Core Animation:

Simplified Animation Techniques for Mac and iPhone Development

第三部分

核心动画的层

第六章

层的滤镜

版本 1.0

翻译时间：2012-11-19

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

1.1.5. 第五章 Core Animation 中文翻译_第四章_层的变换

第 6 章　层的滤镜

就像很多的苹果技术一样，核心动画同样提供了一个链接到其他核心功能的桥梁。这个桥梁可以允许开发者直接和核心图像(Core Image)进行交互---苹果的图像处理和控制框架。这就意味这作为核心动画的程序员，你可以拥有了处理核心图像的滤镜能力。

具体的说，这意味着例如，你可以在一个层的内容上应用高斯模糊。你可以在滤镜上设置模糊率并且应用它，或者你可以用模糊率做动画使之模糊，然后你指定一段时间之后再恢复到清晰。或者说，你想要让图像具有一个玻璃曲解的效果。你可以创建一个核心图像的滤镜，指定的名字为 CIGlassDistorion，并且根据核心图片的文档给它提供滤镜需要的参数。

这一章，我们深入的探讨怎么有效的应用滤镜到核心动画的层上，怎么使用键值对编码来控制输入参数，并且怎么使用滤镜变换。

在著此文时，核心图像滤镜是不支持 iphone 的，因此这一章的内容仅仅适用于 OS X。你可以在 iphone 上做一些图像处理，然而，这些功能作为核心动画一部分目前不可用，并且讲解这些功能也超越了本书的范围。

6.1.　在核心动画层上应用滤镜

苹果的核心图像程序开发向导和关联的核心图像滤镜参考都提供了在层和视图上使用滤镜的详细 API 讲解。你需要进入到这个参考中，看滤镜中的那些条目对你来说是有用的。

为一个层增加核心图像滤镜，只要简单的创建一个 CIFilter 对象，并且增加它到层的滤镜数组参数上，如清单 6-1 所示。

```
CIFilter *blurFilter =
[CIFilter filterWithName:@"CIGaussianBlur"];
[blurFilter setDefaults];
[blurFilter setValue:[NSNumber numberWithFloat:5.0f]
forKey:@"inputRadius"];
[layer setFilters:[NSArray arrayWithObject:blurFilter]];
```

清单 6-1 给层增加高斯模糊滤镜

当你运行清单 6-1 的代码时，滤镜会被应用到层上只要它加入到了层树中，不管视图中包含什么。在这个例子中，层的视图应用了一个 5 像素的弧度的高斯模糊。你可以看到代码输出的结果如图 6-1.如果你想看完整的代码例子，从合作网站(informit.com/coreframeworks)打开叫做 GaussianBlurLayer 的项目工程。

许多可用的滤镜都不要太多的东西，除了通过调用-setDefauts 这个方法指定默认的参数。简单的改变下滤镜的名字，之后可以看到我们的用途。

用这种方法在层上应用滤镜不一定非常有用。就像许多技术条目，控制滤镜的细节需要更多的调用。因为这是一本关于核心动画的书，我们可能需要的方法是应用滤镜到动画的上下文。你需要在你的应用程序上做的是使滤镜应用在层的动画上，而非简单应用在一个静态的层上，然后离开。下面我们继续讲解。

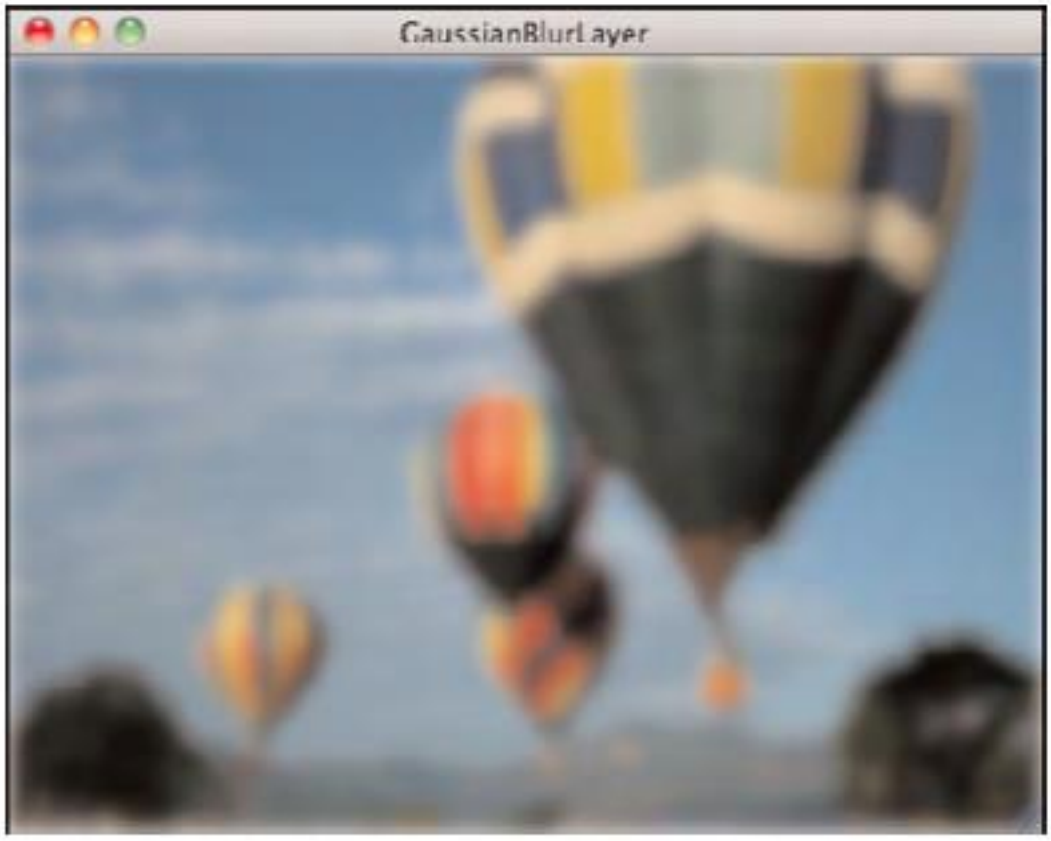

图 6-1 应用到图像的高斯模糊

滤镜动画

为了响应用户的输入，你可能需要给层应用添加一个滤镜。例如，你想要创建一个基于层的核心动画的按钮，当按下去时有一个 5 像素的高斯弧度。为了做这些，创建一个有背后的层的继承于 NSView 的视图，其中包含了做动画的层。当你在初始化时增加滤镜到你的视图的层中，设置它的 inputRadius 这个字段为 0.0，以达到第一次显示时可视效果没有影响。然后，响应点击，你可以改变 inputRadius 这个属性，使之做动画，让按钮变糊，最后再清楚掉。在合作网站上，例子程序点击变糊演示了这个效果。打开工程立刻可以看到这个效果。

在图 6-2 你看到的每个图片都是在一个视图上；每个视图都包含了一个核心动画的层。你看到的图像是视图层的内容。在每个层的背后都是一个叫做 BlurView 的视图，做了如下的动作：

有一个继承于 CALayer 的层叫做 BlurLayer，在层上动画可以执行。

接收一个鼠标按下的事件，可以触发动画。

每一个层都有一个 filters 字段，用来设置一个核心图像滤镜对象的数组。考虑这个数组作为数组可能有点模棱两可。你可以建立一个你需要使用的滤镜的链表，然后当你接收用户输入时例如鼠标点击，你可以通过键值对编码激活它们。同时，我们获得键值对编码。点击变糊的演示程序中引起图标变糊，并且当用户再次点击时会变清晰。

图 6-2 应用到图像的高斯模糊的滤镜

键值对编码可以操作滤镜的值，在做这个之前我们需要为每个我们需要的滤镜命名。滤镜的名字是重要的，因为它将在关键字路径上被使用给滤镜指定参数，例如 inputRadius。进一步的讨论可以看下，“在键值对编码中的关键字路径”。我们也需要改变初始化的输入半径值为 0.0f。这是输入模糊时半径的开始值。我们开始使用 0，因此模糊效果不可见。

清单 6-2 反映出了代码的改变。改变是我们初始化了 inputRadius 的值为 0.0，然后我们为滤镜增加了一个名字，以便于之后调用。完成的视图初始化如清单 6-3 的代码。

```
CIFilter *blurFilter =
[CIFilter filterWithName:@”CIGaussianBlur”];
[blurFilter setDefaults];
[blurFilter setValue:[NSNumber numberWithFloat:0.0f] forKey:@”inputRadius”];
[blurFilter setName:@”blur”];
[layer setFilters:[NSArray arrayWithObject:blurFilter]];
```

清单 6-2 增加滤镜的名字和设置输入半径

```
- (void)awakeFromNib; {
[self setWantsLayer:YES]; // Initialize the layer
blurLayer = [[[BlurLayer alloc]init] retain]; [blurLayer setMasksToBounds:YES];
// Set the layer to fill the entire frame
[blurLayer setFrame:CGRectMake(0.0, 0.0,
[self frame].size.width, [self frame].size.height)];
[[self layer] addSublayer:blurLayer];
// Initialize the Gaussian blur filter
CIFilter *blurFilter =
[CIFilter filterWithName:@”CIGaussianBlur”];
[blurFilter setDefaults];
[blurFilter setValue:[NSNumber numberWithFloat:0.0]
forKey:@”inputRadius”];
// Give the Gaussian blur filter a name
[blurFilter setName:@”blur”];
[blurLayer setFilters:[NSArray arrayWithObject:blurFilter]];
}
```

清单 6-3 滤镜初始化代码

我们已经创建了一个继承自 CALayer 类名字叫 BlurLayer 的类，我们增加这个类到 BlurView 视图对象中。BlurLayer 类管理层中的图像。每个实例视图的图像都是在-applicationDidFinishLaunching 方法中设定，此方法可以在应用程序的代理类中发现，如清单 6-4.

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification; {
NSString *iconPath1 = [[NSBundle mainBundle] pathForResource:@”desktop”
ofType:@”png”];
NSString *iconPath2 = [[NSBundle mainBundle]
pathForResource:@”fwdrive”
ofType:@”png”];
NSString *iconPath3 = [[NSBundle mainBundle]
pathForResource:@”pictures”
ofType:@”png”];
NSString *iconPath4 = [[NSBundle mainBundle]
pathForResource:@”computer” ofType:@”png”];
NSImage *image1 = [[NSImage alloc] initWithContentsOfFile:iconPath1];
NSImage *image2 = [[NSImage alloc] initWithContentsOfFile:iconPath2];
NSImage *image3 = [[NSImage alloc] initWithContentsOfFile:iconPath3];
NSImage *image4 = [[NSImage alloc] initWithContentsOfFile:iconPath4];
[blurView1 setLayerImage:image1]; [blurView2 setLayerImage:image2]; [blurView3 setLayerImage:image3]; [blurView4 setLayerImage:image4];
}
```

清单 6-4 BlurView 对象的初始化

变量 blurView1,blurView2,blurView3,和 blurView4 都是 BlurView 的一个实例，并且每一个都在 interface Builder 中链接到自定义的视图中。你可以看到自定义的视图如图 6-3.

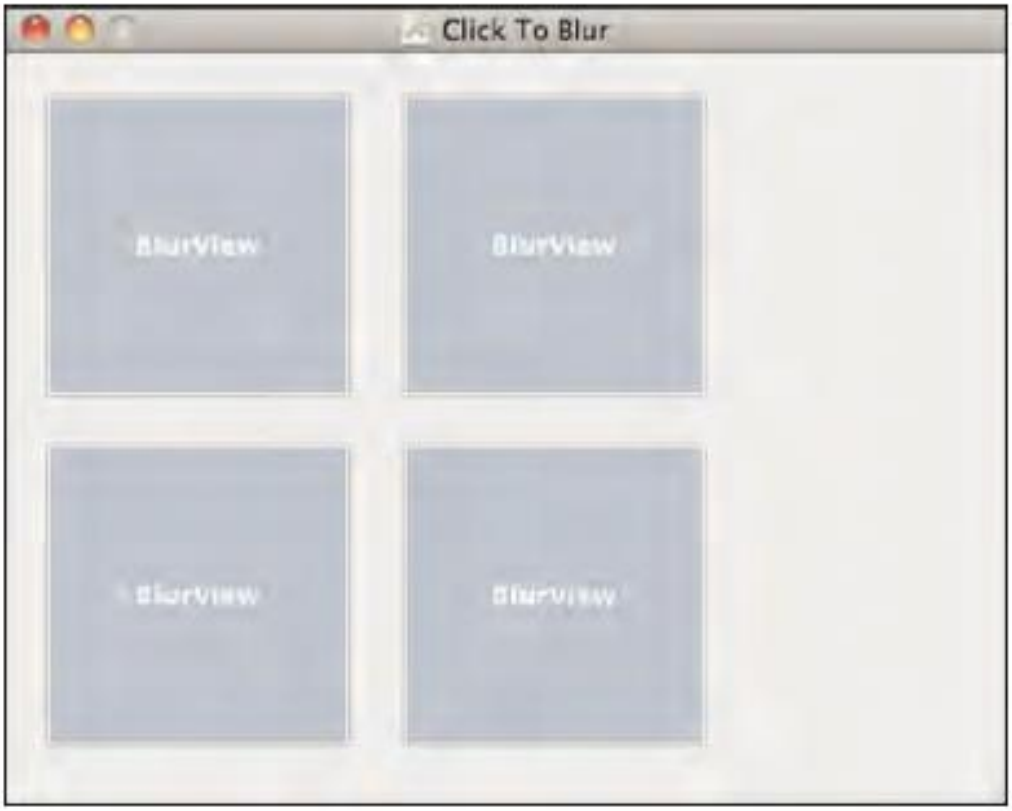

图 6-3 在 interface builder 中的自定义视图

接收用户的输入

核心动画的层不能接收鼠标时间。然而，它后面的视图可以。在视图的初始化代码中(清单 6-4)，当视图接收一个点击时，我们就应用层的滤镜。

为了接收点击，告诉 BlurView 来接收鼠标点击，需要给视图增加清单 6-5 代码。

```
-(BOOL)acceptsFirstResponder; {
return YES; }
```

清单 6-5 使视图接收事件

第一响应者是，在事件响应链中有机会第一个响应鼠标点击或者某个键按下去的事件。在清单 6-5 中，我们告知应用程序，我们的窗口成为第一个响应者，来响应发来的事件。在这个例子中，就是简单的鼠标点击。在我们用这种方法建立了我们的视图可以控制我们的事件后，我们就可以实现接收事件的代码了。在-mouseDown 视图代理中，安装一个基础动画，用来改变模糊滤镜输入半径的值，如清单 6-6.

```
- (void)mouseDown:(NSEvent *)theEvent; {
CABasicAnimation *anim = [CABasicAnimation animationWithKeyPath:
@"filters.blur.inputRadius"];
// Set the filter's start value
[anim setFromValue:[NSNumber numberWithFloat:0.0f]]; // Set the filter's ending value
[anim setToValue:[NSNumber numberWithFloat:5.0f]]; [anim setDuration:0.2f];
[anim setAutoreverses:YES];
[anim setRemovedOnCompletion:YES];
[blurLayer addAnimation:anim forKey:@"filters.blur.inputRadius"]; }
```

清单 6-6 增加模糊动画给鼠标按下的事件

就像清单 6-6 所示的，我们使用关键路径 filters.blur.inputRadius 来创建一个基础动画。你可能注意到了，我们在清单 6-2 中给滤镜增加的名字，这里就说明了它的重要性了。Filters 这个字段进入，然后我们查询滤镜的名字(我们在清单 6-2 中增加的高斯模糊滤镜)，因而我们可以使用它的半径做动画。

输入半径开始和结束的值，通过使用-setFromValue:和-setToValue:这些方法来实现，例如下面两行所示的。我们准备做动画使输入半径的值从 0.0 到 5.0。下面，我们设置动画的执行时间。接近一秒的 1/5，使用-setDuration:0.2f。

-setAutoreverses:YES 调用，告诉动画循环到它的原始值。如果我们设置了 NO，当动画完成时，你会看到层瞬间返回到它的原始值，这样看起来有点突兀。这种情况下，给一个动画返回到原始的值，显得平滑和好看。

最后，当动画完成它的第一圈时，动画从层中自动的被移除(-setRemovedOnCompletion:YES)。不过这里这是多余的调用。然而，我们包含它目的是让你理解这个使我们要做的行动。

现在，当按钮被点击时，它就会使整个按钮层模糊 5 像素的半径，然后在 1/5 秒内，返回到它的清晰状态

使效果具有粘性

可能有时候你想要动画具有粘性，就是不要让动画返回到原始的值，你想要新的值附加上。来看这个例子，来到合作的网站然后在 xcode 中导入 blur sticky 这个工程，点解 build 在运行。当你点击每个图标时，你会注意到图标就会停留在模糊状态，直到你下次再点击时才恢复，如图 6-3.

为了创建这种效果，就是点击时让图标保持模糊，然后再次点击时返回到正常状态，你需要跟踪按钮的状态。这里使用名字叫 toggle 的 bool 值来实现。鼠标点击的方法-mouseDown 如清单 6-7.

```
- (void)mouseDown:(NSEvent *)theEvent; {
CABasicAnimation *anim = [CABasicAnimation
animationWithKeyPath:@"filters.blur.inputRadius"];
if(toggle) {
[anim setFromValue:[NSNumber numberWithFloat:0.0f]];
[anim setToValue:[NSNumber numberWithFloat:5.0f]]; }
else
{
[anim setFromValue:[NSNumber numberWithFloat:5.0f]]; [anim setToValue:[NSNumber numberWithFloat:0.0f]];
}
[anim setDuration:0.2f];
```

```
[anim setRemovedOnCompletion:NO];
[anim setFillMode:kCAFillModeForwards]; [blurLayer addAnimation:anim
forKey:@"filters.blur.inputRadius"]; toggle = !toggle;
}
```

清单 6-7 实现拴的功能

因此什么改变了？首先我们要依赖于 toggle 这个变量的状态，这样我们才能决定开始的值是 0.0 还是 5.0，在设置-setToValue:相反的值之前。如果 toggle 的值尚未设定，我们做模糊的动画，因而开始的值 fromValue 为 0.0，结束的值，toValue 设定为 5.0。然而，如果 toggle 的值是设定了，我们就开始做变清晰的动画，因而开始的值为 5.0，结束的为 0.0。

注意到了在清单 6-7 中没有包含-setAutoreverses 这个调用，而在清单 6-6 中有。为了达到粘性的效果这个调用必须被移除或者设定为 NO；否则，它就会返回它的开始值，并且覆盖-setFillMode:这个方法。

下面，我们就需要改变-setRemovedOnCompletion 这个方法为 NO 了。这告诉了层当它完成时，动画不要从层中移除。

最后，我们增加一个调用-setFillMode:kCAFillModeForwards，来告诉动画当完成时，停留在-setToValue 中指定的字段的结果。

这样每个图标第一次点击时，就会变模糊。然后再点击就会返回到它原始状态，这种就是核心动画基于按钮的层的粘性效果。在图 6-4 中，你会看到有两个图标是模糊的，因为他们已经被点击了一次，保留他们的粘性状态了。

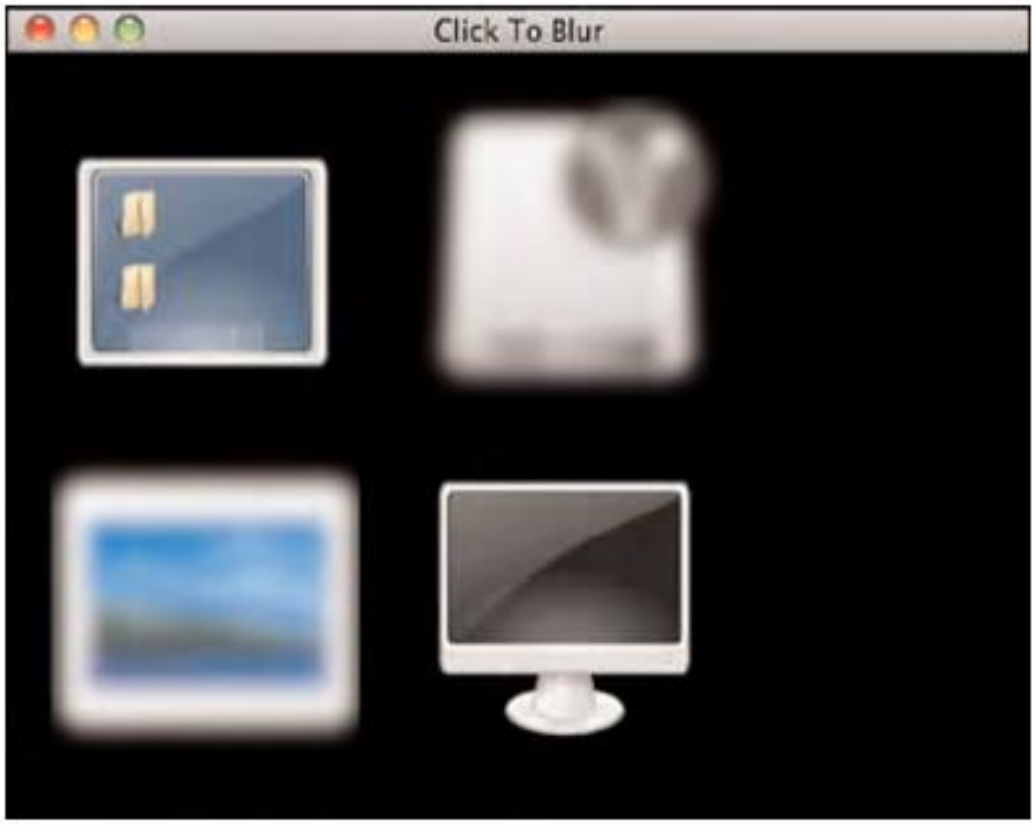

图 6-4 粘性模糊

6.2. 使用数据绑定控制滤镜的值

滤镜的参数通过使用 KVC 进入和改变。这就让你使用数据绑定来创建带着复杂滤镜的层，通过使用标准的 Cocoa，例如滑动条，来控制这些滤镜。

你可能回忆起来先前的段落中讲到，当我们在鼠标按下时实例化了 CABasicAnimation 对象，我们在层上使用关键路径 filters.blur.inputRadius.filters 代表了滤镜数组可用，blur 是我们后面定义的，给滤镜的

名字。inputRadius 是我们想操控的滤镜字段的名字。当我们在 Interface Builder 中，这些关键路径同样可用。不同的是，我们必须首先让 CALayer 可以链接到 interface Builder，通过给 AppDelegate 类设定属性，如列表 6-8.

```
@interface AppDelegate : NSObject {
IBOutlet NSView *displayView;
CALayer *imageLayer; }
@property (retain)CALayer *imageLayer; @end
```

清单 6-8 应用代理的声明

在 AppDelegate 中实现，我们需要@synthensize imageLayer 对象。在 Objective-c 2.0 之前，你不得不为你的实例变量显示的创建 setters 和 getters 方法，并且你改变实例变量时，还要通过调用-willChangeValueForKey 和-didChangeValueForKey 通知对象，来服从 KVC，这些都是绑定工作要求的。现在，用 objective-c 2.0，当你通过@property 和使用@synthensize 关键字时，这些功能都自动创建了。同步 imageLayer 对象的实现代码如清单 6-9.

```
@implementation AppDelegate
@synthesize imageLayer;
...
```

清单 6-9

转换到 Interface Builder 中，然后为应用到 imageLayer 对象中的每一个滤镜都增加一个滑动块。在图 6-5 中，你可以看到我们增加了那些滑动块，为高斯模糊、色调、饱和度，对比度，明亮。

在我们把数据绑定到滑动条之前，滤镜需要增加到 imageLayer 中。为了做这些，在层上调用-setFilters，如清单 6-10 所示。

图 6-5 图像调整滤镜的滑动条

```
- (NSArray*)filters; {
CIFilter *blurFilter =
[CIFilter filterWithName:@"CIGaussianBlur"];
[blurFilter setDefaults];
[blurFilter setValue:[NSNumber numberWithFloat:0.0]
forKey:@"inputRadius"]; [blurFilter setName:@"blur"];
```

```
CIFilter *hueFilter =
[CIFilter filterWithName:@"CIHueAdjust"];
[hueFilter setDefaults];
[hueFilter setValue:[NSNumber numberWithFloat:0.0]
forKey:@"inputAngle"]; [hueFilter setName:@"hue"];
CIFilter *colorFilter =
[CIFilter filterWithName:@"CIColorControls"];
[colorFilter setDefaults]; [colorFilter setName:@"color"];
return [NSArray arrayWithObjects: blurFilter,
hueFilter, colorFilter, nil];
}
```

清单 6-10 滤镜的方法

这个代码创建了 3 个滤镜：模糊滤镜，色调滤镜和颜色滤镜。最后的颜色滤镜，能使我们改变三个不同的值：饱和度，对比度和明亮度。每个滤镜我们都通过-setDefaults 方法给它们设定了默认的值。注意我们给每个滤镜指定的名字。这些名字在绑定和 KVC 时用来进入到滤镜的。

通过增加这些滤镜到层中，我们使用关键路径能够进入每个滤镜的参数。每个滑动块的全部的关键路径展示如下表 6-1.

滤镜的名字	滤镜的参数名字	关键路径
模糊	inputRadius	imageLayer.filters.blur.inputRadius
色调	inputAngle	imageLayer.filters.hue.inputAngle
颜色	inputSaturation	imageLayer.filters.color.inputSaturation
颜色	inputContrast	imageLayer.filters.color.inputContrast
颜色	inputBrightness	imageLayer.filters.color.inputBrightnes

表 6-1

如果你在 interface Builder 中绑定滑动块的检测器，你就可以看到如何应用关键路径了。如图 6-6 所示的色调滤镜的关键路径。

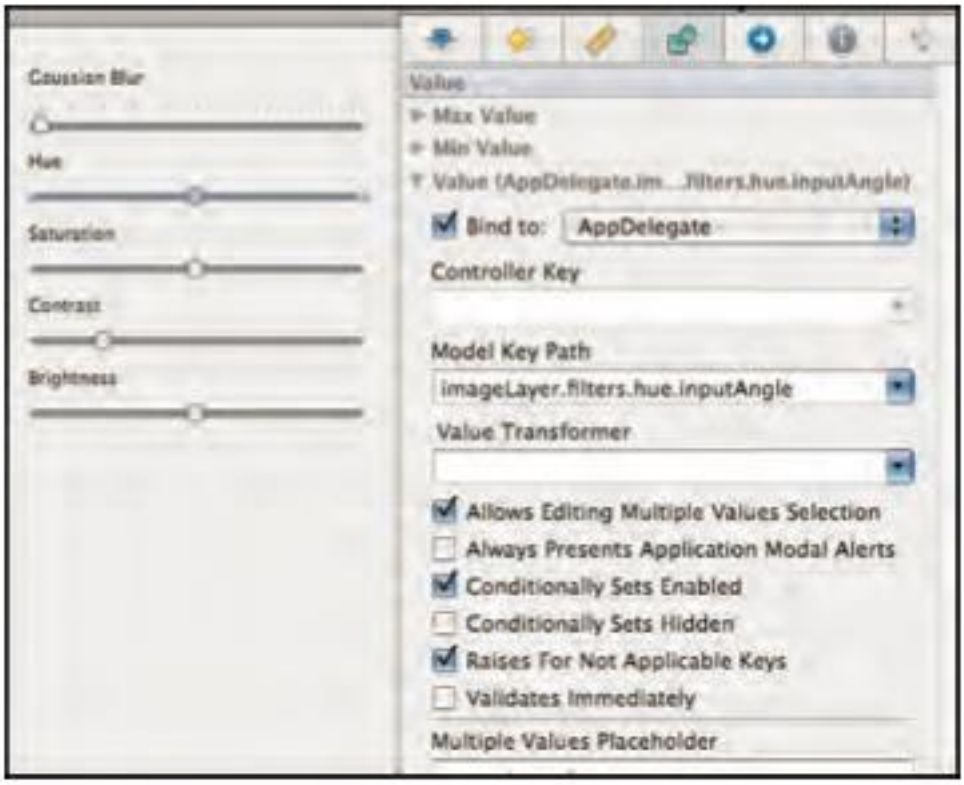

图 6-6 绑定色调滤镜的滑动块

剩下的滑动块用同样的方法进行链接，并且指定关键路径如表 6-1 所示的。

完成这个应用程序需要很少的代码，这要归功于数据绑定的力量。在 appDelegate 中的实现如下清单 6-11.

```
@interface AppDelegate : NSObject { IBOutlet NSView *displayView; CALayer *imageLayer;
}
@property (retain)CALayer *imageLayer;
- (NSArray*)filters;
- (CGImageRef)nsImageToCGImageRef:(NSImage*)image;
@end
@implementation AppDelegate @synthesize imageLayer;
- (void)awakeFromNib; {
[displayView setWantsLayer:YES];
imageLayer = [[CALayer layer] retain];
CGRect frame = CGRectMake(0.0f, 0.0f,
[displayView frame].size.width, [displayView frame].size.height);
[imageLayer setFrame:frame]; [imageLayer setCornerRadius:15.0f]; [imageLayer setBorderWidth:3.0f]; [imageLayer
setMasksToBounds:YES];
NSString *imagePath = [[NSBundle mainBundle] pathForResource:@”balloon”
ofType:@”jpg”];
[imageLayer setContents: (id)[self nsImageToCGImageRef:
[[NSImage alloc] initWithContentsOfFile:imagePath]]];
[imageLayer setFilters:[self filters]];
[[displayView layer] addSublayer:imageLayer]; }
- (NSArray*)filters; {
CIFilter *blurFilter =
[CIFilter filterWithName:@”CIGaussianBlur”];
[blurFilter setDefaults];
[blurFilter setValue:[NSNumber numberWithFloat:0.0]
forKey:@”inputRadius”]; [blurFilter setName:@”blur”];
```

```
CIFilter *hueFilter =
[CIFilter filterWithName:@"CIHueAdjust"];
[hueFilter setDefaults];
[hueFilter setValue:[NSNumber numberWithFloat:0.0]
forKey:@"inputAngle"]; [hueFilter setName:@"hue"];
CIFilter *colorFilter =
[CIFilter filterWithName:@"CIColorControls"];
[colorFilter setDefaults]; [colorFilter setName:@"color"];
return [NSArray arrayWithObjects: blurFilter,
hueFilter, colorFilter, nil];
}
- (CGImageRef)nsImageToCGImageRef:(NSImage*)image; {
NSData * imageData = [image TIFFRepresentation]; CGImageRef imageRef;
if(imageData)
{
CGImageSourceRef imageSource = CGImageSourceCreateWithData((CFDataRef)imageData, NULL);
imageRef =
CGImageSourceCreateImageAtIndex(imageSource, 0, NULL); }
return imageRef; }
- (void)dealloc; {
if( imageLayer ) [imageLayer release];
[super dealloc]; }
@end
```

清单 6-11

6.3. 变换应用到滤镜上

用核心动画开启动画的属性最简单的方法是在你想要做动画的视图上使用动画者的代理对象。当你应用动画者代理对象的时候，它就会调用到默认的变换，这些可能你没有注意到。例如，如果你设定一个层的透明度，那么渐变的转换就会被默认的增加。(渐入是增加不透明，渐出是减少不透明)。为了看到默认的转换，你需要在动画代理者上调用-setOpacity，而不是在视图上调用，那么渐变的变换就会自动的做动画，而不需要你增加额外的代码。最能体现变换价值的地方是，你增加你自定义的变换，并且用滤镜也是很容易做到的。

默认的变换

核心动画提供了一系列默认的变换(如表 6-2),当某些属性改变时，你可以指定这些变换来使用。

字段	功能
渐变	常量 kCATransitionFade 表明层的内容应该渐入，当变换发生时。
滑入	常量 kCATransitionMoveIn 表明层的内容应该从一边滑入。方向是由变换的一个子属性决定的。当你使用 kCATransitionMoveIn 这个主类型时，如果什么都不指定，那么 kCATransitionFromLeft 这个子属性就会指定，内容就会从左边出来。
推	常量 kCATransitionPush 表明层的内容应该从某个方向推走层中任何存在的内容。方向是有变换的子类型决定的（下一个段中包含了”变换的子类型”）。如果你不指定，默认的主类型 kCATransitionPush 是被使用，而子类型 kCATransitionFromLeft 被指定，这样就会指定的内容就会从左边出来。
显示	常量 kCATransitionReveal 表明层的内容应该从某个方向暴露。方向是变换的子类型

	（下一段可以看到变换的子类型）。当你使用 kCATransitionReveal 这个主类型时，如果你不指定默认的子变换，那么指定的内容就会从左边出来。

表 6-2 默认的变换类型和功能

默认的子类型变换如表 6-3

字段	功能
从右边	常量 kCATransitionFromRight 表明动画从右边进入
从左边	常量 kCATransitionFromRight 表明动画从左边进入
从上面	常量 kCATransitionFromRight 表明动画从上边进入
从底端	常量 kCATransitionFromRight 表明动画从底边进入

表 6-3 默认的变换子类型和它的功能

如果你想要使用默认的变换，简单的给层 actions 字段增加变换就行了，这个字段是个 NSDictionary。当你增加 CATransition 对象给层的 actions 字段中时，通过使用关键路径指定你想要变换的层的属性。当你创建你想要变换的层时，你简单的附加上就行了。在下个段落中，我们会详解返回的行为，那是一个变换属性的特定的行为，但是这里是基础，这个代码就是在层中，使用给定的属性的默认变换。

1 选着你想要动画的属性，这里，我们使用不透明度

2 为 actions 这个属性创建一个新的字典。

3 给新的 actions 字典增加关键字 opacity。

4 设定层的 actions 属性，使用新创建的字典。

设定不透明度默认变换的代码如清单 6-12

```
CATransition *transition=[CATransition animation];
[transition setType:kCATransitionMoveIn];
[transition setSubtype:kCATransitionFromTop];
[transition setDuration:1.0f];
// Get the current list of actions
NSMutableDictionary *actions = [NSMutableDictionary dictionaryWithDictionary:[transitionLayer actions]];
[actions setObject:transition forKey:@”opacity”];
[transitionLayer setActions:actions];
```

清单 6-12 不透明度的默认的变换

首先，你需要实例化 CATransition 对象。使用表 6-2 和 6-3 指定的变换属性，这里设定属性 kCATransitionMoveIn，这个会引起变换从某个方向移动。然后，在子类型字段中，指定变换从那个方向进入；例如，我们是从顶端移进的(kCATransitionFromTop)。下一步，指定变换的时间；在这里是一秒。

这里，我们需要给层的行动列表中，增加变换。通过关键路径层就会使用这些变换，而不适用默认的变换。我们指定的字典的关键路径是 opacity。无论什么时候层的 opacity 改变时，它就会使用特定的变换而不会使用默认的变换。

为了做这个工作，我们首先实例化一个字典，从 transitionLayer 中使用原始的 actions 字段。我们做这些，仅仅因为其他的 actions 已经在集合中指定了。

我们知道在字典中没有任何其他的东西。然而，我清楚的指出。你需要理解下面的事，如果你创建一个新的字典并且给层设定了行动，而没有首先获取先前已经有的，你可能会困惑为什么其他的变换不能正常的工作了。

最后，设定层的行动字典到指定的创建的字典，该字典包含了 CATransition 对象。

使用自定义的变换

指定你想要使用的变换的 CIFilter 是简单的。为了应用这个变换，使用 CATransition 对象的滤镜参数来指定你想要指定的关键字的行动。当使用时，滤镜参数会重载 CATransition 对象的类型和子类型参数。如果你使用这两个参数指定默认变换的话，他们将会被忽略，当滤镜参数是被设定时。

用代理或者封转

对于很多应用程序，在你自己的 CALayer 的继承类中你应该封转来实现自定义的变换。然而，你也可以使用 CALayer 的代理方法，这样会更方便。这个使用代理的唯一的理由-方便。如果考虑代码的整洁性，选择封装好的代理，但是有时代理是一个快速直接的方法。有这两个不同的技术，这样你可以通过函数调用来实现了，如表 6-4 所示

你要做的	做法
封装	重载 CALayer 类的方法 +(id<CAAction>)defaultActionForKey:(NSString *)aKey
代理	设定代理指向控制器，然后实现下面的代理方法：-(id<CAAction>)actionForLayer:(CALayer*)layer forkey:(NSString *)key;

表 6-4 使用封装和代理的方法

有时这是快速和方便的，使用代理来指定你想要使用的变换滤镜。如果你使用代理实现了自定义的变换，首先要在应用程序代理的代码中，设定层的代理。例如：

```
[layer setDelegate: self];
```

然后实现代理的方法，就是设定一个行动为指定的关键字和层。

```
- (id<CAAction>)actionForLayer:(CALayer *)layer forKey:(NSString *)key;
```

这个代理方法，提供给你 2 个条目，让你来指定那种滤镜要应用到变换中：层和关键字字段。你可以这样使用这两个特点：

你需要简单的核对看是否传给你了你工作的层对象，通过使用==操作符来比较内存地址。

如果你给层分配了一个名字，你能匹配名字字符串。然后核对关键字是否为你使用关键路径。

如果你更喜欢在自己的继承自 CALyaer 类中封装代码的话，你可以重载-defaultActionForKey，以便于返回一个特定关键字的行动。层就会知道因为我们已经封装了它；例如：

```
+ (id<CAAction>)defaultActionForKey:(NSString*)aKey
```

涟漪变换

我们包含了 2 个例子来演示怎么来实现涟漪变换在这个段落中。一个使用代理展示了怎么实现它，另一个使用封装。你或许迫不及待打开工程来看解决的方法了。

例如，我们使用层的核心图片的涟漪效果滤镜改变层的变换，CIRippleTransition。它要求的字段显示在表 6-5 中。

字段的名字	功能
inputImage	使用 CIImage 的开始图像
inputTargetImage	使用 CIImage 作为结束的图像
inputShadingImage	使用 CIImage 设定涟漪效果的阴影
inputCenter	CIVector 代表效果的中心点，也就是视觉效果的开始。默认的是 150，150
inputExtent	CIVector 代表输入图像的范围。默认的是 0,0,300,300.
inputTime	一个 NSNumber 代表变换的输入时间。默认的是 default:0.0,minimum:0.00,maximum:1.00,slider minimum:0.00 和 slider maximu:1.00
inputWidth	一个 NSNumber 代表输入图像的宽度。默认的值:default:100.00,minimum:1.00,maximum:0.00,slider minimum:10.00,slider maximum:300.00
inputScale	一个 NSNumber 代表输入缩放．默认的值是 :default:50.00,minimum:-50.00,maximum:0.00,slider minimum:-50,slider maximum:50.00.

涟漪变换看起来复杂。然而，当你创建它时，你仅仅需要在滤镜调用-setDefaults 就可以了。在层上使用滤镜时你必须指定的字段是 inputShadingImage。你也可能要指定 inputCenter 字段，以便于你可以指定变换原点的位置。

例如，我们为关键字”bounds”使用涟漪变换滤镜。这意味着无论什么时候层的边框字段改变时，涟漪的效果将会被使用代替默认的变换。在图 6-7 中，你可以看到不同阶段的效果。

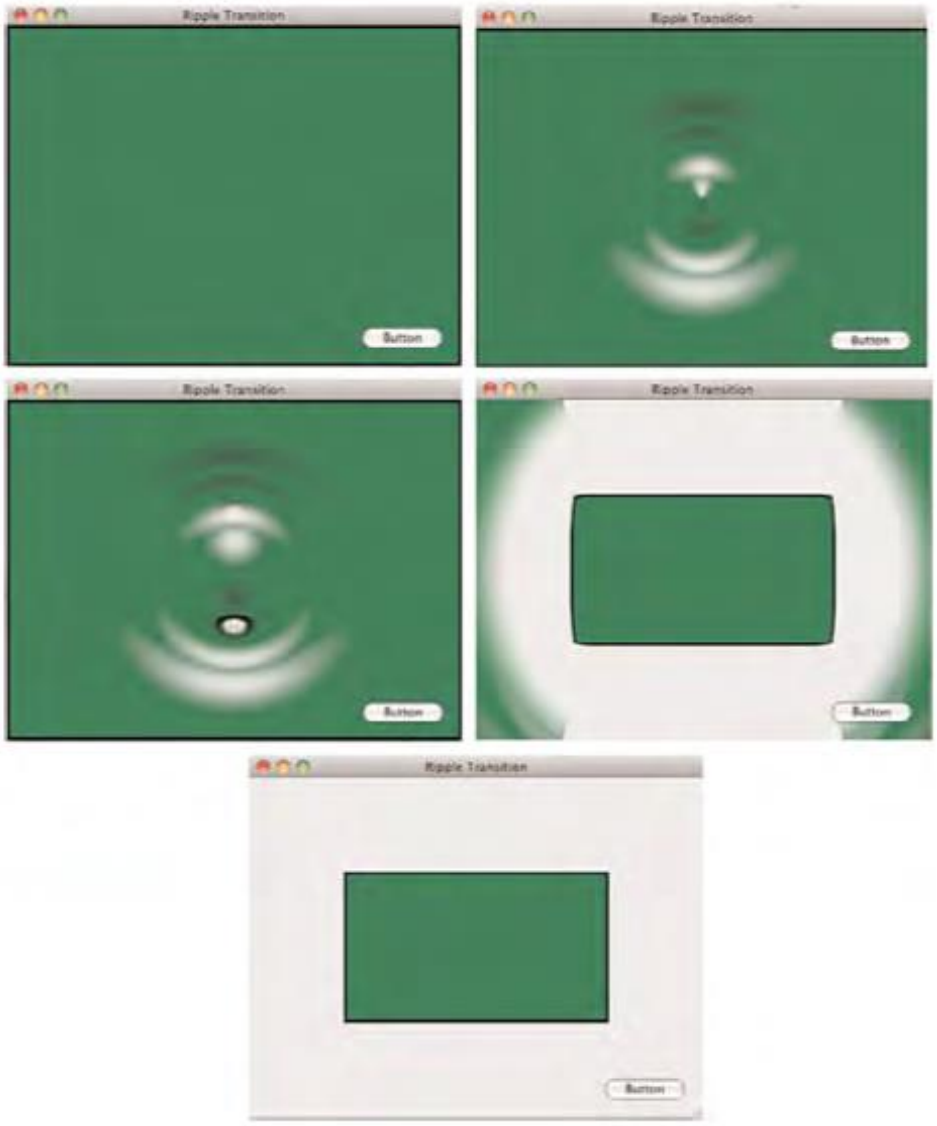

图 6-7 在核心动画层中涟漪变换层

使用封装来实现涟漪变换，只要简单的创建一个层的衍生类 RippleLayer。这个层的代码实现如清单 6-13.

```
@interface RippleLayer : CALayer { }
@end
@implementation RippleLayer
+ (id<CAAction>)defaultActionForKey:(NSString *)key; {
if( [key isEqualToString@”bounds”] )
{}
// Going to cheat a little bit here. Since this is a class
// method, we can’t get the dimensions from self. Instead,
// we need to obtain this size some other way in a real-world // application.
float w = 480.0;
```

```
float h = 360.0;
NSURL *url = [NSURL fileURLWithPath: [[NSBundle mainBundle]
pathForResource:@”Shading” ofType:@”tiff”]];
CIImage *shadingImage =
[[CIImage alloc] initWithContentsOfURL: url];
CIFilter *rippleFilter =
[CIFilter filterWithName:@”CIRippleTransition”]; [rippleFilter setDefaults];
[rippleFilter setValue:shadingImage
forKey:@”inputShadingImage”];
[rippleFilter setValue:[CIVector vectorWithX: 0.5*w Y: 0.5*h]
forKey:@”inputCenter”];
CATransition *theTransition=[CATransition animation]; [theTransition setFilter:rippleFilter]; [theTransition setDuration:2.0f];
return theTransition;
// Cause the layer to use its default transition
return nil; }
@end
```

清单 6-13 涟漪变换层

新的 rippleLayer 类在应用程序的代理类中使用，在 appDelegate 代码中实现如清单 6-14.

```
#import “RippleLayer.h”
@interface AppDelegate : NSObject { IBOutlet NSWindow *window; RippleLayer *transitionLayer;
BOOL toggle; }
-(IBAction)doTransition:(id)sender; @end
@implementation AppDelegate -(void)awakeFromNib;
{
[[window contentView] setWantsLayer:YES]; CGColorRef green =
CGColorCreateGenericRGB(0, 0.45, 0, 1);
transitionLayer = [[[RippleLayer alloc] init] retain]; [transitionLayer setFrame:
NSRectToCGRect([[window contentView] frame])];
[transitionLayer setBackgroundColor:green]; [transitionLayer setDelegate:self]; [transitionLayer setBorderWidth:3];
// Keep the layer behind the button
[[[window contentView] layer] insertSublayer:transitionLayer atIndex:0];
CFRelease(green); }
-(IBAction)doTransition:(id)sender; {
CGRect contentRect =
NSRectToCGRect([[window contentView] frame]);
if(toggle) {
[transitionLayer setBounds:contentRect]; }
else
{
CGRect newFrame = CGRectMake((contentRect.size.width / 2) - 100.0,
(contentRect.size.height / 2) - 100.0, contentRect.size.width - 200.0, contentRect.size.height - 200.0);
[transitionLayer setBounds:newFrame]; }
toggle = !toggle; }
- (void)dealloc; {
[transitionLayer release], transitionLayer = nil;
[super dealloc]; }
@end
```

清单 6-14 使用封装的代理实现

简单的，你可以实现涟漪滤镜的变换，当使用代理时，代码如清单 6-15.

```
@interface AppDelegate : NSObject {
  IBOutlet NSWindow *window;
CALayer *transitionLayer;
CIImage *shadingImage;
```

```
CIVector *extent;
BOOL    toggle;}
- (CIImage *)shadingImage;
-(IBAction)doTransition:(id)sender; @end
@implementation AppDelegate -(void)awakeFromNib;
{
[[window contentView] setWantsLayer:YES]; CGColorRef green =
CGColorCreateGenericRGB(0, 0.45, 0, 1);
transitionLayer = [CALayer layer]; [transitionLayer setFrame:
NSRectToCGRect([[window contentView] frame])]; [transitionLayer setBackgroundColor:green]; [transitionLayer setDelegate:self];
[transitionLayer setBorderWidth:3];
// Keep the layer behind our button.
[[[window contentView] layer] insertSublayer:transitionLayer atIndex:0];
CFRelease(green); }
- (id<CAAction>)actionForLayer:(CALayer *)layer forKey:(NSString *)key;
{
if( layer == transitionLayer ) {
if( [key compare:@"bounds"] == NSOrderedSame ) {
float w = [[window contentView] frame].size.width; float h = [[window contentView] frame].size.height;
CIFilter *rippleFilter =
[CIFilter filterWithName:@"CIRippleTransition"];
} }
[rippleFilter setDefaults];
[rippleFilter setValue:[self shadingImage]
forKey:@"inputShadingImage"];
[rippleFilter setValue:
[CIVector vectorWithX: 0.5*w Y: 0.5*h]
forKey:@"inputCenter"];
CATransition *theTransition=[CATransition animation]; [theTransition setFilter:rippleFilter]; [theTransition setDuration:2.0f];
return theTransition;
// Cause the layer to use its default transition
return nil; }
- (CIImage *)shadingImage {
if(!shadingImage) {
NSURL *url;
url = [NSURL fileURLWithPath:
[[NSBundle mainBundle] pathForResource:@"Shading" ofType:@"tiff"]];
shadingImage = [[CIImage alloc] initWithContentsOfURL: url];
}
return shadingImage; }
-(IBAction)doTransition:(id)sender; {
CGRect contentRect =
NSRectToCGRect([[window contentView] frame]);
if(toggle) {
[transitionLayer setBounds:contentRect];
}
else
{
CGRect newFrame = CGRectMake((contentRect.size.width / 2) - 100.0,
(contentRect.size.height / 2) - 100.0, contentRect.size.width - 200.0, contentRect.size.height - 200.0);
[transitionLayer setBounds:newFrame]; }
toggle = !toggle; }
@end
```

清单 6-15

无论你实现变换滤镜的方法是什么，-doTransition 方法是恒定不变的。它使用一个叫 toggle 的变量，来跟踪层是否覆盖了整个内容区域或者减小了尺寸。在层上调用-setBounds 方法就会触发变换动画的发生。当 toggle 是被设定时，我们就变换成整个视图的框架，而当它没有设定时，就变换成更小的框架。然后我们

就变换这个 toggle 变量。

总结，在封装和代理的主要不同是：

封装要求你创建一个自己的继承类，需要重载+(id<CAAction>)defaultActionForKey:(NSString*)key 方法；

代理要求你的应用程序代理实现-(id<CAAction>)actionForLayer: (CALayer *)layer，同时要设定的层的代理为 appDelegate 的实例，通常也就是自己。

6.4. 总结

无论是提供一个简单的视图效果，抑或增加用户界面的动画，或者改变滤镜的参数，或者改变默认的变换，核心图像滤镜在核心动画工具包中都是一个强大的有用的组件。