Section 5

Core Graphics
UIImage and UIImageView
Threading Techniques

Core Graphics

UIView and CALayer

- Odd split of functionality if you are coming from OS/X
- UIView “wraps” at least one CALayer
- Drawing is into the CALayer
- CALayers can have sublayers
- UIView handles user interaction
- There are different kinds of CALayer for different UI Needs
- Layers are the substrate that CoreGraphics writes on

Properties of CALayer

contentsGravity, opacity, hidden, masksToBounds, mask, doubleSided, cornerRadius, borderWidth, borderColor, backgroundColor, shadowOpacity, shadowRadius, shadowOffset, shadowColor, shadowPath, style

Types of CALayer

- CAEmitterLayer
  Used to implement a Core Animation–based particle emitter system. The emitter layer object controls the generation of the particles and their origin.
- CAGradientLayer
  Used to draw a color gradient that fills the shape of the layer (within the bounds of any rounded corners).

- CAEAGLLayer/CAOpenGLLayer
  Used to set up the backing store and context needed to draw using OpenGL ES (iOS) or OpenGL (OS X).
- CAReplicatorLayer
  Used when you want to make copies of one or more sublayers automatically. The replicator makes the copies for you and uses the properties you specify to alter the appearance or attributes of the copies.

- CAScrollLayer
  Used to manage a large scrollable area composed of multiple sublayers.
- CAShapeLayer
  Used to draw a cubic Bezier spline. Shape layers are advantageous for drawing path-based shapes because they always result in a crisp path, as opposed to a path you draw into a layer’s backing store, which would not look as good when scaled. However, the crisp results do involve rendering the shape on the main thread and caching the results.

- CATextLayer
  Used to render a plain or attributed string of text.
- CATiledLayer
  Used to manage a large image that can be divided into smaller tiles and rendered individually with support for zooming in and out of the content.
- CATransformLayer
  Used to render a true 3D layer hierarchy, rather than the flattened layer hierarchy implemented by other layer classes.

Core Graphics Objects

- CGContext, CGBitmapContext, CGPDFContext
- CGColor, CGColorSpace, CGShading, CGGradient, CGPattern
- CGPath, CGFont,
- CGFunction, CGImage, CGLayer
- CGPDFArray, CGPDFContentStream, CGPDFDictionary, CGPDFDocument, CGPDFObject, CGPDFOperatorTable, CGPDFPage, CGPDFScanner, CGPDFStream, CGPDFString

Graphics Contexts

- Contain drawing parameters and device-specific information needed to draw
- Do not have to be associated with a specific device in your code (happens at run time)
- Are stacked (i.e. can be pushed and popped)
- Can be vector or raster contexts

Graphics Contexts

Define basic drawing attributes such as:

- the colors to use when drawing
- the clipping area
- line width and style information
- font information
- compositing options

Bezier Curves/Paths and Vector vs Raster graphics

- Bezier curves are “parametric” in the mathematical sense
- They model smooth curves that can be scaled indefinitely
- Bezier paths are combinations of linked Bezier curves
- Linkage can be smooth in the mathematical sense
- Bezier Paths are not bound by the limits of rasterization
- Also used in animation to make smooth time steps

Example of a parametric equation

An ellipse can be defined as all points that satisfy:

$$x = a \cos t$$
$$y = b \sin t$$

where:

1. (x, y) are the coordinates of any point on the ellipse,
2. a, b are the radius on the x and y axes respectively,
3. t is the parameter, which ranges from 0 to 2π radians.

CGPath

- General path implementation
- UIKit's UIBezierPath wraps and manipulates one of these
- You can generate a UIBezierPath and then get its CGPath if you like
- CGPathAddArc, CGPathAddRelativeArc, CGPathAddArcToPoint, CGPathAddCurveToPoint, CGPathAddLines, CGPathAddLineToPoint, CGPathAddPath, CGPathAddQuadCurveToPoint, CGPathAddRect,

CGColor and a note on color in general

“Many methods in UIKit require you to specify color data using a UIColor object, and for general color needs it should be your main way of specifying colors. The color spaces used by this object are optimized for use on iOS-based devices and are therefore appropriate for most drawing needs. If you prefer to use Core Graphics colors and color spaces instead, however, you may do so.”

- UIColor wraps a CGColor
- UIColor can be initialized with RGB, HSB, or Mono
- CGColor can be initialized with RGB, CMYK, Mono or a custom color space
- CIColor is RGB since it is about working with image input

CGFont

Text is a mess and I give up.

“Underlying the text views in UIKit is a powerful layout engine called Text Kit. If you need to customize the layout process or you need to intervene in that behavior, you can use Text Kit. Text Kit is a set of classes and protocols that provide high-quality typographical services which enable apps to store, lay out, and display text with all the characteristics of fine typesetting, such as kerning, ligatures, line breaking, and justification.”

“For most apps, you can use the high-level text display classes and Text Kit for all their text handling. For smaller amounts of text and special needs requiring custom solutions, you can use alternate, lower level technologies, such as the programmatic interfaces from the Core Text, Core Graphics, and Core Animation frameworks as well as other APIs in UIKit itself.”

UIImage and UIImageView

Image Representations

- UIImage
- CGImage
- CIImage

UIImage

- wraps one CGImage (or more in the case of animated images) in the same way that UIView wraps a CALayer.
- Can be initialized from a variety of input formats and streams
- Abstracts the underlying format away from the drawing system
- Contains drawing information not associated with the bit map (alignment, stretchability, size, scale)

CGImage

- represents the actual bit mapped data
- supports jpeg, png, gif and stuff you've never heard of
- Lets you work at the bitmapped level if you need to: CGImageGetAlphaInfo, CGImageGetBitmapInfo, CGImageGetBitsPerComponent, CGImageGetBitsPerPixel, CGImageGetBytesPerRow, CGImageGetColorSpace, CGImageGetDataProvider, CGImageGetDecode, CGImageGetHeight, CGImageGetShouldInterpolate, CGImageGetRenderingIntent, CGImageGetWidth,

CIImage

- *Represents* an image BUT.. is not an image.
- Has image data associated with it and all information necessary to produce an image on demand
- Is the abstraction mechanism for GPU-optimized image filtering
- Will generate a CGImage lazily when embedded in a UIImage and the UIImage is asked for its CGImage

iOS Barbie:
“Threads are HARD!”

Threading 101

- All operating systems run “processes”, this is how they do multiple things at once
- The operating system schedules its processes transparently to most code, i.e. it stops your process and runs another one at its discretion without you knowing it
- Does this by “virtualizing” your memory space, i.e. your process thinks its the only one on the machine

Threading 101 (cont'd)

- To communicate between processes requires copying data from one process space to another
- 20 years ago this was how threading was done
- Recent vintage OS's allow you to have multiple threads of execution in the same process space
- Means copying is not required, you can use data in place

Threading 101 (cont'd)

- Each thread has a stack
- All threads share the heap
- You are responsible for your own autoreleasepool and error handling on threads you create
- You have no control over when threads execute
- On iOS devices you only have two concurrent cores
- More than two threads is wasteful. UNLESS, they are idle a lot of the time

Threading 101 (cont'd)

THE RULES

- Immutability is preferred
- Synchronization is required to avoid incorrect serialization of operations
- Synchronization is done by locking objects
- Deadlocks can be avoided only by specifying lock order

Threading Techniques

- NSThread / NSRunloop
- Grand Central Dispatch
- NSOperationQueue/NSOperation

NSThread/NSRunLoop

- Not well supported in Swift (needs an ObjC interface to take full advantage)
- Necessary if you need a specific thread to stay active, e.g. when:
  ** Handling sockets directly (e.g. non-HTTP protocols)
  ** Handling external accessories
  ** Utilizing NSTimers in background threads
- Allows interleaving tasks in a thread (like Ruby `yield`)

Grand Central Dispatch

- C-language threading library
- Queue handling system for iOS and OS/X
- Abstracts away thread management as queues
- Allows simple dispatch of closures into queues
- Manage QoS for various queues
- NB Queues != Threads

NSOperationQueue and NSOperation

- Swift/ObjC Objects rather than C functions
- Abstracts away GCD queues
- Essentially a queue of closures with dependencies
- FIFO execution of tasks with no outstanding dependencies