Core Animation

A layer deeper

Animation under the hood

Core Animation is the framework that UIKit uses to perform animations.

UIView animations are built on Core Animation

It’s got a bunch of classes and protocols that work together, but underlying everything are CALayer and CAAnimation.

CALayers: behind views

While we've been working with views up until today, CALayers are the objects that actually draw on screen.

Layers store information about what to draw. For most, that's a bitmap. When you were drawing a custom view, that drawing code made a bitmap which was then given to a layer for display.

Some layers draw shapes, which means they generate bitmaps from points and paths.

CALayer Subclasses

CAEmitterLayer: throw a bunch of particles onscreen
CAGradientLayer: show a nice gradient, with animation.
CAReplicatorLayer: make copies with transforms
CAScrollLayer: like UIScrollView, but a layer
CAShapeLayer: draw a shape path, can animate changes
CATextLayer: basic text drawing
CATiledLayer: how we draw those tiles on a map
CATransformLayer: view sublayers in 3D, not flattened
And **more...**

CAAnimation

CAAnimation describes an animation. It has:

- A timing function, for custom animation curves
- A delegate, for notifying on start/stop
- NSCopying methods, so animations can be copied and given to each version of the layer.
- duration and speed properties

Subclasses of CAAnimation are used for different cases.

CAAnimation Subclasses

- CABasicAnimation
  - Animate a single property from one value to another
- CAKeyframeAnimation
  - Animate a single property along a series of values
- CAAnimationGroup - group CAAnimations
- CATransition - transition between two views

A CABasicAnimation. Very basic.

To define a basic animation, we need to create a CABasicAnimation object, set a few properties, and add it to a layer.

```
CABasicAnimation* fadeAnim = [CABasicAnimation
animationWithKeyPath:@"opacity"];
fadeAnim.fromValue = @(1.0);
fadeAnim.toValue = @(0.0);
fadeAnim.duration = 1.0;
[theLayer addAnimation:fadeAnim forKey:@"opacityAnim"];
```

Presentation vs model

There are two ‘versions’ of a layer: The presentation layer and the model layer.

The presentation layer is the one actually shown on screen, and the model layer is the one with the current values of the layer. That means that if you ask a layer for a value mid-animation, it will return the value of the model, not the presentation, and it won’t match what’s on screen. You can ask a layer for either its model version or its presentation version.

What can you animate?

anchorPoint, backgroundColor, borderColor, borderWidth, bounds, cornerRadius, frame, mask, opacity, position, shadow, transform, zPosition...

Animations work on key paths, a string of period-delimited property names.

E.g. - @“bounds.origin.x”

Timing is everything

Animations have a durations and a speed.

Speed is a multiplier that affects the duration.

You can set it to 2 to be half as long, 0.5 to be twice as long, and to 0 for not changing at all.

You can also set the offset of the animation, so that it starts part way through. Layers also have a speed.

Keyframe animations

CAKeyframeAnimation is used for multi-step animations.

Give it an array of values, an array of corresponding times from 0.0 to 1.0, and it calculates the inbetween steps between each value.

It also can take an array of timing functions, to describe the rate as it moves between each value.

Animating along a path

CAKeyframeAnimation has the ability to animate using a CGPath. This means you can animate along curves.

Use CAKeyframeAnimation's path property instead of its values property.

To have the layer rotate along the path, set the rotationMode property to kCAAnimationRotateAuto or AutoReverse.

LIGHTHOUSE LABS

Animation delegate

Animations have delegates too, and you can use them to get callbacks when an animation starts and stops.

It’s an informal delegate. That means no protocol. What?!

You can also set up blocks to run when a whole animation *transaction* is complete.

Transactions

CATransaction groups a bunch of animations together into a single transaction, so they are all added to the animation system in a single atomic action.

They can be used to add completion blocks to your animations.

Unlike CAAnimationGroup, animations added as part of a transaction can be removed individually