CGImage以及Mask

**1.UIImage,CGImage和CGImageRef有什么区别**

**CGImage和CGImageRef这两个应当是用来重绘图形的类，它们在应用时是按照图像的像素矩阵来绘制图片的，它们可以用来处理bitmap。**

**（1）CGImageRef与UIImage的互转**

**iOffscreen = CGBitmapContextCreateImage(context);**

**UIImage\* image = [UIImage imageWithCGImage: iOffscreen];**

**（2）UIImage转换成CGImageRef**

**UIImage \*loadImage=[UIImage imageNamed:@"comicsplash.png"];**

**CGImageRef cgimage=loadImage.CGImage;**

**2.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。

**[plain]** view plaincopy

1. - (void)viewDidLoad {

2. [super viewDidLoad];

3. UIImage \*balloon = [UIImageimageNamed:@”balloon.jpg”];

4. [[[self view] layer] setContents:(id)[balloonCGImage]];

5. CGMutablePathRef path =CGPathCreateMutable(); CGPathMoveToPoint(path, NULL, 0, 100);

6. CGPathAddLineToPoint(path, NULL, 200, 0);

7. CGPathAddLineToPoint(path, NULL, 200,200);

8. CGPathAddLineToPoint(path, NULL, 0, 100);

9. shapeLayer = [[CAShapeLayeralloc] init];

10. [shapeLayersetBounds:CGRectMake(0, 0, 200, 200)];

11. [shapeLayer setFillColor:[[UIColorpurpleColor] CGColor]];

12. [shapeLayer setPosition:CGPointMake(200, 200)];

13. [shapeLayer setPath:path];

14. [[[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iPhoneOS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。

**[plain]** view plaincopy

1. - (void)viewDidLoad {

2. [super viewDidLoad];

3. UIImage \*balloon = [UIImageimageNamed:@”balloon.jpg”];

4.  [[[self view] layer] setContents:(id)[balloonCGImage]];

5. CGMutablePathRef path =CGPathCreateMutable();

6. CGPathMoveToPoint(path, NULL, 0, 100);

7. CGPathAddLineToPoint(path, NULL, 200, 0);

8. CGPathAddLineToPoint(path, NULL, 200,200);

9. CGPathAddLineToPoint(path, NULL, 0, 100);

10. shapeLayer = [[CAShapeLayeralloc] init];

11. [shapeLayersetBounds:CGRectMake(0, 0, 200, 200)];

12. [shapeLayer setFillColor:[[UIColorpurpleColor] CGColor]];

13. [shapeLayer setPosition:CGPointMake(200, 200)];

14. [shapeLayer setPath:path];

15. [shapeLayersetStrokeColor:[[UIColor redColor] CGColor]];

16. [shapeLayer setLineWidth:10.0f];

17. [shapeLayersetLineJoin:kCALineJoinRound];

18. [shapeLayersetLineDashPattern:

19. [NSArrayarrayWithObjects:[NSNumber numberWithInt:50], [NSNumber numberWithInt:2],

20. nil]];

21. [[[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，macbookpro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。

**[plain]** view plaincopy

1. - (void)viewDidLoad {

2. [super viewDidLoad];

3. UIImage \*balloon = [UIImageimageNamed:@”balloon.jpg”];

4. [[[self view] layer] setContents:(id)[balloon CGImage]];

5. CGMutablePathRef path =CGPathCreateMutable();

6. CGPathMoveToPoint(path, NULL, 0, 100);

7. CGPathAddLineToPoint(path,NULL, 200, 0);

8.  CGPathAddLineToPoint(path, NULL, 200, 200);

9. CGPathAddLineToPoint(path, NULL, 0, 100);

10. shapeLayer = [[CAShapeLayeralloc] init];

11. [shapeLayersetBounds:CGRectMake(0, 0, 200, 200)];

12. [shapeLayer setFillColor:[[UIColorpurpleColor] CGColor]];

13. [shapeLayer setPosition:CGPointMake(200, 200)];

14. [shapeLayer setPath:path];

15. [[[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。

**[plain]** view plaincopy

1. - (void)viewDidLoad {

2. [super viewDidLoad];

3. UIImage \*balloon = [UIImageimageNamed:@”balloon.jpg”];

4.  [[[self view] layer] setContents:(id)[balloonCGImage]];

5. CGMutablePathRef path =CGPathCreateMutable();

6. CGPathMoveToPoint(path, NULL, 0, 100);

7.  CGPathAddLineToPoint(path,NULL, 200, 0);

8. CGPathAddLineToPoint(path, NULL, 200, 200);

9. CGPathAddLineToPoint(path, NULL, 0, 100);

10. shapeLayer = [[CAShapeLayeralloc] init];

11. [shapeLayersetBounds:CGRectMake(0, 0, 200, 200)];

12. [shapeLayer setFillColor:[[UIColor purpleColor]CGColor]];

13. [shapeLayer setPosition:CGPointMake(200, 200)];

14.  [shapeLayersetPath:path];

15. [shapeLayersetStrokeColor:[[UIColor redColor] CGColor]];

16.  [shapeLayer setLineWidth:10.0f];

17. [shapeLayersetLineJoin:kCALineJoinRound];

18. [shapeLayer setLineDashPattern:

19. [NSArrayarrayWithObjects:[NSNumber numberWithInt:50], [NSNumber numberWithInt:2],

20. nil]];

21. [[[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

**CAGradientLayer**

你在iphone上或者MacOS 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

**[plain]** view plaincopy

1. - (void)viewDidLoad {

2. [super viewDidLoad];

3. [[[self view] layer]setBackgroundColor:

4. [[UIColor blackColor]CGColor]];

5. UIImage \*balloon = [UIImageimageNamed:@”balloon.jpg”];

6. // Create the top layer; thisis the main image

7. CALayer \*topLayer = [[CALayeralloc] init];

8. [topLayersetBounds:CGRectMake(0.0f, 0.0f, 320.0, 240.0)];

9. [topLayersetPosition:CGPointMake(160.0f, 120.0f)];

10. [topLayer setContents:(id)[balloonCGImage]];

11. // Add the layer to the view

12. [[[self view] layer]addSublayer:topLayer];

13. // Create the reflectionlayer; this image is displayed beneath // the top layer

14. CALayer \*reflectionLayer =[[CALayer alloc] init];

15. [reflectionLayer setBounds:CGRectMake(0.0f, 0.0f,320.0, 240.0)];

16.  [reflectionLayer setPosition:CGPointMake(158.0f, 362.0f)];

17. // Use a copy of the imagecontents from the top layer // for the reflection layer

18. [reflectionLayersetContents:[topLayer contents]];

19. // Rotate the image 180degrees over the x axis to flip the image

20. [reflectionLayersetValue:DegreesToNumber(180.0f) forKeyPath:@”transform.rotation.x”];

21. // Create a gradient layer touse as a mask for the

22. // reflection layer

23. CAGradientLayer\*gradientLayer = [[CAGradientLayer alloc] init];

24. [gradientLayersetBounds:[reflectionLayer bounds]];

25.  [gradientLayer setPosition:

26. CGPointMake([reflectionLayerbounds].size.width/2, [reflectionLayer bounds].size.height/2)];

27. [gradientLayersetColors:[NSArray arrayWithObjects: (id)[[UIColor clearColor] CGColor],

28. (id)[[UIColor blackColor]CGColor], nil]];

29. // Override the default startand end points to give the gradient // the right look

30. [gradientLayersetStartPoint:CGPointMake(0.5,0.35)];

31.  [gradientLayersetEndPoint:CGPointMake(0.5,1.0)];

32. // Set the reflection layer’smask to the gradient layer

33. [reflectionLayersetMask:gradientLayer];

34. // Add the reflection layerto the view

35. [[[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：[reflectionLayersetValue: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。

**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 [CAReplicatorLayerclass]; }

  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。

**[plain]** view plaincopy

1. - (id)initWithFrame:(CGRect)frame{

2. if (!(self = [superinitWithFrame:frame])) return nil;

3. CGFloat reflectionBase =230.0f;

4. CATransform3D transform =CATransform3DMakeScale(1.0, -1.0, 1.0);

5. transform =CATransform3DTranslate(transform, 0.0, frame.size.height

6. - 2.0 \* reflectionBase, 0.0);

7. CAReplicatorLayer\*replicatorLayer = (CAReplicatorLayer\*)[self layer];

8.  [replicatorLayersetInstanceTransform:transform];

9. [replicatorLayersetInstanceRedOffset:-0.5];

10. [replicatorLayersetInstanceGreenOffset:-0.5];

11. [replicatorLayersetInstanceBlueOffset:-0.5];

12. [replicatorLayersetInstanceAlphaOffset:-0.45];

13.  [replicatorLayer setInstanceCount:2];

14. 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.

**[plain]** view plaincopy

1. -(void)applicationDidFinishLaunching:(UIApplication \*)application {

2. CGRect frame = [[UIScreenmainScreen] applicationFrame];

3. UIView \*replicatorView = nil;

4. replicatorView = [[ReplicatorViewalloc] initWithFrame:frame];

5. [windowaddSubview:replicatorView];

6. UIImage \*lacey = [UIImageimageNamed:@”Lacey1.png”];

7. UIImageView \*imageView =[[UIImageView alloc] initWithImage:lacey];

8. [imageViewsetFrame:CGRectMake(10, 10, 300, 220)];

9. [imageViewsetContentMode:UIViewContentModeScaleAspectFit];

10. [replicatorViewaddSubview:imageView];

11. [windowsetBackgroundColor:[UIColor blackColor]];

12. [window makeKeyAndVisible];

13. [imageView release],imageView = nil;

14. [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。

**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。

3.Bitmap Images and Image Masks 英文文档翻译

4.刮奖的示例分析见（文档附录）