Core Animation:

Simplified Animation Techniques for Mac and iPhone Development

第三部分

核心动画的层

第十章

其它所有层

版本 1.0

翻译时间：2012-12-27

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

第 10 章 其它所有层

核心动画提供了很多种层，来帮助我们完成许多的任务。这一章讨论几个比较有用的层，包括：

CAShapeLayer,这个层提供了一个简单的可以使用核心图像路径在层树中组成一个阴影的方法。

CAGradientLayer,这个层你可以通过指定颜色，一个开始的点，一个结束的点和梯度类型使你能够简单的在层上绘制一个梯度。

CAReplicatorLayer，可以复制任何增加到层中的子层。这个复制的子层还可以被变换(在第 5 章讨论的层的变换)来产生一个耀眼的效果。

在这一章被讨论的层不是常用的层，但是它可以提供一些特殊的效果。当一个特效或者一些不常见的阴影需要时，这些层能用来完成目标。

CAShapeLayer

到目前为止，我们讨论的 CALayer 对象，他们都是矩形的。这些都是自然而然的，毕竟视图和窗口都是矩形的。然而有时我们想要层成为另一个形状，三角形或者圆形。在第一个核心动画的 release 版本中(iPhone2.x 和 Mac OS X10.5)，我们不得不创建一个透明的层，然后画自己需要的形状到矩形中。更远的说，如果我们需要在一个层上点击，我们需要做一个复杂的点击测试来决定是否该点击落在了我们渴望的区域中，还是落在了外面。

幸运的是，在 Iphone3.0 和 Mac OS X10.6 中核心动画是被升级了版本，有了另外的方法：CAShapeLayer，这个层可以解决这个问题。使用 CAShapeLayer，你可以通过创建一个核心图像路径，并且分配给 CAShaperLayer 的 path 属性，从而为需要的形状指定路径。并且你可以使用-setFillColor 方法给该形状指定一个填充颜色。

清单 10-1 演示了如何创建一个 CAShapeLayer，并且增加它到层树中。在这个例子中，一个新的 CAShapeLayer 是被创建，然后它是被构造成简单的三角形。

```
- (void)viewDidLoad {
[super viewDidLoad];
UIImage *balloon = [UIImage imageNamed:@"balloon.jpg"]; [[[self view] layer] setContents:(id)[balloon
CGImage]];
CGMutablePathRef path = CGPathCreateMutable(); CGPathMoveToPoint(path, NULL, 0, 100);
CGPathAddLineToPoint(path, NULL, 200, 0); CGPathAddLineToPoint(path, NULL, 200, 200);
CGPathAddLineToPoint(path, NULL, 0, 100);
shapeLayer = [[CAShapeLayer alloc] init];
[shapeLayer setBounds:CGRectMake(0, 0, 200, 200)]; [shapeLayer setFillColor:[[UIColor purpleColor]
CGColor]]; [shapeLayer setPosition:CGPointMake(200, 200)]; [shapeLayer setPath:path];
[[[self view] layer] addSublayer:shapeLayer]; }
```

清单 10-1 初始化一个 CAShapeLayer 并且增加它到层树上

在清单 10-1 中，你需要注意的是-viewDidload 方法，那告诉了我们这个代码是一个 iphone 的工程。我们通过调用[UIImage imageNamed:]方法，获取图像，然后设置给视图层的内容来展示图像。其次，通过调用 CGPathCreateMutable 方法创建一个核心图像的路径。我们增加线条来绘制矩形。下面我们创建一个 200x200 像素的形状层。我们用紫色填充，然后设定我们创建的路径。现在，当我们通过调用-addSublayer 增加层到层树上。你可以看到想图 10-1 中那样的图像。

图 10-1 CAShapeLayer 展示的三角形

尽管这个例子是运行在 iPhone OS 上，CAShapeLayer 在 Mac OS X10.5 中也是可用的。

操作路径笔画

CAShapeLayer 能够使你操控你已经绘制的形状的笔画。例如，你可以通过 lineWidth 属性设置笔画的宽度，或者你可以通过调用-setStrokeColor 传递一个 CGColorRef 来设置笔画的颜色。清单 10-2 演示了你可以改变的字段来控制笔画的形状。

```
- (void)viewDidLoad {
[super viewDidLoad];
UIImage *balloon = [UIImage imageNamed:@"balloon.jpg"]; [[[self view] layer] setContents:(id)[balloon
CGImage]];
CGMutablePathRef path = CGPathCreateMutable(); CGPathMoveToPoint(path, NULL, 0, 100);
CGPathAddLineToPoint(path, NULL, 200, 0); CGPathAddLineToPoint(path, NULL, 200, 200);
CGPathAddLineToPoint(path, NULL, 0, 100);
shapeLayer = [[CAShapeLayer alloc] init];
[shapeLayer setBounds:CGRectMake(0, 0, 200, 200)]; [shapeLayer setFillColor:[[UIColor purpleColor]
CGColor]]; [shapeLayer setPosition:CGPointMake(200, 200)]; [shapeLayer setPath:path];
[shapeLayer setStrokeColor:[[UIColor redColor] CGColor]]; [shapeLayer setLineWidth:10.0f];
[shapeLayer setLineJoin:kCALineJoinRound];
[shapeLayer setLineDashPattern:
```

```
[NSArray arrayWithObjects:[NSNumber numberWithInt:50], [NSNumber numberWithInt:2],
nil]];
[[[self view] layer] addSublayer:shapeLayer]; }
```

清单 10-2 路径笔画的控制

我们需要做的第一件事是设置笔画的颜色为红色，并且设置它的宽度为 10 像素。为了获得圆角路径，加入的 kCALineJoinRound 是被用来链接线段和圆角。圆角率是基于线的厚度和转角处线的角度，因而，不能直接调整。最后我们设定了线的虚线模式。在清单 10-2 中，用 50 单元的红线来画层，去创建虚线，留了 2 单元的区域不画，作为虚线的空白。

备注：什么是单元？
单位 unit 是被使用替代像素，是因为分辨率的独特性。苹果在 wwdc2006 上，给开发者介绍了分辨率独特性的概念，作为 Mac OS X 迁移特性的一部分。分辨率的独特性保证了无论该应用程序被使用在 iphone，ipod touch，macbook pro 或者是 30 英寸的 Apple Cinema HD 上显示看起来都一样，也无论用户的空白单元是什么。如果你使用了像素设定，你会看到从一个屏幕到另一个上面会有非常大的不同。

如果你要使用这个模式，下面有点复杂。当你使用时，奇数的值被绘制，然后偶数的值不被绘制。例如，如果你指定 5，10，15，20，笔画将会有 5 个单元被绘制，接下来 10 不被绘制，15 被绘制，20 不被绘制。这种模式可以使用你喜欢的间隙来指定。请记住：奇数等于绘制而偶数不绘制。这些单元是被放在了一个放置 NSNumber 对象的 NSArray 的数组中，如果你在 NSSArray 中放置其他东西，会带来一些异常的效果。

图 10-2 展示了形状层的绘制。圆角是不见了，因为在绘制模式中，圆角正好被空白代替了。

图 10-2 使用红色虚线的形状层

使用 CAShapeLayer 作为一个层的面罩

所有继承于 CALayer 的核心动画层都有一个属性叫做 mask.这个属性能够使你给层的所有内容做遮罩，除了层面罩中已经有的部分，它允许仅仅形状层绘制的部分显示那部分的图像。清单 10-3 中，如果你使用形状层作为一个面罩代替作为一个子层，这里展示了不同之处。

```
- (void)viewDidLoad {
[super viewDidLoad];
UIImage *balloon = [UIImage imageNamed:@"balloon.jpg"]; [[[self view] layer] setContents:(id)[balloon
CGImage]];
CGMutablePathRef path = CGPathCreateMutable(); CGPathMoveToPoint(path, NULL, 0, 100);
CGPathAddLineToPoint(path, NULL, 200, 0); CGPathAddLineToPoint(path, NULL, 200, 200);
CGPathAddLineToPoint(path, NULL, 0, 100);
shapeLayer = [[CAShapeLayer alloc] init];
[shapeLayer setBounds:CGRectMake(0, 0, 200, 200)]; [shapeLayer setFillColor:[[UIColor purpleColor]
CGColor]]; [shapeLayer setPosition:CGPointMake(200, 200)]; [shapeLayer setPath:path];
[[[self view] layer] setMask:shapeLayer]; }
```

清单 10-3 使用 CAShapeLayer 作为一个层的遮罩

初始化的代码是被定义在清单 10-1 中。清单 10-3 不同之处是我们改变了调用的方法，从-addSublayer 到-setMask 并且传给它一个 CAShapeLayer。图 10-3 展示了在应用了这个改变之后视图展示的效果。

图 10-3 使用一个层的面罩来创建一个三角形图片的区域

当使用形状层作为面罩时，笔画的操作可以被使用。事实上，如果你改变清单 10-2 的代码设定层的面罩代替清单 10-3 中增加一个子层的话，视图将会看起来想图 10-4 那样。看清单 10-4 特殊代码的改变。

```
- (void)viewDidLoad {
[super viewDidLoad];
UIImage *balloon = [UIImage imageNamed:@"balloon.jpg"]; [[[self view] layer] setContents:(id)[balloon
CGImage]];
CGMutablePathRef path = CGPathCreateMutable(); CGPathMoveToPoint(path, NULL, 0, 100);
CGPathAddLineToPoint(path, NULL, 200, 0); CGPathAddLineToPoint(path, NULL, 200, 200);
CGPathAddLineToPoint(path, NULL, 0, 100);
shapeLayer = [[CAShapeLayer alloc] init];
[shapeLayer setBounds:CGRectMake(0, 0, 200, 200)]; [shapeLayer setFillColor:[[UIColor purpleColor]
CGColor]]; [shapeLayer setPosition:CGPointMake(200, 200)]; [shapeLayer setPath:path];
[shapeLayer setStrokeColor:[[UIColor redColor] CGColor]]; [shapeLayer setLineWidth:10.0f];
```

```
[shapeLayer setLineJoin:kCALineJoinRound];
[shapeLayer setLineDashPattern:
[NSArray arrayWithObjects:[NSNumber numberWithInt:50], [NSNumber numberWithInt:2],
nil]];
[[[self view] layer] setMask:shapeLayer]; }
```

清单 10-4 使用 CAShapeLayer 作为一个面罩

就像你在图 10-4 中看到的，笔画的操作都影响了图像的面罩，改变了边缘成了锯齿状的形状。

图 10-4 使用笔画层作为层的面罩

10.1.1. CAGradientLayer

你在 iphone 上或者 Mac OS X 上经常看到一张图片的倒影效果。这种效果是困难去做的，但是用 CAGradientLayer 就很简单了。

创建一个倒影的基本方案是（就像在 iChat 或者在 iWeb 展示的图片）使用主图片的一个拷贝翻转放置到主图片的下方。然后你应用一个梯度到翻转的图片上，像是背景的阴影一样，如图 10-5.

在清单 10-5 中实例代码创建了 3 个层。一个是主图片和另一个倒影图片，然后应用一

个梯度层作为倒影层的一个面罩放置到底部。

图 10-5 图像倒影的梯度层

```
- (void)viewDidLoad {
[super viewDidLoad];
[[[self view] layer] setBackgroundColor:
[[UIColor blackColor] CGColor]];
UIImage *balloon = [UIImage imageNamed:@”balloon.jpg”];
// Create the top layer; this is the main image
CALayer *topLayer = [[CALayer alloc] init];
[topLayer setBounds:CGRectMake(0.0f, 0.0f, 320.0, 240.0)]; [topLayer setPosition:CGPointMake(160.0f,
120.0f)]; [topLayer setContents:(id)[balloon CGImage]];
// Add the layer to the view
[[[self view] layer] addSublayer:topLayer];
// Create the reflection layer; this image is displayed beneath // the top layer
CALayer *reflectionLayer = [[CALayer alloc] init]; [reflectionLayer setBounds:CGRectMake(0.0f, 0.0f, 320.0,
240.0)]; [reflectionLayer setPosition:CGPointMake(158.0f, 362.0f)];
// Use a copy of the image contents from the top layer // for the reflection layer
[reflectionLayer setContents:[topLayer contents]];
// Rotate the image 180 degrees over the x axis to flip the image
[reflectionLayer setValue:DegreesToNumber(180.0f) forKeyPath:@”transform.rotation.x”];
```

```
// Create a gradient layer to use as a mask for the
// reflection layer
CAGradientLayer *gradientLayer = [[CAGradientLayer alloc] init]; [gradientLayer setBounds:[reflectionLayer
bounds]]; [gradientLayer setPosition:
CGPointMake([reflectionLayer bounds].size.width/2, [reflectionLayer bounds].size.height/2)];
[gradientLayer setColors:[NSArray arrayWithObjects: (id)[[UIColor clearColor] CGColor],
(id)[[UIColor blackColor] CGColor], nil]];
// Override the default start and end points to give the gradient // the right look
[gradientLayer setStartPoint:CGPointMake(0.5,0.35)]; [gradientLayer setEndPoint:CGPointMake(0.5,1.0)];
// Set the reflection layer's mask to the gradient layer
[reflectionLayer setMask:gradientLayer];
// Add the reflection layer to the view
[[[self view] layer] addSublayer:reflectionLayer]; }
```

清单 10-5 使用梯度层的倒影

这个应用程序使用了 3 个层完成了渴望的效果。顶部的层占据了苹果的一半，显示了原始的图像。底部的或者说倒影的层占据了屏幕的一半，是属于顶端图片的拷贝，并且在 x 轴的方向翻转。图像翻转用到键值对编码使用下面的调用：[reflectionLayer setValue:DegreesToNumber(180.0f) forKeyPath:@"transform.rotation.x"];

这个设置了层，使之从原始的位置变换到 180 度，给了你一个翻转的图像的效果。

最后，一个梯度层是被应用到镜像层上。因为我们想要使用梯度层作为面罩，梯度层使用同样的边框大小和位置同镜像层一样，然后增加它到镜像层的层树上。你可以通过改变 startPoint 和 endPoint 属性的值，来调整梯度层的展示。

10.1.2. CAReplicatorLayer

CAReplicatorLayer 是一个不常用的但是很强大的 CALayer 的子类。它的主要工作是反射任何增加到它上面的子层。这些子类可以被反射很多次基于-instanceCount 这个属性。另外反射它的子层，CAReplicatorLayer 将会基于下面的属性，改变他们的颜色和变换层：

instanceTransform
instanceColor
instanceRedOffSet
instanceGreenOffSet
instanceBlueOffset
instanceAlphaOffset

一个用途就是用 CAReplicatorLayer 来模拟图像的倒影类似与 CoverFlow。你可以创建一个 UIView，那会自动创建一个子层的镜像。我们创建的例子如图 10-6.

图 10-6 用 CAReplicatorLayer 生成的镜像层

创建 UIView

你可以在 Xcode 中通过选择基于窗口的 iphone 模板，开始这个工程。增加一个 UIView 子类到这个模板中，叫做 ReplicatorView。作为 UIView 的子类目的是我们可以重载 +layerClass 方法，指出那种层会被放到后面，如清单 10-6 所示。

```
+ (Class)layerClass {
return [CAReplicatorLayer class]; }
```

清单 10-6 重载+layerClass 方法

无论何时 ReplicatorView 的实例被初始化这个类方法都会被调用。替代原来的 CALayer 在视图的后面，我们将自动的有 CAReplicatorLayer 作为背后的层。

因为我们是子类化了 UIView，如清单 10-7 在-initWithFrame:方法中增加启动的代码。这些启动的代码告诉 CAReplicatorLayer 即将接收的子层要做什么。

```
- (id)initWithFrame:(CGRect)frame {
if (!(self = [super initWithFrame:frame])) return nil;
CGFloat reflectionBase = 230.0f;
CATransform3D transform = CATransform3DMakeScale(1.0, -1.0, 1.0);
transform = CATransform3DTranslate(transform, 0.0, frame.size.height
```

```
- 2.0 * reflectionBase, 0.0);
CAReplicatorLayer *replicatorLayer = (CAReplicatorLayer*)[self layer];
  [replicatorLayer setInstanceTransform:transform];
[replicatorLayer setInstanceRedOffset:-0.5];
[replicatorLayer setInstanceGreenOffset:-0.5];
[replicatorLayer setInstanceBlueOffset:-0.5];
[replicatorLayer setInstanceAlphaOffset:-0.45];
  [replicatorLayer setInstanceCount:2];
return self; }
```

清单 10-7

在调用了父类的方法之后，一个转换是被使用，用来滑动他联系的层和用来把层往下拉 230 个像素。下拉层 230 个像素的原因是，因为我们知道 RepicatorView 会是 220 个像素高，因此我们需要要定位这个层在它的父视图的顶部的下面的 10 个像素处 。

在应用这个转换后，设置靠近后面的那个复制层的颜色（红，绿，蓝和透明度）。因为窗口的背景是黑色的，在这个黑色窗口上增加这个视图。这就给了层一个好看的高斯倒影。

这些实例属性都会被使用，来告知 CAReplicatorLayer 当层是被生成的时候需要做什么。初始化的层任然不可见，但是从第一个的每个子序列的层（基于 instanceCoun 属性）都会转换和偏移一个值。另外，如果层是被复制不止一个，每个子序列的层都将接收先前拷贝层的转换，和它拥有的一个增加。在这个工程中，如果你有了不止一分拷贝，我们会看到变得越来越靠近黑色，当越来越远离屏幕的下方时,他们会被垂直的交替。

利用 ReplicatorView

最后，我们需要初始化 ReplicatorView，增加它到窗口上，然后给它一个子视图来反射。这将在 AppDelegae 中完成，如清单 10-8.

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
CGRect frame = [[UIScreen mainScreen] applicationFrame];
UIView *replicatorView = nil;
replicatorView = [[ReplicatorView alloc] initWithFrame:frame];
[window addSubview:replicatorView];
UIImage *lacey = [UIImage imageNamed:@"Lacey1.png"];
UIImageView *imageView = [[UIImageView alloc] initWithImage:lacey];
[imageView setFrame:CGRectMake(10, 10, 300, 220)];
[imageView setContentMode:UIViewContentModeScaleAspectFit];
[replicatorView addSubview:imageView];
[window setBackgroundColor:[UIColor blackColor]];
[window makeKeyAndVisible];
[imageView release], imageView = nil;
[replicatorView release], replicatorView = nil; }
```

清单 10-8 增加一个 ReplicatorView 到窗口上

下面，我们用应用程序窗口同样的大小，初始化了 ReplicatorView。图像是被装载到一个 UIImageView 中，然后那个 UIImageView 的-contentMode 是被设定为适合框架大小的。最后 UIImageView 是被增加到 ReplicatorView 上。

如果你想要增加不止一个组件到 ReplicatorView 上，你需要调整 instanceCount 属性，以便于每个子层都有一个倒影层。

10.1.3. 总结

CALayer 的子层的每一个都是非常有用的。然而，他们可以被一起应用，随着其他 CALayer 的子层来提供一些特效。例如，CAReplicatorLayer 和 CAShapeLayer 联合使用可以产生一个拼图的效果。

在下一章中，我们讨论核心动画层如何交互。CAShapeLayer 联合一些交互可以创造一些神奇的用户体验，会超越桌面或者 Iphone 提供的一些标准小工具的效果。


DevDiv
点击这里访问：DevDiv.com 移动开发论坛