CS 47

Beginning iPhone Application Development

Week 6: Quartz, Animations and Touch Handling

V in MVC

- Today we are going to focus on the view component of the MVC framework
- Quartz 2D (QuartzCore)
- Animations
- Custom touch handlers

Quartz 2D

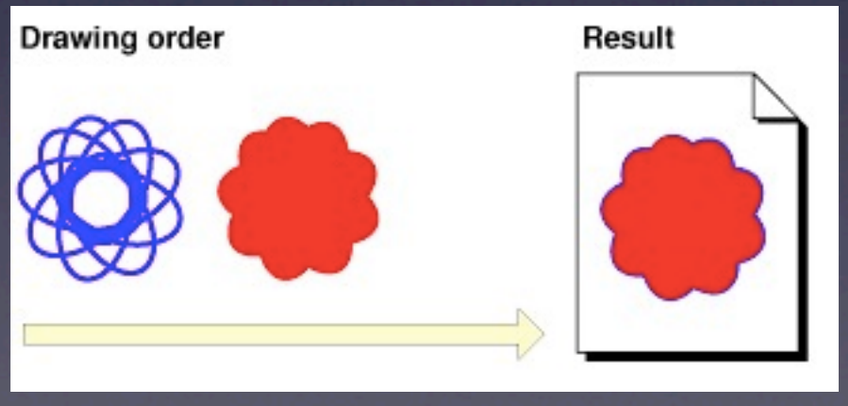
- What can you do with Quartz?
    - Draw custom graphics (shapes, lines, patterns, etc)
    - Provide graphics editing behavior (e.g. erase, cut/copy graphics, etc)
    - Image, PDF creation
- Standard UIViews use QuartzCore (drawRect:)

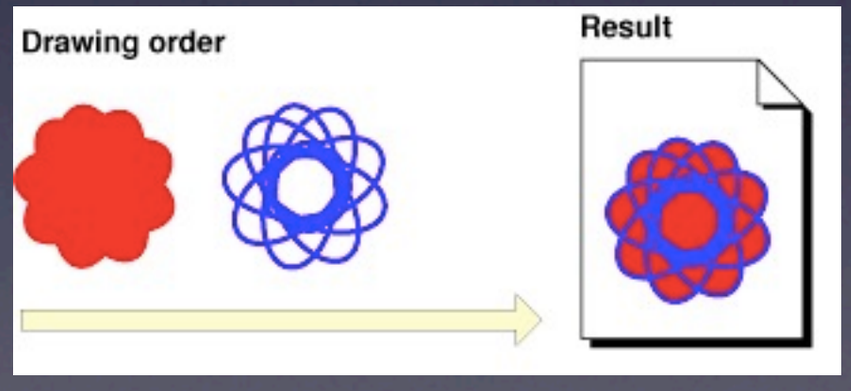
Quartz

- Some other typical uses
  - When you need a view that cannot be made through a combination of standard views (and more than just images)
  - Applying shading effects to dynamic strings (UILabel does not have blur)
  - Applying round corners and shading to images (e.g. portraits)

Quartz

- Quartz follows the painters model
  - Start with a blank canvas
  - Perform sequential operations, each immediately affecting the canvas state





Quartz

- How do we represent the canvas?
- CGContextRef
- Part of the CoreGraphics library (CG)
- The CGContextRef encapsulates whatever destination you are painting to (on iPhone, just screen or image)

CGContextRef

- Note: There are a boatload of functions in the CGContextRef family
- We can't go over them all
- Please read the CGContextRef API documentation for a thorough list of functions

CGContextRef

- How do we get a CGContextRef value?
- If you want the current "screen" context:

```
UIGraphicsGetCurrentContext
```

- If you want to draw a custom image:

```
UIGraphicsBeginImageContext
UIGraphicsGetImageFromCurrentImageContext
UIGraphicsEndImageContext
```

CGContextRef

- The typical usage of the context is inside the drawRect: method of a UIView subclass you create

```
- (void) drawRect:(CGRect)rect {
   CGContextRef context = UIGraphicsGetCurrentContext();
   ... stuff with context ...
}
```

- You can use the image context creation anywhere (does not need to be in a code area related to graphics)

CGContextRef

- The CGContextRef tracks many drawing states
  - Transform matrix, clipping area, line configuration, colors, text-drawing, blending mode, etc
- Drawing operations respect the immediate state of the context

CGContextRef

- Example of some states you can modify

```
CGContextGetInterpolationQuality
CGContextSetFlatness
CGContextSetInterpolationQuality
CGContextSetLineCap
CGContextSetLineDash
CGContextSetLineJoin
CGContextSetLineWidth
CGContextSetMiterLimit
CGContextSetPatternPhase
CGContextSetFillPattern
CGContextSetRenderingIntent
CGContextSetShouldAntialias
CGContextSetShouldSmoothFonts
CGContextSetStrokePattern
CGContextSetBlendMode
CGContextSetAllowsAntialiasing
```

CGContextRef

- You can take a snapshot of the current context state and save it on a stack

  `CGContextSaveGState`

- And then, after making changes to the state, you can pop a saved state off the stack and restore it

  `CGContextRestoreGState`

- Useful for iterating through “stamp” functions

Applying Paint

- When you apply paint, you need to specify:
- A geometry (line, rectangle, arc, path, text, etc)
  `e.g. CGContextFillRect vs. CGContextFillPath`
- Fill vs. stroke (solid color vs. outline)
  `e.g. CGContextFillRect vs. CGContextStrokeRect`

Creating a Path

- Think of a drawing a path on a piece of paper
- **Start:** `CGContextBeginPath`
- **Add routes:** `CGContextAddArc, CGContextAddLines, CGContextAddRect, etc`
- **Or lift the pen and move:** `CGContextRefMoveToPoint`
- **End:** `CGContextEndPath` **-or- fill/stroke the path**

Colors

- CoreGraphics represents color with the CGColorRef object
- Created from a CGColorSpaceRef, and space-specific components (example later)
- Often convenient to use the CGColor getter of a UIColor object

```
UIColor   *myRed  = [UIColor redColor];
CGColorRef redRef = myRed.CGColor;
```

Colors

- Set the color states with CGColorRef
  `CGContextSetFillColorWithColor`
  `CGContextSetStrokeColorWithColor`
- Set the color states with component array
  `CGContextSetFillColor`
  `CGContextSetStrokeColor`
- Set the color states with RGB components
  `CGContextSetRGBFillColor`
  `CGContextSetRGBStrokeColor`

Shadows

- Shadows have: color, offset and blur

  `CGContextSetShadowWithColor`

- When shadows are enabled, the shape is drawn first with the shadow parameters (special color, offset and blur), then drawn a second time with the normal parameters

- Turn shadows off by restoring state, or passing a NULL color

Shadows

- Increasing the blur value will increase the blur bleed radius, but decrease the intensity of the color
- If you want a larger blur radius with a more intense color, you will have to draw the shape multiple times with the proper blend mode

Blending

- CGContextSetBlendMode
- You will almost always use `kCGBlendModeNormal`
- The normal blend mode uses the alpha value of the source to blend with the destination
- Extensive examples in the Paths section of the Quartz 2D Programming Guide

Memory Management

- CG is a C API, but acts much like Objective-C memory management
- Any value you get from a “Create” or “Copy” function, you must call the corresponding “Release” function on.
- You can use the “Retain” functions to increment the reference counter

Memory Management

- Example

```
CGColorSpaceRef *colorSpace = CGColorSpaceCreateDeviceRGB();
CGColorRef *color = CGColorCreate(colorSpace, comps);

/* This does an implicit retain of the color */
myLayer.backgroundColor = color;

CGColorRelease(color);
CGColorSpaceRelease(colorSpace);
```

Drawing Custom Fonts

- With the standard UIKit views, you are limited to the fonts provided by apple (e.g. the fonts accessible with UIFont).
- With Quartz, you can draw any TrueType font - use this to make your own custom UILabel class

Drawing Custom Fonts

```
/* Create the CGFontRef from a .ttf file */

NSString* fontPath = [[NSBundle mainBundle]
                      pathForResource:fontName ofType:@"ttf"];

NSURL *fontURL = [NSURL fileURLWithPath:fontPath isDirectory:NO];

CGDataProviderRef fProv = CGDataProviderCreateWithURL((CFURLRef) fontURL);

/* Create the font reference object from the data provider */
CGFontRef fontRef = CGFontCreateWithDataProvider(fontProvider);

/* Release handle */
CGDataProviderRelease(fontProvider);
```

Drawing Custom Fonts

```
/* Now we have a CGFontRef object, let's apply it to our CGContextRef */
CGContextSetFont(context, fontRef);
CGContextSetFontSize(context, size);

/* We need to flip over the X-axis since it wants to draw upside down */
CGAffineTransform xfrm = CGAffineTransformMake(1.0, 0.0, 0.0, -1.0, 0.0, 0.0);
CGContextSetTextMatrix(context, xfrm);

/* We need to draw glyphs (normally, ASCII value - 29) */
CGGlyph _glyphStr[512];
const char *utfstr = [myText UTF8String];
for (int i = 0; i < [myText length]; i++) _glyphStr[i] = utfstr[i] - 29;

CGContextSetFillColorWithColor(context, textColor.CGColor);
CGContextShowGlyphsAtPoint(context, xPos, size, _glyphStr, [myText length]);
```

Animation

- There are a few ways to think about animation
  - Cycling images per frame (like an animated GIF file)
  - Changing the higher-level properties of a view over a period of time

Animation

- Image cycling (like an animated GIF) is usually accomplished with the UIImageView class - what is it good for?

```
Properties:
.animationImages
.animationDuration
.animationRepeatCount

Methods:
- startAnimating
- stopAnimating
```

Animation

- What you will use way more often is the concept of animating the structure of the view hierarchy
- Moving views around the screen smoothly, rotating them, fading them in and out, etc
- Think about what a UINavigationController does when switching screens

Animation

- Two general styles of high level animation
  - Modify the UIView objects (high level)
  - Modify the CALayer objects (low level)

UIView Animation

- You should be familiar with UIViews by now. Most of the UI elements you've worked with are subclass from UIView
- You can apply the generic UIView animations to any UIView
- Very simple, easy interface, but limited to a few types of animations

UIView Animation

- All UIView animations are done in blocks
- Animation blocks must begin with

```
[UIView beginAnimations:nil context:NULL];
```

  - You can name the animation/context if you want, but this necessary only if you need to track the lifecycle of multiple animations. Most animations are fire and forget

- Animation blocks must end with

```
[UIView commitAnimations];
```

UIView Animation

- UIView animation blocks can be modified by calling these class methods inside of the block

```
+ setAnimationStartDate:
+ setAnimationDelegate:
+ setAnimationWillStartSelector:
+ setAnimationDidStopSelector:
+ setAnimationDuration:
+ setAnimationDelay:
+ setAnimationCurve:
+ setAnimationRepeatCount:
+ setAnimationRepeatAutoreverses:
+ setAnimationBeginsFromCurrentState:
```

UIView Animation

- UIView animation blocks can be nested, creating a stack of animation blocks
- Animation blocks are executed when the corresponding commitAnimations method is called
- Setting animation parameters affects the block on the top of the stack

UIView Animations

- So what properties can I animate about a UIView?

```
frame      - Change the rectangle of the view
bounds     - Same as above, but relative to view
center     - Move the center relative to parent
transform  - Scale, rotate, transform
alpha      - Change opacity/transparency
```

- It's a short list, but you can still achieve 90% of animations you'd want (move, resize, rotate, fade)

CoreAnimation

- What if want more fine grain, programmatic control of animation?
- Need to apply CAAnimation objects to CALayers
- CA = CoreAnimation, part of QuartzCore

CoreAnimation

- UIViews are essentially wrappers for their underlying layers

- You can access the UIView's layer with the .layer accessor

```
CALayer *myLayer = myView.layer;
```

CoreAnimation

- Layers have their own hierarchy

```
CALayer *myLayer  = myView.layer;
CALayer *newLayer = [CALayer layer];
[myLayer addSublayer:newLayer];

/* DON'T MIX AND MATCH UIView HIERARCHY
   WITH CALayer HIERARCHY - THIS IS BAD: */
CALayer *myLayer  = myView.layer;
CALayer *myLayer2 = myView2.layer;
[myLayer addSublayer:myLayer2];
```

CoreAnimation

- CALayer objects have many animatable properties
- Too long to list - check the Core Animation Programming Guide (section: “Layer Style Properties”)

CoreAnimation

- Modifying a layer's property directly performs an implicit animation

```
CALayer *myLayer = myView.layer;

/* Animates to red using default animation parameters */
myLayer.backgroundColor = [UIColor redColor].CGColor;

/* Moves to new position using default animation params */
myLayer.position = CGPointMake(50.0, 50.0);
```

CoreAnimation

- Use CATransaction to nest implicit animations (just like UIView animation)

```
[CATransaction begin];

[CATransaction setValue:[NSNumber numberWithFloat:2.0f]
                 forKey:kCATransactionAnimationDuration];

theLayer.position = CGPointMake(0.0,0.0);

[CATransaction end];
```

CoreAnimation

- Or you can explicitly define a CAAnimation object and apply it to the layer

```
CABasicAnimation *animation;
animation = [CABasicAnimation animationWithKeyPath:@"position"];
animation.delegate  = self;
animation.duration  = 0.25;
animation.fromValue = [NSValue valueWithCGPoint:(myLayer1.position)];
animation.toValue   = [NSValue valueWithCGPoint:(p)];
[myLayer1 addAnimation:animation forKey:@"animatePosition"];
```

- Important: the `animationWithKeyPath` argument must be the name of an animatable property (e.g. position, backgroundColor, opacity, etc)

CoreAnimation

- Notice that I assigned a delegate to the previous animation?
- The delegate receives these protocol messages

```
- (void)animationDidStart:(CAAnimation *)theAnimation
- (void)animationDidStop:(CAAnimation *)theAnimation finished:(BOOL)flag
```

- Good for chaining animations, or taking an action after an animation is complete

CoreAnimation

- Just Mentioning: CAKeyframeAnimation
- Used to animate through a specific path at various time intervals

Handling Touches

- Shifting gears to touch handling
- We already know how to catch generic events in generic UIViews with the addTarget method (like a button press)
- But what if we want fine-grain touch tracking?

Handling Touches

- If you want to monitor all touch events, you must subclass UIView and implement these methods

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
```

- If you want to track multiple events, be sure to set multipleTouchEnabled to YES for the UIView

```
myView.multipleTouchEnabled = YES;
```

Handling Touches

- In the touch handler, query the event for touches belonging to your view

```
NSArray *myTouches = [[event touchesForView:self] allObjects];
/* Remember: self is the UIView we're subclassing */
```

- You can get up to 5 touch objects in this array if multipleTouchEnabled is true
- Extract the UITouch objects from this array

```
UITouch *myTouch = [myTouches objectAtIndex:0];
```

Handling Touches

- Once we have the touch object, we can query for its location

```
UITouch *touch = [myTouches objectAtIndex:0];
CGPoint currentLocation = [touch locationInView:self];
CGPoint previousLocation = [touch previousLocationInView:self];
```

- You can repeat this for each touch to get the current and previous position of each one

Handling Positions

- What you do with the touch locations is entire up to you
- e.g. Use your subclass to track things like distance, swipe speed, rotating touches in a circle, etc
- You could use that info to directly manipulate graphics contained in your view, or...

Handling Positions

- Think MVC: Your view should have as little specific logic as possible
- It may make more sense to implement a protocol to send gesture handling back to a controller
- Let the controller process the gestures and update your view accordingly