БОРИС ОХАЙОН(BORIS OHAYON)

iOS — Извлечение кадров камеры

УЧЕБНОЕ РУКОВОДСТВО

Перевод: С. Кузнецов, 2022 г.

iOS — Camera Frames Extraction

Boris Ohayon

Jan 3, 2017 · 14 min

https://medium.com/ios-os-x-development/ios-camera-frames-extraction-d2c0f80ed05a

iOS — Извлечение кадров камеры

Борис Охайон

Янв 3, 2017 · 14 мин

https://medium.com/ios-os-x-development/ios-camera-frames-extraction-d2c0f80ed05a

Перевод: С. Кузнецов, 2022 г.

iOS — Извлечение кадров камеры

Сегодня мы изучим **получение доступа к каждому кадру канала камеры(access each frame of the camera feed)**, осваивая некоторые части фреймворка **AVFoundation Framework**.

Хотите сделать некоторую **обработку изображений(image processing)**? А **Машинное зрение(Computer vision)**? Тогда доступ к каждому кадру является первым шагом! Здесь мы вместе собираемся построить простой класс, уведомляющий **вызывающую сторону(caller)** каждый раз, когда доступен кадр. Затем вызывающая сторона свободна сделать все что угодно, что она пожелает, с данными изображениями, в режиме реального времени.

Без дальнейшей суматохи давайте перейдем прямо к делу!

TL; DR? Это - то, что мы создаем в этом учебном руководстве!

```
import UIKit
import AVFoundation


protocol FrameExtractorDelegate: class {
    func captured(image: UIImage)
}
```

```
    class FrameExtractor: NSObject,
AVCaptureVideoDataOutputSampleBufferDelegate {

        private let position = AVCaptureDevicePosition.front

        private let quality = AVCaptureSessionPresetMedium

        private var permissionGranted = false

        private let sessionQueue = DispatchQueue(label:
"session queue")

        private let captureSession = AVCaptureSession()

        private let context = CIContext()

        weak var delegate: FrameExtractorDelegate?

        override init() {

            super.init()

            checkPermission()

            sessionQueue.async { [unowned self] in
                self.configureSession()

                self.captureSession.startRunning()

            }

        }

        // MARK: AVSession configuration

    private func checkPermission() {

        switch AVCaptureDevice.authorizationStatus(forMediaType:
AVMediaTypeVideo) {

        case .authorized:

            permissionGranted = true

        case .notDetermined:

            requestPermission()

        default:

            permissionGranted = false

    }

}
```

```
    private func requestPermission() {
        sessionQueue.suspend()
        AVCaptureDevice.requestAccess(forMediaType:
AVMediaTypeVideo) { [unowned self] granted in
            self.permissionGranted = granted
            self.sessionQueue.resume()
        }
    }

    private func configureSession() {
        guard permissionGranted else { return }
        captureSession.sessionPreset = quality
        guard let captureDevice = selectCaptureDevice() else
{ return }
        guard let captureDeviceInput = try?
AVCaptureDeviceInput(device: captureDevice) else { return }
        guard captureSession.canAddInput(captureDeviceInput) else
{ return }
        captureSession.addInput(captureDeviceInput)
        let videoOutput = AVCaptureVideoDataOutput()
        videoOutput.setSampleBufferDelegate(self, queue:
DispatchQueue(label: "sample buffer"))
        guard captureSession.canAddOutput(videoOutput) else
{ return }
        captureSession.addOutput(videoOutput)
        guard let connection =
videoOutput.connection(withMediaType:
AVFoundation.AVMediaTypeVideo) else { return }
        guard connection.isVideoOrientationSupported else
{ return }
        guard connection.isVideoMirroringSupported else
{ return }
        connection.videoOrientation = .portrait
        connection.isVideoMirrored = position == .front
    }
```

```
    private func selectCaptureDevice() -> AVCaptureDevice? {
        return AVCaptureDevice.devices().filter {
            ($0 as AnyObject).hasMediaType(AVMediaTypeVideo) &&
            ($0 as AnyObject).position == position
        }.first as? AVCaptureDevice
    }

    // MARK: Sample buffer to UIImage conversion
    private func imageFromSampleBuffer(sampleBuffer:
CMSampleBuffer) -> UIImage? {
        guard let imageBuffer =
CMSampleBufferGetImageBuffer(sampleBuffer) else { return nil }
        let ciImage = CIImage(cvPixelBuffer: imageBuffer)
        guard let cgImage = context.createCGImage(ciImage, from:
ciImage.extent) else { return nil }
        return UIImage(cgImage: cgImage)
    }

    // MARK: AVCaptureVideoDataOutputSampleBufferDelegate
    func captureOutput(_ captureOutput: AVCaptureOutput!,
didOutputSampleBuffer sampleBuffer: CMSampleBuffer!, from
connection: AVCaptureConnection!) {
        guard let uiImage = imageFromSampleBuffer(sampleBuffer:
sampleBuffer) else { return }
        DispatchQueue.main.async { [unowned self] in
            self.delegate?.captured(image: uiImage)
        }
    }
}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Шаг 1 — Создайте новый iOS-проект

Создайте новый проект приложения по шаблону **Single View Application (Приложение с одним видом)**, здесь нет ничего нового. Дайте ему имя на свой выбор, **FrameExtraction**, например. Выберите язык программирования **Swift**, и все сделано.

Шаг 2 — Наш фреймворк: AVFoundation

В этом демонстрационном проекте мы будем использовать **раскадровку(storyboard)** и уже созданный файл **ViewController.swift** для интерфейса пользователя **UI**-стандарта. Сердце нашей программы будет находиться в новом **Swift**-файле с именем извлекателя кадров **FrameExtractor.swift**.

2.1 Создайте этот файл

Первой и единственной строкой, созданной **Apple**, является строка импорта **import Foundation**. Мы хотим использовать фреймворк **AVFoundation**, чтобы иметь возможность использовать камеру и извлечь кадры. Замените существующую строку следующим.

2.2 Существующую строку замените следующим кодом

```
import AVFoundation
```

исходник FrameExtractor.swift находится на сервисе GitHub

2.3 Создайте героя дня - класс извлекателя кадров FrameExtractor

```
class FrameExtractor {


}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Давайте кратко опишем предполагаемые операции нашего алгоритма.

1. Операция должна получить доступ к камере
2. Операция должна быть настраиваема(**фронтальная камера/тыльная камера(front/back camera)**, **ориентация(orientation)**, **качество(quality)** …)
3. Операция должна возвращать **каждый захваченный кадр(every frame captured)**

Шаг 3 — Представление класса сеанса захвата кадра AVCaptureSession

Внутри фреймворка **AVFoundation** нашим лучшим другом является класс сеанса захвата кадра **AVCaptureSession**. Сеанс координирует **Xow**-операцию данных от **ввода(input)** к **выводу(output)**.

3.1 Создайте сильную ссылку(strong reference) на сеанс захвата кадра в виде атрибута

```
private let captureSession = AVCaptureSession()
```

исходник FrameExtractor.swift находится на сервисе GitHub

Некоторые вещи, которые мы собираемся сделать с сеансом, должны выполняться **асинхронно(asynchronously)**. Поскольку мы не хотим блокировать **главный поток(main thread)**, мы должны создать **последовательную очередь(serial queue)**, которая выполнит работу, связанную с сеансом. Чтобы создать последовательную очередь, давайте использовать инициализатор очереди отправки **DispatchQueue** и назовем ее **очередью сеанса(session queue)**. В начале класса извлекателя кадров **FrameExtractor** мы добавим ссылку на эту очередь, в качестве атрибута для того, чтобы мы могли получить доступ к ней позже и приостановить или возобновить ее, когда будет необходимость.

```
        private let sessionQueue = DispatchQueue(label: "session
    queue")
```

[исходник](#) [FrameExtractor.swift](#) находится на сервисе [GitHub](#)

Шаг 4 — Разрешите снять?

Чтобы получить доступ к камере, приложению требуется у пользователя запросить **разрешение(permission)**. В фреймворке **AVFoundation** мы можем найти класс с именем устройства захвата кадра **AVCaptureDevice**, содержащий свойства, имеющие отношение к используемому устройству, лежащему слоем ниже в программном стеке. Этот класс, через функцию авторизации статуса **authorizationStatus()**, также помнит, авторизовал ли пользователь ранее использование устройства захвата кадра. Эта функция возвращает константу **перечисления(enum)** с именем статуса авторизации **AVAuthorizationStatus**, которое может содержать несколько значений.

```
    switch AVCaptureDevice.authorizationStatus(forMediaType:
AVMediaTypeVideo) {

        case .authorized:

            // Пользователь явно предоставил разрешение
            // на получение медиа
            // The user has explicitly granted permission
            // for media capture

            break


        case .notDetermined:

            // Пользователь еще не предоставил или
            // отклонил разрешение
            // The user has not yet granted or denied permission

            break


        case .restricted:

            // Пользователю не разрешают получить доступ
            // к устройствам получения медиа
            // The user is not allowed to access media
            // capture devices

            break


        case .denied:

            // Пользователь явно отклонил разрешение
            // на получение медиа
            // The user has explicitly denied permission
            // for media capture

            break

    }
```

[исходник](#) [FrameExtractor.swift](#) находится на сервисе [GitHub](#)

Нас интересуют случаи `.authorized` и `.notDetermined`.

1. Если пользователь уже `предоставил разрешение для получения медиа(granted permission for the media capture)`, то мы ничего не должны делать

2. Если `разрешение ограничено или отклонено(permission is restricted or denied)`, пользователь ранее отказался от получения медиа, мы решаем не запрашивать его снова
3. Если пользователь `еще не предоставил или отклонил разрешение(not yet granted or denied permission)`, мы подскажем ему

После совершения выбора пользователь может пойти каждый раз туда, куда он хочет в параметрах настройки телефона, и измените разрешение приложения.

4.1 Объявите переменную класса для отслеживания, если разрешение предоставлено

```
private var permissionGranted = false
```

исходник FrameExtractor.swift находится на сервисе GitHub

Прежде, чем проверить статус авторизации `authorizationStatus`, мы должны поместить наш код где-нибудь! Мы хотим проверить это, как только создан объект экстрактор кадра `FrameExtractor`.

4.2 Создайте метод инициализации init класса, пока без параметра

```
class FrameExtractor {

    private var permissionGranted = false

    init() {

    }

}
```

исходник FrameExtractor.swift находится на сервисе GitHub

4.3 Заполните предыдущий блок случаями(вариантами case), интересующими нас

```
    switch AVCaptureDevice.authorizationStatus(forMediaType:
AVMediaTypeVideo) {

        case .authorized:

        // Пользователь явно предоставил разрешение
        // для захвата медиа
        // The user has explicitly granted
        // permission for media capture

        permissionGranted = true

            break


        case .notDetermined:

        // Пользователь еще не предоставил или
        // отклонил разрешение
        // The user has not yet granted or denied permission

            break

        default:

        // Пользователь отклонил разрешение
        // The user has denied permission

        permissionGranted = false

            break

    }
```

исходник FrameExtractor.swift находится на сервисе GitHub

В классе устройства захвата кадра `AVCaptureDevice` также есть метод запроса доступа `requestAccess`, предлагающий пользователю разрешение и берущий в качестве параметра `обработчик комплектного завершения(completion handler)`, вызываемый в ситуации, как только пользователь принял решение предоставить или отклонить разрешение

4.4 Комплектное завершение блока запроса разрешения

```
    switch AVCaptureDevice. authorizationStatus(forMediaType:
AVMediaTypeVideo) {

        case .authorized:

         // Пользователь явно предоставил разрешение
        // для захвата медиа
        // The user has explicitly granted
        // permission for media capture

         permissionGranted = true

            break


        case .notDetermined:

         // Пользователь еще не предоставил или
        // отклонил разрешение
        // The user has not yet granted or
        // denied permission

         AVCaptureDevice.requestAccess(forMediaType:
AVMediaTypeVideo) { granted in

                self.permissionGranted = granted

            }

            break

        default:

         // Пользователь отклонил разрешение
        // The user has denied permission

         permissionGranted = false

            break

    }
```

[исходник](#) [FrameExtractor.swift](#) находится на сервисе [GitHub](#)

Здесь, мы должны не упустить для **сохраняемых циклов(retain cycles)**, поскольку мы обращаемся по указателю **self** в **обработчике комплектного завершения(completion handler)**, следующее: внутри блока объявите указатель **self** в виде **ссылки без владельца(unowned reference)**. В этом коде особенно, нет никакой фактической необходимости добавлять указатель **self** в виде **ссылки без владельца(unowned reference)**, потому что даже при том, что **закрытие(closure)** сохраняет указатель **self**, но указатель **self** нигде не сохраняет **закрытие(closure)** и как только мы выходим из **закрытия(closure)**, то она выпустила бы, сохраненный указатель **self**. Но

потому что мы могли бы в будущем добавлять сущности и сохранять **закрытие(closure)**, то это может быть способом не забыть хорошую сущность.

Мы могли бы также задаться вопросом почему **ссылка без владельца(unowned reference)**, а не **слабая ссылка(weak reference)**? Всегда рекомендуется использовать **слабую ссылку(weak reference)**, когда неясно о том, что указатель **self** «переживет» **закрытие(closure)**. Здесь, кажется, что мы можем быть вполне уверены, что **закрытие(closure)** не переживет указатель **self** и поэтому нам не требуется объявить его в виде **слабой ссылки(weak reference)** (и тащить указатель **self**, который теперь был бы опционным). Почитайте в сети некоторые статьи о **ссылке без владельца(unowned reference)** и о **слабой ссылке(weak reference)** и там найдете некоторые качественные объяснения.

4.5 Объявите указатель self в виде ссылки без владельца (unowned reference)

```
    switch AVCaptureDevice.authorizationStatus(forMediaType:
AVMediaTypeVideo) {

        case .authorized:

        // Пользователь явно предоставил разрешение
        // для получения медиа
        // The user has explicitly granted permission
        // for media capture

        permissionGranted = true

            break


        case .notDetermined:

        // Пользователь еще не предоставил или
        // отклонил разрешение
        // The user has not yet granted or denied permission

        AVCaptureDevice.requestAccess(forMediaType:
AVMediaTypeVideo) { [unowned self] granted in
            self.permissionGranted = granted
        }

        break


    default:

    // Пользователь отклонил разрешение
    // The user has denied permission

    permissionGranted = false

        break

}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Если когда-нибудь мы заканчиваем в случае неопределено `.notDetermined`, потому что вызов запроса доступа `requestAccess` является синхронным(на произвольной очереди отправки), то мы должны временно приостановить очередь сеанса и возобновить ее, как только мы получаем результат от пользователя.

4. 6 Приостановите и возобновите очередь

```
        switch AVCaptureDevice.authorizationStatus(forMediaType:
AVMediaTypeVideo) {

            case .authorized:

            // Пользователь явно предоставил разрешение
            // для получения носителей
            // The user has explicitly granted permission
            // for media capture

            permissionGranted = true

                break


            case .notDetermined:

            // Пользователь еще не предоставил или
            // отклонил разрешение
            // The user has not yet granted or denied permission

            sessionQueue.suspend()

            AVCaptureDevice.requestAccess(forMediaType:
AVMediaTypeVideo) { [unowned self] granted in

                self.permissionGranted = granted

                self.sessionQueue.resume()

            }

            break


        default:

        // Пользователь отклонил разрешение
        // The user has denied permission

        permissionGranted = false

            break

    }
```

исходник FrameExtractor.swift находится на сервисе GitHub

Как только мы выходим из механизма запроса разрешения, мы должны ранее созданному сеансу получения позволить знать о том, что он может продолжать **процесс конфигурации(configuration process)**.

Когда входим в ветвь кода неопределенного **.notDetermined**, то очередь сеанса **временно приостановлена(suspended)** и она **возобновлена(resumes)**, как только мы получаем ответ от пользователя. Следовательно, когда **поток выполнения(execution flow)** выходит из переключателя **switch**, очередь сеанса могла бы все еще быть временно приостановлена. Чтобы продолжать конфигурировать сеанс только, когда у нас есть допустимое решение от пользователя, то мы можем поместить работу внутри очереди сеанса **асинхронным способом(async way)**. Как только **очередь сеанса возобновляется(session queue resumes)**, она сделает работу внутри этого блока. Снова, поскольку мы используем указатель **self** в **обработчике завершения(completion handler)**, мы не должны забывать использовать указатель **self** в виде **ссылки без владельца(unowned reference)**.

4.7 Добавьте асинхронную конфигурацию на очереди сеанса(session queue)

```
        init() {


            switch AVCaptureDevice.authorizationStatus(forMediaType:
        AVMediaTypeVideo) {

                case .authorized:

                // Пользователь ранее предоставил доступ
                // The user has previously granted access

                permissionGranted = true


                case .notDetermined:

                // Пользователь еще не предоставил с
                // The user has not yet been presented with

                sessionQueue.suspend()

                AVCaptureDevice.requestAccess(forMediaType:
        AVMediaTypeVideo) { [unowned self] granted in
                        self.permissionGranted = granted

                        self.sessionQueue.resume()

                    }


                default:

                // Пользователь отклонил разрешение
                // The user has denied permission

                permissionGranted = false

        }


        sessionQueue.async { [unowned self] in

        self.configureSession()

                }

        }

        private func configureSession() {

        }
```

[исходник](#) [FrameExtractor.swift](#) находится на сервисе [GitHub](#)

4.8 Очистите инициализатор и создайте специфичные методы.

```
        init() {
            checkPermission()
            sessionQueue.async { [unowned self] in
                self.configureSession()
                self.captureSession.startRunning()
            }
        }

        // MARK: AVSession configuration
        private func checkPermission() {
            switch AVCaptureDevice.authorizationStatus(forMediaType:
AVMediaTypeVideo) {
            case .authorized:
                permissionGranted = true
            case .notDetermined:
                requestPermission()
            default:
                permissionGranted = false
            }
        }

    private func requestPermission() {
        sessionQueue.suspend()
        AVCaptureDevice.requestAccess(forMediaType: AVMediaTypeVideo)
    { [unowned self] granted in
            self.permissionGranted = granted
            self.sessionQueue.resume()
        }
    }
```

исходник FrameExtractor.swift находится на сервисе GitHub

Нужно объяснить одну последнее замечание, связанное с разрешением. Начиная с ОС `iOS 10`, ваше приложение потерпит крах, если в `plist`-файле информации приложения вы не зададите строку, описывающую использование запрашиваемого разрешения. Ошибка будет похожа на следующий текст `The app's Info.plist must contain an NSCameraUsageDescription key(Файл информации приложения Info.plist должен содержать ключ описания использования NS-камеры NSCameraUsageDescription)`.

Направьтесь в файл информации приложения `Info.plist` и добавьте ключ описания использования `NS`-камеры `NSCameraUsageDescription` со значением `Used to capture frames(Использована для захвата кадров)` или другим желаемым вами значением, соответствующим цели приложения.

Шаг 5 — Уникально настройте меня!

Нам теперь разрешают записать. Давайте выберем, на какую камеру мы хотим записать, желаемое качество изображения и т.д. …

На данный момент, скажем, мы используем `фронтальную камеру(переднюю камеру; front camera)` устройства и значение качества изображения установлено в `среднее(medium)`. Выбранное качество изображения должно быть компромиссом между красотой изображения и затратами на вычисления, которые будут необходимы после экстракции изображения, как только мы хотим проанализировать его. Извлечение изображения не является самоцелью, если вы хотите сделать анализ изображения каждого кадра, например, то выбор высокого качества изображения не советуем.

В фреймворке `AVFoundation` мы можем найти искомые атрибуты, чтобы уникально настроить фотографирование.

5.1 Добавьте ссылки на те атрибуты

```
class FrameExtractor {


    private let position = AVCaptureDevicePosition.front
    private let quality = AVCaptureSessionPresetMedium


}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Давайте помещать всю конфигурацию сеанса в метод конфигурирования сеанса `configureSession()`. После проверки, если пользователь предоставил разрешение, то мы установим сеанс.

5.2 Создайте функцию конфигурирования сеанса configurationSession()

```
private func configureSession() {
    guard permissionGranted else { return }
    captureSession.sessionPreset = quality
}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Шаг 6 — Ввод

Чтобы выбрать желаемое используемое записывающее устройство, мы должны установить `AV`-устройство съемки(захвата) `AVCaptureDevice`. Чтобы получить все доступные устройства съемки(захвата), мы можем выполнить следующее.

```
AVCaptureDevice.devices()
```

исходник FrameExtractor.swift находится на сервисе GitHub

Используя функцию фильтра, мы можем попытаться выбрать только предназначенное устройство съемки(захвата). Для каждого устройства, в массиве устройств, проверьте:

1. Если это - **устройство видеозаписи(recording device)**
2. Если это - **фронтальная камера(передняя камера; front camera)**

```
AVCaptureDevice.devices().filter {
    ($0 as AnyObject).hasMediaType(AVMediaTypeVideo) &&
    ($0 as AnyObject).position == position
}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Первое устройство в списке результатов, если оно существует и доступно, является искомым устройством.

6.1 Продолжите функцию конфигурирования сеанса configurationSession() со вспомогательным методом

```
    private func configureSession() {
        guard permissionGranted else { return }
        captureSession.sessionPreset = quality
        guard let captureDevice = selectCaptureDevice() else
{ return }
    }

    private func selectCaptureDevice() -> AVCaptureDevice? {
        return AVCaptureDevice.devices().filter {
            ($0 as AnyObject).hasMediaType(AVMediaTypeVideo) &&
            ($0 as AnyObject).position == position
        }.first as? AVCaptureDevice
    }
```

исходник FrameExtractor.swift находится на сервисе GitHub

Теперь, когда у нас есть допустимое устройство съемки (захвата), мы можем попытаться создать класс ввода `AV`-устройства съемки(захвата) `AVCaptureDeviceInput`. Это - класс, манипулирующий конкретным способом данными, фотографированными(захваченными) камерой. Стоит не упустить одну особенность того, что создание класса ввода `AV`-устройства съемки(захвата) `AVCaptureDeviceInput` с классом `AV`-устройства съемки(захвата) `AVCaptureDevice` может перестать работать, если устройство не может быть открыто: оно могло бы быть долго не доступно, или оно могло бы уже использоваться, например. Поскольку это может перестать работать, то мы обертываем его в операторы защиты `guard` и попытки `try?`. Не стесняйтесь обрабатывать ошибки, как пожелаете.

6.2 Создайте класс ввода AV-устройства съемки(захвата) AVCaptureDeviceInput

```
        private func configureSession() {

            guard permissionGranted else { return }

            captureSession.sessionPreset = quality

            guard let captureDevice = selectCaptureDevice() else
    { return }

            guard let captureDeviceInput = try?
    AVCaptureDeviceInput(device: captureDevice) else { return }

        }
```

исходник FrameExtractor.swift находится на сервисе GitHub

6.3 Проверьте, может ли ввод устройства съемки(захвата) быть добавлен к сеансу, и добавьте его

```
            guard captureSession.canAddInput(captureDeviceInput) else { return }

            captureSession.addInput(captureDeviceInput)
```

исходник FrameExtractor.swift находится на сервисе GitHub

Шаг 7 — Вывод

Теперь мы должны прервать каждый кадр(фрейм). Класс вывода **AV**-снятых(захваченных) видеоданных **AVCaptureVideoDataOutput** - класс, который мы собираемся использовать: он обрабатывает несжатые кадры из снятого(захваченного) видео. Сразу после добавления ввода устройства съемки(захвата) перейдем к сеансу.

7.1 Создайте экземпляр класса вывода снятых (захваченных) видеоданных AVCaptureVideoDataOutput

```
        guard captureSession.canAddInput(captureDeviceInput) else
    { return }

        captureSession.addInput(captureDeviceInput)

        let videoOutput = AVCaptureVideoDataOutput()
```

исходник FrameExtractor.swift находится на сервисе GitHub

Класс вывода **AV**-снятых(захваченных) видеоданных **AVCaptureVideoDataOutput** работает только при наличии объекта делегата, которому он может передать каждый кадр. Наш класс экстрактора кадра **FrameExtractor** может отлично быть этим делегатом и принять те кадры.

7.2 Измените объявление класса и подтвердите соответствие протоколу

```
    class FrameExtractor:
AVCaptureVideoDataOutputSampleBufferDelegate {



    // Все мы сделали Everything we did


    }
```

исходник FrameExtractor.swift находится на сервисе GitHub

Заметьте, что здесь появляется ошибка:

Тип экстрактора кадра `"FrameExtractor"` не подтвердил соответствие протоколу `"NSObjectProtocol"`.

Если мы поднимемся по цепочке наследования, то мы увидим, что протокол требует методов, которые тип экстрактора кадра `FrameExtractor` не имеет. Чтобы устранить ошибку, мы можем дать ему те методы, делая `FrameExtractor` наследуемым от `NSObject`.

7. 3 Сделайте FrameExtractor наследуемым от NSObject

```
class FrameExtractor: NSObject,
AVCaptureVideoDataOutputSampleBufferDelegate {


    // Мы все сделали / Everything we did


    }
```

исходник FrameExtractor.swift находится на сервисе GitHub

Поскольку у `NSObject` уже есть `init()`, то построенный нами метод инициализации `init` нужно `переопределить(override)` метод инициализации `init` у `NSObject`. Не забывайте вызывать реализацию у супер-объекта с указателем `super`.

7. 4 Переопределите(override) метод инициализации init

```
class FrameExtractor: NSObject,
AVCaptureVideoDataOutputSampleBufferDelegate {


    override init() {
        super.init()
    }


}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Теперь, когда класс экстрактора кадра **FrameExtractor** соответствует протоколу, мы просто должны специфицировать(задать), что делегатом видеовыхода является сам класс экстрактора кадра **FrameExtractor**.

7. 5 Установка класса экстрактора кадра FrameExtractor в качестве делегата

```
        guard captureSession.canAddInput(captureDeviceInput) else
    { return }

        captureSession.addInput(captureDeviceInput)

        let videoOutput = AVCaptureVideoDataOutput()

        videoOutput.setSampleBufferDelegate(self, queue:
    DispatchQueue(label: "sample buffer"))
```

исходник FrameExtractor.swift находится на сервисе GitHub

У этого протокола есть два опционных метода, один метод вызывается каждый раз, когда кадр **доступен(available)**, другой метод вызывается каждый каждый раз, когда кадр **отброшен(discarded)**. Устанавливая делегата, мы должны специфицировать(задать) **последовательную очередь(serial queue)**, которая обработает съемку(захват) кадров. Два предыдущих метода вызывают **на этой последовательной очереди(on this serial queue)**, и каждая обработка кадра **должна быть сделана на этой очереди(must be done on this queue)**.

Иногда, обработка кадра может потребовать затрат времени и большой вычислительной мощности, и следующий кадр может быть снят(захвачен), в то время как текущий кадр еще не был полностью обработан. Если это происходит, следующий снятый(захваченный) кадр должен быть отброшен!

Если мы должны были каждый доступный кадр передать другой очереди, и обработать их все, то мы могли закончить в ситуации, где кадры навалены грудой и груда все увеличивается. Кадры прибывают быстрее, чем мы можем рассматривать их, и мы должны были бы обработать сами управление памятью, которое они инициируют!

Метод, интересующий нас, является методом, вызываемым каждый раз, когда новый кадр **доступен(available)**:

```
    func captureOutput(_ captureOutput: AVCaptureOutput!,
didOutputSampleBuffer sampleBuffer: CMSampleBuffer!, from
connection: AVCaptureConnection!) {


    }
```

исходник FrameExtractor.swift находится на сервисе GitHub

На данный момент, давайте просто добавим простой оператор печати каждый раз, когда мы приняли кадр.

7.6 Добавьте метод делегата с простым оператором печати

```
    func captureOutput(_ captureOutput: AVCaptureOutput!,
didOutputSampleBuffer sampleBuffer: CMSampleBuffer!, from
connection: AVCaptureConnection!) {

        print("Получил кадр! / Got a frame!")

    }
```

исходник FrameExtractor.swift находится на сервисе GitHub

Давайте завершим метод конфигурирования сеанса `configureSession()`, добавив наш видеовыход к сеансу:

```
    private func configureSession() {

        guard permissionGranted else { return }

        captureSession.sessionPreset = quality

        guard let captureDevice = selectCaptureDevice() else
{ return }

        guard let captureDeviceInput = try?
AVCaptureDeviceInput(device: captureDevice) else { return }

        guard captureSession.canAddInput(captureDeviceInput) else
{ return }

        captureSession.addInput(captureDeviceInput)

        let videoOutput = AVCaptureVideoDataOutput()

        videoOutput.setSampleBufferDelegate(self, queue:
DispatchQueue(label: "sample buffer"))

        guard captureSession.canAddOutput(videoOutput) else
{ return }

        captureSession.addOutput(videoOutput)
    }
```

исходник FrameExtractor.swift находится на сервисе GitHub

Шаг 8 — Пауза и тест

Давайте сделаем перерыв и протестируем наш код!

Перед выполнением проекта осталось сделать две последних вещи. Сначала мы должны запустить сеанс съемки(захвата) и не забыть, что сеанс съемки(захвата) должен быть запущен на ранее созданной нами специальной `последовательной очереди(serial queue)`, поскольку запуск сеанса является `вызовом блокировки(blocking call)`, а мы не хотим блокировать `UI`-интерфейс пользователя.

Помните, что здесь у нас в действии есть две очереди. Первая очередь - очередь сеанса, вторая очередь - очередь, в которую передан каждый кадр. Они отличаются.

8.1 Завершите блок очереди сеанса(session queue block), чтобы запустить сеанс

```
sessionQueue.async { [unowned self] in
    self.configureSession()
    self.captureSession.startRunning()
}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Наконец, в изначальном файле контроллера вида `ViewController.swift`, созданном `IDE`-средой разработки `Xcode`, давайте просто создадим ссылку на `экстрактор кадра(frame extractor)`, чтобы строго сохранить ее.

8.2 Создайте ссылку на экстрактор кадра(frame extractor)

```
import UIKit

class ViewController: UIViewController {

    var frameExtractor: FrameExtractor!

    override func viewDidLoad() {
        super.viewDidLoad()
        frameExtractor = FrameExtractor()
    }


}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Здесь, мы должны отметить, что `экстрактор кадра(frame extractor)` объявлен за пределами вызова метода сделанной загрузки

вида(представления) `viewDidLoad`, иначе, создаваемый объект не был бы `сохранен(retained)`, и ничто не произойдет. Хуже того, при вызове `закрытий(closures)` класса экстрактора(извлекателя) кадра `FrameExtractor` была бы брошена ошибка `swift_abortRetainUnowned`, где мы объявили [указатель `self` в виде `ссылки без владельца(unowned reference); unowned self`], приводя к краху приложения.

Теперь вы можете выполнить проект, и фраза `Получил кадр! / Got a frame!` должна распечатана снова и снова в консоли. Выражаем триумф, `Yay!`

Шаг 9 — О Изображение, где нарисовано ты?

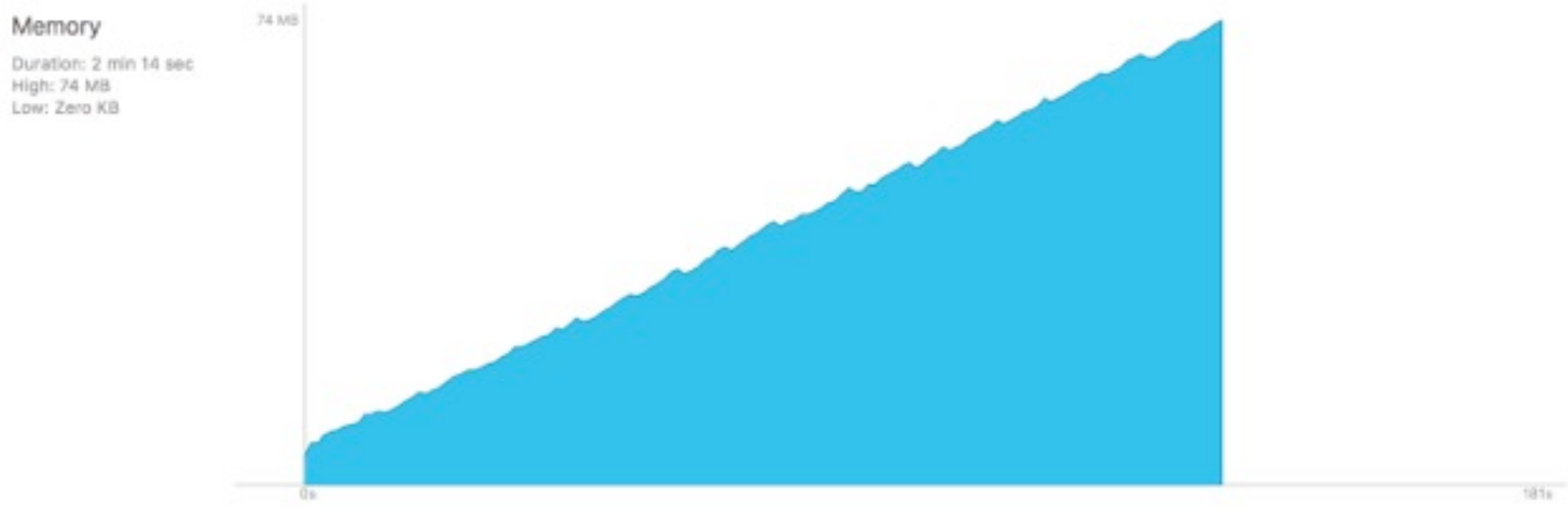
Теперь, когда мы можем снять(захватить) каждый кадр, давайте попытаемся преобразовать их в фактические изображения и отобразить их в виде(представлении) изображения.

Давайте вернемся к методу вывода `captureOutput(_: didOutputSampleBuffer:from)`. Помните, что этот метод снимает(захватывает) каждый кадр. `Буфер съемки/захвата(captured buffer)`, содержащий информацию о кадре, задан в виде параметра функции, именуемой буфером выборки `sampleBuffer`.

Первый алгоритм, который мы могли реализовать для преобразования `буфера выборки(sample buffer)` в фактический класс `UI`-изображения интерфейса пользователя `UIImage`, мог быть таким:

1. Данные в `буфере выборки(sample buffer)` преобразуйте в данные класса буфера `CV`-изображения `CVImageBuffer`
2. Данные класса буфера `CV`-изображения `CVImageBuffer` преобразуйте в данные класса `CI`-изображения `CIImage`
3. Наконец данные класса `CI`-изображения `CIImage` преобразуйте в данные класса `UI`-изображения интерфейса пользователя `UIImage`

Проблема с этим методом состоит в том, что для определенной версии ОС **iOS**, инициализатор класса **UI**-изображения интерфейса пользователя **UIImage**, при преобразовании из данных класса **CI**-изображения **CIImage**, дает **утечку памяти(memory leak)**:



Использованная память, кажется, не освобождена и поэтому она накапливается, и свободная память заканчивается в ОС, в конечном счете уничтожающей процесс нашего приложения.

Второй алгоритм, который мы можем сделать, начинается как предыдущий алгоритм:

1. Данные в **буфере выборки(sample buffer)** преобразуйте в данные класса буфера **CV**-изображения **CVImageBuffer**
2. Данные класса буфера **CV**-изображения **CVImageBuffer** преобразуйте в данные класса **CI**-изображения **CIImage**
3. Данные класса **CI**-изображения **CIImage** преобразуйте в данные класса **CG**-изображения **CGImage**
4. Наконец данные класса **CG**-изображения **CGImage** преобразуйте в данные класса **UI**-изображения интерфейса пользователя **UIImage**

1 шаг алгоритма.

Ниже метода выбора устройства съемки(захвата) **selectCaptureDevice()**, создайте функцию, которая в качестве параметра принимает данные в

буфере выборки(sample buffer) и возвращает, если все проходит хорошо, то возвращает данные класса **UI**-изображения интерфейса пользователя **UIImage**.

9.1 Добавьте следующий метод-скелетон

```
private func imageFromSampleBuffer(sampleBuffer: CMSampleBuffer) ->
UIImage? {


}
```

исходник FrameExtractor.swift находится на сервисе GitHub

9.2 Данные в буфере выборки(sample buffer) преобразуйте в данные класса буфера CV-изображения CVImageBuffer

```
private func imageFromSampleBuffer(sampleBuffer: CMSampleBuffer) ->
UIImage? {

    guard let imageBuffer =
    CMSampleBufferGetImageBuffer(sampleBuffer) else { return nil }

}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Поскольку функция может перестать работать, то мы ее оборачиваем в оператор защиты **guard**.

2 шаг алгоритма.

Затем, поскольку мы используем класс **CI**-изображения **CIImage**, не забывайте импортировать набор **UIKit**.

9.3 Данные класса буфера CV-изображения CVImageBuffer преобразуйте в данные класса CI-изображения CIImage

```
private func imageFromSampleBuffer(sampleBuffer: CMSampleBuffer) ->
UIImage? {

    guard let imageBuffer =
    CMSampleBufferGetImageBuffer(sampleBuffer) else { return nil }

    let ciImage = CIImage(cvPixelBuffer: imageBuffer)

}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Поскольку функция может перестать работать, то мы ее оборачиваем в оператор защиты **guard**.

3 шаг алгоритма.

Этот шаг крайне важен, и суть его в том, что не соглашаемся с предложенной реализацией шага первого алгоритма.

9.4 Создайте CIContext и создайте данные класса CG-изображения CGImage из этого контекста

```
private func imageFromSampleBuffer(sampleBuffer: CMSampleBuffer) ->
UIImage? {

    guard let imageBuffer =
    CMSampleBufferGetImageBuffer(sampleBuffer) else { return nil }

    let ciImage = CIImage(cvPixelBuffer: imageBuffer)

    let context = CIContext()

    guard let cgImage = context.createCGImage(ciImage, from:
    ciImage.extent) else { return nil }

}
```

исходник FrameExtractor.swift находится на сервисе GitHub

4 шаг алгоритма.

Наконец, мы можем из данных класса **CG**-изображения **CGImage** создать и возвратить лежание в основе данные класса **UI**-изображения интерфейса пользователя **UIImage**.

9.5 Из данных класса CG-изображения CGImage создайте и возвратите данные класса UI-изображения интерфейса пользователя UIImage

```
private func imageFromSampleBuffer(sampleBuffer: CMSampleBuffer) ->
UIImage? {

    guard let imageBuffer =
    CMSampleBufferGetImageBuffer(sampleBuffer) else { return nil }

    let ciImage = CIImage(cvPixelBuffer: imageBuffer)

    let context = CIContext()

    guard let cgImage = context.createCGImage(ciImage, from:
    ciImage.extent) else { return nil }

    return UIImage(cgImage: cgImage)
}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Я знаю, что процесс затянулся, но потерпите еще, мы почти все сделали!

Шаг 10 — Вот изображение!

Теперь у нас есть фактический кадр класса **UI**-изображения интерфейса пользователя **UIImage**! Это - что-то, о котором мы знаем, как работать с ним! Последний шаг должен позволить **вызывающей стороне(caller)** знать, что изображение **доступно(available)**. Для этого мы можем создать протокол только с одной функцией снятого(захваченного) изображения **captured(image)**, которую вызывают каждый раз, когда кадр класса **UI**-изображения интерфейса пользователя **UIImage** доступен. Затем

пользователь свободен сделать с ним все, что он захочет. Единственная вещь, которую должен сделать класс вызова, состоит в объявлении в качестве делегата протокола, в который класс экстрактора кадра **FrameExtractor** отправит каждый кадр класса **UI**-изображения интерфейса пользователя **UIImage**.

10.1 Наверху текста в файле класса экстрактора кадра FrameExtractor.swift создайте протокол

```
protocol FrameExtractorDelegate {
    func captured(image: UIImage)
}
```

исходник FrameExtractor.swift находится на сервисе GitHub

10.2 Внутри класса экстрактора кадра FrameExtractor удерживайте делегат(delegate) с атрибутом слабой ссылки(weak attribute)

```
weak var delegate: FrameExtractorDelegate?
```

исходник FrameExtractor.swift находится на сервисе GitHub

Не забывайте, что, чтобы избежать ошибки **сохранения цикла(retain cycle)**, делегат должен быть объявлен с **атрибутом слабой ссылки(weak attribute)**. Поскольку он объявлено **слабым(weak)**, то мы к объявлению протокола добавляем ключевое слово класса **class**.

10.3 Измените объявление протокола

```
protocol FrameExtractorDelegate: class {
    func captured(image: UIImage)
}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Вернемся к методу вывода **captureOutput(_: didOutputSampleBuffer:from)**, мы можем теперь отправить делегату доступный кадр класса **UI**-изображения интерфейса пользователя **UIImage**. Помните, что этот метод вызывают на **последовательной очереди(serial queue)**. Мы не хотим **вызывающую сторону(caller)** напрягать на контакт с последовательной очередью и поэтому мы ему отправим изображение на **главном потоке(main thread)** и **вызывающая сторона(caller)** сможет сразу же обновить **UI**-интерфейс пользователя.

10.4 Диспетчеризируйте главную очередь(main queue), созданным кадром класса изображения UI-интерфейса пользователя UIImage

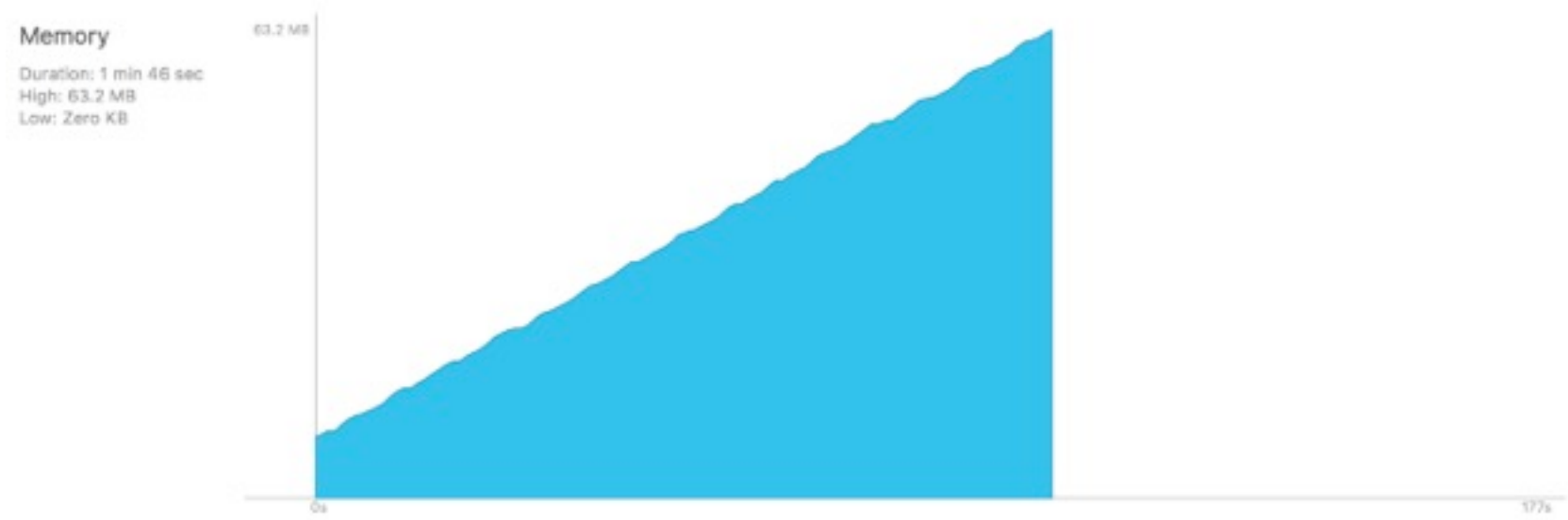
```
func captureOutput(_ captureOutput: AVCaptureOutput!,
didOutputSampleBuffer sampleBuffer: CMSampleBuffer!, from connection:
AVCaptureConnection!) {

    DispatchQueue.main.async { [unowned self] in

        guard let uiImage = self.imageFromSampleBuffer(sampleBuffer:
        sampleBuffer) else { return }

        self.delegate?.captured(image: uiImage)

    }

}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Шаг 11 — Память утекает!

Если мы пристально смотрим на предыдущий код, мы видим, что достойно не все, что мы сказали прежде! Во время шага `7` мы сказали, что каждая обработка изображений **`должна быть сделана на последовательной очереди(must be done on the serial queue)`**, из которой вызывают метод вывода `captureOutput`. Здесь, мы сделали **`ошибку преобразования буфера в изображение(mistake of converting the buffer to the image)`** внутри **`главного потока(main thread)`**!

Далее показано то, что происходит с памятью:



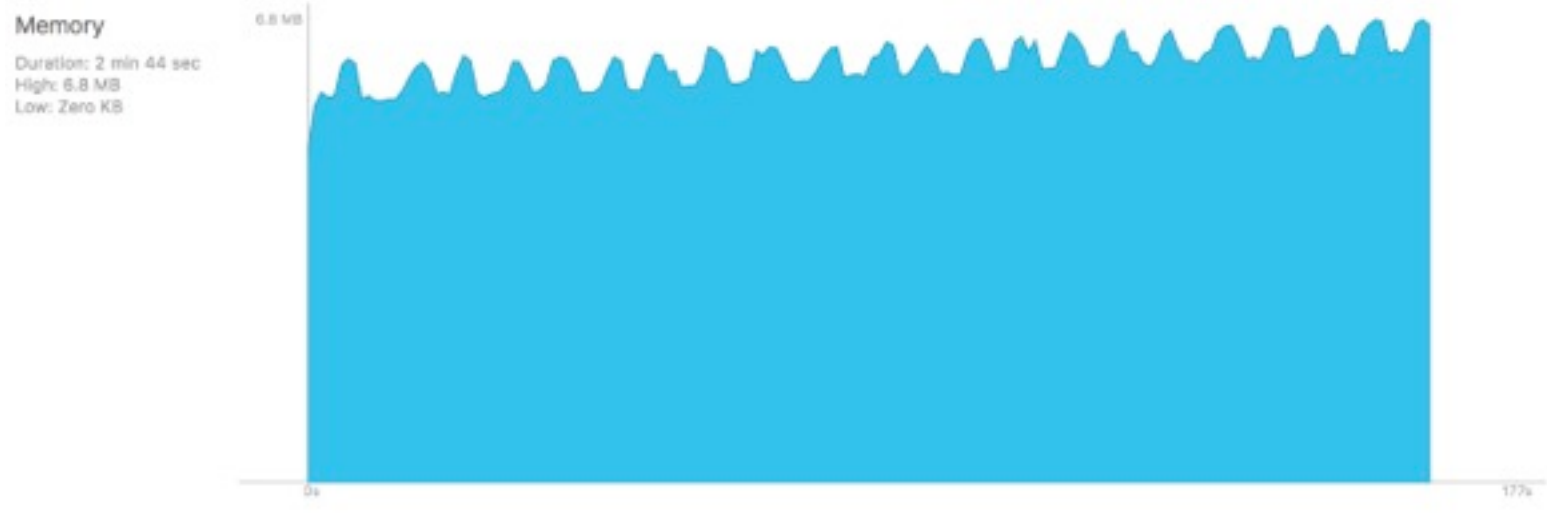
11.1 Переместите обработку изображений за пределы главного потока(main thread)

```
func captureOutput(_ captureOutput: AVCaptureOutput!,
didOutputSampleBuffer sampleBuffer: CMSampleBuffer!, from connection:
AVCaptureConnection!) {

    guard let uiImage = imageFromSampleBuffer(sampleBuffer:
    sampleBuffer) else { return }

    DispatchQueue.main.async { [unowned self] in

        self.delegate?.captured(image: uiImage)

    }

}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Обратите внимание на то, что перед вызовом функции, мы больше не нуждаемся в указателе `self`, и мы удалили его для соответствия инструкции по лучшему `Swift`-кодированию.

Далее показано теперешнее состояние памяти:



Это более приятно, объем памяти кажется устойчивым в течение долгого времени, но мы видим некоторые взлеты и падения, которыми мы также можем заняться. Они происходят, потому что, преобразовывая буфер в изображение, мы каждый раз создаем контекст. Мы хотим, только один раз создать контекст.

11.2 Переместите контекст как переменную класса

```
    private let context = CIContext()
```

исходник FrameExtractor.swift находится на сервисе GitHub

Далее показано теперешнее состояние памяти:

Кажется, утечка памяти устранена!

Шаг 12 — Покажите мне, что получили!

Давайте быстро установим `UI`-интерфейс пользователя.

1. Перейдите к файлу `Main.storyboard` и добавьте вид изображения `ImageView`, берущий весь экран.
2. Установите значение `режима содержания(content mode)` в `aspect fill(аспект заливки)`
3. Нажмите Control и ссылку на вид изображения `ImageView` перетащите в файл контроллера вида `ViewController.swift`

Теперь перейдите в в файл контроллера вида `ViewController.swift`.

12.1 Контроллер вида ViewController приспособьте на соответствие делегату экстрактора кадра FrameExtractorDelegate

```
class ViewController: UIViewController, FrameExtractorDelegate {


    var frameExtractor: FrameExtractor!


    @IBOutlet weak var imageView: UIImageView!

    override func viewDidLoad() {
        super.viewDidLoad()
        frameExtractor = FrameExtractor()
     }

}
```

исходник FrameExtractor.swift находится на сервисе GitHub

12.2 Установите ViewController так, чтобы он стал делегатом

```
class ViewController: UIViewController, FrameExtractorDelegate {


    var frameExtractor: FrameExtractor!


    @IBOutlet weak var imageView: UIImageView!

    override func viewDidLoad() {
        super.viewDidLoad()
        frameExtractor = FrameExtractor()
        frameExtractor.delegate = self
     }

}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Наконец, добавьте метод совершенной съемки(захвата) **captured** из созданного нами протокола делегата экстрактора кадра **FrameExtractorDelegate** и обновите вид (представление) изображения каждый раз, когда мы получаем изображение. Мы можем сразу же обновить **UI**-интерфейс пользователя, потому что метод совершенной съемки(захвата) **captured** вызван на **главном потоке(main thread)**.

12.3 В протоколе добавьте метод совершенной съемки(захвата) captured

```
func captured(image: UIImage) {
    imageView.image = image
}
```

исходник FrameExtractor.swift находится на сервисе GitHub

Теперь вы можете нажать кнопку выполнить приложение!

Шаг 13 — Ой, мы - перевернуты

Точно, по умолчанию **сенсор изображения(image sensor)** ориентирован на этот способ. Это является причиной того, почему мы должны правильно установить **ориентацию видеовыхода(orientation the video output)**!

Для этого, мы должны получить соединение от видеовыхода и проверить, можем ли мы установить ориентацию видео. Кроме того, если мы используем переднюю(фронтальную) камеру, то канал передачи изображения должен быть отражен зеркально.

13. 1 Завершите полностью метод конфигурации сеанса configureSession()

```
guard let connection = videoOutput.connection(withMediaType:
AVFoundation.AVMediaTypeVideo) else { return }
guard connection.isVideoOrientationSupported else { return }
guard connection.isVideoMirroringSupported else { return }
connection.videoOrientation = .portrait
connection.isVideoMirrored = position == .front
```

исходник FrameExtractor.swift находится на сервисе GitHub

Кроме того, перейдите на панель **General project settings(Общие параметры настройки проекта)** и в группе **Deployment Info(Информация развертывания)**, снимите флажок в поле **Landscape Left(Альбомный, слева)** и в поле **Landscape Right(Альбомный, справа)**, что вынудит приложение остаться в портретной ориентации.

Выполните приложение снова и **voilà**!

Вот именно!

Теперь вы свободны использовать камеру нравящимся вам способом и выпустите свою креативность в **мире машинного зрения(computer vision world)**!

Не стесняйтесь тщательно продумать это! Например, дайте возможность запустить и остановить **извлечение кадра(frame extraction)**, потому что в реальном состоянии, если мы продвигаем контроллер вида на вершину, то извлечение кадра все еще происходит! В текущем состоянии приложения приостановлено извлечение кадра, когда происходит прерывание (например, телефонный вызов) и впоследствии возобновляется.

Много материала может быть построено на основе этого, например, предоставляет пользователю возможность изменить настройки **снятых/захваченных кадров(captured frames)**, **используемой камеры(camera used)**, **качество(quality)** и т.д. … Много значимых учебных руководств доступно в онлайне, но не забывайте, что, изменяете **настройки камеры(camera settings)**, то это должно быть сделано, блокируя его. Совершенно новый предмет!

Чтобы сделать обработку изображений на каждом снятом(захваченном) кадре класса `UI`-изображения интерфейса пользователя `UIImage`, вы могли бы использовать библиотеку компьютерного зрения `OpenCV`! Некоторые аккуратные функции, например функция преобразования снятого(захваченного) кадра класса `UI`-изображения интерфейса пользователя `UIImage` в матрицу `UIImageToMat` дает в результате лежащую в основе `матрицу пикселей(pixel matrix)` позади изображения. Как только вы получаете матрицу, то начинается реальное шоу. Смотрите мои другие учебные руководства, объясняющие, как использовать `OpenCV` с языком программирования `Swift`!

Сообщите мне о проектах, построенных на основе этого руководства, я хотел бы взглянуть на них!

Понравилось, что вы прочитали? Давайте нажмем кнопку одобрения так, чтобы все могли прочитать ее также!

Кроме того

Эта статья на `Medium`-сайте:

• iOS — Извлечение кадров камеры(iOS — Camera Frames Extraction)

https://medium.com/ios-os-x-development/ios-camera-frames-extraction-d2c0f80ed05a