Structure

vImage.PixelBuffer

An image buffer that stores an image's pixel data, dimensions, bit depth, and number of channels.

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst | macOS 13.0+ | tvOS 16.0+ | visionOS | watchOS 9.0+

```
struct PixelBuffer<Format> where Format : PixelFormat
```

Mentioned in

- Enhancing image contrast with histogram manipulation
- Applying vImage operations to regions of interest
- Building a basic image conversion workflow
- Applying color transforms to images with a multidimensional lookup table
- Converting bitmap data between Core Graphics images and vImage buffers

Overview

Use a `vImage.PixelBuffer` to represent an image from a `CGImage` instance, a `CVPixelBuffer` structure, or a collection of raw pixel values. Pixel buffers are typed by their bits per channel and number of channels. For example, `vImage.Interleaved8x4` indicates a 4-channel, 8-bit-per-channel pixel buffer that contains image data such as RGBA or CMYK.

Pixel buffers expose methods that are available for the buffer's pixel format. For example, the fast box convolution functions are only available for one- and four-channel 8-bit per channel buffers:

```
let src = vImage.PixelBuffer<vImage.Interleaved8x4>(cgImage: cgImage,
                                                    cgImageFormat: &cgImageFormat)
```

```
let dest = vImage.PixelBuffer<vImage.Interleaved8x4>(src.size)

src.boxConvolve(kernelSize: vImage.Size(width: 64, height: 64),
                edgeMode: .truncateKernel,
                destination: dest)
```

Typed pixel buffers provide a simple API to convert between pixel formats. For example, the following code converts 8-bit unsigned integer pixels to 32-bit floating point pixels:

```
let src = vImage.PixelBuffer<vImage.Interleaved8x4>(cgImage: cgImage,
                                                    cgImageFormat: &cgImageFormat)
let dest = vImage.PixelBuffer<vImage.InterleavedFx4>(size: src.size)

src.convert(to: dest)
```

vImage pixel buffers manage their memory, therefore, you don't need to call `deallocate()` when you're finished with the buffer.

Topics

Pixel buffer essentials

- Creating vImage pixel buffers

  Allocate and initialize pixel buffers from raw pixel data, Core Graphics images, and Core Video buffers.

- Pixel formats

  Specify a pixel buffer's bit depth, number of channels, and data storage format.

- Working with underlying data

  Access a pixel buffer's underlying pixel data.

Inspecting a pixel buffer

`var width: Int`

The width of the pixel buffer.

`var height: Int`

The height of the pixel buffer.

`var size: vImage.Size`

The size of the pixel buffer.

`var channelCount: Int`

Returns the number of channels.

`var rowStride: Int`

The width, in pixels, of the underlying memory, including any additional row byte padding.

`var byteCountPerPixel: Int`

Returns the number of bytes per pixel.

`var count: Int`

The total number of pixels multiplied by the number of channels in the buffer, including any row padding.

`var array: [Format.ComponentType]`

An array of `width * height * channelCount` values that's a copy of the buffer's visible contents.

Pixel buffer methods

`func copy(to: vImage.PixelBuffer<Format>)`

Copies the contents of the pixel buffer to another pixel buffer.

`func copy(to: CVPixelBuffer, cvImageFormat: vImageCVImageFormat, cgImageFormat: vImage_CGImageFormat) throws`

Copies the contents of a pixel buffer to a Core Video pixel buffer.

`func makeCGImage(cgImageFormat: vImage_CGImageFormat) -> CGImage?`

Returns a Core Graphics image from the pixel buffer's contents.

`func withCVPixelBuffer(readOnly: Bool, body: (CVPixelBuffer) -> Void)`

Calls the given closure with a locked 32-bit BGRA Core Video Pixel Buffer.

Pixel buffer operations

Applying geometric operations to pixel buffers

Reflect, shear, rotate, scale, and apply affine transforms to image data.

Applying color transforms to pixel buffers

Adjust the colors of an image by applying gamma, polynomials, or multidimensional lookup.

- Blending and compositing pixel buffers

  Composite two pixel buffers to create a single image.

- Convolving and applying morphology

  Apply convolution, dilation, or erosion to a pixel buffer.

- Thresholding and clipping pixel buffer values

  Limit the values in a pixel buffer to a threshold or a range.

- Calculating and transforming histograms

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

- Converting pixel buffers

  Convert pixel buffer data between different bit-depths.

- Interleaving and deinterleaving pixel buffers

  Convert pixel buffer data between interleaved and planar formats.

- Cropping and working with regions of interest

  Crop images and apply operations to regions of interest.

- Applying channel operations

  Extract, flatten, permute, and overwrite the individual color channels of a pixel buffer.

- Applying arithmetic operations

  Multiply the pixel values of a buffer by scalar values or matrices.

Instance Properties

```
var bytesPerRow: Int
```

The width, in bytes, of the underlying memory including any additional row byte padding.

Instance Methods

```
func applyLookup([Pixel_16U], destination: vImage.PixelBuffer<vImage.Planar16U>)
```

```
func applyLookup([Pixel_8], destination: vImage.PixelBuffer<vImage.Planar8>)
```

`func applyLookup([Pixel_F], destination: vImage.PixelBuffer<vImage.PlanarF>)`

`func applyMorphology(operation: vImage.MorphologyOperation<Format.ComponentType>, destination: vImage.PixelBuffer<Format>)`

Applies a morphology operation to the buffer.

`func applyMorphology(operation: vImage.MorphologyOperation<Format.ComponentType>, destination: vImage.PixelBuffer<Format>)`

Applies a morphology operation to the buffer.

`func convolve(with: vImage.ConvolutionKernel2D<Float>, bias: Float?, edgeMode: vImage.EdgeMode<Pixel_8888>, destination: vImage.PixelBuffer<Format>)`

`func separableConvolve(horizontalKernel: [Float], verticalKernel: [Float], bias: Float, edgeMode: vImage.EdgeMode<Pixel_8888>, destination: vImage.PixelBuffer<Format>)`

`func withBNNSTensor<R>((BNNSTensor) throws -> R) rethrows -> R`

Calls the given closure with a pointer to a BNNS tensor that shares memory with the pixel buffer.

Default Implementations

- AccelerateBuffer Implementations
- AccelerateMatrixBuffer Implementations
- AccelerateMutableMatrixBuffer Implementations

Relationships

Conforms To

`AccelerateBuffer`

Conforms when `Format` conforms to `StaticPixelFormat`.

`AccelerateMatrixBuffer`

Conforms when `Format` conforms to `StaticPixelFormat`.

`AccelerateMutableBuffer`

Conforms when `Format` conforms to `StaticPixelFormat`.

`AccelerateMutableMatrixBuffer`

Conforms when `Format` conforms to `StaticPixelFormat`.

`Copyable`

See Also

vImage Pixel Buffers

{} Using vImage pixel buffers to generate video effects

Render real-time video effects with the vImage Pixel Buffer.

{} Applying tone curve adjustments to images

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

{} Adjusting the brightness and contrast of an image

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

{} Adjusting the hue of an image

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

{} Sharing texture data between the Model I/O framework and the vImage library

Use Model I/O and vImage to composite a photograph over a computer-generated sky.

{} Calculating the dominant colors in an image

Find the main colors in an image by implementing k-means clustering using the Accelerate framework.