Core Animation:

Simplified Animation Techniques for Mac and iPhone Development

第三部分

核心动画的层

第五章

层的变换

版本 1.0

翻译时间：2012-11-12

DevDiv 翻译：animeng
DevDiv 校对：symbian_love BeyondVincent(破船)
DevDiv 编辑：BeyondVincent(破船)

写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com 移动开发论坛特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 DevDiv 移动开发论坛。

技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和技术需要支持的话，请访问网站 www.devdiv.com 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助你。

关于本文的翻译

感谢 animeng 对本文的翻译，同时非常感谢 symbian_love 和 BeyondVincent(破船)在百忙中抽出时间对翻译初稿的认真校验。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

推荐资源

iOS

iOS 5 Programming Cookbook 中文翻译各章节汇总

iOS6 新特征：参考资料和示例汇总

Android

DEVDIV 原创 ANDROID 学习系列教程实例

Windows Phone

Windows Phone 8 新特征讲义与示例汇总

Windows 8

Building Windows 8 apps with XAML and C#中文翻译全部汇总

Building Windows 8 apps with HTML5 and JavaScript 中文翻译汇总

Windows 8 Metro 开发书籍汇总

Windows 8 Metro App 开发 Step by Step

其它

DevDiv 出版作品汇总

目录

本书翻译贴各章汇总

Core Animation 中文翻译各章节汇总

第一部分 核心动画开篇

1.1.1. 第一章 什么是核心动画

1.1.2. 第二章 我们可以和应该做哪些动画

1.1.3. 第三章 **Core Animation 中文翻译_第三章_基础动画**

1.1.4. 第四章 **Core Animation 中文翻译_第四章_关键帧动画**

第 5 章　层的变换

到现在，我们已经讨论了如何在屏幕周围移动一些几何元素，改变颜色，和多种多样其他有趣的效果。在这一章中，我们将进一步讨论。转换器是一个容器，用来描述应用到层上的一些矩阵转换，从而达到一些惊奇的效果。

变换又是什么呢？一个变换是一个术语，它包含了一些改变尺寸，位置或者更多面旋转一个物体的功能。变换是对应用使用一个矩阵函数，幸运的是我们不需要关心这个矩阵函数。无论我们需要旋转或者缩放一个层，我们必须使用一个变换去完成这个渴望的效果。

矩阵变换这个话题可以涉及到一个很深的数学问题上，这个超越了我们本章的范围。相反，这一章触及了一些更加普通和有趣的变换，例如在 3D 空间旋转一个层，或者一个有趣的缩放效果，这些如何发生。

5.1.1.　缩放变换

为了展示一些矩阵变化的效果，我们使用一个简单的层，然后进行一些变换。第一个变换是缩放一个层从一个尺寸到另一个尺寸。开始做这个例子，如清单 5-1

```
- (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];
}
```

清单 5-1

在-applicationDidFinishLaunching:方法中，我们获得了 contentView 的一个引用，设置了它的层和标记它作为层的背景。通过设置这个层，我们保证了那个视图背景使用了什么类型的层。

当 contentView 是被安装好时，我们下一步就就是构造这个我们需要操作的层。它的背景颜色设置为灰色，并且通过使用-setCornerRadius:设置角是圆形的。下面，把边框的颜色用 2 个像素设置为绿色的。最后，那个层的大小设定为 contentView 的 1/4，剧中显示。

在 Interface builder 中，给窗口增加 3 个按钮；我们要演示的每个变换：缩放，旋转，和 3D 旋转。结果窗口显示如下

图 5-1 窗口

scale 按钮是关联到方法-scaleTransform:，实现如下清单 5-2

```
- (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);
value = [NSValue valueWithCATransform3D:transform]; [animation setFromValue:value];
[animation setAutoreverses:YES]; [animation setDuration:1.0f]; [animation setRepeatCount:100];
[workLayer addAnimation:animation forKey:kScaleKey]; }
```

清单 5-2 缩放转换

这个方法会移除所有已经存在的动画，如果用户点击了好几个按钮，那些存在的动画就会堆积起来。下一步，我们想构造一个动画对象，那可以指导层怎么去变换。为了做这些，构造一个 CABasicAnimation 使用变换路径，这将告诉动画它要改变 transform 这个属性，无论这个层是否被应用，然后开始应用矩阵的变换到动画中。你可以使用很多种方法构造矩阵变换。这里的方法，我们使用 CATransform3DMakeScale 这个方法，需要传递 x,y,z 轴用来完成转换。就像列表 5-2 所示，我们设定 CATransform3DMakeScale 的 x，y 分别为 0.5，然后我们单独留下 z 轴。为了创建抖动效果，下一步就要设定 CATransform3DMakeScale 为 1.0。通过设定 x 和 y 为 1.0，我们就可以触发放大恢复的效果。

当 CATransform3DMakeScale 值被设定后，设定 autoReverse 为 YES，给它设定一个 1 秒的慢的时间段，然后设定一个最大的重复数；这里我们使用 100。最后，我们增加创建好的动画到层上面，通过使用关键字指派动画。我们要使用一个大一点重复数，以至于给于一个动画一直运行的假象。如果动画真的遍历到了 100 遍，那么就会停止。

5.1.2. 使用旋转变换

下一个变换我们来旋转层。层就沿着一个轴旋转，然后就自动恢复这个旋转，如清单 5-3 所示

```
- (IBAction)rotateTransform:(id)sender; {
NSValue *value = nil;
CABasicAnimation *animation = nil; CATransform3D transform;
[[self workLayer] removeAllAnimations];
animation = [CABasicAnimation animationWithKeyPath:@"transform"]; transform = CATransform3DMakeRotation(1.57f, 0.0f, 0.0f, 1.0f); value = [NSValue valueWithCATransform3D:transform];
[animation setToValue:value];
transform = CATransform3DMakeRotation(0.0f, 0.0f, 0.0f, 1.0f); value = [NSValue valueWithCATransform3D:transform];
[animation setFromValue:value];
[animation setAutoreverses:YES];
[animation setDuration:1.0f];
[animation setRepeatCount:100];
[workLayer addAnimation:animation forKey:kScaleKey];
}
```

清单 5-3 旋转转换

在这个例子中，CATransform3DMakeRotation 被应用沿着一个轴在层上旋转。不像清单 5-2 所展示的那样，这个例子用了 4 个参数：第一个参数是角度，用弧度表示，和下面三个数字是 x，y 和 z 轴。层是在 x 轴上被旋转 1.57 个弧度（那是 90 度）。x,y,z 这些值有些不寻常。这个值涉及到了旋转的向量值，这些值在-1.0 和 1.0 之间。旋转是被设置成全正的沿 z 轴，那就会产生一个顺时针的 2D 旋转。

5.1.3. 使用 3D 旋转

下面的例子，比先前的例子更进一步了，使用了 2 个轴的旋转；看清单 5-4

```
- (IBAction)rotate3DTransform:(id)sender; {
NSValue *value = nil; CABasicAnimation *animation = nil; CATransform3D transform;
[[self workLayer] removeAllAnimations];
animation = [CABasicAnimation animationWithKeyPath:@"transform"]; transform = CATransform3DMakeRotation(1.57f, 1.0f, 1.0f, 0.0f); value = [NSValue valueWithCATransform3D:transform];
[animation setToValue:value];
transform = CATransform3DMakeRotation(0.0f, 1.0f, 1.0f, 0.0f); value = [NSValue valueWithCATransform3D:transform];
[animation setFromValue:value];
[animation setAutoreverses:YES];
[animation setDuration:1.0f];
[animation setRepeatCount:100];
[workLayer addAnimation:animation forKey:kScaleKey];
}
```

清单 5-4

清单 5-4 的方法和清单 5-4 的方法接近，除了 CATransform3DMakeRotation 这个方法传递的值不一样。这个例子设置 x 和 y 轴为 1.0，这会产生一个旋转，这种旋转会给你一种沿着两个轴的对角线滑过的幻觉。

因为我们旋转层 90 度并且自动恢复，这个例子貌似是沿着 2 个轴滑动。这是有用的，当你有一个两面的层（如一个图标或者一个戳），这样就让你在两面可以翻转。

5.1.4. 锚点

关于层中先前已经提到过这个概念，在用变换时，锚点非常的重要。当你要给一个层应用一个变换时，变换会使用锚点来决定那一点来旋转，缩放等等。

到目前为止某些例子，缩放的变换（在清单 5-2 中展示的）会引起层在窗口的中间抖动，就像图 5-2 所示

的那样。这是因为任何层默认的锚点都是在中间。

然而，如果我们改变-applicationDidFinishLaunching:并且移动锚点到左下角，就像列表 5-5 所示的那样，我们就可以得到一个信服的结果。

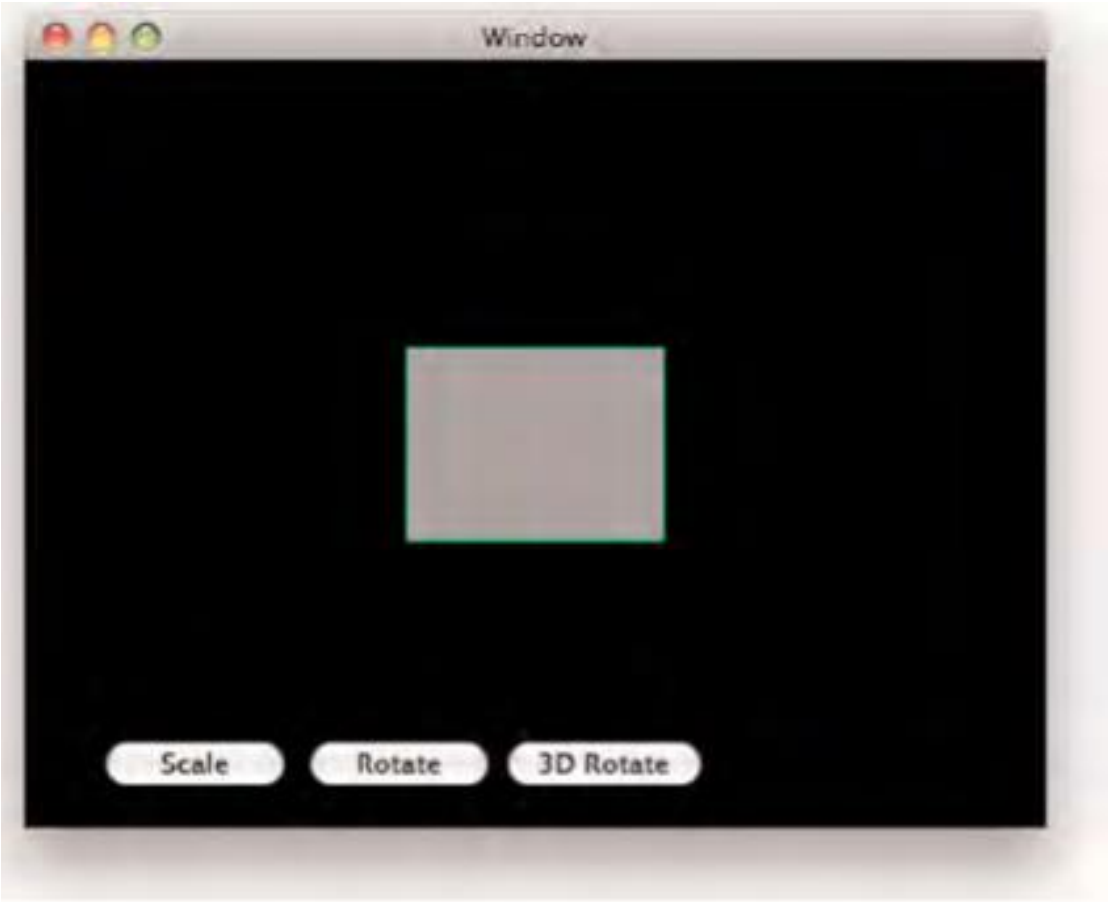

图 5-2 锚点在中间

```
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];
```

清单 5-5 升级锚点

在清单 5-5 中，我们增加了一行代码[worklayer setAnchorPoint:CGPointMake(0,0)];这将重新定位锚点到层的左下角。当缩放的变换运行时，你可以看到层好像在左下角抖动，就像图 5-3 所示的那样。

通过锚点和变换的联合控制，你可以生产一些有趣的结果。例如，我们想要在一个固定的角落中，让一个层沿着 z 轴旋转。通过移动锚点到希望的角落，这里旋转仅仅沿着 z 轴，那么这个层旋转就像是粘在一个东西上面一样。假如我们放置另一个层在同样的角落，这将有一个难以置信的效果。

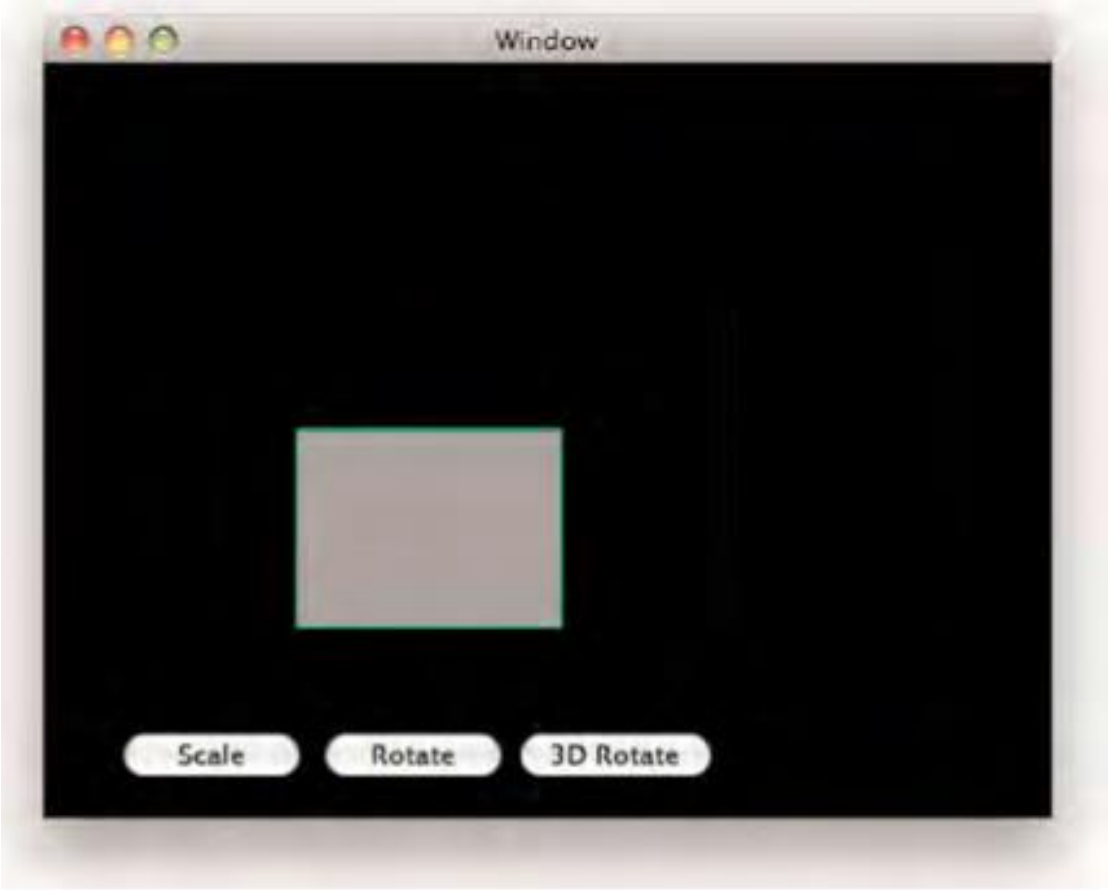

图 5-3 左下角的锚点

5.1.5. 联合变换

到目前为止，我们所涉及的都是单一的变换，要么旋转要么缩放层。但是如果你想要同时执行多个变换怎么办？幸运的是我们将会展示给你。

为了演示怎么来联合变换，我们建立了一个不同的工程。我们开始一个标准的 cocoa 应用程序从 xcode 模板中，然后增加一个 appDelegate。那个 appDelegate 会获得一个 NSWindwo 的引用计数，以便于可以控制它。然后，在-applicationDidfinishLauching:这个方法中，我们安装到层上，展示如清单 5-6.

```
- (void)applicationDidFinishLaunching:(NSNotification*)notification {
CGColorRef color;
NSView *contentView = [[self window] contentView]; CALayer *rootLayer = [CALayer layer];
color = CGColorCreateGenericRGB(0.0, 0.0, 0.0, 1.0); [rootLayer setBackgroundColor:color];
[contentView setLayer:rootLayer]; [contentView setWantsLayer:YES];
layer = [CALayer layer];
color = CGColorCreateGenericRGB(0.5f, 0.5f, 0.5f, 1.0f);
[layer setBackgroundColor:color]; [layer setCornerRadius:5.0f];
color = CGColorCreateGenericRGB(0.0f, 1.0f, 0.0f, 1.0f); [layer setBorderColor:color];
[layer setBorderWidth:2.0f];
[layer setBounds:CGRectMake(0, 0, 100, 100)]; [layer setPosition:CGPointMake(55, 55)];
[rootLayer addSublayer:layer]; }
```

清单 5-6 联合转换

-applicationDidFinishLaunching:方法获得了 contentView 的一个引用，然后在开启这个层的后面增加一个 CALayer 给它。rootLayer 的背景颜色也是黑的。

下一步，创建我们准备操控的那个层，并且设置它的颜色为灰绿色，2 像素的边框和 5 像素的圆角率。不

像清单 5-1，这个层会是 100x100 像素和位置在 contentView 的左下角。

在 interface Builder 中有另一个改变。增加一个方块按钮 NSButton 给 contentView，设置为透明，并且调整大小和窗口一样。这就使用户无论点击窗口的那一点，都可以触发我们下面要做的行动。

绑定这个大按钮的行动给 appDelegate，然后它的-(IBAction)action:(id)sender 这个方法就是我们下面要声明的。

当用户点击窗口时，我们想要 workingLayer 移动到右上角。伴随这个，我们也要旋转层 180 度平且缩放到原来尺寸的 1/10。在清单 5-7 就是这个问题的解决方法。

```
- (IBAction)action:(id)sender; {
NSRect frame = [[[self window] contentView] frame]; float x = frame.origin.x + frame.size.width - 30; float y = frame.origin.y +
frame.size.height - 30; CATransform3D rotate;
[CATransaction begin];
[CATransaction setValue:[NSNumber numberWithFloat:5.0f]
forKey:kCATransactionAnimationDuration];
[layer setPosition:CGPointMake(x, y)];
scale = CATransform3DMakeScale(0.1f, 0.1f, 1.0f);
rotate = CATransform3DMakeRotation(1.57f, 0.0f, 0.0f, 1.0f); [layer setTransform:rotate];
[layer setTransform:scale];
[CATransaction commit]; }
```

清单 5-7 行动

然而，当代码运行时，这个层移动和旋转，但是它不能缩放。这是因为调用-setTransform:这个方法仅仅能设置一个属性，之后最后设置的那个值将替换先前设置的值。尽管在一些情况下，这看起来是有用的。即使老的变换的移除和新的变换的设置也是 2 个动画，但是它没有影响我们例子中看到的效果。

因为变换覆盖了彼此，变换需要在应用到层上时，被联合起来。这个可以用 CATransform3DConcat 方法，这将用 2 个 CATransform3D 的引用，然后返回一个联合的 CATransform3D，例如清单 5-8.

```
- (IBAction)action:(id)sender; {
NSRect frame = [[[self window] contentView] frame]; float x = frame.origin.x + frame.size.width - 30; float y = frame.origin.y +
frame.size.height - 30; CATransform3D rotate;
CATransform3D scale; CATransform3D combine;
[CATransaction begin];
[CATransaction setValue:[NSNumber numberWithFloat:5.0f]
forKey:kCATransactionAnimationDuration];
[layer setPosition:CGPointMake(x, y)];
scale = CATransform3DMakeScale(0.1f, 0.1f, 1.0f);
rotate = CATransform3DMakeRotation(1.57f, 0.0f, 0.0f, 1.0f); combine = CATransform3DConcat(rotate, scale);
[layer setTransform:combine];
[CATransaction commit]; }
```

清单 5-8

在这个完整的-action:方法中，CALayer 最后应该留下来的位置在开始 CATransaction 块开始之前就计算好了。这确保了改变动画在同一期间执行。我们给 CATransaction 设定执行时间段，通过使用+setValue:forKey:这个方法，传递一个 kCATransactionAnimationDuration 这个关键字。

在 CATransaction 已经开始之后，我们可以在层上设定我们想要的动画属性，然后它就使用 CATransaction 的执行时间自动的执行动画。代替构造若干个 CABasicAnimation 对象然后应用到层中，你可以直接设定属性。

第一个设定的属性就是位置。因为锚点就在层的中间，你能容易的计算出这个位置离窗口的右上角 5 个像素的位置，然后设定这个位置属性给层。

下一步，你需要构造变换来应用到这个层中。开始，先构造 2 个子变换，例如：

创建一个缩放变换，缩放层到原来尺寸的 1/10。

构造一个旋转变换，旋转层 1.57 个弧度。

在这里带着这两个变换，下一步就是使用 CATransform3DConcat 方法来联合它们。

当这两个变换是被联合时，应用它们到层上，然后提交 CATransaction。结果就是那个层会优雅的从左下角移到右上角，缩放成原来尺寸的 1/10 并且移动时旋转 180 度。这给你的印象就像是随着层滑动到指定的位

置，从无到有的缩放。

5.1.6. 缩放 vs 边框

在联合变换的例子中，我们联合了缩放和旋转变换以达到渴望的效果。你可能说：为什么不仅仅用改变层的边框来替代，以避免联合变换的麻烦？

原因是在缩放层和改变它的边框之间有着非常重要的不同之处。当一个层缩放时，任然会认为层是一个原始的尺寸，并且在缩放被应用之前绘制原始的尺寸。然而，如果我们改变了尺寸，然后缩放层，层就看起来就不对了，因为它知道自己是一个不同的尺寸了。

例如，如果我们改变层的边框，在联合变换的例子中代替做一个层的变换，动画的结果将看起来如图 5-4.

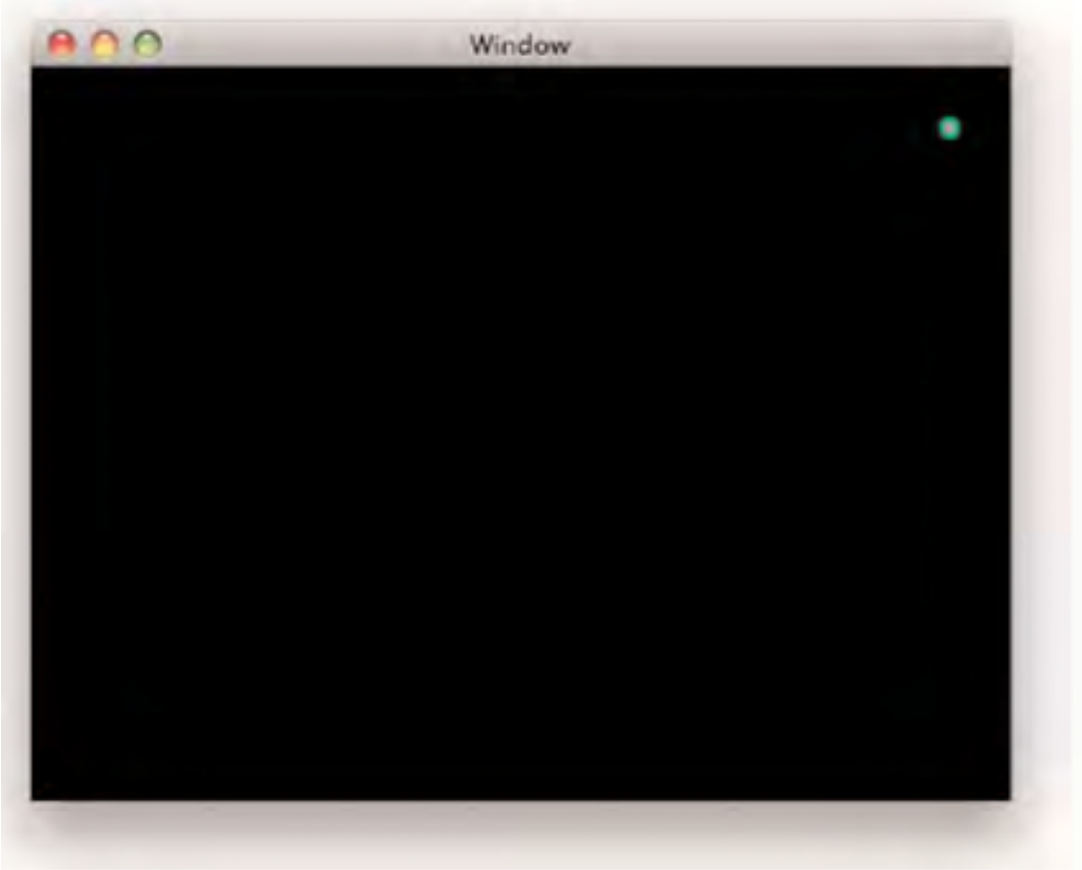

图 5-4 边框和缩放的动画

注意到最后层好像是一个圈。这是因为我们改变了层的边框，但是没有改变层的圆角率和边框的宽度。然而，当缩放变换被使用时，结果就是我们看到的那样，如图 5-5.这会是一个正方形。当你在看图 5-5 时，你会发现一个特点边框不可见。这是因为缩放后的边框比 0.5 个像素要小。同样，因为圆角(原始是 5 个像素)比 1 个像素要少，它也不再可见。当用一个复杂的层树工作时，改变边框和缩放的不同之处就富有戏剧性了。如果我们在放大这个层到 100%，那么边框就将成为 50 个像素宽了。

图 5-5 缩放和旋转的动画

5.1.7. 总结

当你第一次使用变换进行工作时，会令人生畏。更糟糕的是，如果你决定在互联网上搜索时，它会使你更加的迷惑。这里过了一遍概念希望能帮助你意识到变换的有用和便捷之处。