

硕 士 研 究 生 读 书 报 告



题目 OpenGL在Cocos2dx中的应用

作者姓名 陈伟哲

作者学号 21651027

指导教师 李启雷

学科专业 移动互联网与游戏开发

所在学院 软件学院

提交日期 二○一七年四月

Application of Cocos2dx in OpenGL

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Qilei Li

By

Weizhe Chen

Zhejiang University, P.R. China

2017

摘要

本文重点探讨了OpenGL ES着色程序。OpenGL ES渲染管线包含一个可编程的顶点着色阶段和一个可编程的片段着色器阶段，其他的阶段则通过一些固定的方法做有限的控制。每个可编程阶段使用一个着色器作为一个处理单元，分别对顶点和片段进行自定义的处理。顶点着色器程序被上传至GPU，被OpenGL ES作用在渲染管线上。

本文将描述顶点数组、顶点属性的绑定，着色器程序的编写方法，着色器的内置方法、变量，以及怎样在Cocos2dx中使用这一切。

**关键词**：OpenGL，Cocos2dx，着色程序

Abstract

This paper focuses on the OpenGL ES rendering program. The OpenGL ES rendering pipeline contains a programmable vertex rendering stage and a programmable fragment rendering stage, while the other stages are limited by some fixed methods. In each programmable phase, a rendering is used as a processing unit, respectively. Vertex rendering program was uploaded to GPU, OpenGL ES role in the rendering pipeline.

In this article, we will describe the vertex array, the binding of vertex attributes, the preparation of the rendering program, the built-in method of the rendering, the variables, and how to use it in Cocos2dx.

**Keywords：**OpenGL, Cocos2dx, rendering program

1 顶点和顶点数组

首先我们需要了解下OpenGL ES渲染管线的工作原理。应用程序通过OpenGL ES API将绘制命令发送至OpenGL ES渲染管线，其中一些命令定义了组成一个物体的图元的集合，而另一些命令控制这些图元在渲染管线的各个阶段将怎样被处理。OpenGL ES渲染管线实际上就是将一个3D几何绘制到2D屏幕上。

在OpenGL ES中，图元由一系列属性组成的顶点数组定义。图元的类型可以分为7种，分别是：点；未连接的线段；闭合且连接的线段；未连接的线段；连接的三角形；扇形三角形；未连接的三角形。每个顶点由多个顶点属性定义，如顶点坐标、颜色、纹理坐标等。每个顶点属性值由1～4个标量值组成，顶点属性的值通过OpenGL ES命令传入，可以被顶点着色器程序读取。

顶点属性的值由客户端应用程序通过调用OpenGL ES命令进行设置，而它们会被传输到GL服务端，被顶点着色器读取。客户端的顶点属性称为通用顶点属性，应用程序可以以两种方式定义一个顶点属性的值，它们对于顶点着色器而言没有区别。

相较于常量属性，应用程序中更常使用的方式是由顶点数组给给个顶点设置不同的值，应用程序中大部分顶点属性都是由这种方式定义的。存储顶点属性的数据通常有两种方式。一是存储所有的顶点属性在一个缓冲对象中。这种存储顶点属性的方式称为结构数组，这个结构定义了一个顶点中所有属性。二是所有顶点的每个顶点属性存储一个独立的缓冲对象，这种方式称为数组结构。在顶点着色器程序中，使用attribute限定符修饰的全局变量称为顶点属性变量。顶点属性变量的类型只能是float、vec2、vec3、vec4、mat2、mat3及mat4，每一个非矩阵变量都会占据1个存储空间，mat2、mat3、mat4分别占据2、3、4个存储空间，每个存储空间包含4个float的分量，因此，应该尽可能使用vec4类型。在客户端程序中，我们通过一个通用属性索引值表示一个顶点属性。在顶点着色器中，我们通过一个字符串名称定义属性变量。通过VertexAttribPointer命令指定了各个顶点属性的值，以及构成这些值的顶点数组在内存中的地址，这些顶点数组的数据存储在客户端内存中，最后需要将这些数据传输至GL服务端内存中，并最终将这些图元绘制出来。

**2 顶点缓冲对象**

顶点数组的数据存储在客户端内存中，它们在每一次绘制的时候被传输至GL服务端内存中。有时需要将一些频繁使用的数据存储在更高速的服务端内存中，OpenGL提供了一种方法使得客户端可以分配，初始化服务端内存，以及从服务端内存读取顶点数据进行绘制。

缓冲对象代表服务端的一块内存空间，可以用来存储顶点数组、顶点索引数组等。在客户端程序中缓冲对象的名称用一个整数表示，0为GL的保留值。顶点数组可以以与客户端相同的格式和方式存储在服务端缓冲对象中，这样就使绘制命令及其他GL操作不会因为缓冲对象的使用而发生改变。

顶点索引数组也可以与客户端相同的格式和方式存储在服务端缓冲对象中。默认情况下，0被绑定ELEMENT\_ARRAY\_BUFFER，它表示DrawElements命令应该从客户端通过indices参数复制顶点索引数组。

对于不同类型3D物体的绘制，我们通常会使用不同的顶点缓冲对象、顶点索引缓冲对象及设置其他特定的GL状态。这就需要在应用程序中频繁切换顶点属性的客户端状态参数，而这些参数往往要从客户端复制，并上传至服务端，如VertexAttribPointer设置的顶点属性状态。OpenGL ES扩展提供了一种方法，可以缓存顶点属性状态至GL服务端，这样就可以实现快速的顶点属性状态切换。

**3 Cocos2d-x着色器子系统**

着色器是可编程管线最灵活和强大的部分，一方面，应用程序可以完全控制渲染管线的每个顶点和图元，能作出一些更精细的控制；另一方面，可以将一些图形处理算法放到高度并行的，同时对一些向量、矩阵运算等进行优化过的GPU中去执行，这些都大大提升了图形渲染的质量及能力。

然而这种灵活性也带来了使用上的复杂度的上升。如绘制一个最简单的三角形也需要编写顶点、片段着色器，以及编译和链接它们，绑定和设置各种顶点属性变量，全局属性变量，还需要计算每个顶点的坐标变换等。

编写一个着色器程序往往是简单的，我们甚至可以从互联网上找到很多很酷的算法来计算片段的颜色。然而，将着色器程序与应用程序中的元素结合起来，以及使用适当的流程来设置和修改参数，却是使用OpenGL ES最枯燥的部分。因为CPU和GPU处于不同的内存空间，它们不能共同操作一个内存地址。我们只有理解任何数据的封装，传输过程及读取方式，才能真正灵活地使用强大的可编程管线。

不同的游戏引擎都有自己独特的着色器使用流程，这些流程往往是根据游戏引擎的特点来设计的。例如在Unity3D中，着色器程序首先和纹理及其他一些参数构成一个材质，然后材质用来构成一个渲染组件，最后将这个渲染组件添加到一个元素中去。

**4 顶点着色器与片段着色器**

顶点着色器提供了一种方法，使得可以针对渲染管线的每个顶点进行操作，顶点着色器接收一些输入参数，进行着色器的计算后输出另一些参数以供渲染管线的后续阶段使用。每个顶点之间是并行执行的，每个顶点从顶点数组获取对应的顶点属性值，相互之间没有任何依赖。

片段着色器是OpenGL ES可编程渲染管线最强大的地方，3D应用程序中的很多特效都是在这里完成的，它不仅能精细控制每个片段的颜色值，还可以使用多重纹理来计算一些融合效果，我们可以使用多张不同意义的纹理，例如法线贴图，来在片段着色器中按照一定的算法进行处理。利用GPU的并行特征可以在这里实现一些更复杂的图形算法。每个片段之间也是并行处理的，所以不能在片段之间传递和保存状态信息，GPU每秒处理的片段数量的能力很大程度上决定了该GPU的并行处理能力。

**5 着色器编辑工具**

由于着色器是在处于OpenGL ES渲染管线的环境中执行的，它还需要和其他GL管线中的参数一起工作，并且最后需要在GPU中执行，所以着色器的编写显得有些复杂。

有很多工具可以用来编辑着色器程序，它们提供着色器程序的编辑、编译和链接，开发者可以及时发现一些错误信息。其中一些工具还提供一些功能，如可以设置纹理、着色器的参数、GL绘制的参数、投影矩阵等，使其可以可视化预览着色器的最终效果，大大提升了开发者的工作效率。

**6 示例**

最后以一个简单的示例来展示怎样编写着色器程序，以及怎样在Cocos2d-x中使用自定义的着色器程序。主要展示两个方面：一方面是在不继承Sprite的情况下，怎样让Sprite等元素支持ETC压缩纹理，这对于Android平台的开发者至关重要；另一方面是了解怎样动态设置顶点属性或者全局属性的值。

使用ETC压缩纹理，即对纹理生成一个不含Alpha通道的ETC压缩纹理，以及一个仅包含Alpha通道的ETC压缩纹理。注意：这个包含Alpha通道的纹理实际上是一个3个分量RGB都为Alpha值的纹理，着色器可以通过任意一个RGB分量来获取原纹理的Alpha通道值，所以两张纹理的大小几乎是一样的，这样，一个ETC纹理将占用两倍的存储空间和内存。

首先创建一个名为ETCShader的Cocos2d-x项目，然后将使用Mali Textrue Compression Tool 生成的HelloWord.pkm和HelloWorld\_alpha.pkm文件复制至其Resources目录，并将它们添加至工程中。

接下来在Resources目录下新建一个顶点着色器和一个片段着色器，命令分别为etc\_shader.vert和etc\_shader.frag。因为使用ETC压缩纹理并不需要对顶点着色器进行修改，所以我们使用Sprite默认使用的顶点着色器程序。值得注意的是，Sprite的顶点着色器并没有使用变换矩阵CC\_MVPMatrix，而是使用了投影矩阵CC\_PMatrix。这是因为QuadCommand要进行批绘制，需要将具有不同模型视图变换矩阵的Sprite合并在一起绘制，QuadCommannd在应用程序中提前将其转换到世界坐标系下，所以每个顶点只需要执行一个投影变换即可。

所以，我们对Sprite或者其他使用QuadCommand进行绘制的元素使用自定义的着色器程序时，其顶点着色器中输入的顶点坐标并不是本地坐标系啊的坐标值，而是世界坐标系下的坐标值。当然，如果开发者一定需要取本地坐标值，也可以通过乘以CC\_MVMartix的逆矩阵来还原本地坐标值。能这么处理的原因是，对于使用自定义着色器的元素，QuadCommand不会对其执行批绘制，因此，CCMVMatrix存储的是其元素对应的模型视图矩阵。在片段着色器中，需要新增一个纹理采样器将Alpha通道融合进来。我们增加了一个名为“u\_texturel”的纹理采样器变量。

将片段着色器的实现很简单，它仅仅是从新增的保存Alpha通道值的纹理中取出Alpha值并将添加到原始纹理的Alpha通道。由于HelloWorld\_alpha.pkm纹理的R、G、B3个分量均保存着Alpha值，所以我们可以从任意一个通道取值作为Alpha值。接下来需要让Sprite运用我们自定义的着色器程序。我们只需要新建一个GLProgramState实例，它包含自定义的着色器程序，然后将其赋值给Sprite对象即可。

注意：虽然这里GLProgramState并没有加入GLProgramStateCache进行管理，但是我们并不需要管理其内存释放，create方法返回一个autorelease对象，其引用计数仅被sprtie对象持有，sprite会在其被移除或者替换GLProgramState对象时释放其引用计数来释放该对象。最后还需要做一件事情，那就是将带有Alpha通道的纹理传入着色器程序，这可以通过GLProgramState很简单地实现。SetUniformTexture将属性信息保存在一个名为UniformValue的变量中，然后在每次绘制的时候使用纹理，并设置u\_texturel变量的值为textureAplha纹理。

Cocos2d-x 3.1大大简化了多重纹理的使用，并且很容易对已有元素使用自定义的着色器。着色器子系统复杂度的降低使得更多的开发者可以在应用程序中使用自定义的着色器程序来实现更酷的视觉表现。由于应用程序中很多精灵可能都使用ETC压缩纹理，所以可以通过一个统一的入口来创建Sprite等实例，这样就可以很简单地处理整个应用程序的ETC压缩纹理的使用。

**7 着色器程序最佳实践**

（1）相对于其他一些OpenGL ES状态切换等操作，创建一个着色器，编译链接是很耗时的，应用在初始化的时候创建并缓存起来，每个自定义的着色器应用始终使用一个GLProgram实例。对于不是在外部修改属性变量的情况，自定义元素的所以实例应该共享一个GLProgramState实例。

（2）不要在Release模式下检查着色器程序的状态。OpenGL ES命令是在GPU中执行的，虽然它提供大部分命令的状态及错误信息，应用程序通常不会在运行时读取这些信息，因为这需要从GPU传回应用程序，但是我们应该在Debug模式下读取这些信息来方便调试。

（3）如果一个变量对一次绘制始终是相同的，那么不要在着色器程序中去计算它的值，因为着色器程序会被执行多次，尤其是片段着色器。这些变量应该设为全局属性或者常量顶点属性在CPU中计算好，然后直接传入渲染管线。

（4）在着色器程序中尽量使用常量来索引数组，因为在着色器程序中使用临时变量来索引数组效率是很低的。尤其是在顶点着色器中，应该避免较大的数组迭代操作，因为片段着色器将会被执行上万次，这些代价是非常高的。如果不是变更十分频繁的，可以考虑将其放到CPU中计算，使用一定形式的缓存，然后将值传递给着色器程序。因为着色器程序是并行的处理单元，它无法缓存任何数据，每一帧、每一个单元都得重新计算。

（5）尽可能地使用多重纹理。多重纹理不仅用于实现一些高级的图形效果，它还可以提升游戏性能。多个纹理被传入同一个管道，然后字啊片段着色器中被融合，可以有效地减少绘制次数，并且可以高度利用GPU的并行性特性，同时还避免了重复的顶点数组的复制。

**8 总结**

OpenGL ES着色器是游戏及图形应用程序开发中的高级内容，它主要用于增强游戏画面的表现，通过直接和渲染管线进行交互，开发者可以很容易地实现一些复杂的图形效果。在OpenGL ES中，着色器并不是一个独立的部分，它和很多其他GL命令一起工作，并且关联性极强。

然后详细描述了Cocos2d-x种着色器子系统对于着色器使用的封装，它的使用方法及流程。Cocos2d-x 3.1提供的新的着色器子系统在架构上更加灵活，使得应用程序可以为系统提供的元素指定自定义的着色器程序，而不需要继承该元素。它也大大简化了着色器程序的实现，我们甚至只需要编写着色器程序并给属性指定值即可。其他大部分的编译、链接、绑定等操作都被隐藏起来。

最后讲述了顶点着色器和片段着色器程序。在顶点着色器中最重要的是顶点坐标变换，详细描述了相关的变换矩阵，并解释了Cocos2d-x怎样构建和传递这些变换矩阵到顶点着色器。对于片段着色器，最重要的则是通过顶点着色器传输易变变量及使用纹理采样。

参考文献

[1]我所理解的Cocos2d-x．秦春林[M]．北京：电子工业出版社，2014．

[2]OpenGL编程指南.王锐 等译．北京：机械工业出版社，2016．