CALayers

Hands-on Challenges

CALayers Hands-On Challenges

Challenge C: Animated Layers

So far the control looks great, but the foreground layer doesn't move or animate as you change the control's value.

Luckily, this is super easy, thanks to the power of Core Animation! One of the key advantages of using `CALayers` is how easy they are to animate; in fact almost every property on a `CALayer` (such as `borderWidth` and `cornerRadius`) is animatable.

This includes properties on CAShapeLayer, such as `strokeBegin` and `strokeEnd`. Currently we've hard-coded the `strokeEnd` on the `fgLayer` to 0.5. Our goal is to make an animation every time the number of chapters updates. We want to animate from its current `strokeEnd` to the desired `strokeEnd` (based on percentage of chapters read).

Open **StatView.swift** and add this new method at the end of the class:

```
private func animate() {
  var fromValue = fgLayer.strokeEnd
  let toValue = curValue / range
  let percentChange = abs(fromValue - toValue)
}
```

Here you set `fromValue` to where the stroke currently is, `toValue` to the percentage of chapters read, and also calculate the absolute value of the percentage changed.

Next add this to the bottom of `animate()`:

```
// 1
let animation = CABasicAnimation(keyPath: "strokeEnd")
animation.fromValue = fromValue
animation.toValue = toValue
// 2
animation.duration = CFTimeInterval(percentChange * 4)
// 3
fgLayer.removeAnimationForKey("stroke")
fgLayer.addAnimation(animation, forKey: "stroke")
```

Let’s go over this section by section:

1. This is how you create an animation on a layer property. You create a `CABasicAnimation`, passing in the name of the layer property you wish to animate. You then pass in the from and to values.
2. Here we base the duration based on the amount changed, so that the longer the bar has to move, the longer it takes (i.e. so the movement is at a constant rate no matter how long the bar is).
3. Finally you remove any existing animation named “stroke” (to avoid running two animations at once) and then add the animation to the foreground layer named “stroke” (to make it run).

You need to tweak a few more things before you can test this. First modify the `curValue` property at the top of the class to the following:

```
var curValue: CGFloat = 0 {
  didSet {
    animate()
  }
}
```

Now when `curValue` changes you call `animate()` rather than `configure()`, to trigger the animation.

Second inside `setup()` set the `fgLayer`’s `strokeEnd` back to 0:

```
fgLayer.strokeEnd = 0
```

You want the animation to start at 0 each time the view loads; it was only set to 0.5 earlier for testing purposes.

Finally move the line below from the top of `configure()` to the top of `animate()`:

```
percentLabel.text = String(format: "%.0f/%.0f", curValue, range)
```

Build and run, and at first the animation seems to work great, except after the animation completes, everything disappears. What gives?

Presentation Layers and Implicit Animations

When you run an animation on a layer, the animation actually takes place on a special sublayer called the presentation layer. Think of this as a copy of the layer that is used for animations only.

The property you are animating changes over time on the presentation layer, but when the animation is complete, the presentation layer disappears and you only see the original layer. The animation does not automatically update the value of the property on the original layer; that's up to you.

To fix this, simply add this line to the end of `animate()`:

```
fgLayer.strokeEnd = toValue
```

Now you'll see it (mostly) works, except that in a few books which don't have many chapters (such as the iOS Apprentice and Core Data by Tutorials), there's a strange 'wiggle' action when you change the chapters up and down.

This is because changing a layer property (like `strokeEnd`) actually triggers something called an implicit animation. This is the automatic animation Apple triggers for you when you change the value. This can be handy sometimes, but since you're already running your own animation you want to disable this so they don't conflict.

To do this, replace the line you just added with the following:

```
CATransaction.begin()
CATransaction.setDisableActions(true)
fgLayer.strokeEnd = toValue
CATransaction.commit()
```

Build and run again, and you'll see the animation is quite smooth.

There's just one problem left – try opening the iOS Apprentice and before the animation completes, hit the – sign. Notice how it starts animating from the completed position rather than the current animation value?

This is because you are currently setting the `fromValue` to `fgLayer.strokeEnd`, but remember that this might not match up to what you are seeing in the presentation layer. To fix this, add these lines right before you set `percentChange`:

```
if let presentationLayer = fgLayer.presentationLayer() as? CAShapeLayer
{
  fromValue = presentationLayer.strokeEnd
}
```

This checks to see if the presentation layer is active, and if so gets the stroke end from that as the starting point, rather than what the stroke end will be after the animation completes.

Build and run, and the animation now proceeds smoothly in all cases. You are now a layer animation pro, and you can take what you learned and apply it to any layer property! :]

> **Note**: If you'd like to learn more about cool layer animations like this, check out our iOS Animations with Swift video tutorial series, or our book iOS Animations by Tutorials.