iOS Dev Accelerator
Week 2 Day 1

- UIAlertController
- UIImagePickerController
- Core Image
- Debugging

The MVC layout of our Week 2 App

Model Layer

- Post
- FilterService
- ParseService

Controller Layer

- Photo ViewController
- Gallery ViewController
- Timeline ViewController
- Tab Bar Controller
- AlertController

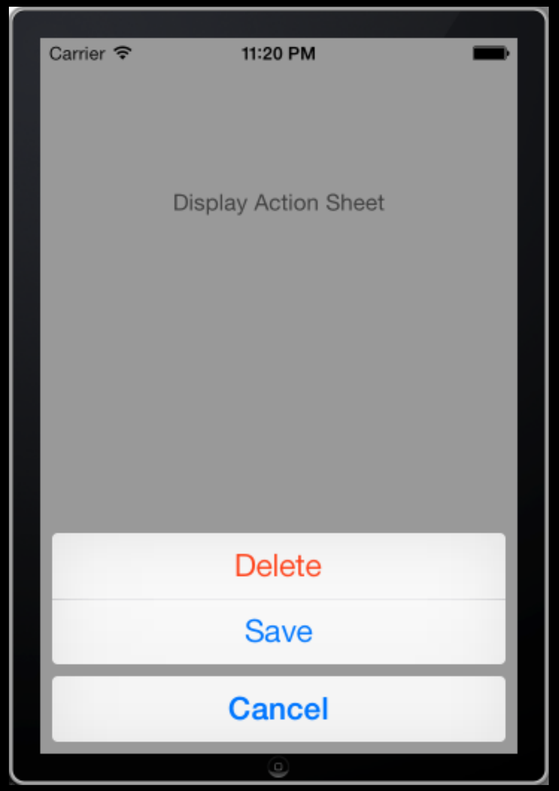
View Layer

- Storyboard
- FilterCell
- GalleryCell
- Buttons, Labels, Bars
- CollectionViews

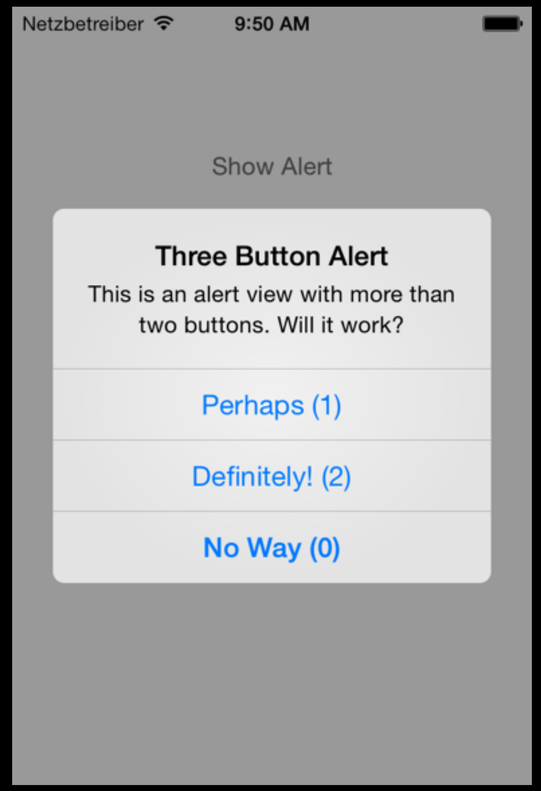
UIAlertController

UIAlertController

- “UIAlertController object displays an alert message to the user”
- Replaces both UIActionSheet and UIAlertView in iOS8
- After configuring the Alert Controller present it with presentViewController:animated: Completion:



ActionSheet



AlertView

UIAlertController Setup

`init(title:message:preferredStyle:)`

Creates and returns a view controller for displaying an alert to the user.

Declaration

SWIFT

```
convenience init(title title: String!,
                 message message: String!,
           preferredStyle preferredStyle: UIAlertControllerStyle)
```

Parameters

title	The title of the alert. Use this string to get the user's attention and communicate the reason for the alert.
message	Descriptive text that provides additional details about the reason for the alert.
preferredStyle	The style to use when presenting the alert controller. Use this parameter to configure the alert controller as an action sheet or as a modal alert.

UIAlertController Configuration

- In order to add buttons to your alert controller, you need to add actions.
- An action is a instance of the UIAlertAction class.
- “A UIAlertAction object represents an action that can be taken when tapping a button in an alert”
- Uses a closure expression (great!) to define the behavior of when the button is pressed. This is called the handler.

UIAlertAction Setup

`init(title:style:handler:)`

Create and return an action with the specified title and behavior.

Declaration

SWIFT

```
convenience init(title title: String!,
                 style style: UIAlertActionStyle,
               handler handler: ((UIAlertAction!) -> Void)!)
```

Parameters

title	The text to use for the button title. The value you specify should be localized for the user's current language. This parameter must not be `nil`.
style	Additional styling information to apply to the button. Use the style information to convey the type of action that is performed by the button. For a list of possible values, see the constants in UIAlertActionStyle.
handler	A block to execute when the user selects the action. This block has no return value and takes the selected action object as its only parameter.

Return Value

A new alert action object.

Feedback

Adding Actions

- Adding actions to the AlertController is as easy as calling addAction: on your AlertController and passing in the UIAlertAction(s)
- The order in which you add those actions determines their order in the resulting AlertController.

Presenting the alert controller

- To present the alert controller, you can call presentViewController:animated:completion: on the parent view controller
- This will work out of the box for iPhone, but on iPad it takes a bit more configuration
- On iPad you have to tell the alert controller where to present from, since its going to be a pop out menu.
- You can do this by setting the sourceView and sourceRect on the alert controller's popoverPresentationController.
- For some reason, Apple designed it so you have to set the sourceView and sourceRect EVERY time you are going to display the alert controller.

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:

- In order to conform to the UIImagePickerControllerDelegate, you must also conform to the UINavigationControllerDelegate. Both have no required methods.

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

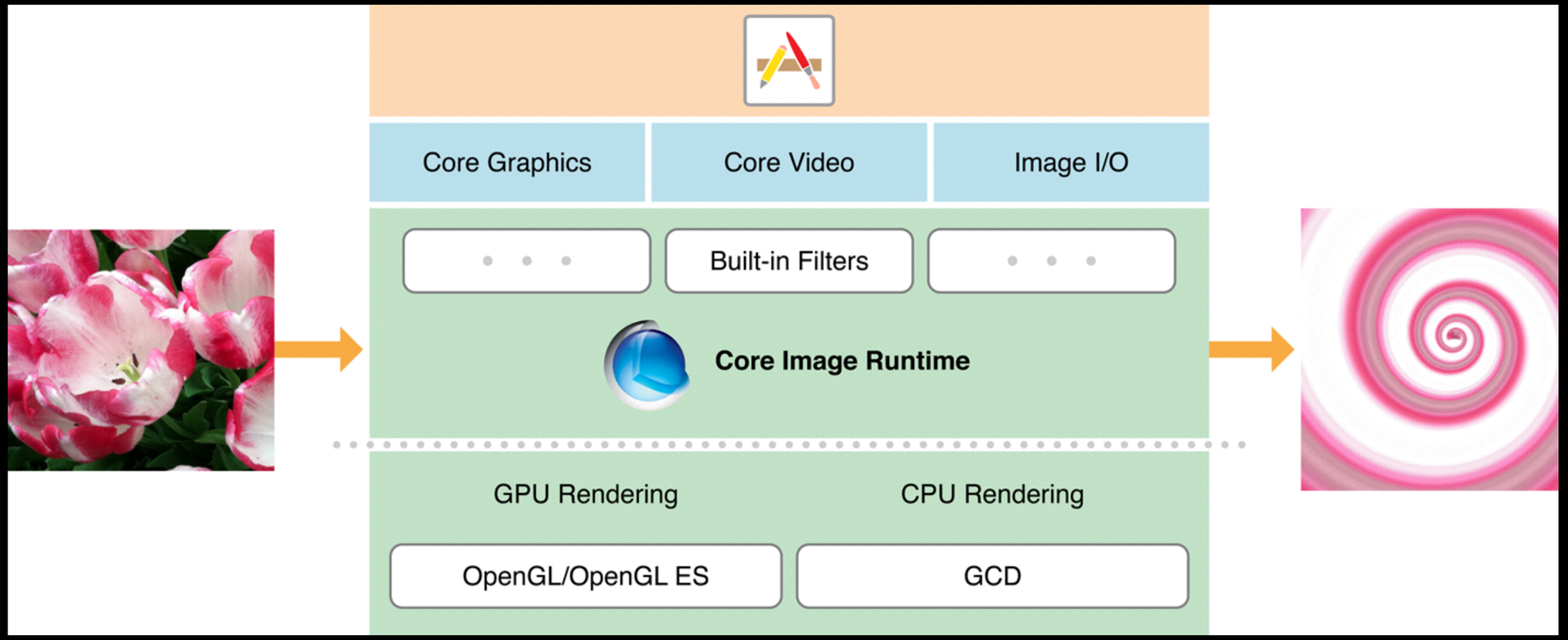
Demo

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

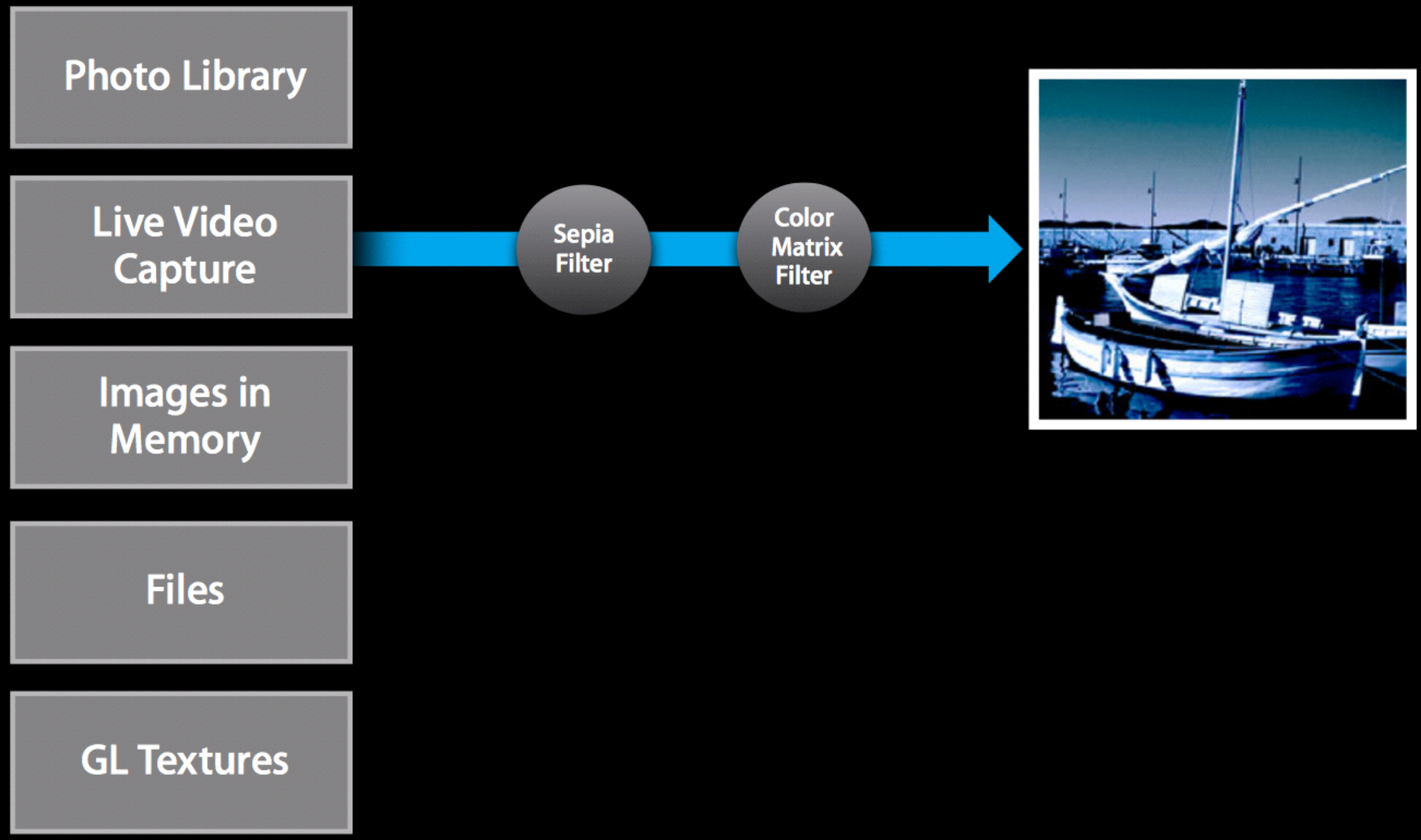
- Built-in image processing filters (90+ on iOS)
- Face and Feature detection capability
- Support for automatic image enhancement
- Ability to chain multiple filters together to create custom effects

guy using CoreImage

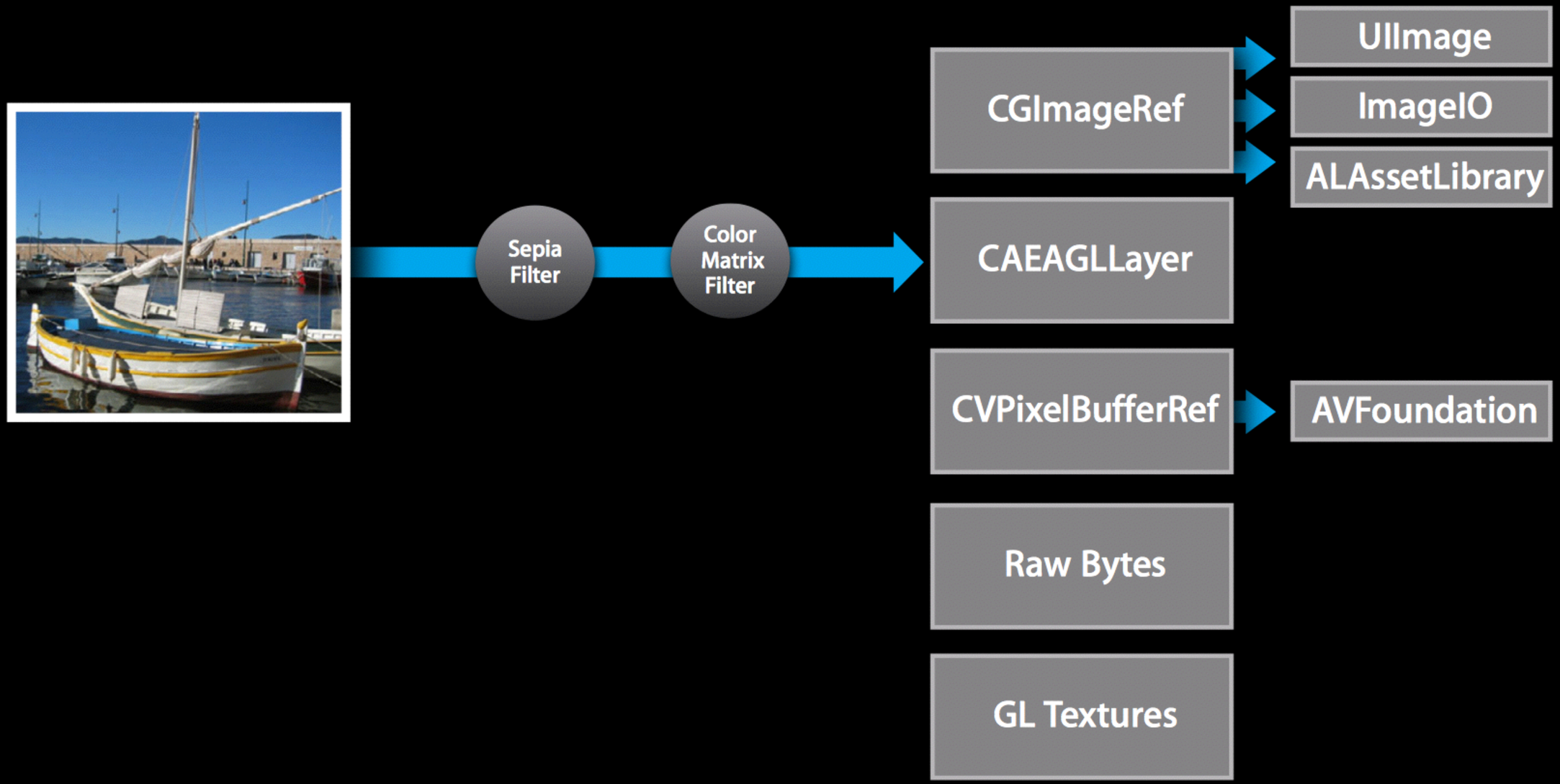
Filtering

- 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

Demo

Debugging

Debugging

- Having the feature you just built work perfectly on the first time you try it pretty much never happens. Everyone makes mistakes all the time.
- When things go wrong, which they will, instead of immediately tweaking the code, try debugging to compare the actual results vs the intended results.
- So how do you debug?

Breakpoints

- A breakpoint is a way to pause the execution of your app at a specific line
- You set breakpoints by clicking in the left hand margin of your code:

```
21      override func viewDidLoad() {
22        super.viewDidLoad()
23
24        self.tableView.dataSource = self
25        self.tableView.delegate = self
26        let cellNib = UINib(nibName: "TweetTableViewCell", bundle: NSBundle.
         mainBundle())
```

- After you have set your breakpoint, there is no need to recompile your app. If your app has to run that code again, the breakpoint will hit and your execution will pause.

Modifying your Breakpoint

- There are a few things you can do to your breakpoints after you have placed it:
- Click it once to disable it. It will look faded out.
- To remove it, drag it off of the ‘gutter’
- You can right click a breakpoint and choose from a few options (the ones above, plus edit breakpoint)

Conditional Breakpoint

- Using a conditional breakpoint, you can designate specific conditions where the breakpoint should pause execution.
- In addition, you can specify actions to take place upon the breakpoint being triggered.
- These actions can be a wide range of things: AppleScripts, Capturing OpenGL frames, Log or speak a message, Execute Shell command, play a sound.

Exception Breakpoints

- Exception breakpoints are super useful and super simple to set.
- These special breakpoints allow you to pause execution of your app when an exception is thrown, not caught.
- Which is to say, before your app actually crashes because of the exception.
- This is helpful because often times Xcode does not produce a very helpful debug statement when your app crashes.

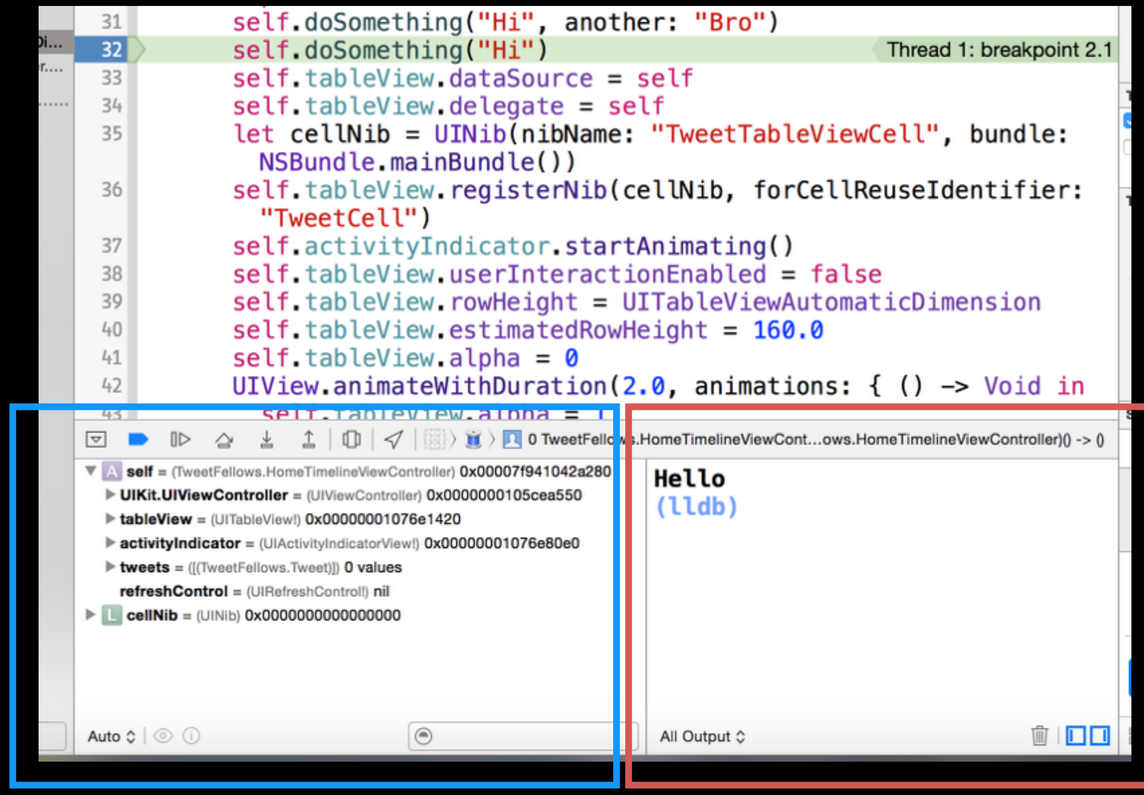
Symbolic Breakpoints

- Symbolic Breakpoints stop program execution when a specific function or method starts executing.
- Right now its a little wonky with Swift and defining the exact functions/methods you want to trigger

Demo

Debug Area

- Once your application is paused because of a breakpoint, you can then do the actual debugging using the Debug Area at the bottom of Xcode.
- The two different views are the variable view and debugger console



Variable View

Debugger Console

Variable View

- The Variable View allows you to inspect the value of a variable to help uncover problems in your code
- The Variable View allows you to see all variables in the current scope at the time of the paused execution
- You can specify which items you want to see by using the little popup menu at the bottom:
  - Auto: Displays recently accessed Variables
  - Local: Displays local variables
  - All: Displays all variables

Demo

Debugger Console & LLDB

- The Debugger Console is a great tool to use for debugging.
- It is made possibly by LLDB.
- LLDB is an open source debugger that comes bundled inside Xcode and lives in the debugger console
- When you add breakpoints, you are actually telling LLDB when it should pause execution of the app.

print

- The print command allows you print the value of a variable
- You can also use p for short.

po

- The po command (print object) prints the result of calling description on the object

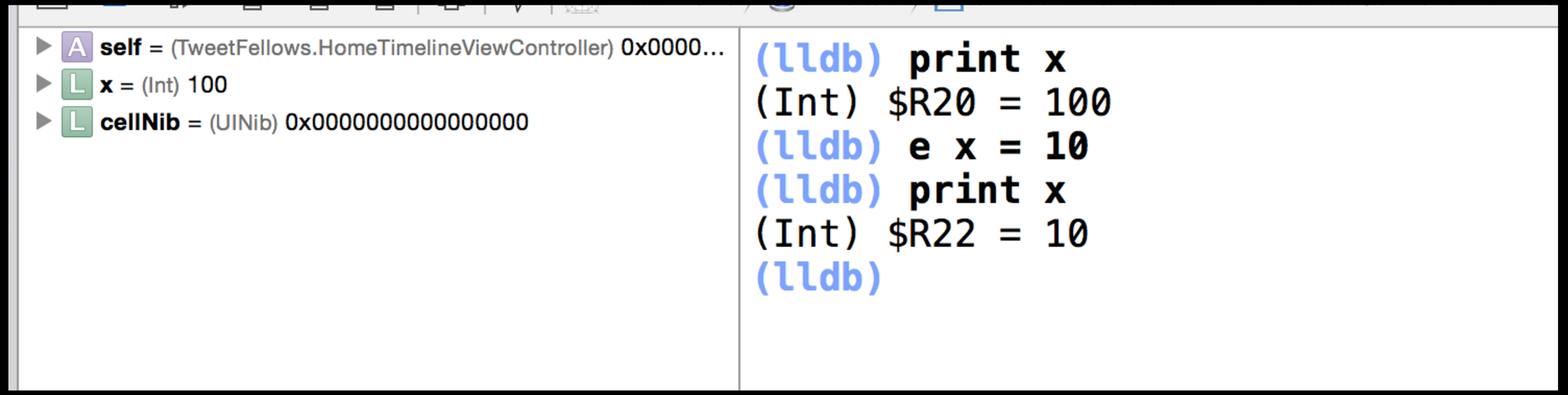
```
(lldb) print self.view
(UIView!) $R13 = Some {
  Some = 0x00007f89d8c75240 {
    UIKit.UIResponder = {
      ObjectiveC.NSObject = {}
    }
  }
}
(lldb) po self.view
<UIView: 0x7f89d8c75240; frame = (0 0; 375 667); autoresize
+H; layer = <CALayer: 0x7f89d8c614b0>>

(lldb) po x
10

(lldb) print x
(Int) $R16 = 10
(lldb)
```

expression

- The expression command lets you modify the value held by a variable from the debug console.
- It doesn't just modify the variable in the debug console, it actually modifies the value in the program!
- Also can be shorted to just e



Demo

Flow Control

- There are 4 buttons in the debug bar that you can use to control the flow of the program while paused for a breakpoint:

Continue Step Over Step Into Step Out

Flow Control

- Continue: Un-pauses the program, allowing it to continue executing normally. It will continue forever or until it hits the next breakpoint.
- Step Over: Executes a line of code as if it were a blackbox. If the line you are at is a function call, it will not go inside that function. Instead it will just execute the function and keep going.
- Step In: Steps inside the function call of the current line in order to debug or examine its execution.
- Step Out: Use this if you want to leave the current function you are in. It will execute until it hits the first return statement. If you accidentally Step In, just use Step Out to get out.

Demo