iOS Dev Accelerator Week2 Day2

- Custom Protocols
- Resizing an Image
- CoreImage
- UIImagePickerController
- PhotosFramework

Custom Protocols

Protocols

- In the real world, people are often required to follow strict procedures in certain situations.
- For example, firefighters are supposed to follow a specific protocol during emergencies.
- In the world of object oriented programming, its critical to be able to define a set of behaviors that is expected of an object in certain situations.
- This is what a protocol is used for.

Protocols

- Table views are an example that we have already used in our apps.
- A table view expects to be able to communicate with a data source object in order to find out what it is required to display.
- This means that which ever object is the data source must respond to specific messages
- **The datasource could be an instance of any class**, such as a UIViewController or some custom data source class, but its important that it implements the required methods to function as a table view datasource
- Swift & Objective-C allow you to create protocols which declare methods expected to be used in a particular situation.
- This is similar to interfaces in Java

Declaring a Protocol

- Declaring protocols is rather simple in Swift:

```
protocol RandomNumberGenerator {
    func random() -> Double
}
```

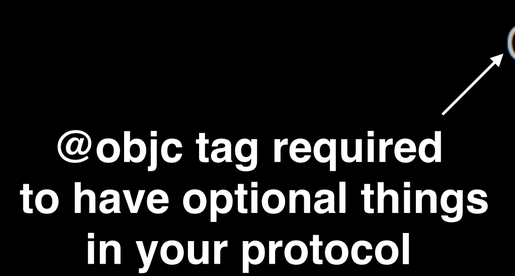
- Above is an example of a protocol called RandomNumberGenerator
- It has one required method, its called random and it returns a Double.
- So any class that conforms to this protocol must have a method called RandomNumberGenerator that return a Double.
- It is important to remember that protocols don't define the implementation of these methods. That is up to the object that conforms to the protocol!

Declaring a Protocol

- Protocols can also declare properties that it expects its conformers to have:

```
protocol SomeProtocol {
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}
```

- Properties in a protocol must be designated as read write or read only, as seen above
- Methods and properties declared in a protocol can be marked as optional:




```
@objc protocol CounterDataSource {
    optional func incrementForCount(count: Int) ->
        Int
    optional var fixedIncrement: Int { get }
}
```

Conforming to a protocol

- Conforming to a protocol is also a pretty simple operation.
- Here is an example of a protocol called FullyNamed and then an example of a struct that conforms to it:

```
protocol FullyNamed {
    var fullName: String { get }
}


struct Person: FullyNamed {
    var fullName: String
}
```

Best practice delegate method convention

- Delegation methods should begin with the name of object doing the delegating — application, control, controller, etc.
- The name is then followed by a verb of what just occurred — willSelect, didSelect, openFile, etc.
- For example, our protocol will be called imageSelectedDelegate and the method we will define in it will be called controllerDidSelectImage

Creating a delegate property

- Once your protocol is setup, you need to add a delegate property to whatever class is going to have the delegate:

```
var delegate : ImageSelectedDelegate?
```

the delegate property's
type is the protocol. This basically just
says this property can be set to any
type as long as it conforms to this protocol

Demo

Resizing an Image

Resizing an Image

- Par for the course with programming, there are a number of different ways to resize an image in iOS
- NSHipster did a great article on a bunch of the different ways, and which way is the fastest.
- Here is the fastest:

```
var size = CGSize(width: 100, height: 100)
UIGraphicsBeginImageContext(size)
originalImage!.drawInRect(CGRect(x: 0, y: 0, width: 100, height: 100))
let thumbnail = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
```

1. UIGraphicsBeginImageContext() creates bitmap-based graphics context for you to draw in and makes it the ‘current context’
2. drawInRect is a method on UIImage which draws the image in the specified rectangle, in the current context, and scales if needed
3. UIGraphicsGetImageFromCurrentImageContext() just pulls the currently drawn image from the context and returns a UIImage

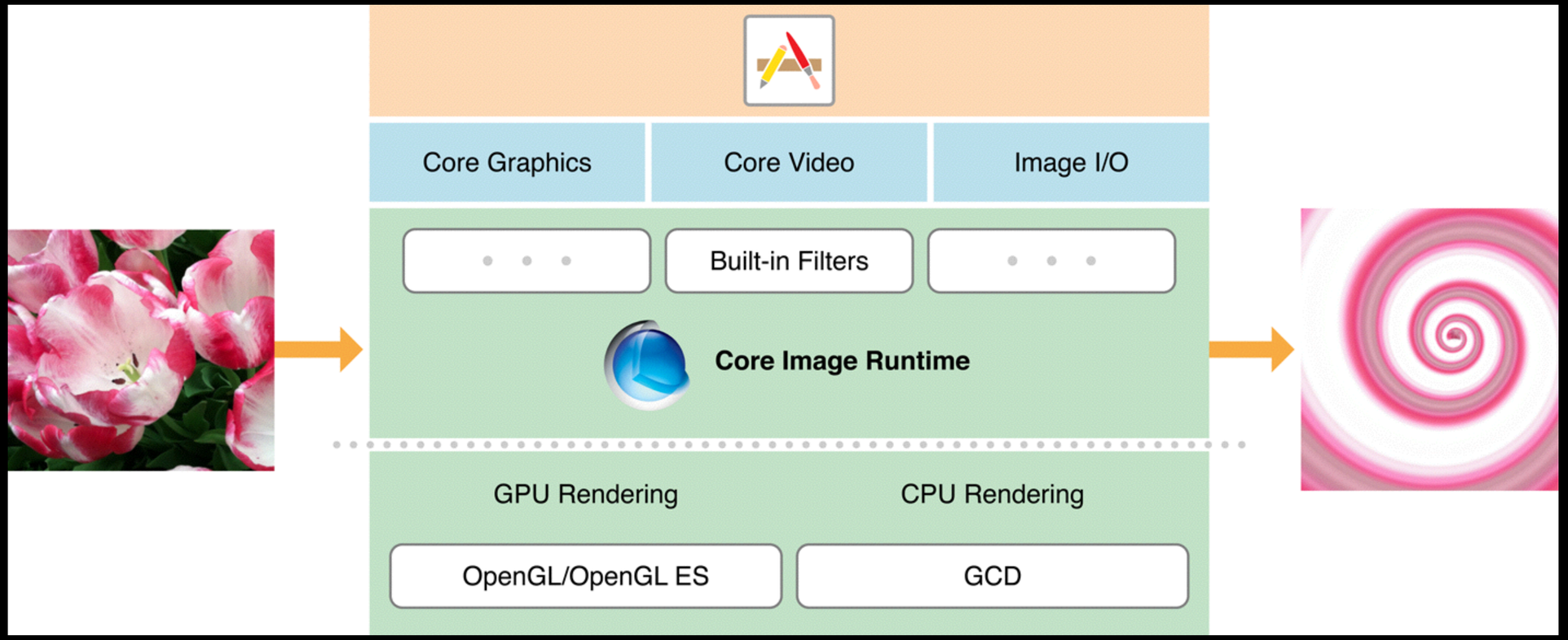
Demo

CoreImage

CoreImage

- “Core Image is an image processing and analysis technology designed to provide near real-time processing for still and video images”
- Can use either the GPU or CPU
- “Core Image hides the details of low-level graphic processing….You don’t need to know the details of OpenGL/ES to leverage the power of the GPU”

CoreImage


Core Graphics
Core Video
Image I/O
Built-in Filters
Core Image Runtime
GPU Rendering
CPU Rendering
OpenGL/OpenGL ES
GCD

CoreImage Offerings

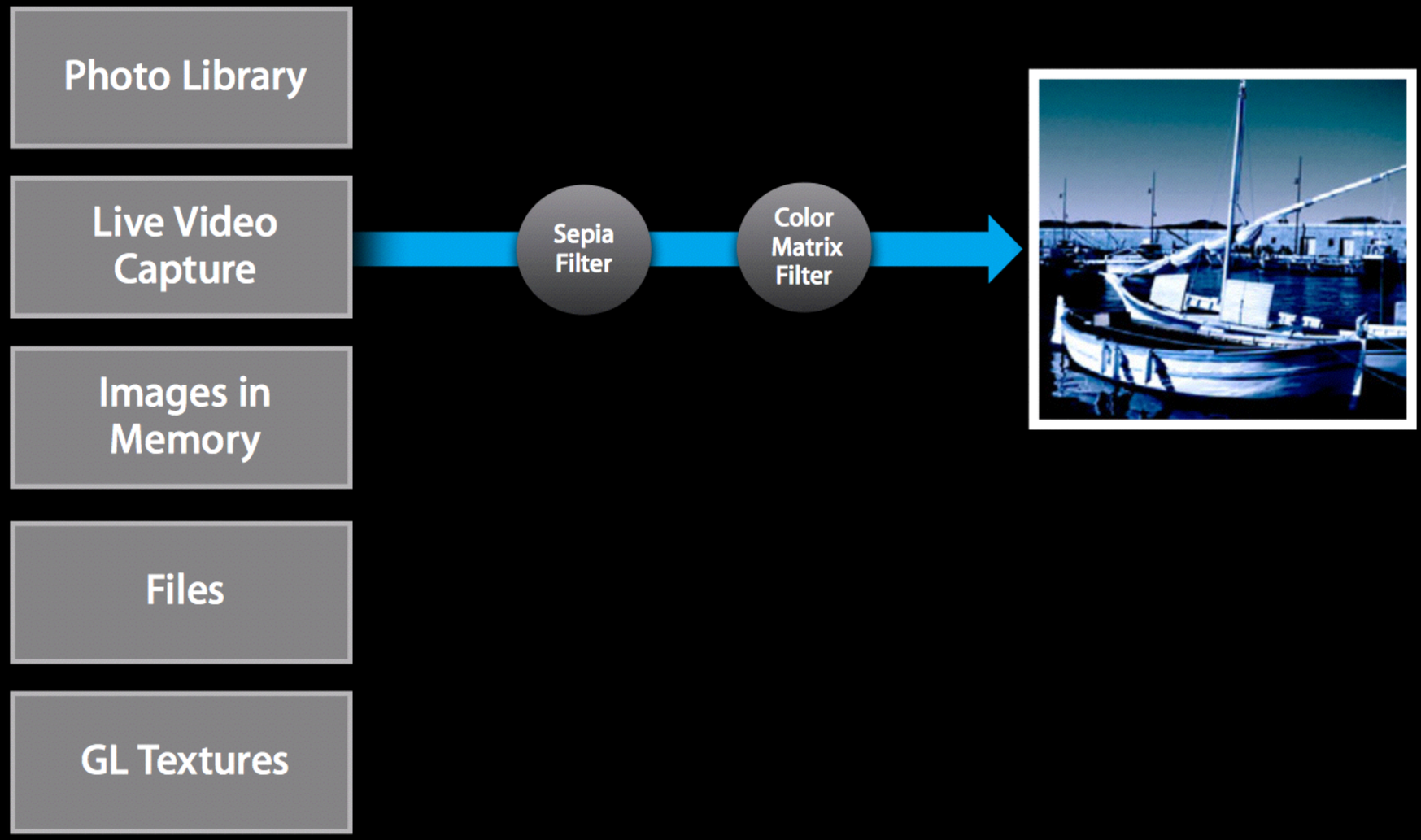
- Built-in image processing filters (90+ on iOS)
- Face and Feature detection capability
- Support for automatic image enhancement
- Ability to chain multiple filters together to create custom effects

guy using CoreImage

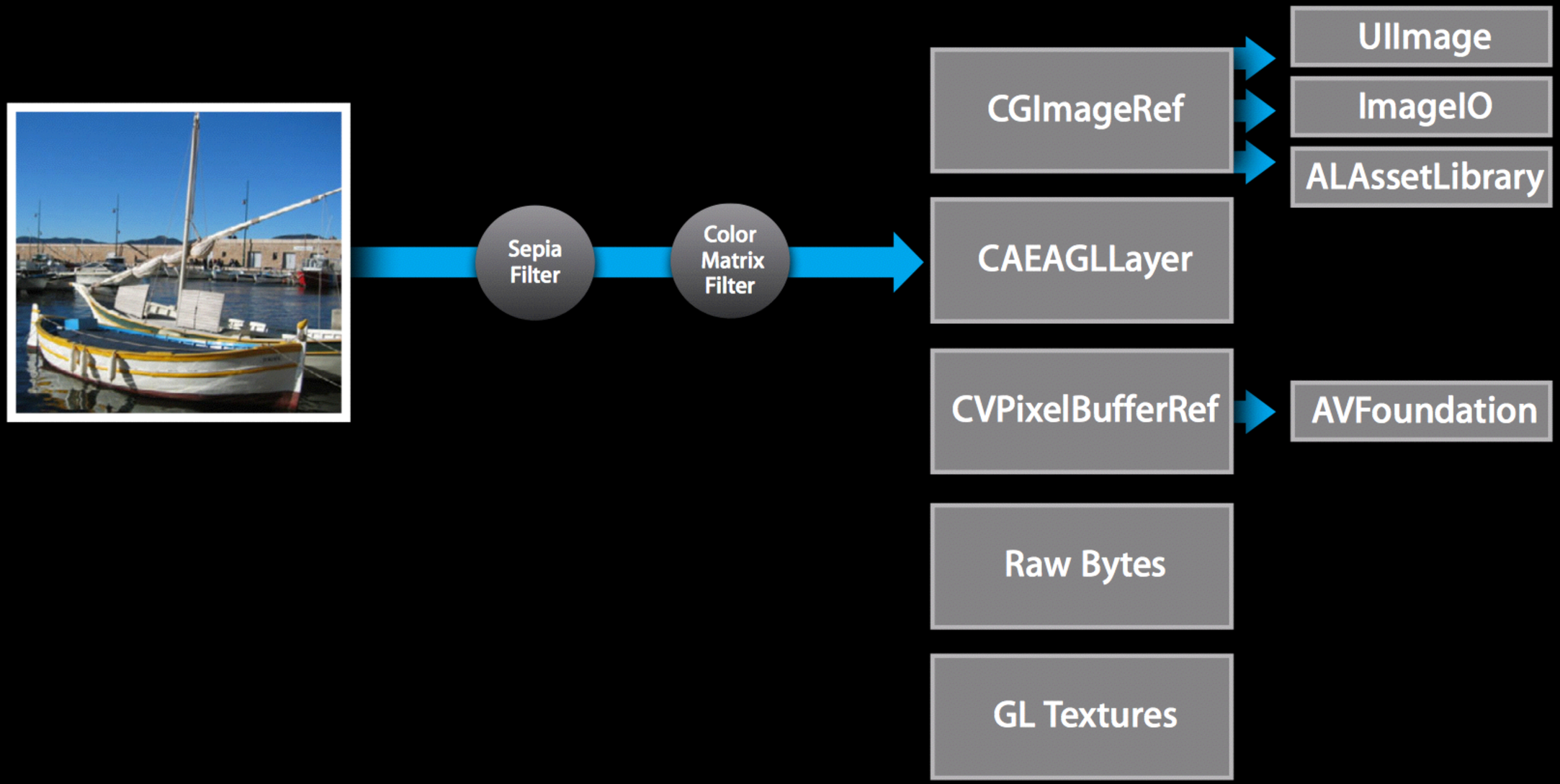
Filtering

- Filters applied on a per pixel basis
- Can be chained together

Filtering Inputs are Flexible



As are the Outputs



CIAdditionCompositing
CIAffineClamp
CIAffineTile
CIAffineTransform
CIBarsSwipeTransition
CIBlendWithMask
CIBloom
CICheckerboardGenerator
CICircleSplashDistortion
CICircularScreen
CIColorBlendMode
CIColorBurnBlendMode
CIColorControls
CIColorCube
CIColorDodgeBlendMode
CIColorInvert
CIColorMap
CIColorMatrix
CIColorMonochrome
CIColorPosterize
CIConstantColorGenerator
CICopyMachineTransition
CICrop
CIDarkenBlendMode
CIDifferenceBlendMode
CIDisintegrateWithMask
CIDissolveTransition
CIDotScreen
CIEightfoldReflectedTile
CIExclusionBlendMode
CIExposureAdjust
CIFalseColor
CIFlashTransition
CIFourfoldReflectedTile
CIFourfoldRotatedTile
CIFourfoldTranslatedTile
CIGammaAdjust
CIGaussianBlur
CIGaussianGradient
CIGlideReflectedTile
CIGloom
CIHardLightBlendMode
CIHatchedScreen
CIHighlightShadowAdjust
CIHoleDistortion
CIHueAdjust
CIHueBlendMode
CILanczosScaleTransform
CILightenBlendMode
CILightTunnel
CILinearGradient
CILineScreen
CILuminosityBlendMode
CIMaskToAlpha
CIMaximumComponent
CIMaximumCompositing
CIMinimumComponent
CIMinimumCompositing
CIModTransition
CIMultiplyBlendMode
CIMultiplyCompositing
CIOverlayBlendMode
CIPerspectiveTile
CIPerspectiveTransform
CIPinchDistortion
CIPixellate
CIRadialGradient
CIRandomGenerator
CISaturationBlendMode
CIScreenBlendMode
CISepiaTone
CISharpenLuminance
CISixfoldReflectedTile
CISixfoldRotatedTile
CISoftLightBlendMode
CISourceAtopCompositing
CISourceInCompositing
CISourceOutCompositing
CISourceOverCompositing
CIStarShineGenerator
CIStraightenFilter
CIStripesGenerator
CISwipeTransition
CITemperatureAndTint
CIToneCurve
CITriangleKaleidoscope
CITwelvefoldReflectedTile
CITwirlDistortion
CIUnsharpMask
CIVibrance
CIVignette
CIVortexDistortion
CIWhitePointAdjust

CIImage

- An Immutable object that represents the recipe for an Image
- Can represent a file from disk or the output of a CIFilter
- Multiple ways to create one:

```
var image  = CIImage(contentsOfURL: url)
```

```
var anotherImage = CIImage(image: UIImage())
```

Also has inits from Raw bytes,
NSData,CGImage,PixelBuffers,etc

CIFilter

- Mutable object that represents a filter (not thread safe since its mutable!)
- Produces an output image based on the input.
- Each filter has a different set of inputKey's you can modify to alter the effect of the filter:

```
var filter = CIFilter(name: "CISepiaTone")
filter.setValue(image, forKey: kCIInputImageKey)
filter.setValue(NSNumber(float: 0.8), forKey: @"inputIntensity")
```

- You can query for all the inputs of a filter with the .inputKeys property on an instance of CIFilter

CIContext

- An object through which Core Image draws results
- Can be based on CPU or GPU
- Always use GPU because the CPU performance sucks in comparison when dealing with graphical computations. All iOS 8 supporting devices support GPU context

```
self.context = CIContext(options: nil)
```
← CPU context

```
var options = [kCIContextWorkingColorSpace : NSNull()]
var myEAGLContext = EAGLContext(API: EAGLRenderingAPI.OpenGLES2)
self.gpuContext = CIContext(EAGLContext: myEAGLContext, options: options)
```
← GPU context

Our filtering workflow

- Today we will use CIFilters to produce filtered thumbnails of the image that was selected from our gallery vc
- The general workflow of this is:
    1. generate thumbnail of selected photo
    2. loop through our array of Thumbnail objects and set the original thumbnail property on each one
    3. tell collection view to reload data
    4. Lazily filter each thumbnail in cellForItemAtIndexPath

Demo

Camera Programming

- 2 ways for interfacing with the camera in your app:
  1. UIImagePickerController (easy mode)
  2. AVFoundation Framework (hard mode)

UIImagePickerController

- The workflow of using UIImagePickerController is 3 steps:
    1. Instantiate and modally present the UIImagePickerController
    2. ImagePicker manages the user's interaction with the camera or photo library
    3. The system invokes your image picker controller delegate methods to handle the user being done with the picker.

UIImagePickerController Setup

- The first thing you have to account for is checking if the device has a camera.
- If your app absolutely relies on a camera, add a UIRequiredDeviceCapabilities key in your info.plist
- Use the isSourceTypeAvailable class method on UIImagePickerController to check if camera is available.

UIImagePickerController Setup

- Next make sure something is setup to be the delegate of the picker. This is usually the view controller that is spawning the picker.
- The final step is to actually create the UIImagePicker with a sourceType of UIImagePickerControllerSourceTypeCamera.
- Media Types: Used to specify if the camera should be locked to photos, videos, or both.
- AllowsEditing property to set if the user is able to modify the photo in the picker after taking the photo.

UIImagePickerControllerDelegate

- The Delegate methods control what happens after the user is done using the picker. 2 big method:
  1. imagePickerControllerDidCancel:
  2. imagePickerController:didFinishPickingMediaWithInfo:

Info Dictionary

The info dictionary has a number of items related to the image that was taken:

```
NSString *const UIImagePickerControllerMediaType;
NSString *const UIImagePickerControllerOriginalImage;
NSString *const UIImagePickerControllerEditedImage;
NSString *const UIImagePickerControllerCropRect;
NSString *const UIImagePickerControllerMediaURL;
NSString *const UIImagePickerControllerReferenceURL;
NSString *const UIImagePickerControllerMediaMetadata;
```

MediaType is either kUTTypeImage or kUTTypeMovie

Demo

Photos Framework

Photos Framework

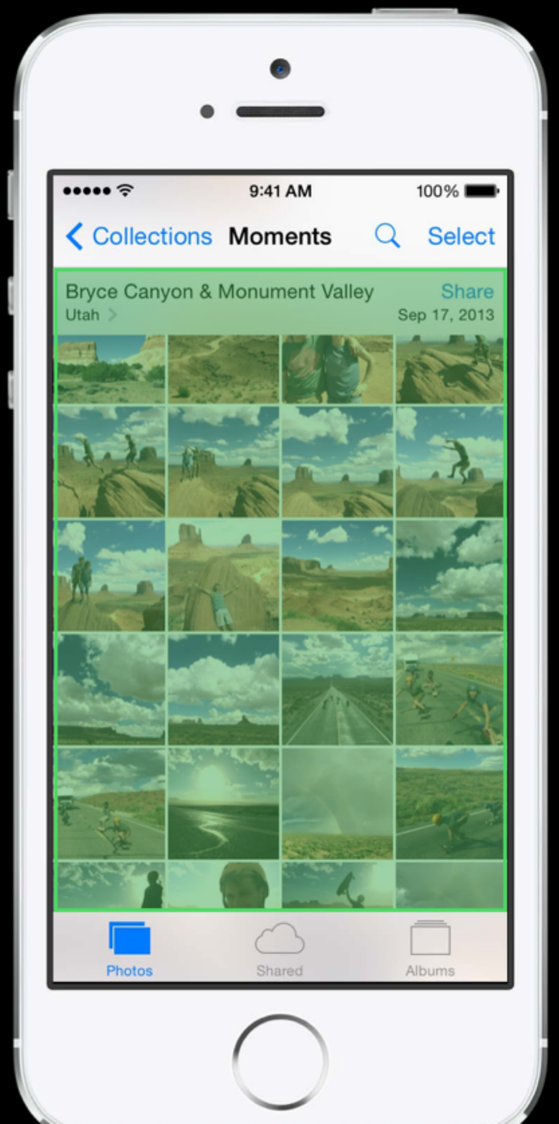
- New Apple Framework that allows access to photos and videos from the photo library.
- Also used for creating photo editing app extensions, a new feature with iOS8
- First-class citizen, you can create a full-featured photo library browser and editor on par with Apple's Photos App.
- Intended to supersede ALAssetsLibrary

PHAsset

- The Photos framework model object that represents a single photo or video.
- Has properties for Media type, Creation date, Location, and Favorite.

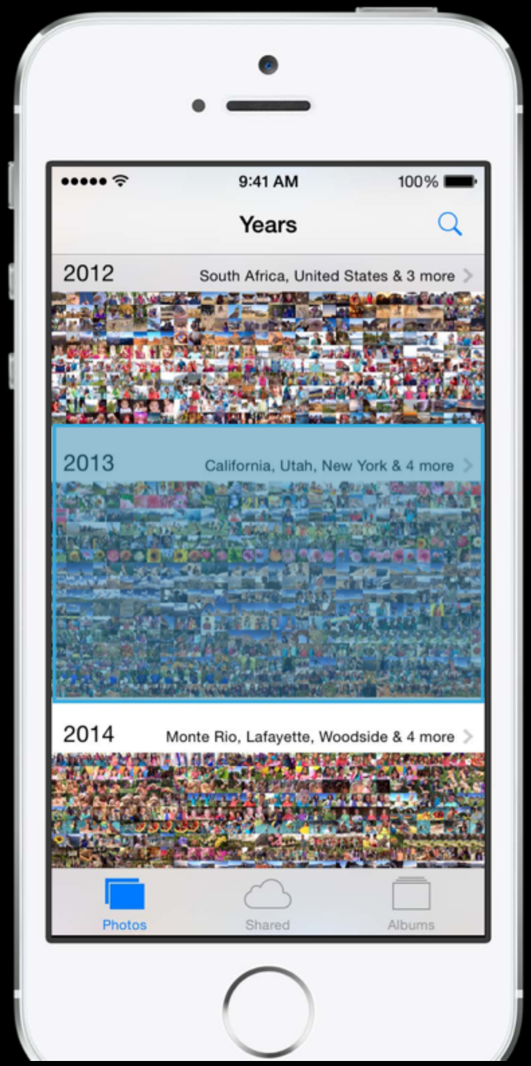
PHAssetCollection

- A Photos framework model object representing an ordered collection of assets.
- Albums, moments, and smart albums.
- Has properties for Type, Title, and Start and End Date.



PHCollectionList

- A Photos framework model object representing an ordered collection of collections.
- This can be a folder, moment, or year
- Has properties for Type, Title, and Start and End Date.



Fetching Model Objects

- You fetch via class methods on the models:
  - PHAsset.fetchAssetsWithMediaType(PHAssetMediaType.Photo, options:nil)
  - PHAssetCollection.fetchMomentsWithOptions(nil)
- Collections do not cache their contents, so you still have to fetch all the assets inside of it. This is because the results of a fetch can be very large, and you dont want all of those objects in memory at once.

PFFetchResult

- Results returned in a PHFetchResult
- Similar to an Array.

Making Changes

- You can favorite a photo and add an asset to an album
- You cannot directly mutate an asset, they are read only (thread safe!)
- To make a change, you have to make a change request.
- Theres a request class for each model class:

```
PHAssetChangeRequest
PHAssetCollectionChangeRequest
PHCollectionListChangeRequest
```

Making Changes

```
func toggleFavorite(asset : PHAsset) {

    PHPhotoLibrary.sharedPhotoLibrary().performChanges({

        //create a change request object for the asset
        var changeRequest = PHAssetChangeRequest(forAsset: asset) as
      PHAssetChangeRequest
        //make your change
        changeRequest.favorite = !changeRequest.favorite

        }, completionHandler: { ( success : Bool,error : NSError!) -> Void in

        //asset change complete
        })

}
```

Making New Objects

Create via creation request

```
var request = PHAssetChangeRequest.creationRequestForAssetFromImage(UIImage())
```

Placeholder objects

```
var placeHolder = request.placeholderForCreatedAsset
```

- Reference to a new, unsaved object
- Add to collections
- Can provide unique, persistent `localIdentifier`

Getting to the actual data

- Many different sizes of an image may be available or different formats of a video
- Use PHImageManager to request images/videos
- Request an image based on target size for displaying
- Request a video based on the usage
- Asynchronous API, because you dont know how long it will take to load the data, it could be very expensive
- Will optionally retrieve the data from the network if its only on iCloud
- Use a PHCachingImageManager when displaying a collection of images for better performance.

Requesting an Image

```
let manager = PHImageManager.defaultManager()

manager.requestImageForAsset(photo,
                             targetSize: cellSize,
                             contentMode: PHImageContentMode.AspectFill,
                             options: nil,
                             resultHandler: {(result : UIImage!, [NSObject : AnyObject]!) -> Void in
                                 if result {
                                     imageView.image = result
                                 } else {
                                     //tell user something went wrong
                                 }
    })
```

Advanced Image Request

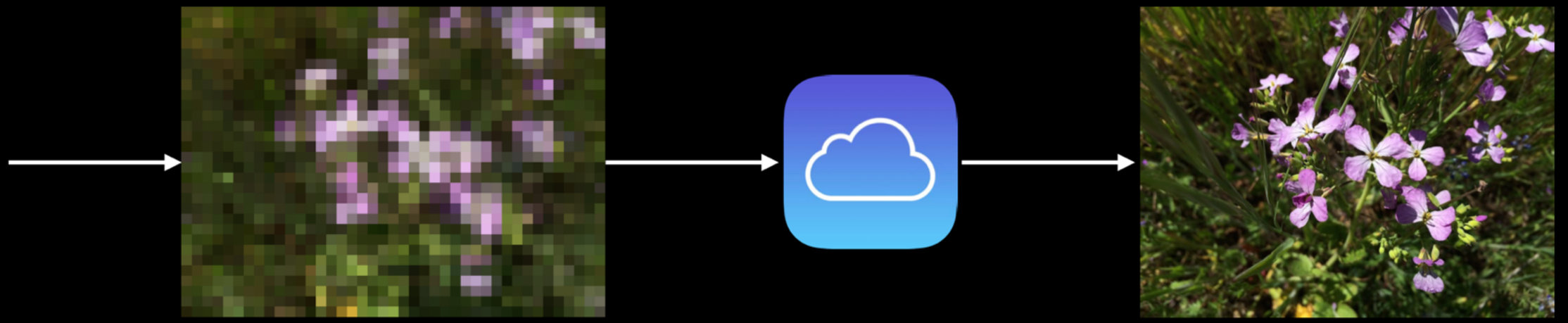
```
var options = PHImageRequestOptions()

options.networkAccessAllowed = true
options.progressHandler = {(progress : Double, error : NSError!, degraded : UnsafePointer<ObjCBool>,
[NSObject : AnyObject]!) in
    //update visible progress UI
    }

//use your options to control the request behavior
manager.requestImageForAsset(photo,
    targetSize: cellSize,
    contentMode: PHImageContentMode.AspectFill,
    options: options,
    resultHandler: {(result : UIImage!, [NSObject : AnyObject]!) -> Void in
```

Advanced Image Request

```
[manager requestImageForAsset: ... ^(UIImage *result, NSDictionary *info) {
    // This block can be called multiple times
}];
```



First callback synchronous

Second callback asynchronous

Demo