Core Animation:

Simplified Animation Techniques for Mac and iPhone Development

第三部分

核心动画的层

第九章

Quartz Composer 层

版本 1.0

翻译时间：2013-01-31

DevDiv 翻译： BeyondVincent(破船)
DevDiv 校对： 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 9 章　Quartz Composer 层

Quartz Composition 为开发复杂的可视化效果提供了一个方法。Quartz Composer 是一个可视化开发工具，通过它可以创建一些可视化的效果，例如屏保程序效果，图形的一些动画效果，这些效果只需要在工具中简单的拖拽，连接好各个 patch 的输入输出就可以了。

当然你也可以创建自己的 patches，不过在 Quartz Composer 的工具箱中有大量的 patches 可以选择使用。只需要简单的从工具箱中拖出喜欢的 patch 到 patch 编辑器中，就可以开始连接输入和输出了。

注意：ok，实际使用的时候没这么简单，从我上面的描述，你可以有一个简单的概念。如果你想学习关于 Quartz Composer 的更多知识，请看有 Graham Robinson 和 Surya Buchwald 写的书籍：Real-Time Motion Graphics with Quartz Composer。

输入一般是一些图片或者是从摄像头获取到的视频文件。输出则是复合的图片，它包含许多效果，比如滤光器、遮罩和转场等。

如果你对 Quartz Composer 不太熟悉，那么你可以在电脑中搜索.qtz 文件，然后再 Quartz Composer 中打开它们，就能看到各种类型的可视化效果。

在本章中，你将使用 Quartz Composer 创建一个简单的 Quartz 合成物。然后将这个合成物加载到 QCCompositionLayer，并添加到 Cocoa 控件中，以便控制合成物的各种输入值。

9.1. 使用 Quzrtz Composer 创建一个 Multi-Video 流

为了与前两章的内容一致，在这里我们继续使用视频来当做渲染的媒体。在这里的合成物有两个 Movie Loader 源 patches 和两个 Billboard patch 组成，其中 Movie Loader patches 是用来加载的 movies，Billboard 则负责显示加载的 movies。Movie Loader 的功能如名字一样——负责从磁盘中加载 movie 并将其用于合成物的输入源。它是源 patch 的一种类型。Billboard 负责渲染并显示视频。你将 Movie Loader 输出的图片连接到 Billboard 的输入图片，视频就将被渲染。

Quartz Composer 可以在/Developer/Applications 目录下找到。现在就打开它开始吧。

按照下面的步骤来创建合成物（composition）：

1、在 Quartz Composer 选择 File > New Blank 来创建一个新的合成物。
2、在 Root Macro Patch 中，从 Patch Browser 拖拽两个 Movie Loader 到编辑器中。
3、在 Root Macro Patch 中，从 Patch Browser 拖拽两个 Billboard 渲染器到编辑器中。
4、在 Root Macro Patch 中，从 Patch Browser 拖拽一个 Clear 渲染器到编辑器中。
5、点击一个 Movie Loader 的图片输出，并将其拖拽到一个 Billboard 的图片输入上。
6、点击另外一个 Movie Loader 的图片输出，并将其拖拽到另外一个 Billboard 的图片输入上。
7、点击 Clear patch 的数字徽章，将其设置为层 1.这样就能确保 clear 在最后面一层。

8、选择其中一个 Movie Loader，并在右边的 inspector 中的 Movie Location 输入参数里面提供一个 movie，如图 9-1 所示。在另外一个 Movie Loader 上重复该操作。

9、如图 9-2 所示，为了让视频并排显示，调整两个 Billboard 的 X 位置和宽度。

Billboard Layer 2，设置宽度为：0.85，X 位置为：-0.4856.

Billboard Layer3，设置宽度为：0.85，X 位置为：0.4856.

这样设置之后，视频就会并排显示了。

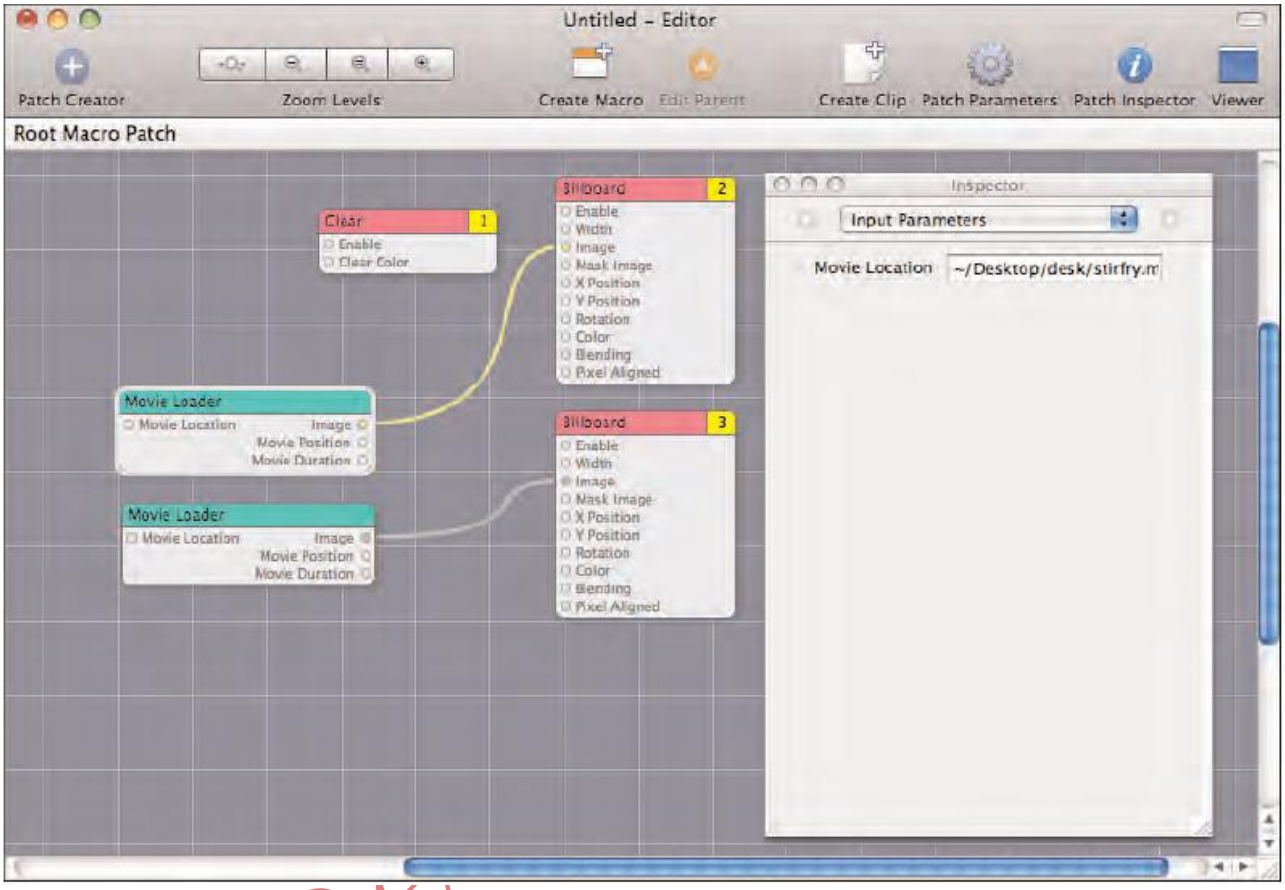

图 9-1 设置 Movie Loader 的 Movie 位置参数

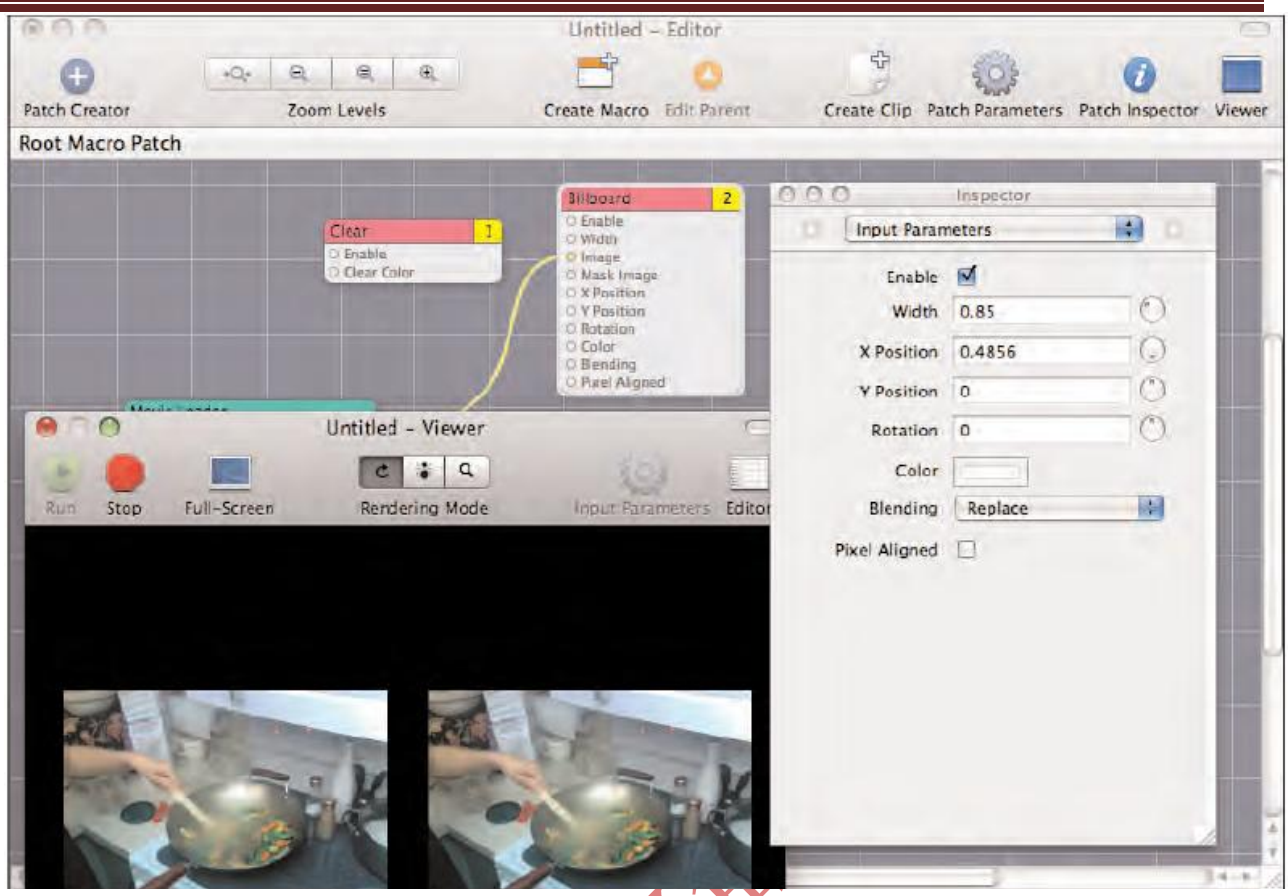

图 9-2 设置 Billboards 的 X/Y 和 Width/Height

10、 将合成物保持为一个.qtz 文件。

11、 如果合成物没有运行，那么点击 Run 按钮，就可以看到两个视频在不放。

9.1.1. 创建可控制的参数

使用 Quartz Composition 的强大之处就是可以提供合成物的可控输入和输出。QCCompositionLayer 提供的方法可以将设置任意参数，并将其发布到合成物中。

当我们创建合成物时，我们明确的指定了我们想要在 Movie Loader 中使用的 movie 文件。那么我们可以这样做吗：由用户来指定视频文件的路径，并将该路径传递给合成物。答案是可以的，通过将发布 Movie Loader 的 Movie Loaction 参数即可。

按照下面的步骤，可以发布 Movie Location 的输入参数：

1、在 Quartz Composer 中，在第一个 Movie Loader 上 Control-click 视频位置输入，然后选择 Published Input->Movie Location。

2、将输入命名为 movie1_location，然后按 Enter。

3、在另外一个个 Movie Loader 上 Control-click 视频位置输入，然后选择 Published Input->Movie Location。

4、将输入命名为 movie2_location，然后按 Enter。

5、保存合成物。

现在，可以通过程序代码来设置视频文件的位置。下面我将创建一个 Xcode 工程，并使

用 QCCompositionLayer 来显示合成物。

9.1.2. 创建一个 Xcode 工程

为了创建一个程序，按照下面的步骤进行：

1、在 Xcode 中，按 Shift-⌘-N，然后再工程模板对话框中选择 Cocoa 程序。

2、将工程命名为 Dual Video QC Composition 并单击保存。

3、展开 Framework group，在 Linked Framework 分组中 Control-click，然后选择 Add->Existing Frameworks

4、在结果对话框中，导航至/System/Library/Frameworks，选择 QuartzCore.framework 和 Quartz.framework。这里需要点击添加两次。

5、在 Classes 分组中 Control-click 并选择 Add>New File。

6、在新建文件模板对话框中，选择 Cocoa 分组下的 Objective-C class，并单击 Next。

7、将文件命名为 AppDelegate.m，并确保同时创建“AppDelegate.h”也被勾选上，然后单击 Finish。

8、在代码编辑器中点击并打开 AppDelegate.h 文件，然后添加如下代码：

```
@interface AppDelegate : NSObject {
IBOutlet NSWindow *window;
}
```

9、在代码编辑器中点击并打开 AppDelegate.m 文件，然后添加如下代码：

```
@implementation AppDelegate
- (void)awakeFromNib;
{
[[window contentView] setWantsLayer:YES];
}
@end
```

10、在工程的 Resources 分组中，双击 MainMenu.xib 文件，以在 Interface Builder 中打开这个 nib 文件。

11、从 Library 中，拖拽一个 NSObject 对象到 MainMenu.xib 文件中，并将其命名为 AppDelegate。

12、确保 AppDelegate 是选中的。然后再 object inspector 中，点击 Identity 选项卡，将 Class 字段修改为 AppDelegate。

13、在 MainMenu.xib 中，Control-click File’s Owner，然后将其拖拽到 AppDelegate 对象上，接着在出现的上下文菜单上选择 delegate。

14、在 MainMenu.xib 中，Control-click AppDelegate，然后将其拖拽连接到 Windows 对象上，接着在出现的上下文菜单上选择 Window。

15、保持好 XIB 文件，并返回到 Xcode 中。

现在工程已经配置好了。在上面的步骤中，我创建了一个程序 delegate，这样就可以再 view 和 data 之间提供一些功能。在下面的小节中，我将演示如何添加 Quartz Composition layer，QCCompositionLayer 到我们的 window 中。

9.1.3. 添加一个 QCCompositionLayer 到 Window 中

现在，在 AppDelegate 类中，已经为 windows 定义好了一个 outlet，我们可以将 QCCompositionLayer 添加到 window 的 view 的 root layer 中。如列表 9-1，演示了如何在-awakeFromNib 方法中实现：

列表 9-1 实现 QCCompositionLayer

```
- (void)awakeFromNib;
{
[[window contentView] setWantsLayer:YES];
NSString *path = [[NSBundle mainBundle]
pathForResource:@”DualStirfry”
ofType:@”qtz”];
qcLayer = [[QCCompositionLayer alloc] initWithFile:path];
// Set the composition frame to be the same
// size as the window content.
[qcLayer setFrame:NSRectToCGRect([[window contentView] frame])];
// Insert the layer at index zero so that any controls we add
// to the window won’t be obscured by the layer.
[[[window contentView] layer] insertSublayer:qcLayer atIndex:0];
}
```

这里的实例变量 qcLayer 被声明在 AppDelegate 头文件中。在-awakeFromNib 方法方法中初始化 qcLayer 可以保证我们的 window 和 view 在这之前即已经初始化好了，可以被使用了。

上面的代码中，注意到我们从 main bundle 中加载之前就创建好的 Quarzt 合成物。在这里，你首先需要将之前创建好的合成物添加到工程中：Control-click Resources 分组，然后选择 Add Existing Files。导航到合成物的位置，然后选中文件并单击 Add。现在文件就被添加到程序的 bundle 中了，当打包好时，你就可以通过列表 9-1 中的代码对其进行访问了。

9.1.4. 将参数传递给 Quartz 合成物

我们之前在创建 Quartz 合成物时，发布的参数命名为 movie1_location 和 movie2_location，现在我们可以在代码中通过键值对进行访问。在实际的合成物中，我们已经设置好了默认值，不过现在我们可以根据用户的选择对其进行设置。我们只需要添加一对按钮，和一个打开文件的对话框就可以让用户选择相关的输入参数了。

为了从选择的文件中获得路径，需要在 Interface Builder 中创建两个按钮，以及在 Xcode 中创建两个 IBActions。首先我们在 Xcode 中创建 action。如列表 9-2 演示了如何使用 NSOpenPanel 来获取视频文件的路径，并将其设置给 movie1_location。

列表 9-2 使用 NSOpenPanel 获得视频路径

```
- (IBAction)setMovie1Location:(id)sender;
{
NSOpenPanel *openPanel;
openPanel = [NSOpenPanel openPanel];
[openPanel setCanChooseDirectories:NO];
[openPanel setAllowsMultipleSelection:NO];
```

```
[openPanel setResolvesAliases:YES];
[openPanel setCanChooseFiles:YES];
if ([openPanel runModalForTypes:nil] == NSOKButton)
{
[qcLayer setValue:[openPanel filename]
forInputKey:@”movie1_location”];
}
}
```

你可以直接复制上面的代码,并粘贴到函数－setMovie2Location 中， 然后简单的将 InputKey 修改为 movie2_location。

Quartz Composition layer 提供了这样一个方法－setValue:forInputKey, 通过 QCCompositionRenderer 协议，来设置在合成物中发布的输入值。感兴趣的是什么呢？我们还可以通过代码调用－setValue:forKeyPath 来设置 movie location 字段的值。只需要简单的将－setValue 替换为如下即可：

```
[qcLayer setValue:[openPanel filename]
forKeyPath:@”patch.movie1_location.value”];
```

在 QCCompositionLayer 可用之前，你必须使用 QCView 对象将 Quartz 合成物显示在一个 Cocoa 程序中。如果你希望使用绑定来控制那些 patch，你还需要实例化一个 QCPatchController 对象。patch.movie1_location.value 是一个 keypath，如果你使用 QCPatchController，那么你是需要用到这个 keypath 的，因为“patch”引用到合成物中 root level patch。这种机制可以在 QCPatchController 中是隐藏的，所以，你可以使用绑定来访问这些值。当你希望使用 IB 的 Cocoa 控件可以绑定到某个值，这非常有用。

9.1.5. 从合成物中获取图片数据

与我们发布的视频位置输入一样，我们还可以发布输出，不过，首先我们需要修改合成物，对图片输出添加一个过滤器。然后我们就尅在 Cocoa 程序中获取到输出的 image 了。

下面几个步骤是如何给合成物中添加一个过滤器：

1、在 Quartz Composer 中打开合成物，然后再 Patch Creator 中从过滤器中拖拽一个 Dot Screen patch 到编辑区域中。

2、将第二个 Movie Loader 的 Image 输出拖拽到 Dot Screen patch 的 Image 输入上。

3、将 Dot Screen patch 的 Image 输出拖拽到第二个 Billboard 的 Image 输入中。

现在可以看到如图 9-3 的界面：

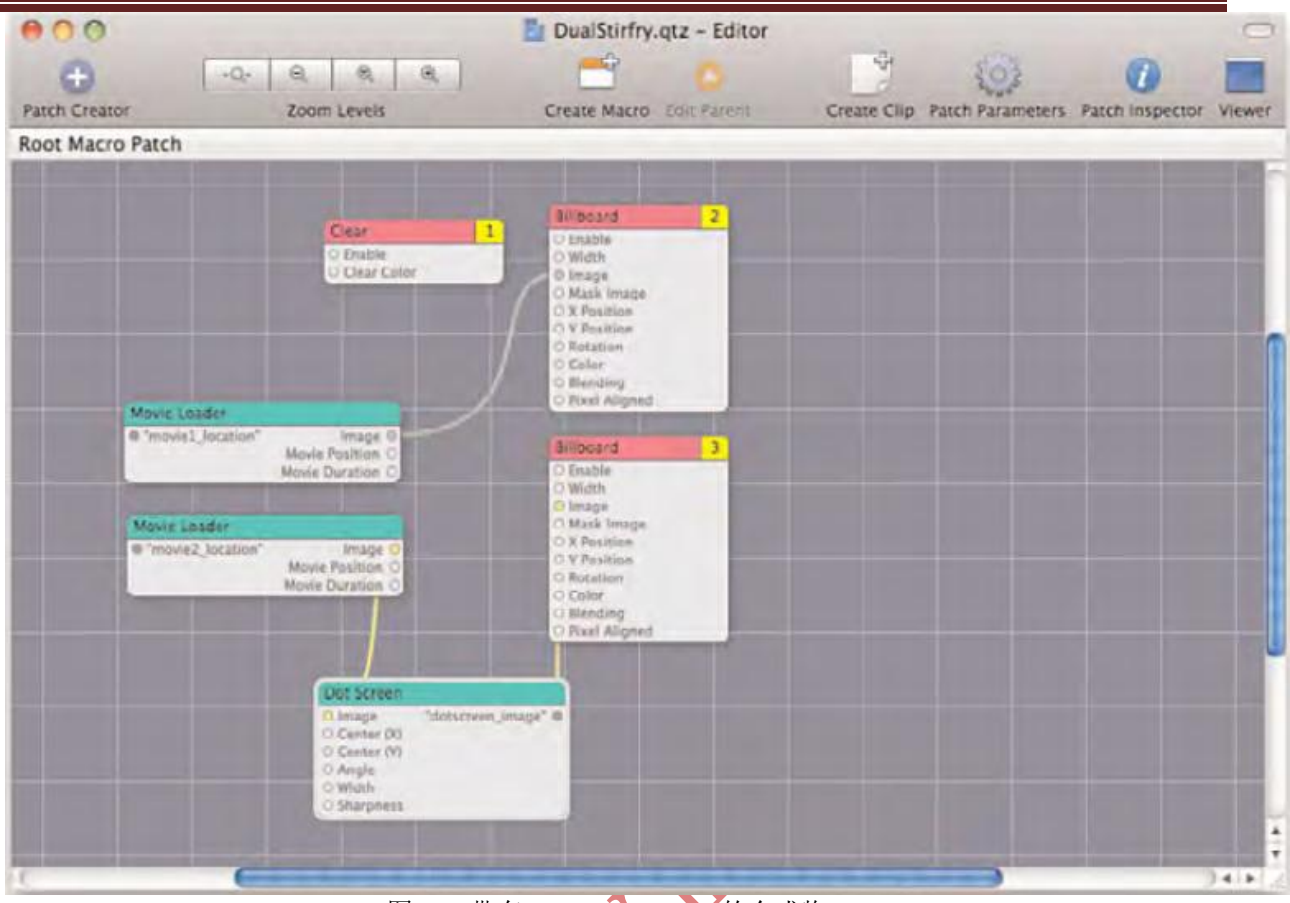

图 9-3 带有 Dot Screen Patch 的合成物

现在我们需要将 Dot Screen 的 Image output 进行发布，这样我们就能够在视频播放的任何时候都能通过过滤器获得当前的图片。按照下面的步骤，可以发布 Dot Screen 的 Image output：

1、在 Dot Screen 上 Control-click
2、选择 Published Outputs->Image
3、将其命名为 dotscreen_image，然后按下 Enter 键
4、保存合成物。

9.1.6. 通过代码获得当前图片

现在我们已经将 Image output 发布到合成物中了，我们可以通过代码获取到当前 image 的数据，并转换为一个 NSImage 对象。实际上，我们可以通过调用 -valueForOutputKey:ofType，并传入不同的格式，来获取不同格式的图片数据。在下面的示例代码中，我简单的将当前图片设置给 content 字段。在这种情况下，我们想要获取一个 CGImageRef，而不是 NSImage。这其实并不重要，因为我们总是需要在调用 valueForOutputKey 方法时指定 CGImage 作为参数。在列表 9-3 中，演示了如何将当前 image 获取为 CGImageRef，并将其设置给顶层的 content。

列表 9-3 获取 Image 数据位 CGImageRef

```
- (IBAction)getDotScreenImage:(id)sender;
{
```

```
CGImageRef image = (CGImageRef)
[qcLayer valueForOutputKey:@”dotscreen_image”
ofType:@”CGImage”];
[[[window contentView] layer] setContents:(id)image];
}
```

在上面的代码中，我请求从合成物中以 CGImage 类型获取 dotscreen_image output，然后得到一个 CGImageRef 对象，可以将其设置给 window 的 content view 层。

调用- valueForOutputKey:ofType 可以返回 NSImage，NSBitmapImageRep，CGImage，CIImage，CVPixelBuffer，CVOpenGLBuffer 和 CVOpenGLTexture。这只需要简单的将 CGImage 替换为你需要的图片类型即可。

注意到-getDotScreenImage 是一个 IBAction。在 Interface Builder 中，需要在 window 中添加一个按钮，并将其连接到 AppDelegate 中的这个 action 上。图 9-4 显示了这个示例程序运行的效果图。

图 9-5 显示了当 Get Image 按钮被按下时，程序运行的效果图。它是将 root layer 的 content 字段设置为从 movie 2 中获得的当前图片，也就是已经应用了过滤器的那个视频。

那个 Quartz 合成物的 input 和 output，你可以决定哪个可以再代码中可用。你可以开发复杂的视觉效果，然后再通过编程的方式控制它们。你只需要记住你只能访问在 Root Patch 中可用的 input 和 output。如果你创建了一个 macro patch（一个 patch 封装了另外一个 patch），那么你需要将 input 和 output 暴露在 Root Patch 级别，否则你将不能通过调用-setValue:forInputKey 和-valueForOutputKey:ofType 方法进行访问。

图 9-4 运行 QCCompositionLayer 示例工程

图 9-5 将 Root Layer 的 content 设置为当前的过滤图片

9.2. Quartz Composition 层与 OpenGL

在 QCCompositionLayer 中，可以随处使用 OpenGL 的强大功能，QCCompositionLayer 是直接继承自 CAOpenGLLayer 的，在第八章中我详细的介绍了“OpenGL 层”。在那章中，我们学到了，首先需要使用 -canDrawInCGLContext 判断是否需要进行绘制，然后使用 -drawInCGLContext 在上下文中进行绘制。如果我们创建自己的 QCCompositionLayer（为了添加一些额外的功能），那么我们可以重载这两个方法。

这将会给你提供强大的功能：在显示当前帧之前可以做任何你想要的渲染功能。记住合成物在播放，也就是说它们是按照最基本的进行渲染到屏幕上的。它的频率不是你能控制的，这跟你的硬件和视频刷新率有关。不过，如果-drawInCGLContext 被调用了，那么下一帧也已经准备好渲染了，而 drawInCGLContext 给你提供了一个 CGLContextObj 对象，你可以对其进行你自己的渲染。列表 9-4 演示了继承自 QCCompositionLayer 的类。

列表 9-4 默认继承自 QCCompositionLayer 的实现

```
@interface QuartzLayer : QCCompositionLayer {
}
@end
@implementation QuartzLayer
- (BOOL)canDrawInCGLContext:(CGLContextObj)glContext
pixelFormat:(CGLPixelFormatObj)pixelFormat
forLayerTime:(CFTimeInterval)timeInterval
displayTime:(const CVTimeStamp *)timeStamp;
```

```
{
return [super canDrawInCGLContext:glContext
pixelFormat:pixelFormat
forLayerTime:timeInterval
displayTime:timeStamp];
}
- (void)drawInCGLContext:(CGLContextObj)glContext
pixelFormat:(CGLPixelFormatObj)pixelFormat
forLayerTime:(CFTimeInterval)timeInterval
displayTime:(const CVTimeStamp *)timeStamp;
{
// Call super to ensure the OpenGL context gets flushed.
[super drawInCGLContext:glContext
pixelFormat:pixelFormat
forLayerTime:timeInterval
displayTime:timeStamp];
}
@end
```

重载中-canDrawInCGLContext 的调用取决于你的具体情况。当在渲染视频时，我调用这个方法来决定视频的下一帧是否准备好绘制了。如果准备好了，返回 YES；否则，返回 NO。然后，如果返回的是 YES，－drawInCGLContext 处理的绘制将在 OpenGL 上下文中进行。如果你想理解相关内容，看一下第八章。

9.3. 总结

Quartz composition 提供了一个创建复杂的可视化效果。它只局限于你的思维。这听起来感觉比较老套，不过这就是事实。当你的程序需要动画，或者其它可视化元素，你可能有时间去尝试用代码来实现处理，不过如果视觉效果非常复杂的时候，你可能需要重新考虑一下使用 Quartz composition 了。

为什么你可以很简单的就能通过代码对画面添加一些元素和效果，并对合成物进行控制？当你开始思考这些时，你的思维已经在发散了。核心动画 Quartz Composition Layer、QCCompositionLayer 提供了可视化的方法来在你的 Cocoa 程序中与合成物进行配合使用。

点击这里访问：DevDiv.com 移动开发论坛