Core Animation:

Simplified Animation Techniques for Mac and iPhone Development

第三部分

核心动画的层

第八章

OpenGL 层

版本 1.0

翻译时间：2012-12-17

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 8 章 OpenGL 层

使用核心动画的 OpenGL 层，CAOpenGLLayer 使你在播放电影时可以做更多的控制，这些控制包括在视频帧中使用核心图像滤镜或者组合混合视频流到同一个上下文中。

这一章展示给你，如何用 CAOpenGLLayer 来渲染一个视频通道，就像第七章 QuickTime 层的功能一样。下面，我们就来看，利用 CAOpenGLLayer，在网格层上来组合多种视频通道，就像你可以在视频墙上看到的一样。这里演示了，当你使用 CAOpenGLLayer 层时，你可以控制那些功能和效果。

8.1. 在 CAOpenGLLayer 上渲染视频

就像第七章讨论的，简单的视频播放可以用 QTMovieView 和 QTMovieLayer 来控制。然而，假如你在渲染之前想改变目前的帧，你最好使用 OpenGl。第一步，我们来看在 CAOpenGlLayer 上，如何简单的展示没有改变的帧。这里模拟了第七章在 QTMovieLayer 上已经实现的功能。

为了利用 CAOpenGLLayer，你需要子类化它，它不可以直接使用。相比于你在核心动画中需要做的东西，用 CAOpenGLLayer 层创建 OpenGL 的内容要显的简单得多。在一个 NSOpenGLView 中，你需要设置每件事，但是在 CAOpenGLLayer 中，下面的都可以免费得到。

预先配置的 OpenGL 上下文。

视频端口自动设定到 CAOpenGLLayer 帧上

预先配置的像素格式对象

设置这些代码如此简单，以至于你需要关心仅仅 2 个功能。第一个功能核对下一帧是否要被渲染，第二个函数就会根据第一个函数是否返回 YES 或者 NO，来决定要不要渲染到内容上。如果你很好的理解了这两个功能函数，你也就很好的理解了 CAOpenGLLayer 如何工作了，重要的是如何来使用它。这俩函数如表 8-1 所示。

```
- (BOOL)canDrawInCGLContext:(CGLContextObj)glContext pixelFormat:(CGLPixelFormatObj)pixelFormat
forLayerTime:(CFTimeInterval)timeInterval displayTime:(const CVTimeStamp *)timestamp;
- (void)drawInCGLContext:(CGLContextObj)glContext pixelFormat:(CGLPixelFormatObj)pixelFormat
forLayerTime:(CFTimeInterval)interval displayTime:(const CVTimeStamp *)timestamp;
```

表 8-1 CAOpenGLLayer 的代理渲染函数

只有当你设定了层的同步属性 asynchronous 为 YES 的时候，函数-canDrawInCGLContext 才会被调用。你可以在你的继承 CAOpenGLLayer 层的初始化方法 init 中，调用此方法：

```
[self setAsynchronous:YES];
```

如果你计划手动的更新内容或者根据定时器进行安排，那么你就不需要设定这些。在这种情况下，无论什么时候想要刷新内容，只需要简单的调用-setNeedsDisplay:YES 这个函数就行了。

对于我们的情况，然而，我们想要-canDrawInCGLContext 被调用，因为当电影播放时，我们需要不停的核对已经准备好的帧。为了获取这些帧，需要设定 asynchronous 属性为

YES。

只有当-canDrawInCGLContext 返回 YES 的时候，函数-drawInCGLContext 会被调用。在它被调用之后，就可以渲染你的 OpenGL 内容到需要的上下文中。

8.1.1. 层的时序

注意到-canDrawInCGLContext 和-drawInCGLContext，这两个关系时间的字段。

forLayerTime, 属于 CGTimeInterval

displayTime, 属于 CVTimeStamp

我们不关心 forLayerTime，因为我们在 CAOpenGLLayer 层上，不需要要使用它。然而，displayTime 对于我们这个练习至关重要。

根据显示的刷新率，正确的播放视频，并且同步音频可能有些微妙的关系。然而，苹果公司有了一个稳定的播放视频方式，这里他们用到了显示链接(display link)。这里有苹果公司在核心视频程序向导中定义的显示链接：

为了简单的同步视频和显示的刷新率，核心视频提供了一个特别的定时器叫做显示链接。显示链接在一个独立的高优先级的线程中运行，这个线程不会被应用程序的交互处理影响到。

过去，同步视频帧和显示的刷新率是一个问题，尤其是如果你也有音频时。很简单的你就可以想到，例如通过定时器输出一个帧时，但是这不可能考虑到用户交互，CPU 装载，窗口组合等等这些问题的时间延时。核心视频中的展示链接，基于展示的类型和延时，做了一个非常智能的评估，从而可以获得什么时候一个帧需要输出。

实质上，也就是说你必须要创建一个回调函数，它会被定期的调用，在这个回调函数中，你可以核对一个新的帧在此刻（也就是回调函数中的 inOutputTime 这个 CVTimeStamp 这个时刻点）是否可用。显示链接的回调如清单 8-2.

```
CVReturn MyDisplayLinkCallback ( CVDisplayLinkRef displayLink,
const CVTimeStamp *inNow,
const CVTimeStamp *inOutputTime, CVOptionFlags flagsIn, CVOptionFlags *flagsOut,
void *displayLinkContext);
```

清单 8-2

然而，在 CAOpenGLLayer 中安装和使用显示链接的回调函数是完全没有必要的，因为这些功能是通过-canDrawInCGLContext 和-drawInCGLContext 这两个函数实现的。你就可以核对在函数参数中 displayTime 这个时刻是否一个新的帧被提供。

因此，通过-canDrawInCGLContext 这个函数，来渲染一个视频帧到 CAOpenGLLayer 上的步骤：

1.核对视频是否正在播放；如果没有返回 no。

2.如果视频是在播放，核对是否视频的上下文已经被设定。如果没有，通过调用-setupVisualContext 这个方法设定它。

3.检查是否一个新的帧是准备好了。

4.如果可以，复制目前的图像到图像缓存中。

5.如果上述每个都是成功的，返回 YES

6.如果-canDrawInCGLContext 返回了 YES，-drawInCGLContext 会被调用。在这里面就用绘制 OpenGL 的线条到视频目前的纹理图像上。

-canDrawInCGLContext 在清单 8-3 中的实现。

```
- (BOOL)canDrawInCGLContext:(CGLContextObj)glContext pixelFormat:(CGLPixelFormatObj)pixelFormat
forLayerTime:(CFTimeInterval)timeInterval displayTime:(const CVTimeStamp *)timeStamp
{
if( !qtVisualContext ) {
```

```
// If the visual context for the QTMovie has not been set up
// we initialize it now
[self setupVisualContext:glContext withPixelFormat:pixelFormat];
}
// Check to see if a new frame (image) is ready to be drawn at // the current time by passing NULL as the second
param if(QTVisualContextIsNewImageAvailable(qtVisualContext,NULL))
{
// Release the previous frame
CVOpenGLTextureRelease(currentFrame);
// Copy the current frame into the image buffer
QTVisualContextCopyImageForTime(qtVisualContext, NULL,
NULL, &currentFrame);
// Returns the texture coordinates for the
// part of the image that should be displayed CVOpenGLTextureGetCleanTexCoords(currentFrame,
lowerLeft, lowerRight, upperRight, upperLeft);
return YES; }
return NO; }
```

清单 8-3 -canDrawInCGLContext 代理的实现

在你要在一个 OpenGL 的上下文中画任何东西，你必须先给 QuickTime 视频安装可视的上下文。-setupVisualContext 的代码如清单 8-4.

```
- (void)setupVisualContext:(CGLContextObj)glContext withPixelFormat:(CGLPixelFormatObj)pixelFormat;
{
OSStatus error;
NSDictionary *attributes = nil;
attributes = [NSDictionary dictionaryWithObjectsAndKeys:
[NSDictionary dictionaryWithObjectsAndKeys:
[NSNumber numberWithFloat:[self frame].size.width], kQTVisualContextTargetDimensions_WidthKey,
[NSNumber numberWithFloat:[self frame].size.height], kQTVisualContextTargetDimensions_HeightKey, nil],
kQTVisualContextTargetDimensionsKey, [NSDictionary dictionaryWithObjectsAndKeys:
[NSNumber numberWithFloat:[self frame].size.width], kCVPixelBufferWidthKey,
[NSNumber numberWithFloat:[self frame].size.height], kCVPixelBufferHeightKey, nil],
kQTVisualContextPixelBufferAttributesKey, nil];
// Create the QuickTime visual context
error = QTOpenGLTextureContextCreate(NULL, glContext,
pixelFormat, (CFDictionaryRef)attributes, &qtVisualContext);
// Associate it with the movie
SetMovieVisualContext([movie quickTimeMovie],qtVisualContext); }
```

清单 8-4 QuickTime 的可视上下文的实现

在-setupVisualContext 中，视频是联系 CAOpenGLLayer 中的 OpenGL 的上下文。这样，当-drawInCGLContext 是被调用时，每件事都被设定好了，就可以调用 OpenGL 的 API 了，就像清单 8-5 所示。

```
- (void)drawInCGLContext:(CGLContextObj)glContext pixelFormat:(CGLPixelFormatObj)pixelFormat
forLayerTime:(CFTimeInterval)interval displayTime:(const CVTimeStamp *)timeStamp
{
NSRect bounds = NSRectFromCGRect([self bounds]);
GLfloat minX, minY, maxX, maxY;
minX = NSMinX(bounds); minY = NSMinY(bounds); maxX = NSMaxX(bounds); maxY = NSMaxY(bounds);
glMatrixMode(GL_MODELVIEW); glLoadIdentity(); glMatrixMode(GL_PROJECTION); glLoadIdentity();
glOrtho( minX, maxX, minY, maxY, -1.0, 1.0); glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
CGRect imageRect = [self frame];
```

```
// Enable target for the current frame glEnable(CVOpenGLTextureGetTarget(currentFrame));
// Bind to the current frame
// This tells OpenGL which texture we want
// to draw so when we make the glTexCord and
// glVertex calls, the current frame gets drawn
// to the context glBindTexture(CVOpenGLTextureGetTarget(currentFrame),
CVOpenGLTextureGetName(currentFrame)); glMatrixMode(GL_TEXTURE);
glLoadIdentity(); glColor4f(1.0, 1.0, 1.0, 1.0); glBegin(GL_QUADS);
// Draw the quads
glTexCoord2f(upperLeft[0], upperLeft[1]); glVertex2f (imageRect.origin.x,
imageRect.origin.y + imageRect.size.height); glTexCoord2f(upperRight[0], upperRight[1]);
glVertex2f (imageRect.origin.x + imageRect.size.width, imageRect.origin.y + imageRect.size.height);
glTexCoord2f(lowerRight[0], lowerRight[1]);
glVertex2f (imageRect.origin.x + imageRect.size.width,
imageRect.origin.y); glTexCoord2f(lowerLeft[0], lowerLeft[1]);
glVertex2f (imageRect.origin.x, imageRect.origin.y);
glEnd();
// This CAOpenGLLayer is responsible to flush // the OpenGL context so we call super
[super drawInCGLContext:glContext
pixelFormat:pixelFormat forLayerTime:interval
displayTime:timeStamp]; // Task the context
QTVisualContextTask(qtVisualContext); }
```

清单 8-5 drawInCGLContext 的实现

要理解上面的代码，你可能需要理解 OpenGL，这就超过了本书讲述的范围。然而，看清单 8-5 中这两行代码。

```
glEnable(CVOpenGLTextureGetTarget(currentFrame));
glBindTexture(CVOpenGLTextureGetTarget(currentFrame),
CVOpenGLTextureGetName(currentFrame));
```

这两行告诉了 OpenGL 能够绑定绘图的纹理到目前的帧上(CVImageBufferRef)，它是通过调用-canDrawInContext 获得的。总之，这是告诉 OpenGL 要绘制什么东西。

8.2. 渲染多个视频渠道

这一章最后的目标是来演示如何在一个 CAOpenGLLayer 层中，渲染多个 QuickTime 视频流。刚才提到了，使用 OpenGL 代替 QTMovieLayer 的原因归结于性能。当你使用多个 QTMovieLayers 导入和播放多个 QTMovies 时，性能会迅速的下降。为了提高性能，我们替代去获得播放视频的每个帧，而是组合他们一起到同一个 OpenGL 上下文中。

为了完成这个目标，我们的做法不同于先前段落中使用 OpenGL 来渲染一个单一的 QuickTime 视频。我们为每个 QuickTime 视频都创建了一个图像缓冲区，实时的来核对是否下一帧准备在-canDrawInCGLContext 中被调用。我们也要通过在每一个动画初始化时为其设置一个可绘矩形的方法，来在网格中显示动画。

我们可以复制粘贴上面段落中我们写的代码，但是这样的话代码会变得笨重和冗余。因此，我们使用面向对象的方法，这里来创建一个继承自 OpenGL 层的对象(叫做 OpenGLVidGridLayer)，一个 VideoChannel 对象代表一个视频流，然后一个 VideoChannelController 对象提供了一个播放和渲染的接口。下面是每个对象要做的事情：

OpenGLVidGridLayer

这个对象用来初始化层以便同步运行，同时设置框架大小，设置背景颜色为黑色，设置

传进来的视频路径的数组，初始化 VideoChannel 和 VideoChannelController 对象，调用-canDrawInCGContext 和-drawInCGLContext 方法，并且作为一个代理，来开始播放我们分配给的 VideoChannel 对象的视频。

VideoChannel

这个代表着每个方格中的视频。它存储了一些区域，这些区域将会在父区域中被使用来组合视频，并且要核对它分配的视频是否准备绘制下一个帧，这里还要使用初始化指定的高和宽来初始化可视区域，使用 OpenGL 的调用来绘制分配的视频，并且为视频的播放提供一个代理方法。

VideoChannelController

这个对象包含了 VideoChannel 对象的数组，这些对象都有一个初始化的视频和区域，然后用来在方格中渲染。它提供了一个代理函数，来指导所有的 VideoChannel 来播放和停止视频，并且还提供了一个代理函数来看是否所有的 VideoChannel 都是准备被绘制到下一个帧上，还提供了一个代理函数来告知所有的 VideoChannel 来设置它们的可视上下文，调用在 VideoChannel 中的 OpenGL 绘图的初始化代码，并且还提供了一个代理函数告知所有的 VideoChannel 来渲染它们的视频到各自的区域。

面向对象的方法是一把双刃剑。尽管它可以使我们用更清晰的方式组织东西，但是必须要用对象去思想考虑每件事，因此学习这个描述方法，以便于你知道工程中的那些代码代表什么在运行。VideoChannelController 对象提供了每个 VideoChannels 对象的控制，所以看这个对象如何工作，是你理解代码的一个最佳的选择。OpenGLVideGridLayer 提供了一些初始化的代码，并且提供了一个功能，决定我们是否要在当前的时间进行绘制。

表 8-1 描述了应用程序的例子，OpenGL VidGrid。窗口中所有展示的区域都是继承自 CAOpenGLLayer 这个类的一个子类，叫做 OpenGLVidGridLayer。屏幕上展示的每个独立的分割的视频都是由 VideoChannel 这个类的代码来呈现的。

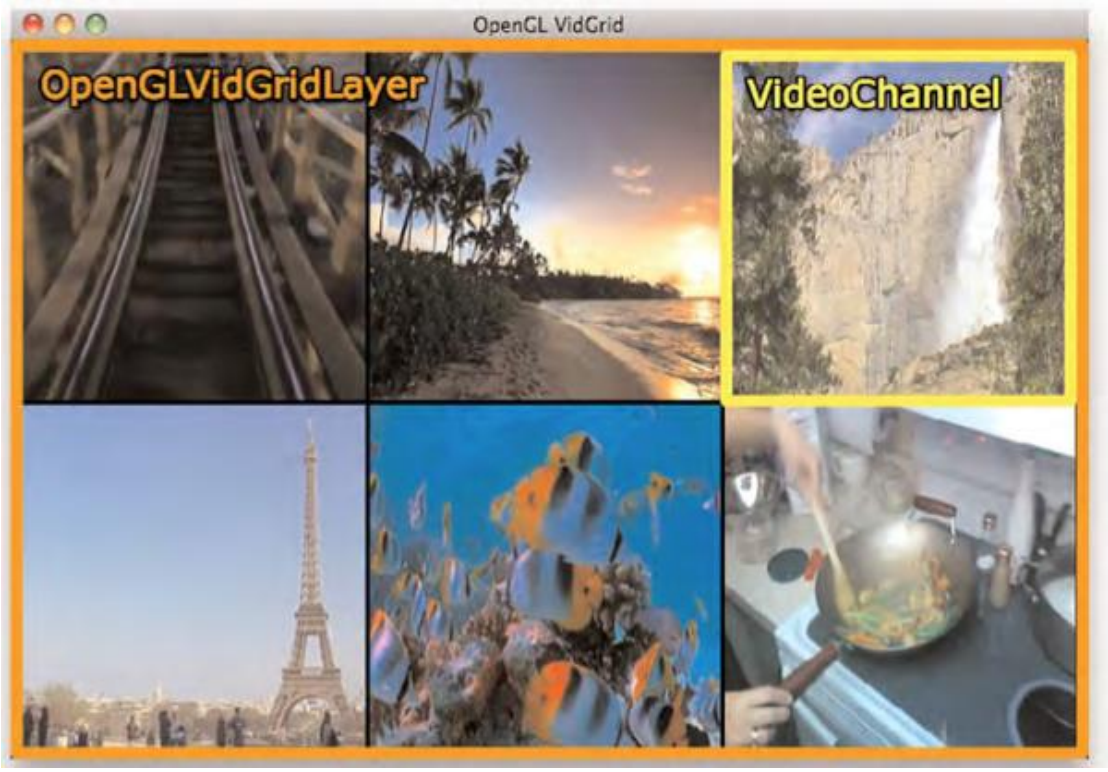

图 8-1 可视的对象

实现继承子 CAOpenGLLayer 的子类 OpenGLVidGridLayer

继承自 CAOpenGLLayer 的类，OpenGLVidGridLayer 提供了绘制所有视频通道到同一个 OpenGL 上下文中的入口。初始化代码接收了一个 QTMovie 对象的列表，同时为这些 QTMovie 对象计算展示的区域。它为每个 QTMovies 对象创建了一个 VideoChannel 对象，并且增加他们到一个数组中。VideoChannelController 对象会被初始化，然后这个 VideoChannels 数组对象就分配给它。实现的代码如清单 8-6.

```
- (void)initVideoChannels; {
int i = 0;
if( videoChannels != nil )
[videoChannels release], videoChannels = nil; videoChannels = [[NSMutableArray alloc] init];
// Create a grid kColCount across
float vidWidth = [self frame].size.width / kColCount; vidWidth = vidWidth - kMargin;
float vidHeight = [self frame].size.height; if( [videoPaths count] > kColCount )
vidHeight = [self frame].size.height / (((float)[videoPaths count])/kColCount);
vidHeight = vidHeight - kMargin; int row = 0;
int col = 0;
for (i = 0; i < [videoPaths count]; ++i) {
NSString *path = [videoPaths objectAtIndex:i]; QTMovie *m = [QTMovie movieWithFile:path error:nil];
// Mute each video
[m setMuted:YES];
// Force the movie to loop
[m setAttribute:[NSNumber numberWithBool:YES] forKey:QTMovieLoopsAttribute];
float x = col*vidWidth; float y = row*vidHeight;
if( i!= 0 && (i+1) % (int)kColCount == 0 ) {
row++; }
CGRect currentRect = CGRectMake(x+kMargin, y+kMargin,
vidWidth-kMargin, vidHeight-kMargin);
// Instantiate a video channel for each QTMovie // object we've created
VideoChannel *channel =
[[[VideoChannel alloc] initWithMovie:m usingDisplayRect:currentRect] autorelease];
[videoChannels addObject:channel];
col++;
if( col > kColCount-1 )
col = 0;
}
if( videoController != nil ) {
[videoController release]; }
videoController = [[VideoChannelController alloc] init];
[videoController setVideoChannels:videoChannels]; }
```

清单 8 - 6

前面的章节我们看到了如何去渲染一个视频到 OpenGL 的上下文中。我们利用这些代码，放置它到封装了这个功能的类里面。VideoChannelController 对象掌控了 VideoChannel 对象。它传递 OpenGL 的上下文给每一个 VideoChannel 对象，那将告知他们去绘制内容到 OpenGL 上下文中。

在这个工程的 AppDelegate 中，我们初始化了 CAOpenGLLayer 层的子类 OpenGLVidGridLayer，并且传递了一个视频的路径地址数组给这个层，如清单 8-7.

```
-(void)awakeFromNib; {
NSString *path1 = @"/System/Library/Compositions/Eiffel Tower.mov";
NSString *path2 = @"/System/Library/Compositions/Fish.mov";
NSString *path3 = [[NSBundle mainBundle] pathForResource:@"stirfry"
ofType:@"mp4"]; NSString *path4 = @"/System/Library/Compositions/Rollercoaster.mov";
NSString *path5 = @"/System/Library/Compositions/Sunset.mov";
```

```
NSString *path6 = @"/System/Library/Compositions/Yosemite.mov";
NSArray *paths = [NSArray arrayWithObjects:path1, path2,
path3, path4, path5, path6, nil];
gridLayer = [[OpenGLVidGridLayer alloc] initWithVideoPaths:paths
usingContentFrame:
NSRectToCGRect([[window contentView] bounds])];
[[window contentView] setWantsLayer:YES];
[[[window contentView] layer] addSublayer:gridLayer]; [gridLayer playMovies];
}
```

清单 8-7

当 OpenGLVidGridLayer 被创建后，我们传递一个视频地址的数组给它和每个视频要渲染的区域。

当窗口的内容视图使用了[[window contentView] setWantLayer:YES]，增加了 OpenGLVidGridLayer 作为子层到根层上时，然后调用它的-playMovies 方法，这个方法也就是调用了[videoController togglePlaybackAll]。这个函数就会迭代在 VideoChannelControlller 中所有的 VideoChannel 对象和指导他们播放每个视频在他们设计好的容器中。

清单 8-8 展示了如何在 OpenGLVideGridLayer 中重载-canDrawInCGLContext 和-drawInCGLContext 的方法。

```
- (BOOL)canDrawInCGLContext:(CGLContextObj)glContext pixelFormat:(CGLPixelFormatObj)pixelFormat
forLayerTime:(CFTimeInterval)timeInterval displayTime:(const CVTimeStamp *)timeStamp
{
if( ![videoController isPlaying] )
return NO;
[videoController setVisualContext:glContext
withPixelFormat:pixelFormat];
BOOL ready = [videoController channelsReadyToDraw:(CVTimeStamp*)timeStamp];
return ready; }
- (void)drawInCGLContext:(CGLContextObj)glContext pixelFormat:(CGLPixelFormatObj)pixelFormat
forLayerTime:(CFTimeInterval)interval displayTime:(const CVTimeStamp *)timeStamp
{
NSRect bounds = NSRectFromCGRect([self bounds]); [videoController drawAllInRect:bounds];
// This forces OpenGL to flush the context
[super drawInCGLContext:glContext pixelFormat:pixelFormat
forLayerTime:interval displayTime:timeStamp];
[videoController taskAll]; }
```

清单 8-8 绘图函数的实现

这个函数看起来非常的简单，因为我们卸载了主要的工作，给了 VideoChannelController 这个对象。VideoChannelController 会指导所有的 VideoChannel 对象来做这些工作，然后调用-canDrawInCGLContext 来核对视频是否正在运行。如果没有，我们不需要渲染，这样返回 NO。在这种情况下，-drawInCGLContext 就不会被调用。当视频正在运行时，然后通过在 VideoChannelController 调用-channelIsReadyToDraw，来核对目前每个 VideoChannel 中的要被绘制的图形缓冲是否准备好，然后渲染 QuickTime 视频的可视区域就会被安装。你可以看到清单 8-9 如何实现的。

```
- (BOOL)channelsReadyToDraw:(CVTimeStamp*)timeStamp; {
BOOL stillOk = NO;
int i = 0;
for (i=0; i<[videoChannels count]; ++i) {
VideoChannel *currentChannel =
```

```
[videoChannels objectAtIndex:i];
if( [currentChannel readyToDrawNextFrame:timeStamp] ) {
stillOk = YES; }
if( !stillOk ) return NO;
}
return stillOk; }
```

清单 8-9 实现 ChannelsReadyToDraw 的功能

代码通过迭代所有的 VideoChannel 对象，然后核对每个对象看是否准备绘制。如果它们中任一一个失败，就在-canDrawInCGLContext 中返回 NO。清单 8-10 展示了 VideoChannel 中-readyToDrawNextFrame 的实现。

```
- (BOOL)readyToDrawNextFrame:(CVTimeStamp*)timeStamp; {
if(QTVisualContextIsNewImageAvailable(qtVisualContext,NULL)) {
CVOpenGLTextureRelease(currentFrameImageBuffer);
QTVisualContextCopyImageForTime( qtVisualContext,
NULL,
NULL, &currentFrameImageBuffer);
CVOpenGLTextureGetCleanTexCoords( currentFrameImageBuffer,
lowerLeft, lowerRight, upperRight, upperLeft);
return YES; }
return NO; }
```

清单 8-10 实现 readyToDrawNextFrame 函数

如果你仔细观察，你会发现这个代码和清单 8-3 渲染一个视频通道的代码非常的相似。这个函数是来核对，对于可视的上下文中在指定的时间 timeStamp 中，是否有新的图像可用。如果有可用的，先前图像的缓冲就会被释放，当前的图像就会拷贝到图像缓冲中，纹理坐标被重设，然后就会返回 YES。否则就返回 NO。

这个调用栈最后会返回到 OpenGLVidGridLayer 中的-canDrawInCGLContext 函数中。如果所有的通道都是被绘制了，那么就会返回 YES，然后调用-drawInCGLContext。

当-drawInCGLContext 是被调用时，我们就会传递窗口的主区域给视频的控制器，然后控制器调用[videoController drawAllInRect:bounds]这个方法，其中 bounds 就是区域，如清单 8-11.

```
- (void)drawAllInRect:(NSRect)rect; {
GLfloat minX, minY, maxX, maxY;
minX = NSMinX(rect); minY = NSMinY(rect); maxX = NSMaxX(rect); maxY = NSMaxY(rect);
glMatrixMode(GL_MODELVIEW); glLoadIdentity(); glMatrixMode(GL_PROJECTION); glLoadIdentity();
glOrtho(minX, maxX, minY, maxY, -1.0, 1.0);
glClearColor(0.0, 0.0, 0.0, 0.0); glClear(GL_COLOR_BUFFER_BIT);
int i = 0;
for (i=0; i<[videoChannels count]; ++i) {
VideoChannel *currentChannel = [videoChannels objectAtIndex:i];
[currentChannel drawChannel]; }
}
```

清单 8-11 实现 drawAllInRect

如果你对比清单 8-11 和清单 8-5，你会清楚的看到非常的相似。OpenGL 调用来绘制了线条的属性。当这些设定调用是被运行时，我们迭代了所有的 VideoChannel 对象，并且告诉它们每一个来绘制它们目前的帧，替代了我们在 8-5 中所做的绘制代码。记住要绘制的每个 VidwoChannel 的区域都已经在初始化的时候设定好了。

当我们迭代所有的 VideoChannel 对象时，我们调用-drawChannel，如清单 8-12

```
- (void)drawChannel; {
[self drawImage]; }
- (void)drawImage; {
[self drawImage:mainDisplayRect
withOpacity:opacity];
withOpacity:(GLfloat)op;
}
- (void)drawImage:(CGRect)imageRect {
glEnable( CVOpenGLTextureGetTarget(
currentFrameImageBuffer));
glBindTexture( CVOpenGLTextureGetTarget(
currentFrameImageBuffer), CVOpenGLTextureGetName(
currentFrameImageBuffer));
glMatrixMode(GL_TEXTURE); glLoadIdentity(); glColor4f(1.0, 1.0, 1.0, op); glBegin(GL_QUADS);
glTexCoord2f(upperLeft[0], upperLeft[1]); glVertex2f (imageRect.origin.x,
imageRect.origin.y + imageRect.size.height);
glTexCoord2f(upperRight[0], upperRight[1]);
glVertex2f (imageRect.origin.x + imageRect.size.width,
imageRect.origin.y + imageRect.size.height);
glTexCoord2f(lowerRight[0], lowerRight[1]);
glVertex2f (imageRect.origin.x + imageRect.size.width,
imageRect.origin.y); glTexCoord2f(lowerLeft[0], lowerLeft[1]);
glVertex2f (imageRect.origin.x, imageRect.origin.y);
glEnd(); }
```

清单 8-12 实现 VideoChannel 的 drawChannel

其中-drawChannel 方法调用另一个方法的默认实现-drawImage，这个函数也调用同样名字的函数但是多了两个参数：

displayRect，这个是在主窗口中 mainDisplayRect 传递的

opacity，这个一般传递一个 1.0，不透明。

这些看起来都是没有必要的重复层级调用，但是它可以让你方便的改变调用，这样给与一个不同的透明度，假如你想要渲染视频带着一些透明度而不是全部不透明。

我们已经展示了渲染多个视频通道的核心代码，这里你需要运行这个实例工程，OpenGLVidGrid 来看下如何工作。

8.3. 总结

核心动画为苹果的技术提供了一个如此强大的抽象功能。你不比成为 OpenGL 领域的专家，就可以利用 OpenGL 的强大功能来帮助开发者。就像本章看到的，渲染一个 OpenGL 上下文是很多 Cocoa 程序员都可以做到的。同时也帮助你介绍了一些 OpenGL 的概念，便于以后你进行更深入的学习。不管你在编程方面多么努力，CAOpenGLLayer 都会在你的应用程序中，给你需要利用的 OpenGL 功能的一些东西。

点击这里访问：DevDiv.com 移动开发论坛