И ещё немного про изображения

UIImage, CGImage, CIImage

Роман Волков

Linearity


Style
Brushes
Brush Mode
Fixed width
Fill
#FFFFFF
Blend Mode
Normal
Opacity
100%
Blur
0.00 pt
Stroke
#2F2C7D
Stroke Width
0.50 pt
Arrows
Dash
Gap
Pencil


MacBook Pro

Что вы узнаете

- Как правильно выбрать **Image
- Что обозначают аббревиатуры RGB, CMYK, YCbYr, P3
- Как компьютер "видит" изображения
- Способы манипуляций изображениями: "по-пиксельный", "декларативный"
- Различие между основными framework'ами для работы с изображениями в iOS/macOS

План

1. Как изображения хранятся в памяти
2. Отличие формата изображения от цветового пространства
3. Основные группы цветовых пространств
4. UIImage - зачем нужен и связь с CGImage и CIImage
5. CGImage. Особенности Quart 2D в iOS
6. CIImage и Core Image - инструмент обработки фото/видео

Немного теории

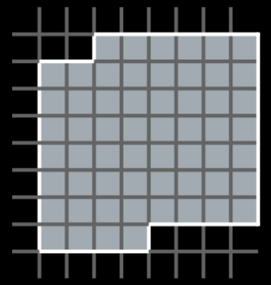
Форматы изображений

- JPEG
- PNG
- Tiff
- RAW
- BMP
- DNG
- И многие другие

Изображения в памяти

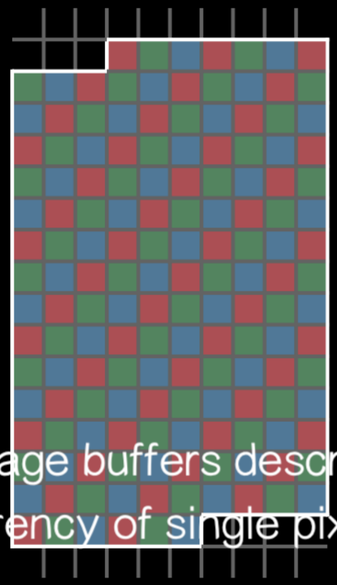
590KB

10MB



Data buffer

Bytes do notdirectly describe pixels

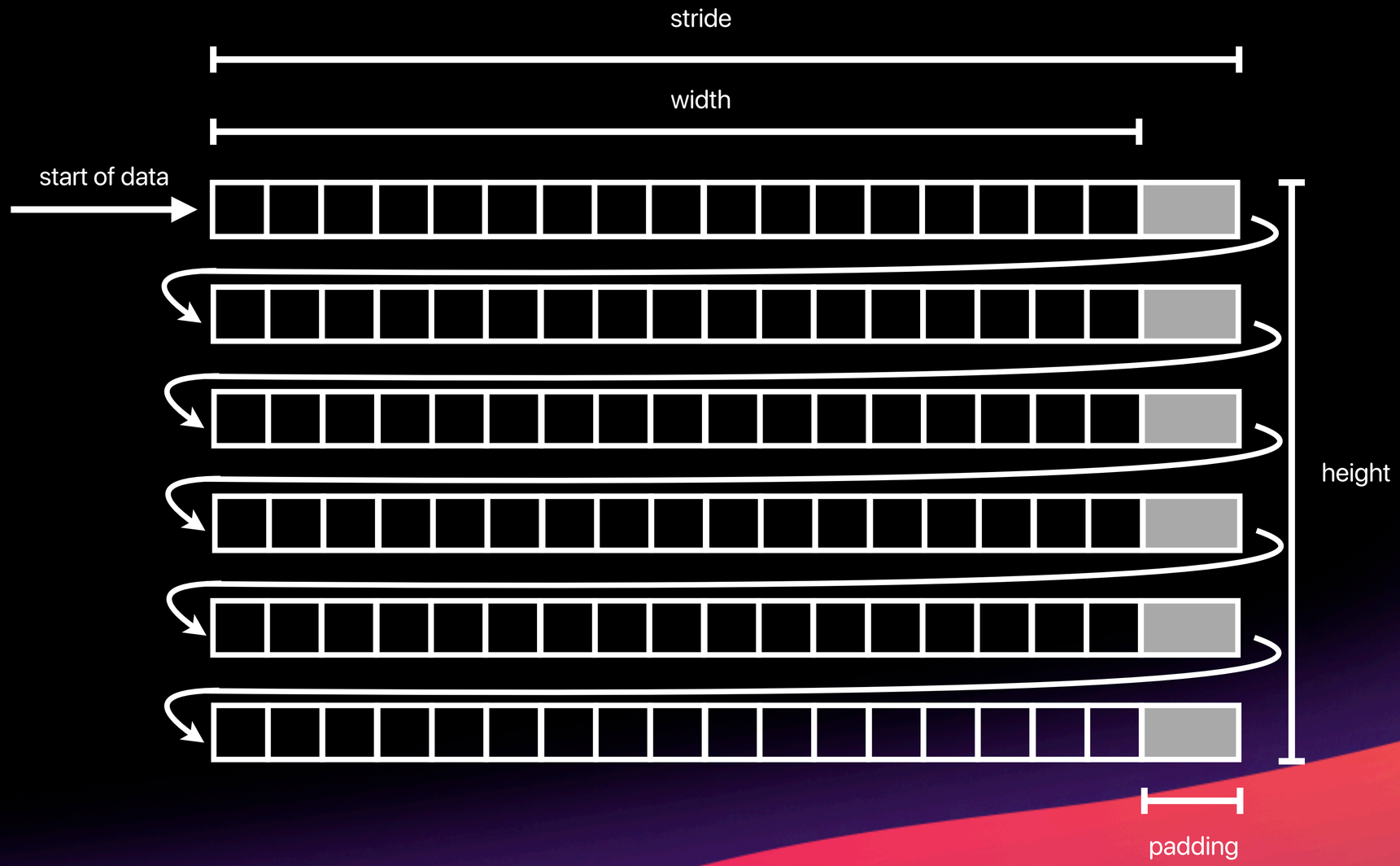


sRGB:

Each 32 bit in image buffers describes the color and the transparency of single pixel in our image.

UIImageView

Stride vs Width



Выравнивание - 16бит

Цветовые пространства

Цветовые пространства - это различные типы цветовых режимов, используемых в обработке изображений, видео и других систем для различных целей. Например:

- RGB
- CMYK
- YCbCr
- YIQ
- HSV
- и другие

RGB

RGB - (R: красный, G: зеленый, B: синий).

У RGB есть разные вариации как Adobe RGB, sRGB и другие.

Предназначен для электронных систем

RGBA. Opaque vs transparent

CMY'K

Четырёхцветная схема формирования цвета, используемая прежде всего в полиграфии.

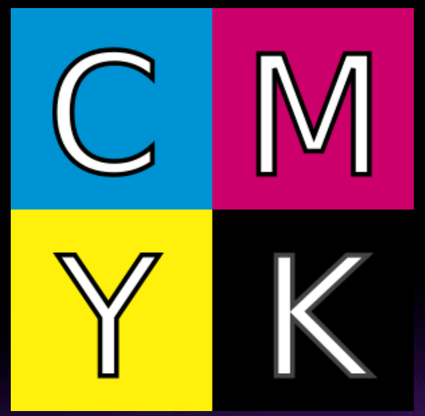
Отличие от RGB: CMYK - для отраженного счета, RGB - для светящихся элементов

C - голубой

M - пурпурный

Y - желтый

K - черный



Y′CbCr

- Семейство цветовых пространств, которые чаще всего используются в сферах цифровой фотографии и записи видео.
- Y′ – отдельная компонента яркости
- CB и CR – разница для красного и синего канала.
- Используется в JPEG
- Появился для обратной совместимости между ЧБ и цветными ТВ

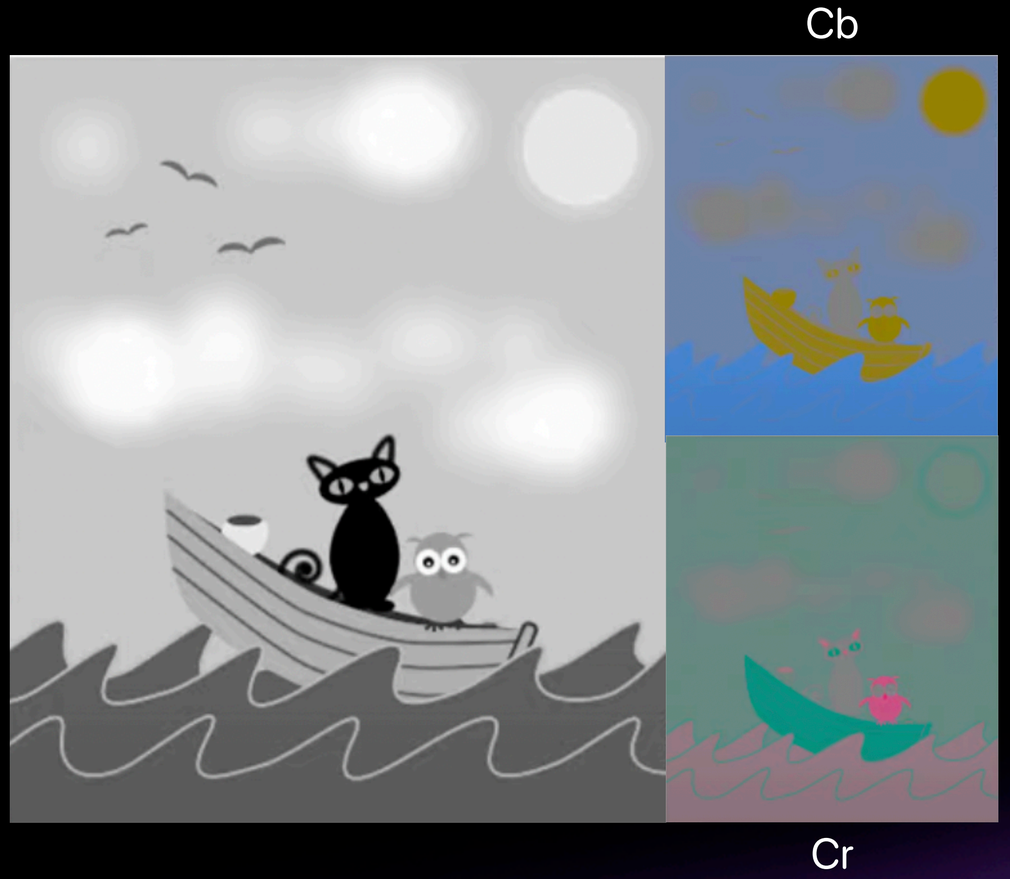
RGB -> Y'CbCr

Y

Cb

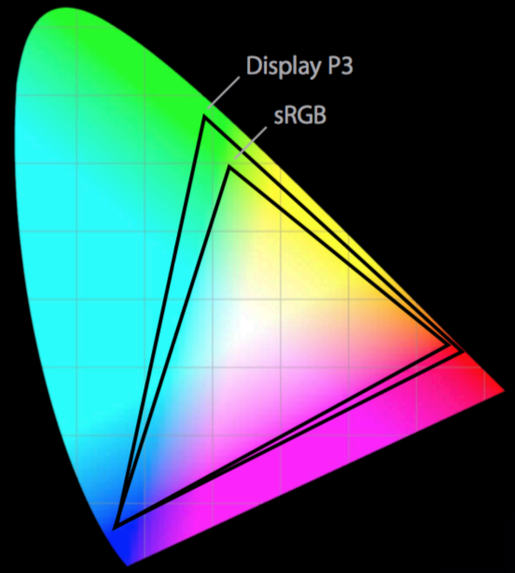
Cr

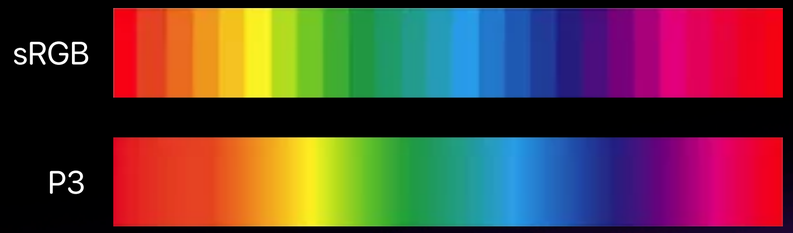
Y′CbCr planars



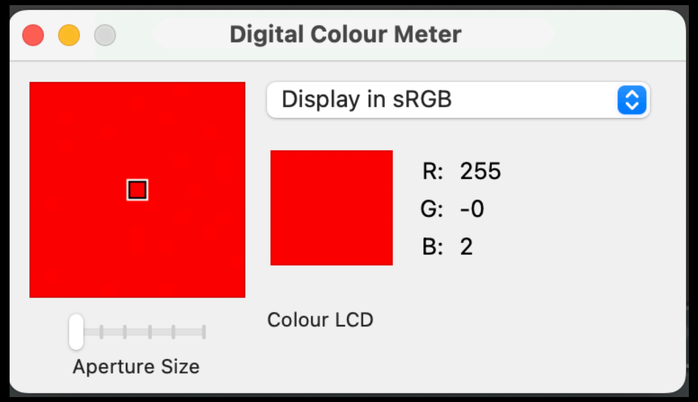
Display P3

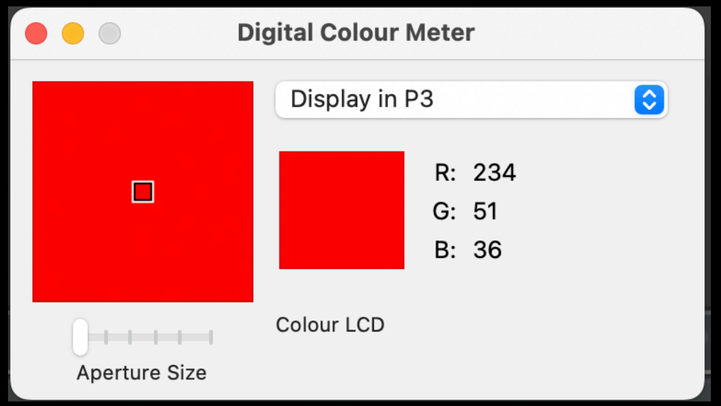
- sRGB - создан в 1996г. HP и Microsoft для использования в мониторах, принтерах и WWW
- Display P3 - расширение от Apple Inc. для дисплеев с расширенной цветовой гаммой

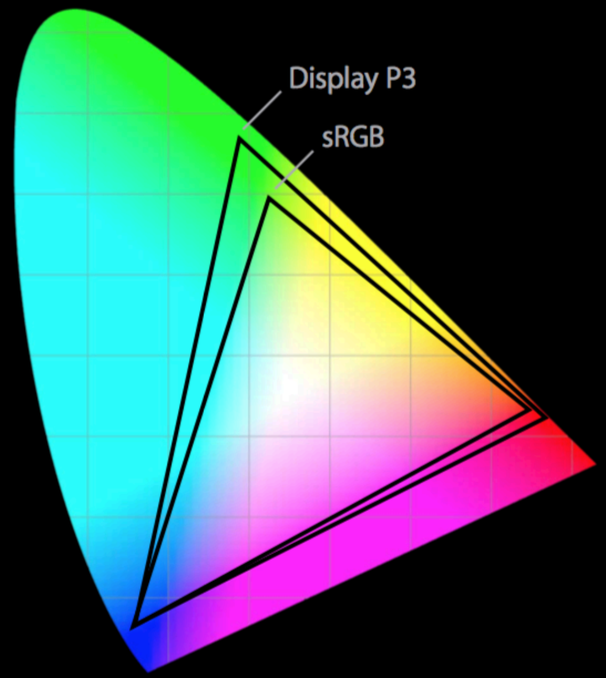




Display P3


Digital Colour Meter
Display in sRGB
R: 255
G: -0
B: 2
Colour LCD
Aperture Size


Digital Colour Meter
Display in P3
R: 234
G: 51
B: 36
Colour LCD
Aperture Size


Display P3
sRGB

UIImage

UIImage

"You use image objects to represent image data of all kinds"

- Удобная "обёртка" вокруг множества вариаций изображений для UIKit: растровые изображения, SF Symbols, вычислительные изображения, etc.
- Доступ к Assets
- Адаптация под TraitCollection
- Tinning
- PNG и JPEG
- и многое другое

UIImage::init из CGImage vs CIImage

```
open var cgImage: CGImage? { get }

open var ciImage: CIImage? { get }
```

Когда они будут **nil**?

UIImage ::cgImage vs ::ciImage

```
let image = #imageLiteral(resourceName: “image.jpg")

image.ciImage // nil
CIImage(image: image) // Optional(<CIImage: 0×6000034e8790 extent [0 0 5464  ...
image.cgImage // Optional(<CGImage 0×13c7141d0> (IP)\n ...

CIImage(image: image)?.cgImage === image.cgImage // true
```

UIImage::init из CGImage vs CIImage

```
let ciImage = CIImage(image: image)!

let image_1 = UIImage(cgImage: image.cgImage!)
let image_2 = UIImage(ciImage: ciImage)

image_1.cgImage // not nil
image_1.ciImage // nil

image_2.cgImage // nil
image_2.ciImage // not nil
// (хотя исходная ciImage создана из растрового изображения)

image_2.ciImage!.cgImage === image_1.cgImage // true
```

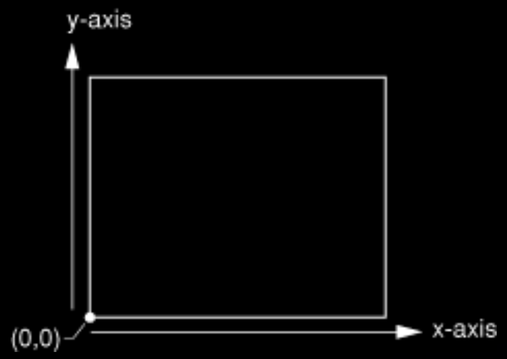
UIImage. Итог

- Значение ::cgImage и ::ciImage зависит использованного конструктора
- Обычно нужно использовать CIImage(image:) для API вроде Vision
- Создание CGImage из CIImage требует дополнительных действий

CGImage

Quartz 2D

- Quartz 2D - two-dimensional drawing engine
- CGContext
- Система координат: слева снизу
- Но UIKit инвертирует y-ось



CGImage

- Растровое изображение
- Изображение-маска

CGImage. Pixel format

- Кол-во битов на каждую компоненту цвета - bitsPerComponent
- Кол-во битов на пиксель - bitsPerPixel
- Кол-во байтов на одну строку - bytesPerRow
- Не всегда равно (кол-во пикселей * кол-во байт на пиксель)

CGImage. Bitmap info

```
public enum CGImageAlphaInfo : UInt32, @unchecked Sendable {
    case none = 0 /* For example, RGB. */
    case premultipliedLast = 1 /* For example, premultiplied RGBA */
    case premultipliedFirst = 2 /* For example, premultiplied ARGB */
    case last = 3 /* For example, non-premultiplied RGBA */
    case first = 4 /* For example, non-premultiplied ARGB */
    case noneSkipLast = 5 /* For example, RGBX. */
    case noneSkipFirst = 6 /* For example, XRGB. */
    case alphaOnly = 7 /* No color data, alpha data only */
}
```

CGImage. premultiplied vs non-premultiplied

- RGBA - **.last**
  - (R: 0.94, G: 0.39, B: 0.2, A: 0.5)
- RGBA - **.premultipliedLast**
  - (R: **0.47**, G: **0.195**, B: **0.1**, A: 0.5)

CGContext

- **CGContext::init**
- **UIGraphics(Begin/End)ImageContextWithOptions**
  - ARGB 32bit
  - opaque == true ?**.noneSkipFirst | kCGBitmapByteOrder32Host**
  - opaque == false?: **.premultipliedFirst | kCGBitmapByteOrder32Host**
- **UIGraphicsGetImageFromCurrentImageContext** - получить итоговое изображение
- Можно вызывать с любого потока.

CGImage. Deep copy

```
let newImage = UIImage(cgImage: image.cgImage!)

newImage.cgImage === image.cgImage // true



let cgImageCopy = image.cgImage!.copy()!

cgImageCopy === image.cgImage // false
```

CGImage. Deep copy

```
let data = image.cgImage!.dataProvider!.data! // returns copy of data

let anotherCopy = CGImage(
    pngDataProviderSource: image.cgImage!.dataProvider!,
    decode: nil,
    shouldInterpolate: false,
    intent: .defaultIntent
)
```

CGImage + CGContext

```
let cgImage = image.cgImage!
let context = CGContext(
    data: nil, //nil - for automatic memory allocation
    width: cgImage.width, // you can upscale/downscale the image
    height: cgImage.height,
    bitsPerComponent: cgImage.bitsPerComponent,
    bytesPerRow: 0, //0 - for automatic, otherwise 4 * cgImage.width for RGBA
    space: cgImage.colorSpace!,
    bitmapInfo: cgImage.bitmapInfo.rawValue
)!

context.draw(cgImage, in: .init(origin: .zero, size: CGSize(width: cgImage.width, height:
cgImage.height)))
// context.clip(to: , mask: )
let redrawnCopy = context.makeImage()!
```

CIImage

Core Image

- Инструмент для высоко-производительной обработки изображений и видео.
- Основной кирпич - **CIFilter**
- Может объединять или менять порядок CIFilter под капотом в целях оптимизации

CIFilter

- **CIFilter::outputImage** – Возвращает объект CIImage, содержащий внутри граф операций
- Применение фильтров последовательно не влечет за собой запуск вычислений
- **CIContext::createCGImage(_:from:)** (или другой способ) - заставляет CoreImage запустить весь граф обработки

CIFilter chain

```
let ciImage = CIImage(image: image)
let EV: Float = 1.0
let sharpness: Float = 1.0

let sharpenFilter = CIFilter(name: "CISharpenLuminance")!
sharpenFilter.setValue(ciImage, forKey: kCIInputImageKey)
sharpenFilter.setValue(sharpness, forKey: kCIInputSharpnessKey)

let exposureFilter = CIFilter.exposureAdjustFilter()
exposureFilter.inputImage = sharpenFilter.outputImage
exposureFilter.EV = EV


let context =  CIContext(mtlDevice: MTLCreateSystemDefaultDevice()!)
let outputBitmap = context.createCGImage(
   exposureFilter.outputImage,
   from: ciImage.extent
)
```

CIColorFilter на ~~стероидах~~ шейдерах

- Если не хватает готовых фильтров - можно реализовать свой на Metal
- Metal код должен быть в файле *.metal
- Обязательно должен иметь
  - `#include <CoreImage/CoreImage.h>`
  - `extern "C" { namespace coreimage {`
- Флаг линковки `-cikernel` и флаг компилятора `-fcikernel`

CIColorFilter на ~~стероидах~~ шейдерах

```
extern "C" {
    namespace coreimage {
        vector_float4 multiply(sample_t left, sample_t right) {
            return vector_float4(
              left.r * right.r,
              left.g * right.g,
              left.b * right.b,
              left.a
            );
        }
    }
}
```

CIColorFilter на ~~стероидах~~ шейдерах

```
public final class MultiplyOperationFilter: CIFilter {
    public var leftImage: CIImage!
    public var rightImage: CIImage!
    private let kernel: CIColorKernel

    public init() {
        let url = Bundle(for: Self.self).url(forResource: "default", withExtension: "metallib")!
        let data = try! Data(contentsOf: url)
        kernel = try! CIColorKernel(functionName: "multiply", fromMetalLibraryData: data)
        super.init()
    }

    @available(*, unavailable)
    required init?(coder: NSCoder) {
        preconditionFailure("init(coder:) has not been implemented")
    }

    public override var outputImage: CIImage? {
        return kernel.apply(
            extent: leftImage.extent,
            arguments: [leftImage, rightImage]
         )
    }
}
```

Выводы

- Рассмотрели отличие цветового пространства от формата изображения
- Изображение в памяти - одномерный массив. Цветовое пространство определяет структуру массива
- RGB <-> CMYK <-> YCbCr
- UIImage - контейнер для CGImage, CIImage
- Пример как сделать полную копию изображения в памяти
- Как эффективно делать множественную обработку изображений готовыми инструментами и с помощью кода на шейдерах

Дополнительный материал

- Форматы изображений
- Каталог CIFilter
- Overview of Quartz 2D
- Ben Sandofsky - Building a Realtime Video Processor with Swift and Metal
- Swift Accelerate and vImage
- Андрей Володин - Байтик к байтику, или Как выжать из телефона всё и не расплавить его

Спасибо!

https://www.vectornator.io/

romanvolkov