Instance Method

withUnsafeMutableBufferPointer(_:)

Calls the given closure with a pointer to the buffer's mutable contiguous storage.

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

```
func withUnsafeMutableBufferPointer<R>(_ body: (inout UnsafeMutableBuffer
Pointer<Format.ComponentType>) throws -> R) rethrows -> R
```

Available when `Format` conforms to `StaticPixelFormat`.

Parameters

`body`

A closure with an `UnsafeMutableBufferPointer` parameter that points to the contiguous storage for the pixel buffer.

Return Value

The return value, if any, of the body closure parameter.

Discussion

You can use this function to simplify interoperation with other libraries and frameworks.

Note

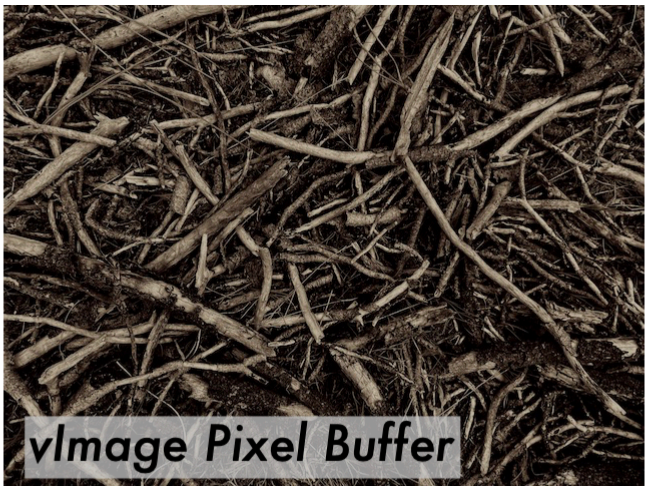
The contiguous storage may include space outside of the buffer’s width that doesn’t contain image information. The storage contains `rowStride * channelCount * height` elements.

Add Text to a Pixel Buffer

The following code creates a `CGContext` instance from the pointer to the buffer’s storage. The code renders a gray rectangle over the image and draws text inside the rectangle by calling `CTLineDraw(_:_:)`.

```
let srcImage =  imageLiteral(resourceName: " ... ").cgImage(
    forProposedRect: nil,
    context: nil,
    hints: nil)!

var cgImageFormat = vImage_CGImageFormat()

let buffer = try vImage.PixelBuffer(
    cgImage: srcImage,
    cgImageFormat: &cgImageFormat,
    pixelFormat: vImage.Interleaved8x4.self)

buffer.withUnsafeMutableBufferPointer { ptr in

    // Define font.
    let fontAttributes = [
        kCTFontFamilyNameAttribute : "Futura",
        kCTFontStyleNameAttribute : "Medium Italic"
    ] as NSDictionary
    let fontDescriptor = CTFontDescriptorCreateWithAttributes(fontAttributes)
    let font = CTFontCreateWithFontDescriptor(fontDescriptor, 48, nil)
    let attributes = [kCTFontAttributeName : font] as CFDictionary

    // Create `CGContext` and attributed string.
    guard
        let context = CGContext(data: ptr.baseAddress!,
                                width: buffer.width,
                                height: buffer.height,
                                bitsPerComponent: Int(cgImageFormat.bitsPerComponent
                                 bytesPerRow: buffer.rowStride * buffer.byteCountPerP
                                 space: cgImageFormat.colorSpace.takeRetainedValue(),
                                 bitmapInfo: cgImageFormat.bitmapInfo.rawValue),
          let text = CFAttributedStringCreate(nil,
                                             "vImage Pixel Buffer" as CFString,
                                             attributes) else {
          return
      }

      let line = CTLineCreateWithAttributedString(text)

      context.textPosition = CGPoint(x: 25, y: 25)

      // Draw text background gray rectangle.
      let boundingRect = CTLineGetImageBounds(line, context).insetBy(dx: -5, dy: -5)
      context.setFillColor(.init(gray: 0.75, alpha: 0.75))
      context.addRect(boundingRect)
      context.drawPath(using: .fill)

      // Draw text to `CGContext`.
      CTLineDraw(line, context)
  }

  let result = buffer.makeCGImage(cgImageFormat: cgImageFormat)!
```

On return, the buffer contains the original image with a text overlay.



Fill a Pixel Buffer with Random Values

The following code uses Accelerate's BNNS library to fill a pixel buffer with random values:

```
let buffer = vImage.PixelBuffer(
    size: vImage.Size(width: 512,
                      height: 512),
    pixelFormat: vImage.Interleaved8x3.self)

buffer.withUnsafeMutableBufferPointer { bufferPtr in

    guard
        var descriptor = BNNSNDArrayDescriptor(
            data: bufferPtr,
            shape: .vector(bufferPtr.count)),
        let randomNumberGenerator = BNNSCreateRandomGenerator(
            BNNSRandomGeneratorMethodAES_CTR,
            nil) else {
                fatalError()
            }

    BNNSRandomFillUniformInt(randomNumberGenerator,
                             &descriptor,
                             0, 255)

    BNNSDestroyRandomGenerator(randomNumberGenerator)
}
```

On return, the buffer contains random color values.