Core Frameworks Series

[Simplified Animation Techniques for Mac and iPhone Development]

CORE ANIMATION

marcus ZARRA
matt LONG

Praise for *Core Animation*

"[This book is] a neat introduction to Core Animation. Both beginners and advanced developers will find many useful nuggets of information and techniques in this book."

—Brian Christensen

"*Core Animation* has prepared me for the future of user interface programming on Macs and iPhones—and I'm glad it's here because the future is now."

—Brent Simmons, NetNewsWire Developer

"Anyone endeavoring to undertake animation of any significance can benefit from owning a copy of this book. Marcus Zarra and Matt Long provide a much-needed guide to the basics of simple and grouped animations along with stylistic guidelines for when and how they should be used. However, it is the treatment of the book's advanced material that will keep this book relevant to the developer long after it has been used to master the basics."

—Daniel Pasco, CEO, Black Pixel

Summary

Basic animation is known as *single keyframe animation*. This definition helps to explain what keyframe animation provides. The keyframes in an animation are like a list of what `CABasicAnimation` refers to as its `toValue` field: the destination value. A list of these values can be supplied to the keyframe animation using either the animation's `values` field or its `path` field. Each of these values are destinations that the animation reaches at some point in its duration.

Keyframe animation provides a powerful way to animate any animatable layer property using a simple list of values, making it easy for the developer because the in-between values are automatically interpolated. This simplicity makes it a snap to create animation-based user interfaces informative, intuitive, and easy to implement.

PART III

Core Animation Layers

IN THIS PART

This page intentionally left blank

CHAPTER 5

Layer Transforms

IN THIS CHAPTER

- Scale Transform
- Using -rotateTransform:
- Using -rotate3DTransform:
- Anchor Points
- Combining Transforms
- Scale Versus Bounds

Up to this point, we have discussed how to move elements around the screen, change their color, and various other interesting effects. In this chapter, we take that quite a bit further. Transforms is a catchall to describe applying a matrix transform to a layer for some startling results.

What is a transform? A transform is a term used to include any function that alters the size, position, or rotation of an object, in our case a layer. Transforms scale a layer up or down and rotate a layer along one or more planes. Transforms are applied using a matrix function that fortunately we do not need to interact with directly. Whenever we want to rotate or scale a layer, we must use a transform to accomplish the desired effect.

The topic of matrix transforms can quickly turn into a deeply mathematical conversation that is beyond the scope of this chapter. Instead, this chapter touches on a few of the more common and interesting transforms, such as rotating a layer in 3D space or creating interesting zoom effects, and how to bring them about.

NOTE

Cocoa Touch

Unless otherwise specified, all the transforms discussed in this chapter can be performed both on the desktop and on any device that uses Cocoa Touch. However, it should be noted that transforms can be computationally-intensive, and on a device running Cocoa Touch, it is wise to test the performance of the animation to confirm that it is within acceptable boundaries.

Scale Transform

To demonstrate some of the capabilities of matrix transforms we take a simple layer and perform several different transforms on it. The first transform scales the layer from one size to another. To start this example, build the layers shown in Listing 5-1.

LISTING 5-1 applicationDidFinishLaunching

```
- (void)applicationDidFinishLaunching:(NSNotification*)notification
{
  NSView *contentView = [[self window] contentView];
  CALayer *layer = [CALayer layer];
  CGColorRef color;
  color = CGColorCreateGenericRGB(0.0f, 0.0f, 0.0f, 1.0f);
  [layer setBackgroundColor:color];
  [contentView setLayer:layer];
  [contentView setWantsLayer:YES];

  workLayer = [CALayer layer];
  color = CGColorCreateGenericRGB(0.5f, 0.5f, 0.5f, 1.0f);
  [workLayer setBackgroundColor:color];

  [workLayer setCornerRadius:5.0f];

  color = CGColorCreateGenericRGB(0.0f, 1.0f, 0.0f, 1.0f);
  [workLayer setBorderColor:color];
  [workLayer setBorderWidth:2.0f];

  CGRect workFrame = [layer bounds];
  workFrame.origin.x = workFrame.size.width / 4;
  workFrame.origin.y = workFrame.size.height / 4;
  workFrame.size.width /= 2;
  workFrame.size.height /= 2;
  [workLayer setAnchorPoint:CGPointMake(0, 0)];
  [workLayer setFrame:workFrame];
  [layer addSublayer:workLayer];
}
```

In the `-applicationDidFinishLaunching:` method, we grab a reference to the `contentView`, set its layer and flag it as layer backed. By setting the layer, we are guaranteeing what type of layer the view uses for its backing.

When the `contentView` is set up properly, we next construct the layer that will be manipulated. Its background color is set to gray, and the corners are rounded using `setCornerRadius:`. Next, the border color is set to green with a width of 2 pixels. Finally,

the layer's frame is set to be a quarter the size of the `contentView` and centered onscreen.

In Interface Builder, add three buttons to the window; one for each transform that we perform: Scale, Rotate, and 3D Rotate. The resulting window is shown in Figure 5-1.

NOTE

On the iPhone, the layer is already in place. You need to override the class method `+layerClass` instead to control the layer that is used.

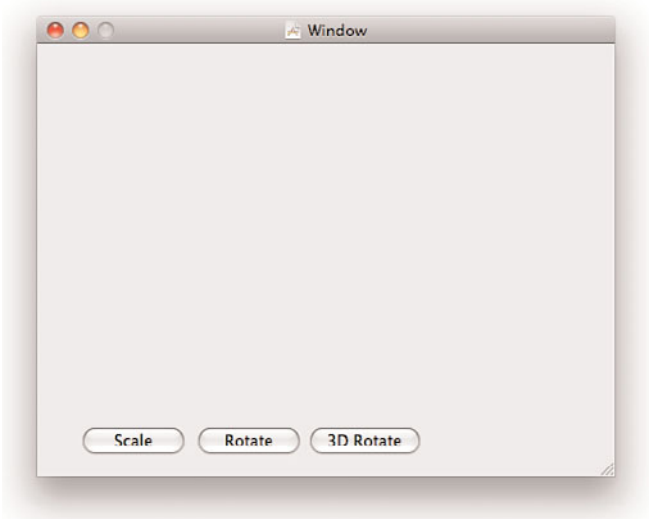

FIGURE 5-1 Interface Builder Window

The Scale button is bound to the method `-scaleTransform:`, implemented as shown in Listing 5-2.

LISTING 5-2 -scaleTransform:

```
- (IBAction)scaleTransform:(id)sender
{
  NSValue *value = nil;
  CABasicAnimation *animation = nil;
  CATransform3D transform;

  [[self workLayer] removeAllAnimations];
  animation = [CABasicAnimation animationWithKeyPath:@"transform"];
  transform = CATransform3DMakeScale(0.5f, 0.5f, 1.0f);
  value = [NSValue valueWithCATransform3D:transform];
  [animation setToValue:value];
  transform = CATransform3DMakeScale(1.0f, 1.0f, 1.0f);
```