

[Accelerate](#) / Adjusting the hue of an image

Sample Code

Adjusting the hue of an image

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

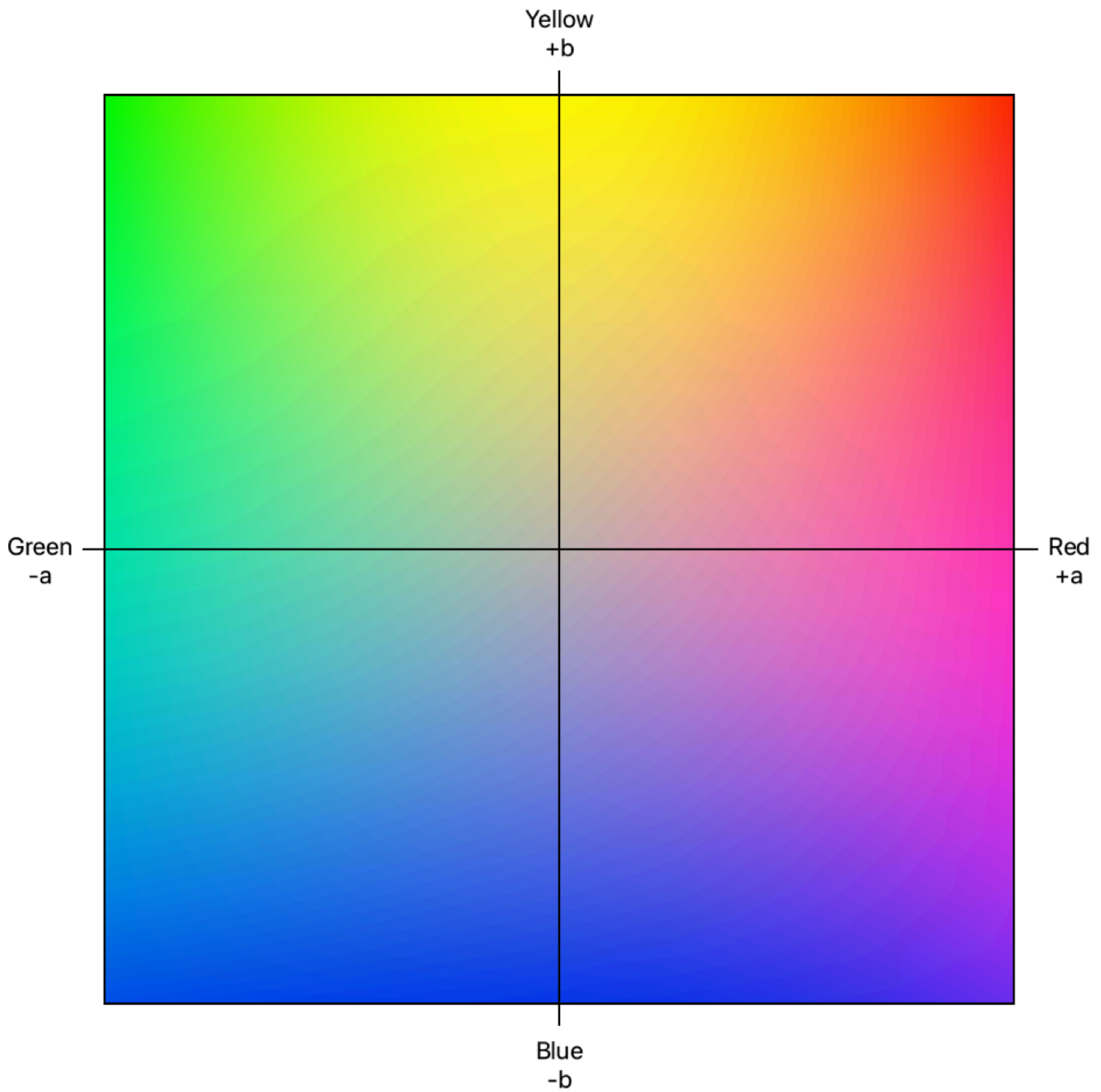
Download

macOS 13.0+ | Xcode 14.0+

Overview

This sample code project allows you to adjust the hue of an image by treating the chrominance information as 2D coordinates, and transforming those values with a rotation matrix. You can convert an RGB image — with its pixels represented as red, green, and blue values — to $L^*a^*b^*$, where luminance and chrominance are stored discretely. The L^* in $L^*a^*b^*$ refers to the lightness, and the a^* and b^* refer to the red-green and blue-yellow values, respectively.

The image below shows an approximation of an $L^*a^*b^*$ color chart. The a^* value transitions horizontally (left to right) from negative, through zero, to positive, and the b^* value transitions vertically (bottom to top) from negative, through zero, to positive. Because this sample code focuses on color rather than lightness, the image doesn't consider L^* .



The sample uses the vImage Any-to-Any converter to convert the source image's color space to L*a*b*. The code converts the interleaved L*a*b* image data to multiple-plane image data that it passes to a matrix multiply operation to apply the hue adjustment.

The following image shows four photographs, from left to right, with a hue adjustment of -90° , 0° (an unchanged hue), 90° , and 180° :



Create the L*a*b* image format

To create the image format for the L*a*b* color space, the sample app uses the [genericLab](#) system-defined [CGColorSpace](#).

```
var labImageFormat = vImage_CGImageFormat(  
    bitsPerComponent: 8,  
    bitsPerPixel: 8 * 3,  
    colorSpace: CGColorSpace(name: CGColorSpace.genericLab!),  
    bitmapInfo: CGBitmapInfo(rawValue: CGImageAlphaInfo.none.rawValue),  
    renderingIntent: .defaultIntent)!
```

On return, `labImageFormat` describes the interleaved L*a*b* pixels over which this sample works. The first channel in each pixel is the lightness, and the second and third channels are the *a** and *b**, respectively.

Generate the pixel buffer and image format from the source image

The converter that the sample uses to convert the source pixels to L*a*b* color space requires two [vImage_CGImageFormat](#) structures that describe the source and destination images. The sample uses the [makeDynamicPixelBufferAndCGImageFormat\(cgImage:\)](#) method to create a dynamic pixel buffer and image format structure from the source Core Graphics image.

```
let source = try vImage.PixelBuffer  
    .makeDynamicPixelBufferAndCGImageFormat(cgImage: sourceCGImage)
```

On return, `source.cgImageFormat` contains the image format of the source image, and `source.pixelBuffer` is a pixel buffer that contains the source image data.

Create the source image color space to L*a*b* converter

The sample app uses the source and L*a*b* image formats to create a [vImageConverter](#) instance to convert between the two color spaces.

```
let rgbToLab = try vImageConverter.make(sourceFormat: source.cgImageFormat,
                                       destinationFormat: labImageFormat)
```

For more information about vImage's convert-any-to-any functionality, see [Building a basic image conversion workflow](#).

Convert the source image to L*a*b*

The sample creates a pixel buffer that's the same size as the source image.

```
labInterleavedSource = vImage.PixelBuffer<vImage.Interleaved8x3>(size: size)
```

The converter's [convert\(from:to:\)](#) function performs the conversion.

```
try rgbToLab.convert(from: source.pixelBuffer,
                    to: labInterleavedSource)
```

On return, the labInterleavedSource contains the L*a*b* representation of the source image.

Convert the interleaved L*a*b* buffer to planar buffers

The function the sample app uses to apply the hue adjustment, [multiply\(by:divisor:preBias:postBias:destination:\)](#), operates on a multiple-plane pixel buffer. To convert the interleaved L*a*b* buffer to planar buffers, the app creates a [vImage.Planar8x3](#) pixel buffer.

```
labPlanarDestination = vImage.PixelBuffer<vImage.Planar8x3>(size: size)
```

It then calls [deinterleave\(destination:\)](#) to populate the planar buffers with the contents of the interleaved buffer.

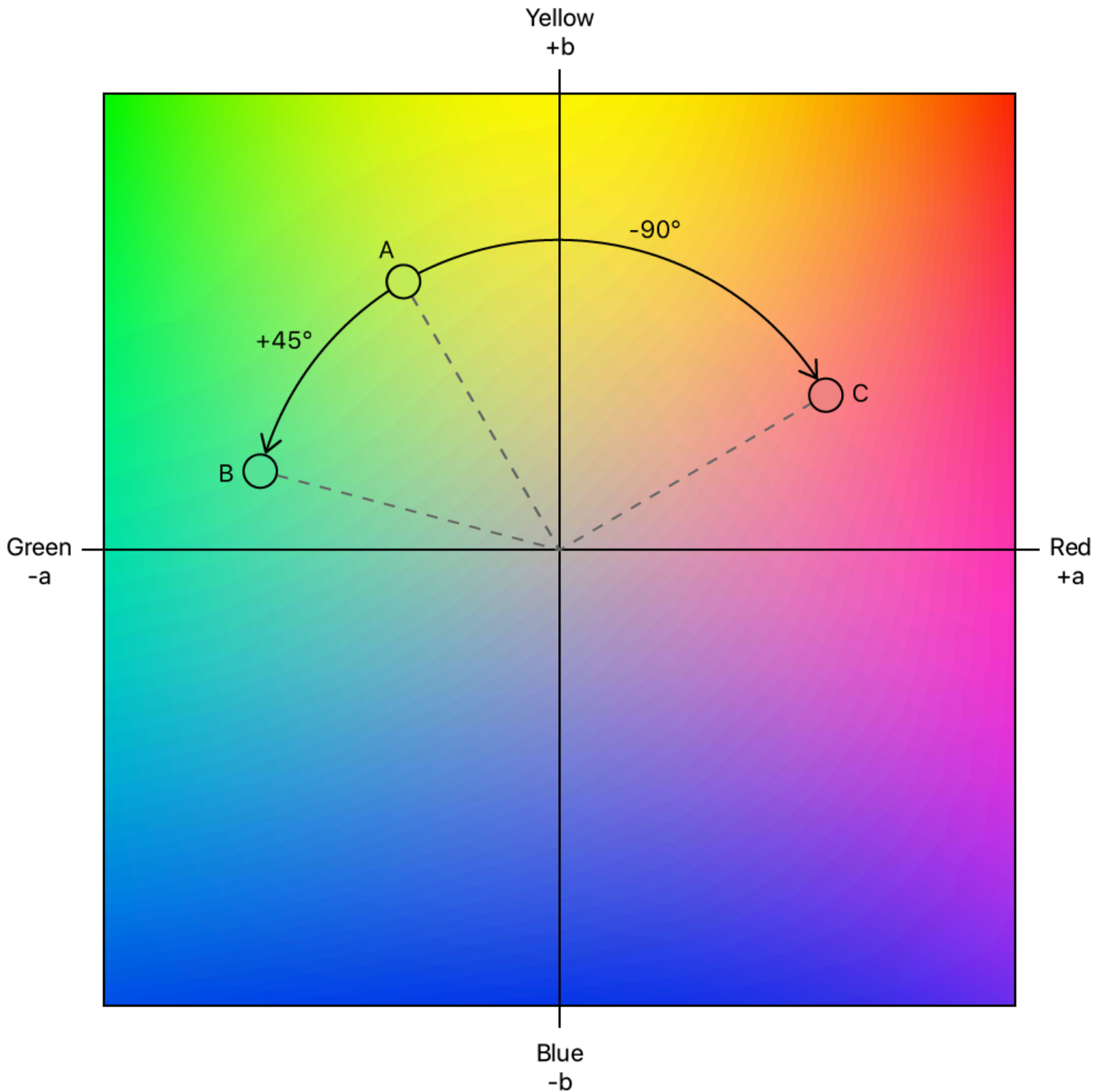
```
labInterleavedSource.deinterleave(destination: labPlanarDestination)
```

For more information about working with planar buffers, see [Optimizing image-processing performance](#).

Apply the hue adjustment

The app adjusts the hue of an image by rotating a two-element vector, described by a^* and b^* . For more information about working with rotation matrices, see [Working with Matrices](#).

The following visualizes a sample color (marked A) rotated by -90° (marked C) and 45° (marked B):



The following code generates the rotation matrix based on hueAngle:

```
let divisor: Int = 0x1000
```

```
let rotationMatrix = [
    1, 0, 0,
    0, cos(hueAngle), -sin(hueAngle),
    0, sin(hueAngle), cos(hueAngle)
].map {
    return Int($0 * Float(divisor))
}
```

The `preBias` and `postBias` values effectively shift the a^* and b^* values from $0 \dots 255$ to $-128 \dots 127$, so the rotation is centered where a^* and b^* are zero.

```
let preBias = [Int](repeating: -128, count: 3)
let postBias = [Int](repeating: 128 * divisor, count: 3)
```

The `multiply(by:divisor:preBias:postBias:destination:)` function multiplies each pixel in the source buffer by the matrix and writes the result to the destination buffers. The code performs the matrix multiplication in-place, so the source and destination point to the same buffers.

The following code performs the matrix multiply operation:

```
labPlanarDestination.multiply(
    by: rotationMatrix,
    divisor: divisor,
    preBias: preBias,
    postBias: postBias,
    destination: labPlanarDestination)
```

On return, `labPlanarDestination` contains the hue-adjusted a^* and b^* channels.

Display the image

Finally, the sample code converts the hue-adjusted planar buffer back to an interleaved buffer.

```
labPlanarDestination.interleave(destination: labInterleavedDestination)
```







The SwiftUI `Image` view supports the $L^*a^*b^*$ color space. The following code creates a Core Graphics image from the interleaved pixel buffer and passes it to the published `outputImage` property that the app displays on the screen:

```
if let result = labInterleavedDestination
    .makeCGImage(cgImageFormat: labImageFormat) {

    DispatchQueue.main.async {
        self.outputImage = result
    }
}
```

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