) + preBias, gamma) +
     postBias
```

In these examples, `exponentialCoefficients` is defined as `(1, 0, 0)` and the calculation can be simplified to `pow(inputValue, gamma)`.

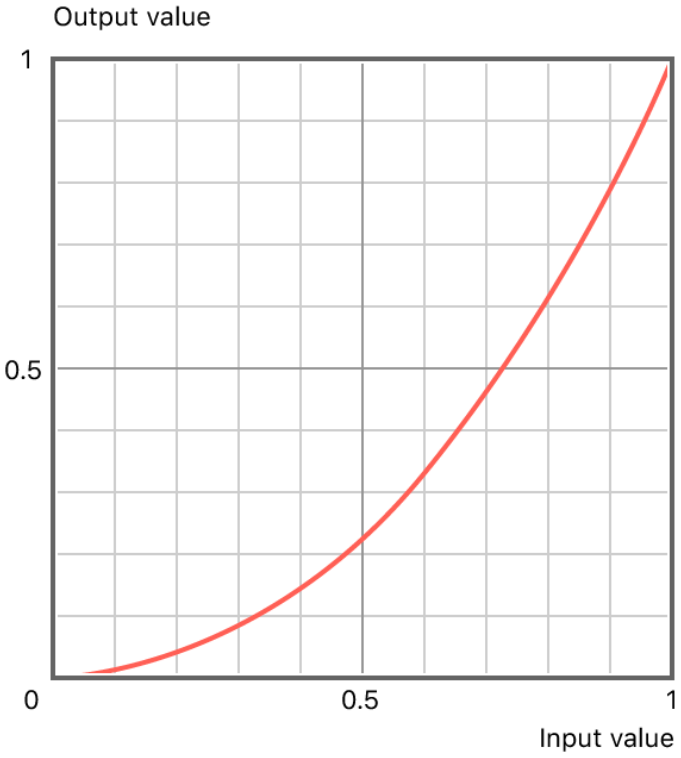
The `E1` preset returns each pixel unchanged.

```
ResponseCurvePreset(id: "E1",
                    boundary: 0,
                    linearScale: 1,
                    linearBias: 0,
                    gamma: 1),
```



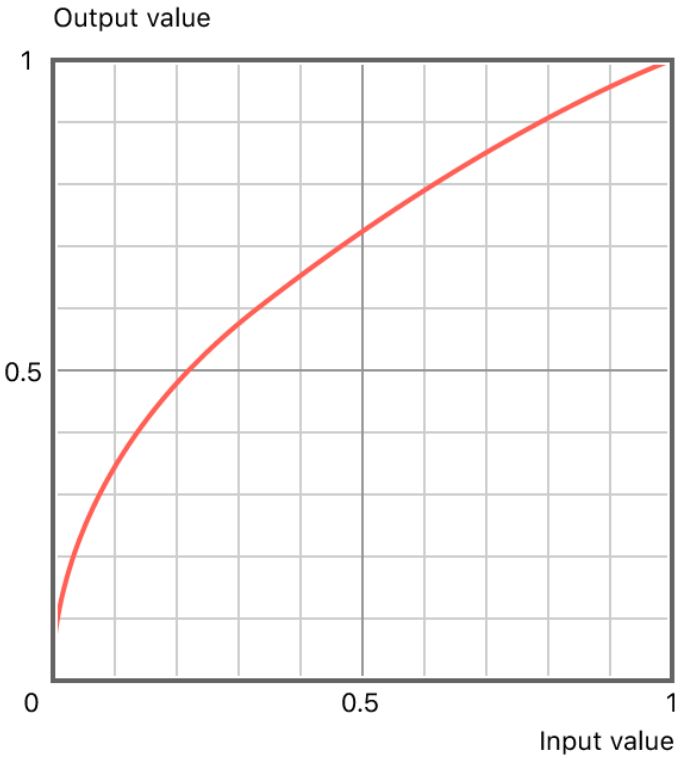
The E2 preset has an overall darkening effect.

```
ResponseCurvePreset(id: "E2",
                    boundary: 0,
                    linearScale: 1,
                    linearBias: 0,
                    gamma: 2.2),
```



The E3 preset has an overall lightening effect.

```
ResponseCurvePreset(id: "E3",
                    boundary: 0,
                    linearScale: 1,
                    linearBias: 0,
                    gamma: 1 / 2.2)
```



Correct gamma before applying operations

Many vImage operations — such as convolution and scaling — provide optimal results when working on images with a linear response curve. When working with nonlinear images — such as sRGB — best practice is to convert them to a linear color space by applying a reciprocal gamma (such as `1/2.2`), performing the operation, and converting them back to their original domain by applying the original gamma (such as `2.2`).

See Also

Color and Tone Adjustment

{} Adjusting saturation and applying tone mapping

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

{} Applying tone curve adjustments to images

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

{} 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.