Core Animation:

Simplified Animation Techniques for Mac and iPhone Development

第三部分

核心动画的层

第七章

视频层

版本 1.0

翻译时间：2012-12-03

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 7 章 视频层

这一章来关注 QTMoviLayer，一个轻量级的核心动画的层，它提供了一个方法，可以利用 QTKit 框架中的 QTMovie 对象来播放音频和视频电影。

我们也会关注 QTCaptureLayer，一个轻量级的核心动画层，它提供了一个框架，那个框架可以捕获图像设备的帧，例如捕获现在的 mac 电脑中都有的视频摄像头中的图像信息。创建捕获会话是最困难的部分，但在你设置了这个部分后，你可以使用任何你想要捕捉图像的摄像机的图像层。

QuickTime 层提供了所有你需要的，比基于视图的部分具有更高水平的功能。这一章展示给你，利用 QuickTime 技术，通过使用层你可以获得多少东西。

7.1. 利用 QTMovieLayer 工作

QTMovieLayer 的 API 是简单的。对于一个 QTMovieLayer 仅仅有三个独立的条目：

+layerWithMovie:
-initWithMovie:
-movie

前两个是初始化层的，第三个是初始化层后，用来得到 QTMovie 对象的引用。另外其他的特性都可以简单的通过父类 CALayer 获得。

来自 QTKit 包的 QTMovie 对象提供了一些你需要的影视控制，例如回放，擦洗，快进等等。不同的是 QTMovie 没有提供可视的元素，而 QTMovieView 则提供了。并且，你还需要绑定你的行动到出口例如滑动块和按钮。在 QTMovie 中可用的行动方法如表 7-1.

方法	描述
-autoplay	Autoplay 和 play 做同样的事情，除了它是利用流媒体的。只有足够的数据可用时，它才开始播放影视
- play	开始影视播放
- stop	停止影视播放
- gotoBeginning	在 QTMovie 对象中，设定 currentTime 字段为 QTMoviZero，然后开始播放。
- gotoEnd	在 QTMovie 对象中，设定 currentTime 字段为影视的时间段。
- gotoNextSelectionPoint	如果一些选择点是被设定，设置 currentTime 字段到选择的点上。
- gotoPreviousSelectionPoint	如果一个选择点是被设定，并且这个选择点的时间早于 currentTime 字段记录的时间，那么就设定 currentTime 字段到选择点的时间。

- gotoPosterFrame	在 QTMovie 对象中，设定 currentTime 字段到预览帧的时间。如果没有预览帧指定，currentTime 就设定影像的开始时间。
- setCurrentTime:	在 QTMovie 对象中，设定 currentTime 字段，使用 QTTime 对象，你指定它作为参数。
- stepBackward	回退一帧，同事 currentTime 字段也被改变。
- stepForward	快进一帧。currentTime 字段同时被改变。

你可以简单的创建一个 IBAction 的方法在控制器中，在 IBAction 方法中，调用上面的方法。看即将来到的部分，”增加基础的播放控制”，来看看是怎么做的。

7.1.1. 创建一个简单的基于 QTMovieLayer 的层

这一章的例子程序叫做层上的影视播放器(在合作网站上有实例代码)，我们创建了一个应用程序的代理类，叫做 AppDelegate，我们在 interface Builder 中连接这个控制器对象。带着这些设定，AppDelegate 类给了我们一个实体的入口来创建和展示 QTMovieLayer。在这个工程中，我们增加了 QuartzCore 框架和 QTKit 框架。它是定位在 /Developer/SDKs/MacOSX10.6.sdk/System/Library/Frameworks/QTKit.framework 这个目录下，前提是要安装了开发工具。

如果你想播放一个电影，你需要在 apDelegate 中简单的实现 playback 在-awakeFromNib 方法中，代码如清单 7-1。

```
- (void)awakeFromNib; {
[[window contentView] setWantsLayer:YES];
NSString *moviePath = [[NSBundle mainBundle]
pathForResource:@”stirfry” ofType:@”mov”];
movie = [QTMovie movieWithFile:moviePath error:nil];
if( movie ) {
QTMovieLayer *layer = [QTMovieLayer layerWithMovie:movie];
[layer setFrame:NSRectToCGRect([[window contentView] bounds])];
[[[window contentView] layer] addSublayer:layer];
[movie play];
} }
```

清单 7-1 简单的实现影视播放

代码的开始，通过用-setWansLayer 方法通知窗口的 contentView 它后面应该放置层。下面就从主要的包中实例化一个 QTMovie 对象 。如果视频是合法的，我们就用 QTMovieLayer 初始化它。下面代码，就是设定层的帧的框架大小为内容视图的框架大小，并且增加层作为内容视图的子层。然后开始视频的播放。就这样。这是你需要在 QTMovieLayer 上播放视频的全部的步骤。话说回来，如果我们不给于一些方法来控制回放，它就不是一个非常有用的播放器。

7.1.2. 增加一些基础的播放控制

现在我们有了基础影视层的创建，下面要做的事就是为视图增加一些控制，使我们在影视中可以播放，暂停和回放或者快进。在 interface builder 中，增加一个按钮来开始和停止，并且伴随 2 个按钮，一个用来快进一帧，一个用来后退一帧。

```
- (IBAction)togglePlayback:(id)sender;
{ if( [movie rate] != 0.0 )
[movie stop];
else
[movie play];
}
- (IBAction)stepBack:(id)sender; {
[movie stepBackward];
[self updateSlider:nil]; }
- (IBAction)stepForward:(id)sender; {
[movie stepForward];
[self updateSlider:nil]; }
- (IBAction)goToBeginning:(id)sender; {
[movie gotoBeginning];
[self updateSlider:nil]; }
- (IBAction)goToEnding:(id)sender; {
[movie gotoEnd];
[self updateSlider:nil]; }
```

清单 7-2 控制按钮的实现

这些行动都是连接了我们在 Interface Builder 中创建的按钮。图 7-1 展示了基础控制按钮的截屏。

图 7-1 播放控制按钮

当你运行工程时，注意到我们在 interface builder 中增加到窗口的按钮不可见。问题是按钮是被影视层盖住了。需要把按钮都放在前面，增加影视层到根层，方法就是通过调用-insertSublayer:atIndex 把它放到第 0 个位置。最后，我们就需要改变-awakFromNib 中-addSublayer 中的调用了，如下：

```
[[[window contentView] layer] insertSublayer:layer atIndex:0];
```

现在，如果你运行工程，按钮都是在视频的上面了，能够使你点击和控制视频了。

7.1.3. 使用滑动块跟踪进度和改变帧

为了完成我们的基础播放器功能，我们给 interface 增加了一个滑块，滑块就提供了一

个播放的进度条，在视频的顶端还展示了播放的时间。首先，在 Xcode 中创建一个行动来控制滑块的改变，如清单 7-3.

```
- (IBAction)sliderMoved:(id)sender; {
long long timeValue = movieDuration.timeValue * [slider doubleValue];
[movie setCurrentTime:QTMakeTime(timeValue,
movieDuration.timeScale)];
    }
```

清单 7-3 实现滑块的行动

滑块有一个 0.0 的最小值和 1.0 的最大值。在 interface Builder 中，通过鼠标点击选择滑块 设置这些值，然后在检测器属性中设定滑块的最小值和最大值。滑块的值提供了一个小数的值，这个小数的值是视频总时间的值乘以目前的值。然后就可以根据滑块改变的位置设定 QTMovie 对象的当前时间了。为了做这些，需要调用 QTMakeTime 函数根据你计算的总时间来构造 QTTime 结构体，然后通过-setCurrentTime 传递给视频。这就可以使 QTMovieLayer 更新视图到目前的帧。

为了获得滑块目前的值(在 Interface Builder 中指定的 0.0 和 1.0 之间的值)，你需要在 AppDelegate 中创建一个 SSlider 的接口，在头文件中增加如下的行：

IBOutlet NSSlider *slider;

在 Interface Builder 中，为窗口增加一个 NSSlider 的控制器，然后从 AppDelegate 到滑块控制器上做一个连接到代理的-sliderMoved 这个行动上。图 7-2 和 7-3 演示了怎么在 Interface Builder 中来拖拽这些连接。

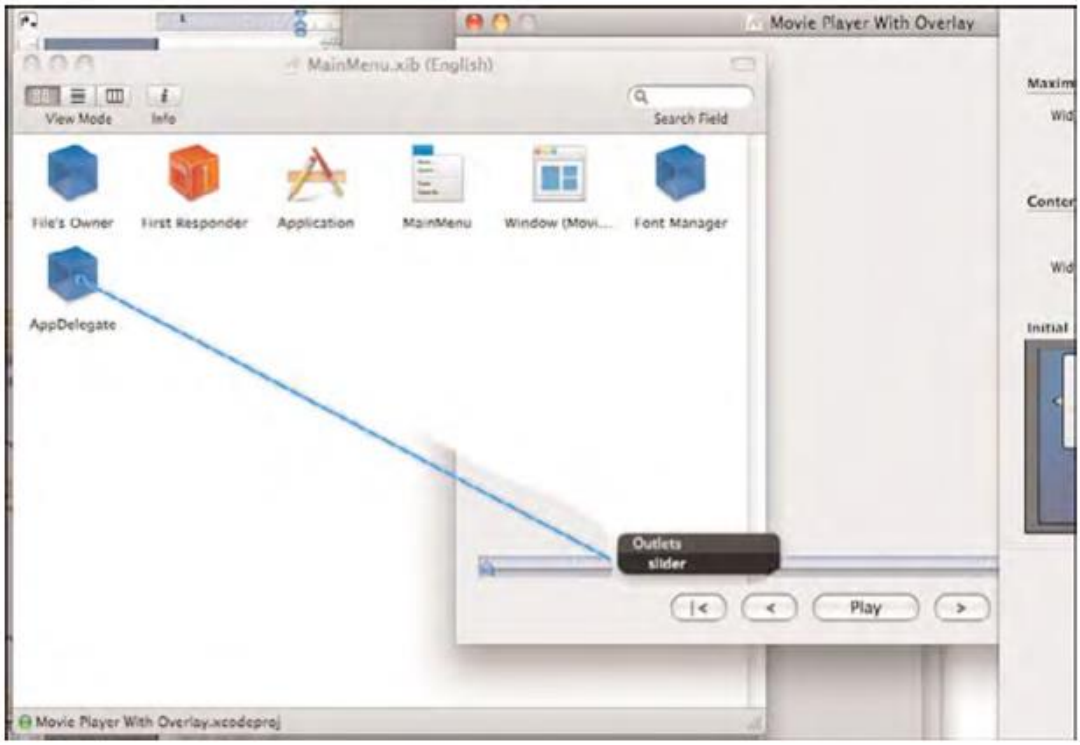

图 7-2 从 AppDelegate 到滑块上拖拽连接

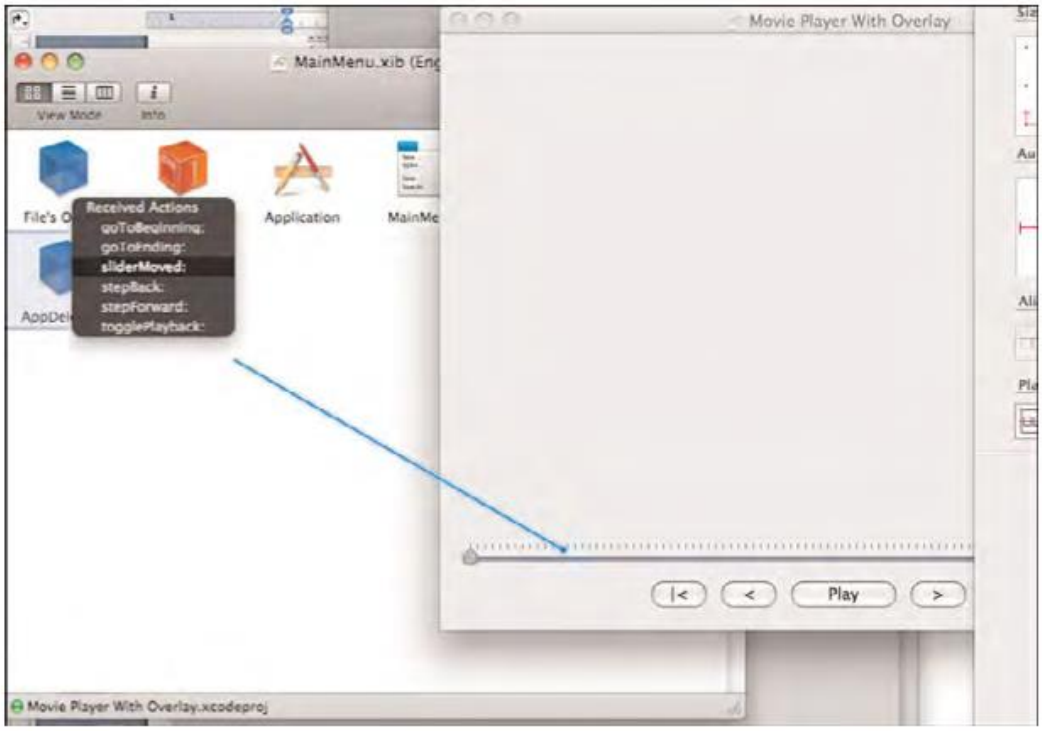

图 7-3 从滑块到 AppDelegate 上拖拽连接

为了更新滑块目前的位置，只要播放视频时，开启一个定时器就可以实现更新了。为了做这些，更新-togglePlayback 行动，代码如清单 7-4.

```
- (IBAction)togglePlayback:(id)sender; {
if( [movie rate] != 0.0 ) {
[movie stop];
[timer invalidate]; }
else
{
[movie play];
timer = [NSTimer scheduledTimerWithTimeInterval:0.02
target:self selector:@selector(updateSlider:) userInfo:NULL
repeats:YES];
} }
```

清单 7-4 实现定时器

无论影视什么时候播放，实例变量 timer 都会被实例化。当影视是被停止时，我们需要通过调用-invalidate 停止定时器。

定时器的每个滴答都调用-updateSlider。实现 selector 的代码如清单 7-5.

```
- (void)updateSlider:(NSTimer*)theTimer; {
QTTime current = [movie currentTime]; double value = (double)current.timeValue /
(double)movieDuration.timeValue;
[slider setDoubleValue:value];
[slider setNeedsDisplay]; }
```

清单 7-5 实现定时器的回调方法

回调函数中首先获取目前的时间，然后除于影视的总时间，总时间是存储在

movieDuration 这个实例变量中。这样就返回一个 0.0 到 1.0 之间的值，就可以用来给滑块设定了。滑块的值被设定成这个百分比，然后调用-setNeedsDisplay 来重绘滑块界面。

7.1.4. 增加覆盖的层

给影视上面覆盖一个层，对于一个 QuickTime 开发者来说是经常做的一件事。当你使用 QTMovieView 时，就有点挑战了，因为视图是重量级的并没有提供简单的方法可以增加子视图，来完成这个特点。如果你使用视图，要需要增加一个无边框的子窗口，该窗口包含覆盖层的内容，或者你需要使用 OpenGL 资源来实现它。这两个解决方案都会增加大量的代码，并使应用程序复杂，因此让我们来看看核心动画如何简单的完成该任务。

为了使用核心动画增加一个覆盖层，需要创建一个继承于 CALayer 的层，然后在 QTMovieLayer 上调用-addSublayer。为了演示这些，-awakeFromNib 中改变了清单 7-1 的代码如清单 7-6，增加了一个 CATextLayer 来覆盖到影视上。

```
- (void)awakeFromNib; {
[[window contentView] setWantsLayer:YES];
  NSString *moviePath = [[NSBundle mainBundle]
pathForResource:@"stirfry" ofType:@"mov"];
  movie = [QTMovie movieWithFile:moviePath error:nil];
if( movie ) {
NSRect contentRect = [[window contentView] bounds];
QTMovieLayer *layer = [QTMovieLayer layerWithMovie:movie];
[layer setFrame:NSRectToCGRect(contentRect)];
textLayer = [CATextLayer layer];
[textLayer setString:@"Do Not Try This At Home!"];
[textLayer setAlignmentMode:kCAAlignmentCenter];
[textLayer setFrame:NSRectToCGRect(contentRect)];
[layer addSublayer:textLayer];
[[[window contentView] layer] addSublayer:layer];
[movie play]; }
}
```

清单 7-6 实现覆盖的层

清单 7-6 增加了实例化 CATextLayer 的代码，作为 QTMovieLayer 的子层。文案居中可以通过调用 setAlignmentMode:kCAAlignmentCenter 来实现。现在开始播放影视，你将会看到文案”Do Not Try This at Home!”展示在框架的顶端。

图 7-4 显示了基本的层看起来怎么样。注意到了设定的文案层的框架大小同窗口视图框架的大小一样。这样，文本就展示在图片的最顶端了。

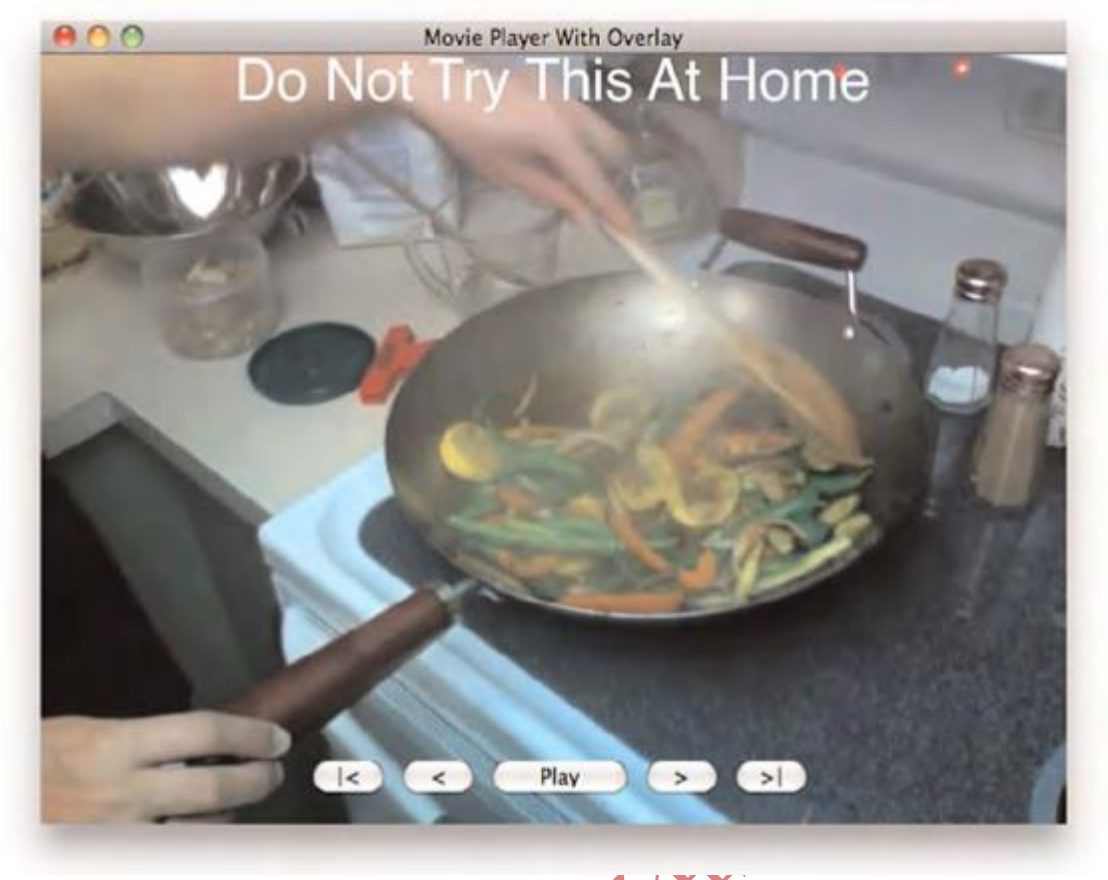

图 7-4 展示了“Do Not Try This at Home”的覆盖层

7.1.5. 覆盖一个时间文案的代码

当播放电影时，一个常见的要求就是能在播放的同时看到电影播放的时间。再次，使用核心动画的层，使显示时间文案层的代码变得简单。为了完成这些，第一步在先前的例子中创建一个 CATextLayer，然后通过使用-setString 更新 CATextLayer 的字符串显示。

在清单 7-5 中我们使用了定时器来更新滑块的位置。这里我们同样利用定时器，通过调用获得影视目前的播放时间，来更新文案的内容。

现在创建一个函数叫做-updateTimeStamp 来升级时间文案展示的代码，如清单 7-7.

```
- (void)updateTimeStamp; {
NSString *time = QTStringFromTime([movie currentTime]);
[textLayer setString:time]; }
```

清单 7-7 实现时间的更新

下一步，你需要做很少的工作，就可以让时间文案层的代码工作。首先，改变-awakeFromNib 的调用，设定文本层开始的字符串时间为 0，QTZeroTime，代码展示如清单 7-8.

```
- (void)awakeFromNib; {
[[window contentView] setWantsLayer:YES];
NSString *moviePath = [[NSBundle mainBundle]
pathForResource:@”stirfry” ofType:@”mov”];
movie = [QTMovie movieWithFile:moviePath error:nil];
```

```
if( movie ) {
NSRect contentRect = [[window contentView] bounds];
QTMovieLayer *layer = [QTMovieLayer layerWithMovie:movie];
[layer setFrame:NSRectToCGRect(contentRect)];
textLayer = [CATextLayer layer];
[textLayer setString:QTStringFromTime(QTZeroTime)];
[textLayer setAlignmentMode:kCAAlignmentCenter];
[textLayer setFrame:NSRectToCGRect(contentRect)];
[layer addSublayer:textLayer];
[[[window contentView] layer] addSublayer:layer];
[movie play]; }
}
```

清单 7-8 设定影视时间的初始化代码

QTZeroTime 是 QTTime 结构体，它代表视频的开始时间。下面，改变定时器设定的-updateSlider 方法，如清单 7-9 所示。

```
- (void)updateSlider:(NSTimer*)theTimer; {
QTTime current = [movie currentTime];
double value = (double)current.timeValue / (double)movieDuration.timeValue;
[slider setDoubleValue:value]; [slider setNeedsDisplay];
[self updateTimeStamp]; }
```

清单 7-9 调用时间间隔的更新

带着这些改变，时间文案层就会随着电影的播放实时的更新。在 Xcode 中，导入电影播放的例子工程，然后点击 build 和看一看层的行动。如图 7-5 展示了运行时的时间文案层的变化。就像你注意的，在正常的情况下时间会更新。

图 7-5 展示时间层

7.1.6. QTMovieLayer 和 contentsRect

在第二章中，“我们可以做什么动画？”，你看到了所有可用的动画参数。这些参数中最有趣的一个参数 contentsRect，这个参数是用来指定你要做的动画需要显示的区域。当关联到 QTMovieLayer 上时，这个非常有趣的，因为你可以复制 QTMovieLayer 中的内容给另一个标准的 CALayer，然后随着这样做，你可以指定 QTMovie 的那些部分需要在层内容中显示。

因此你马上会知道怎么的有用。假如你创建了包含了很多格子的独立视频。你就可以使用这个视频的引用然后复制内容到其他 CALayers 中，然后在每个层中指定视频层的某个要显示的部分。或者你可以吧一个影视分割成很多的格子。图 7-6 展示了这个应用的一个截图，你可以从合作网站获得演示的工程 CopyMovieContents.

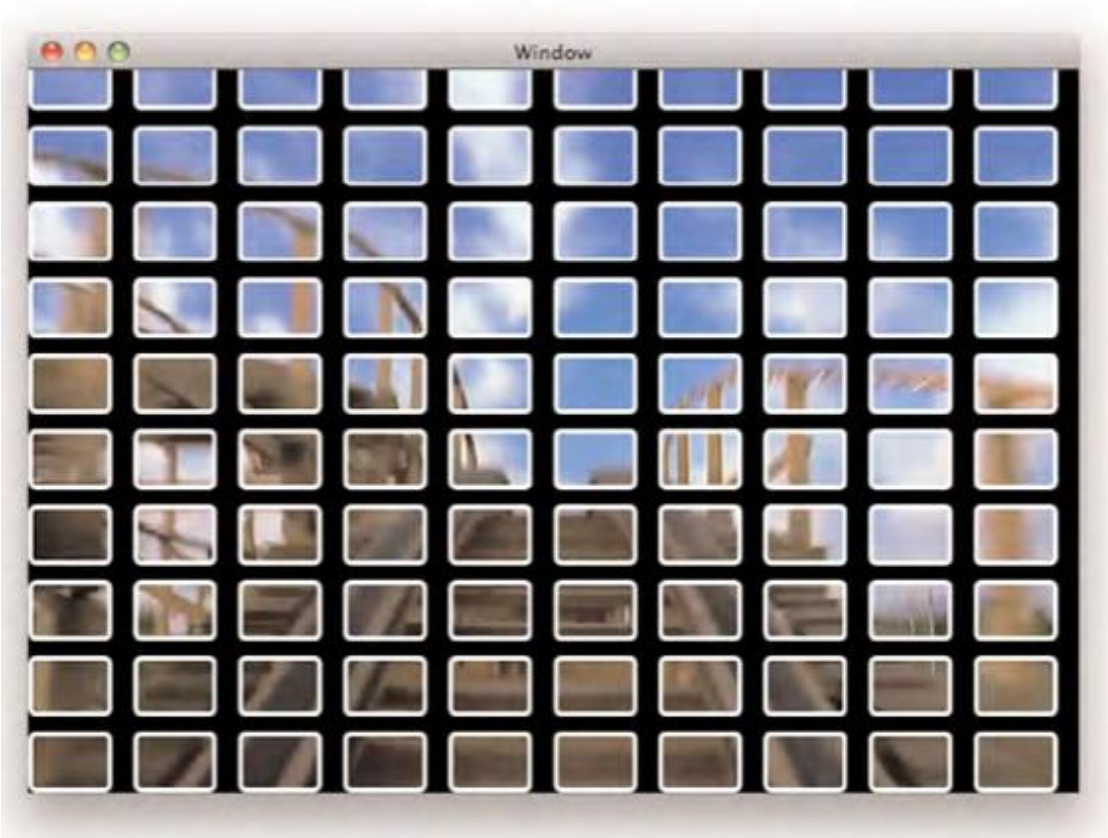

图 7-6 拷贝影视的内容到单独的 CAlayers 上

你看到的格子中的每个区域都是一个独立的 CALayers，通过从相同的 QTMovieLayer 中获得内容。对于这个应用程序，我们创建了一个 QTMovieLayer 的过上车的影视，影视文件的位置在/System/Library/Compositions/Rollercoaster.mov.

我们之后给我们的视图增加层。然后我们增加我们每个独立的 CALayers，这些 CALayers 都是用内容和 contentRect 设定的。

实现的代码非常的简单。你给每个 CALayer 设定影视层的内容，如下面的代码：

```
[layer setContents:[movieLayer contents]];
```

然后设定 contentsRect 来显示你想要的影视层要显示的部分。你设定 contentRect 字段如下：

```
[layer setContentsRect:CGRectMake(0.25f, 0.25f, 0.25f, 0.25f)];
```

就像你在第二章中回忆到的，这就造成了 CAlayer 的内容仅仅显示影视层的 1/4,也就是距离左下角高和宽分别为影视层高和宽的 1/4。

不过，这些技巧在下面我们要讲的 QTCaptureLayer 上不能工作。

7.2. 使用 QTCaptureLayer

QTCaptureLayer 提供了一个方法，来展示连接在电脑上的视频设备捕获的视频。这些设备包含视频摄像头或者一个通过火线连接的视频摄像头。为了捕获这些视频，你需要设定一个 QTCaptureSession，要做 2 件事：

它要提供一个接口，可以接收捕获到的帧数。

它可以使你保存图像到一个影视文件中，并且控制会话来关闭 QTCaptureLayer。

捕获视频最复杂的部分，并不是显示图像到 QTCaptureLayer 上。而是，设定 QTCaptureSession 这个对象。这些设定要求一些步骤，包括获取设备，打开设备，增加设备到捕获对话中，以及创建一个视频输出对象来增加行列帧到捕获对话中。

如果你打开例子工程叫做 Photo Capture，然后建立工程，将会帮助你理解。当你运行时，你会看到如图 7-7 中屏幕所示。

图 7-7 图像捕获窗口的例子工程

清单 7-10 展示了创建 QTCaptureSession 对象所要的代码。

```
- (void)initCaptureSession; {
if(captureSession == nil) {
NSError *error = nil;
captureSession = [[QTCaptureSession alloc] init];
```

```
// This finds a device, such as an iSight camera
QTCaptureDevice *videoDevice = [QTCaptureDevice defaultInputDeviceWithMediaType: QTMediaTypeVideo];
if (videoDevice == nil) {
// Try a different device, such as miniDV camcorder
videoDevice = [QTCaptureDevice defaultInputDeviceWithMediaType: QTMediaTypeMuxed];
}
// No need to continue if we can't find a device
if (videoDevice == nil) return;
// Try to open the device
[videoDevice open:&error];
// No need to continue if device couldn't be opened
if( error != nil ) return;
// Create a device input object to add to the capture session
QTCaptureDeviceInput *input = [[QTCaptureDeviceInput alloc] initWithDevice:videoDevice];
[captureSession addInput:input error:&error];
if( error != nil ) return;
// Create video output to add raw frames to the session
output = [[QTCaptureDecompressedVideoOutput alloc] init];
  [captureSession addOutput:output error:&error];
if ( error != nil ) return;
[self setSession:captureSession];
} }
```

清单 7-10 初始化捕获会话

这些相互关联的部分代码提供了一些相当神奇的东西。苹果用这些抽象使你能够容易的及时捕获和展示帧。随着 QTCaptureSession 的创建成功，下一步要做的就是通过 QTCaptreDecompressedVideoOutPut 的代理对象捕获任意时间的帧。在 QTCaptreDecompressedVideoOutPut 对象被申请时，设置它的代理给自己如下：

```
[output setDelegate:self];
```

它能够使你捕获图像文件或者影视对象。下面，让我们更进一步看下 QTCaptureLayer 是如何在捕获会话中初始化的。

7.2.1. 创建和展示 QTCaptureLayer

尽管可以通过 alloc 和 init 直接创建一个 QTCaptureLayer 对象，或者用便捷的方法像 +layerWithSession，我们还是准备子类化 QTCaptureLayer，这样通过在子类的初始化方法中可以让我们隐藏 QTCaptureSession 的初始化。我们这样做的原因是便于我们封装所有的捕获功能到层上，这样无论应用程序想在哪应用捕获功能时，都能复用这个层。

在图像捕获的实例代码中，我们创建了继承自 QTCaptureLayer 的子类并且命名它 CaptureLayer。清单 7-11 展示了在初始化时的 init 代码。

```
- (id)init; {
self = [super init];
if( !self ) return nil;
[self initCaptureSession]; return self;
}
```

清单 7-11 继承 QTCaptureLayer 的初始化方法

就像你看到的，我们在清单 7-10 中都已经调用了-initCaptureSession 这个方法。这里，当我们初始化一个 CaptureLayer 对象的同时，QTCaptureSession 就已经安装好准备运行了。下一步，增加 CaptureLayer 到窗口的根层树上。你可以看到在 AppDelegate 的-awakeFromNib 代码中是如何实现的，如清单 7-12 所示。

```
-(void)awakeFromNib; {
[[window contentView] setWantsLayer:YES];
captureLayer = [[CaptureLayer alloc] init];
// Use the frame from the generic NSView we have // named captureView
[captureLayer setBounds: NSRectToCGRect([captureView frame])];
[captureLayer setPosition:
CGPointMake([captureView frame].size.width/2, [captureView frame].size.height/2)];
[[captureView layer] insertSublayer:captureLayer atIndex:0];
[captureLayer starCaptureSession]; }
```

清单 7-12 在 AppDelegate 中实现 CaptureLayer

当你运行这些代码时，你会看到我们在窗口的左边创建了一个视图，视图是被你的视频摄像头或者 miniDV 照相机获得的东西填充。

7.2.2. 捕获目前的图像

自然的，你就想要捕获目前的一张图像，就像你使用苹果自带的 Photo Booth 应用程序一样。为了实现这个，需要一个继承自 QTCaptureLayer 的子类叫做 CaptureLayer 的类加入到应用程序中。这里，我们实现了-getCurrentImage 这个函数，当行动触发时，来返回 NSImage 对象，就包含了目前的图像。回头看 7-10，你可以看到我们设定了 QTCaptureDecompressedVidioOutput 对象的代理在-initSession 的代码中。我们现在实现它的代理方法，-captureOutput,如下所示清单 7-13.

```
- (void)captureOutput:(QTCaptureOutput *)captureOutput
didOutputVideoFrame:(CVImageBufferRef)videoFrame
withSampleBuffer:(QTSampleBuffer *)sampleBuffer fromConnection:(QTCaptureConnection *)connection
{
// Store the current frame
CVImageBufferRef imageBuffer;
CVBufferRetain(videoFrame);
// Synchronize access, as this delegate is not // called on the main thread.
@synchronized (self)
{
imageBuffer = currentImageBuffer;
currentImageBuffer = videoFrame; }
CVBufferRelease(imageBuffer); }
```

清单 7-13 实现捕获输出的回调

这个代理按照在场景后面的 QTCaptureSession 的 API，以有规律的时间间隔在不停的调用。currentImageBuffer 对象在代理中不停的更新，因为它要为用户点击拍照按钮随时做准备。当用户点击了拍照按钮时，QTCaptureLayer 的子类就会去查询目前的图像。为了捕获这个图像到层中，我们增加了-getCurrentImage 这个函数，清单如 7-14.

```
- (NSImage*)getCurrentImage; {
CVImageBufferRef imageBuffer;
```

```
@synchronized (self) {
imageBuffer = CVBufferRetain(currentImageBuffer); }
if (imageBuffer) {
// Create an NSImage
NSCIImageRep *imageRep = [NSCIImageRep imageRepWithCIImage:
[CIImage imageWithCVImageBuffer:imageBuffer]];
NSImage *image = [[[NSImage alloc] initWithSize: [imageRep size]] autorelease];
[image addRepresentation:imageRep];
CVBufferRelease(imageBuffer); return image;
}
return nil; }
```

清单 7-14 实现目前的图像捕获

就像在清单 7-13 中，captureOutput 回调函数里进入到 currentImageBuffer 对象时，在常规的基础上加入了同步。这个回调运行在自己的线程中，做一个同步块是很有必要的。如果图像的缓存是被成功的获取，我们就转化它成为 NSImage 对象，然后再返回给要调用的函数。

最后，我们增加一个行动在 Appdelegate 中，目的来触发捕获按钮被按下来的情况。这个行动会抓取 QTCaptureLayer 子类中的图像，然后用它设置 NSImageView 的图像。清单 7-15 展示了如何实现。

```
- (IBAction)grabImage:(id)sender; {
NSImage *image = [captureLayer getCurrentImage];
[imageView setImage:image]; }
```

清单 7-15 在 imageView 上设定图像

当你运行 Photo Capture 应用程序时，在左边的 NSView 的视图中会展示 QTCaptureLayer。当你点击捕获图像按钮时，右边的视图就会捕获层中目前的图像。当你在 image view 上设定图像时，视图就会展示目前帧的图像。如果你想保存图像，你可以通过在 NSImage 上调用-representationUsingType 来获得 NSData 对象。当你获得了 NSData 对象时，你可以用-writeTofile:atomically:方法写入到磁盘中。

7.3. 总结

QuickTime 的核心动画层，提供了一个强大的功能，可以来展示磁盘的影视文件和通过支持的视频捕获设备展示实时的视频。在你自己的视频应用程序中，它使这个复杂的任务变得如此的简单。

点击这里访问：DevDiv.com 移动开发论坛