

硕 士 研 究 生 读 书 报 告



题目 OpenGL ES图形标准的研究与讨论

作者姓名 王晓瑶

作者学号 21651026

指导教师 李启雷

学科专业 软件功能

所在学院 软件学院

提交日期 二○一七年一月三

Research and Discussion of OpenGL ES Graphics Standard

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

XiaoYao Wang

Zhejiang University, P.R. China

2017

# 摘要

随着手机、PDA、掌上游戏机、便捷式GPS等移动设备的普及，3D图形绘制技术也逐渐渗透到各种移动设备中，移动设备成为人们进行信息获取、信息处理和个人娱乐的重要途径。因此，人们迫切希望能在移动设备上进行真实感的3D图形显示，如3D图形、3D游戏以及3D地图等。

由多家国际知名行业领导者创立的Khronos Group于2001年制定了嵌入式3D图形标准OpenGL ES，该标准定义了软件和图形加速硬件间灵活强大的底层交互接口API，为硬件厂商的图形加速器提供了跨平台统一软件接口。随着标准的指定，移动图形硬件加速和渲染技术也得到一定的发展。

本文主要对OpenGL ES进行了简单的介绍，然后对OpenGL里的一些基础知识进行了说明，接着对OpenGL ES 3.0相对于前面版本的更新内容进行了介绍，最后以一个简单的例子说明了OpenGL ES 3.0的大体程序框架，并对整篇读书报告进行了总结和展望。

**关键词**：OpenGL ES，图形标准，移动设备

# Abstract

With the popularity of mobile phones, PDA, handheld game consoles, portable GPS and other mobile devices, 3D graphics rendering technology has gradually penetrated into a variety of mobile devices. Mobile devices have become an important way for people to obtain information, information processing and personal entertainment. Therefore, people are eager to be able to carry out realistic 3D graphics display on mobile devices, such as 3D graphics, 3D games and 3D maps.

Khronos Group which was founded by a number of internationally renowned industry leaders developed an embedded 3D graphics standard OpenGL ES in 2001. The standard defines a flexible and powerful interface between software and graphics acceleration hardware. It provides a cross platform unified software interface for hardware vendors. With the designation of the standard, the mobile graphics hardware acceleration and rendering technology has also been a certain development.

This paper makes a brief introduction to the OpenGL ES, and then describes some basic knowledge of OpenGL, then introduces the OpenGL ES 3.0 compared with the previous version of the update.At last, a simple example is given to illustrate the general framework of OpenGL ES 3.0 .Also,a summary and outlook of the whole reading report are given.

**Keywords:** OpenGL ES, graphic standard, mobile device

# 1 OpenGL ES简介

OpenGL ES(OpenGL for Embedded Systems)是一种高级3D图形应用程序编程接口，是专门针对手机、PDA以及游戏主机等嵌入式设备而设计的，主要包括OpenGL ES 1.0、OpenGL ES 1.1、OpenGL ES 2.0和OpenGL ES 3.0。OpenGL ES 1.0和1.1规范采用固定功能管线，分别从OpenGL 1.3和1.5规范衍生而来。OpenGL ES 2.0规范采用了可编程图形管线，从OpenGL 2.0规范衍生而来。本文主要针对OpenGL ES3.0进行介绍。OpenGL ES3.0实现了具有可编程着色功能的图形管线，由OpenGL ES3.0 API规范和OpenGL ES着色语言规范组成。OpenGL ES 3.0从OpenGL 3.3规范衍生而来，引入了采用阴影贴图、体渲染、基于GPU的粒子动画、几何形状实例化、纹理压缩和伽玛校正等技术的重要功能。

# 2 OpenGL ES基础知识

## 2.1数据类型

OpenGL定义了自己的数据类型，为了保证传递数据的尺寸和精度正确，应坚持使用这些OpenGL的数据类型。以下是OpenGL中主要涉及的一些类型：

1. GLenum，用于GL枚举的无符号整型。通常用于通知OpenGL由指针传递的存储于数组中数据的类型。
2. GLboolean，用于单布尔值。OpenGL ES还定义了自己的真假值（GL\_TRUE和GL\_FALSE）。
3. GLbitfield，用于将多个布尔值（最多32个）打包到单个使用位操作变量的四字节整型。
4. GLbyte，有符号单字节整型，数值从-128到127。
5. GLushort，有符号双字节整型，数值从-32768到32767
6. GLuint，有符号四字节整型，数值从-2147483648到2147483647。
7. GLfloat，四字节精度浮点数。
8. GLclampf，四字节浮点数，特别表示数值为0.0到1.0。
9. GLvoid，void指示一个函数没有返回值或者没有参数。
10. GLfixed，定点数使用整型数存储实数。这通常是对3D系统的优化方式。
11. GLclampx，另一种定点型，用于使用定点运算来表示0.0到1.0之间的实数。

## 2.2点或者顶点

3D图像的最小单位称为点或者顶点，多边形由点构成，而物体由多个多边形组成。在OpenGL ES中，一个顶点数组是包括场景中部分或全部顶点数据的简单数组。

# 3 OpenGL ES 3.0 图形管线

## 3.1 引言

OpenGL ES 3.0实现了具有可编程着色功能的图形管线，如图3.1所示。

片段着色器

变换反馈

顶点缓冲区/数组对象

光栅化

图元装配

顶点着色器

纹理

API

帧缓冲区

逐片段操作

图3.1

其中顶点着色器和片段着色器部分属于可编程阶段。在OpenGL ES中必须创建这两个着色器，才能构成一个完整的着色程序。

## 3.2 顶点着色器

顶点着色器定义了在2D或者3D场景中几何图形是如何处理的。一个顶点指的是2D或者3D中的一点。在图像处理中，有4个顶点，每个顶点代表图像的一个角。顶点着色器设置顶点的位置，并且把位置和纹理坐标这样的参数发送到片段着色器。

其中顶点着色器的输入包括：

1. 着色器程序，描述顶点上执行操作的顶点着色器程序源代码或者可执行文件。
2. 顶点着色器输入，用顶点数组提供的每个顶点的数据。
3. 统一变量，顶点或者片段着色器使用的不变的数据。
4. 采样器，代表顶点着色器使用纹理的特殊统一变量类型。

在图元光栅化阶段，为每个生成的片段计算顶点着色器输出值，并作为输入传递给片段着色器。用于从分配给每个图元顶点的顶点着色器输出生成每个片段值得机制称作插值。OpenGL ES 3.0 相对于之前的版本，增加了变换反馈的新功能，使顶点着色器输出可以选择性地写入一个输出缓冲区。顶点着色器可用于通过矩阵变换位置、计算照明公式来生成逐顶点颜色以及生成或者变换纹理坐标等基于顶点的传统操作。由于顶点着色器是由应用程序规定，所以可用于执行自定义计算，实施各种新的变换。

## 3.3 图元装配和光栅化阶段

图元是指三角形、直线等几何对象。图元的每个定点被发送到顶点着色器的不同拷贝，在图元装配期间，这些顶点被组合成图元。对于每个图元，要根据其在视锥体内的情况进行相应的裁剪。接着就进入光栅化阶段，在此阶段绘制对应的图元。光栅化意为将图元转化为一组二维片段的过程，这些片段会被交给片段着色器处理。

## 3.4 片段着色器和逐片段操作

GPU会使用片段着色器在对象或者图片的每一个像素上进行计算，最终计算出每个像素的最终颜色。图片，是数据的集合。图片的文档包含每一个像素的各个颜色分量和像素透明度的值。因为对每个像素，算式是相同的，GPU流水线作业这个过程，从而有效地进行处理。对光栅化阶段生成的每个片段都需执行片段着色器。片段着色器的输入与顶点着色器类似，除去片段着色器的输入变量是光栅化单元用插值为每个片段生成的顶点着色器输出。片段着色器可以抛弃片段，也可以生成一个或者多个颜色值作为输出。一般除了渲染到多重渲染目标外，片段着色器只输出一个颜色值。在多重渲染目标的情况下，为每个渲染目标输出一个颜色值。光栅化阶段生成的颜色、深度、模板和屏幕坐标位置成为逐片段操作阶段的输入。在逐片段操作阶段，每个片段需要执行以下功能：

1. 像素归属测试，确定帧缓冲区中某个位置的像素是不是属于OpenGL ES。
2. 裁剪测试，确定某个位置的像素是否在裁剪矩形范围内。
3. 混合，将新生成的片段颜色值与保存在帧缓冲区某位置的颜色值组合起来。
4. 抖动，在使用有限精度在帧缓冲区保存颜色值时，会产生伪像，该功能用于最小化此伪像。

在此阶段的最后，片段不然被抛弃，不然就在帧缓冲区的某个位置写入颜色、深度等。

## 3.4 OpenGL ES 3.0 新功能

OpenGL ES 3.0引入了有关纹理、着色器、几何形状。缓冲区对象、帧缓冲区的新功能。下面进行简单的说明。

使用纹理是3D图像渲染最基础的操作。在纹理方面，引入了以下新功能：

1. sRGB纹理和帧缓冲区，允许应用程序执行伽玛校正渲染。纹理可保存在经过伽玛校正的sRGB空间，在着色器中读取时反校正到线性空间，然后输出到帧缓冲区时转换回sRGB伽玛校正空间。通过在线性空间中正确地进行照明和其他计算，可得到更高的视觉保真度。
2. 2D纹理数组，2D纹理是OpenGL ES应用中最普遍的，2D纹理数组保存了一组2D纹理。这种纹理数组可用来执行纹理动画。在这之前，一般使用在单个2D纹理中平铺动画帧并修改纹理坐标改变动画帧来实现这种动画效果。现在可在数组的每个2D切片中指定一个帧。
3. 3D纹理，3D纹理可以认为是多个2D纹理的叠加，对于许多医学成像应用是必不可少的。
4. 深度纹理和阴影比较，即启用存储在纹理中的深度缓冲区。深度纹理常用来渲染阴影。深度缓冲区会从光源的角度渲染，然后在渲染场景时比较，以确定片段是否在阴影中。
5. 无缝立方体，立方图可进行采样来使用相邻面的数据并删除接缝处的伪像。
6. 浮点纹理，访问浮点纹理数据可以实现很多应用，比如高动态范围纹理和多功能计算。

还有ETC2/EAC纹理压缩、整数纹理、非2幂次纹理（NPOT）、纹理细节级别（LOD）功能、纹理调配、不可变纹理、最小尺寸增大等新功能，这里就不进行详细的介绍。

在着色器方面，也有了很多重大的更新：

1. 二进制程序文件，在OpenGL ES 3.0中完全链接过的二进制程序文件可以保存为离线二进制格式，运行时不需要链接步骤，有利于减少应用程序的加载时间。且在3.0中，所有着色器的实现都有在线着色器编译器。
2. 非方矩阵和全整数支持，OpenGL ES 3.0中可支持方阵之外的新矩阵类型。全整数支持即支持整数标量和向量类型以及全整数操作。可实现从整数到浮点数、浮点数到整数的转换以及从纹理中读取整数值和向整数颜色缓冲区中输出整数值的功能。
3. 质点采样，用质点采样声明顶点着色器的输出变量，可以避免在多重采样时产生伪像。
4. 统一变量快，统一变量值可组合为统一变量块。统一变量块可更高效地加载，也可在多个着色器程序间共享。
5. 片段深度，片段着色器可显式控制当前片段的深度值，而不是依赖深度值的插值
6. 平面/平滑插值程序，插值程序即顶点着色器输出或者片段着色器输入可以显式声明为平面或者平滑着色。
7. 布局限定符，顶点着色器输入可用布局限定符声明，以显式绑定着色器源代码中的位置，且无需调用API。布局限定符也可用于片段着色器的输出，在渲染到多个渲染目标时将输出绑定到各个目标。最重要的是，布局限定符可用于控制统一变量块的内存布局。
8. 实例和顶点ID，现在在顶点着色器中可访问到顶点索引，若用实例进行渲染，还可访问实例ID。
9. 宽松的限制，OpenGL ES 3.0放松了对着色器的限制。不再限制指令长度，支持变量为基础的循环和分支，并支持数组索引。

在几何形状方面也引入了多种与几何形状规范和图元渲染控制相关的新功能：

1. 变换反馈，即可在缓冲区对象中捕捉顶点着色器的输出，这对粒子动画有很大的帮助。
2. 布尔遮挡查询，应用程序可查询一个或一组绘制调用的像素是否通过深度测试。这个功能可避免在边界被遮盖的对象上进行几何形状处理的优化。
3. 实例渲染，可有效地对包含类似几何形状但是属性不同的对象进行渲染。这一功能在进行人群的渲染时十分有用。
4. 图元重启，OpenGL 3.0中，可使用特殊的索引值表示新图元的开始。就可以消除使用三角形条带时生成退化三角形的需求。
5. 新顶点格式，包括有符号和无符号规范化顶点属性；8位、16位和32位整数属性以及16位半浮点。

为了提供图形管线各部分指定数据的效率和灵活性，对于缓冲区对象，增加了以下几个新功能：

1. 统一变量缓冲区对象，即为存储或绑定大的统一变量块提供高效的方法。有利于减少将统一变量值绑定到着色器的性能代价。
2. 顶点数组对象，可提供绑定和在顶点数组状态之间切换的高效方法。顶点数组对象实际上是顶点数组状态的容器对象。应用程序可通过顶点数组对象在一次API调用中切换顶点数组状态，而不是发出多个调用。
3. 采样器对象，将采样器状态与纹理对象分离。有利于提高纹理中共享采样器状态的效率。
4. 同步对象，是为了给应用程序提供检查一组OpenGL ES操作是否在GPU上完成执行的机制。与之相关的新功能是栅栏，这个功能为应用程序提供了通知GPU应等待一组OpenGL ES操作结束才能接受更多操作进入执行队列的方法。
5. 像素缓冲区对象，该功能使应用程序能够执行对像素操作和纹理传输操作的异步数据传输。这有利于CPU和GPU之间更快的数据传输，在传输操作期间，应用程序也可继续工作。
6. 缓冲区子界映射和对象间的拷贝，子界映射使得应用程序能够映射缓冲区的一个子区域供CPU访问。对象间的拷贝提供了一个从缓冲区对象向另一个缓冲区对象传输数据的高效机制，无需CPU的干预。

对于帧缓冲区主要增加了以下四个新功能：

1. 多重渲染目标，即允许应用程序同时渲染到多个颜色缓冲区。利用这个功能，片段着色器输出多个颜色，每个用于一个相连的颜色缓冲区。该功能也可以用于延迟着色。
2. 多重采样渲染缓冲区，可以使应用程序能够渲染到具备多重采样抗锯齿功能的屏幕外帧缓冲区。
3. 帧缓冲区失效提示，OpenGL ES 3.0的许多实现是基于块状渲染的GPU。该功能为应用程序提供了通知驱动程序不再需要帧缓冲区内容的机制。这使得驱动程序能够采取优化步骤，跳过不必要的图块恢复操作。
4. 新的混合方程式，OpenGL ES 3.0支持最大值/最小值函数作为混合方程式。

# 4 一个OpenGL ES 3.0示例

该示例说明创建绘制一个三角形所需要的步骤，先说明一个概念EGL。OpenGL ES命令需要渲染上下文和绘制表现。渲顶点和染上下文存储相关的OpenGL ES状态。绘制表面是用于绘制图元的表面，它指定渲染所需的缓冲区类型，如深度缓冲区。模板缓冲区等。EGL是Khronos渲染API和原生窗口系统之间的接口。

1. 用EGL创建屏幕上的渲染。
2. 加载顶点和片段着色器。
3. 创建一个程序对象，连接顶点和片段着色器，并链接程序对象。
4. 设置视口。
5. 清除颜色缓冲区。
6. 渲染简单图元。
7. 使颜色缓冲区的内容在EGL窗口表面中可见。

下面对样板代码的各个部分进行说明。

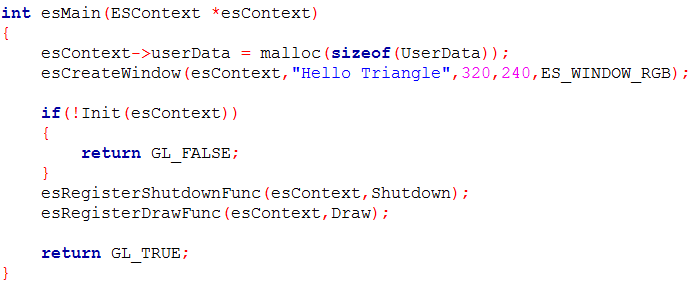


图4.1

如图4.1，首先是Hello Triangle的主函数，OpenGL ES 3.0的代码框架为每个应用程序声明名为esMain的主入口点。ESContext有一个名为userData，类型为void\*的成员变量。样板代码把应用程序所需的所有数据保存在userData中，ESContext结构中还包含窗口宽度和高度、EGL上下文和回调函数指针等信息。esMain函数主要负责分配userData，创建窗口和初始化绘图回调函数。

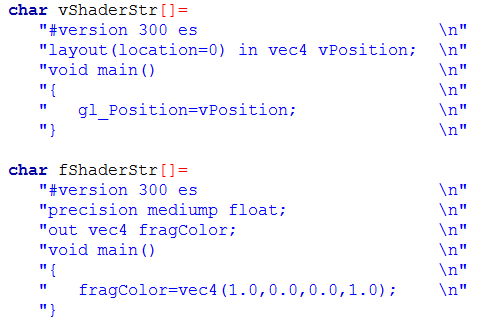


图4.2

如图4.2，所做的主要功能使创建简单的顶点和片段着色器。vShaderStr[]为顶点着色器声明，第一行表示使用的着色器版本，第二行声明一个名为vPosition的4分量向量。之后放在这个变量中的每个顶点的位置会传入如图4.3的Draw函数中。

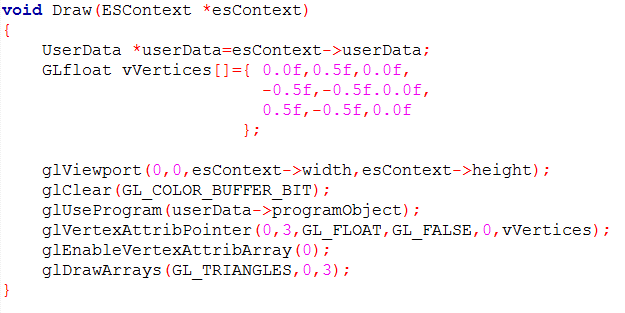


图4.3

layout(location=0)限定符表示这个变量的位置是顶点属性0。着色器声明一个main函数，表示着色器执行开始。它将vPosition输入属性拷贝到名为gl\_Position的特殊输出变量，每个顶点着色器必须在gl\_Position变量中输出一个位置。这个变量定义传递到管线的下一阶段。这里主要介绍的是顶点着色器的样子。

fShaderStr[]片段着色器第一行也是声明着色器的版本，下一条语句声明着色器中浮点变量的默认精度。片段着色器声明一个4分量向量的输出变量fragColor，写入这个变量的值会被输出到颜色缓冲区。

定义完着色器代码后，就使用如图4.4，LoadShader函数负责加载着色器源代码，该函数返回一个着色器对象，便于以后链接到程序对象。

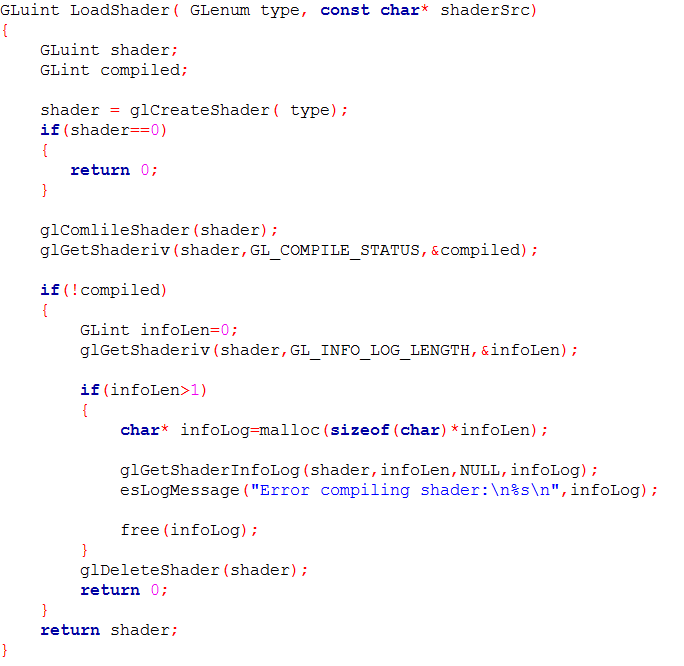


图4.4

应用程序为顶点和片段着色器创建了着色器对象，就需要创建一个程序对象。程序对象即最终链接的程序。不同的着色器编译为一个着色器对象之后，它们必须连接到一个程序对象并一起链接，才能绘制图形。这里将该过程进行简化描述。如图4.5。

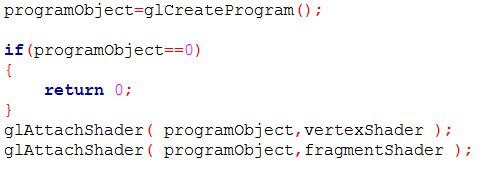


图4.5

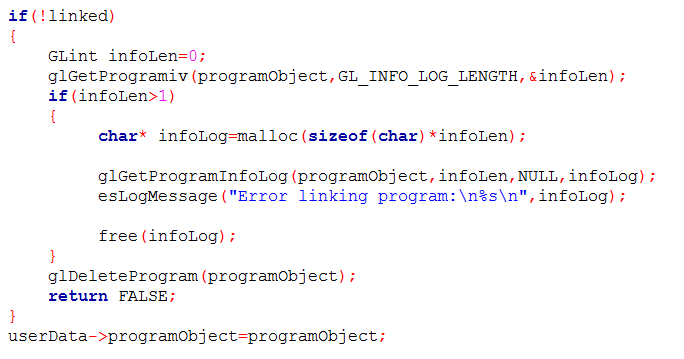
第一步是创建程序对象并将顶点着色器和片段着色器连接到对象上。如图4.6。

图4.6

然后链接程序并检查链接错误。程序对象成功链接之后，可使用程序对象来渲染。使用程序对象句柄调用glUseProgram后，即在Draw函数中的glUseProgram(userData->programObject)。之后后续的渲染将用链接到程序对象的顶点和片段着色器进行。接着需要设置视口和清除颜色缓冲区。Draw回调函数用于绘制帧，在Draw中执行的第一条命令是glViewport，它通知OpenGL ES用于绘制的2D渲染表面的原点、宽度和高度。

在OpenGL ES中，视口定义所有OpenGL ES渲染操作最终显示的2D矩形，即glviewport(0,0,esContext->width,esContext->height)；设置完视口后，下一步就是清除屏幕。在这个示例中，只向颜色缓冲区中绘制图形。在每个帧的开始，用glClear函数清除颜色缓冲区。之后我们需要指定三角形的几何形状。三角形的顶点由vVertices数组中的3个坐标(x,y.z)指定，即Draw函数中vVertices[]中数组所设定的值。

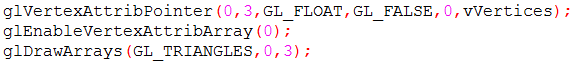


图4.7

如图4.7，顶点位置需加载到GL，并连接到顶点着色器中声明的vPosition属性。绘制三角形的最后一步是告诉OpenGL ES绘制图元。我们用glDrawArrays函数实现这个目的。

最后需要将三角形绘制到帧缓冲区，这里使用双缓冲区进行绘制。双缓冲区分为前台缓冲区和后台缓冲区，所有的渲染发生在后台缓冲区，位于不可见的屏幕内存区域。当渲染完成被交换到前台缓冲区即可见缓冲区，然后前台缓冲区变为下一帧的后台缓冲区。通过EGL函数eglSwapBuffers可以控制，框架在调用Draw函数回调函数之后调用该函数。

通过以上的例子，主要是为了熟悉OpenGL ES 3.0应用程序的几个关键组件。

# 5 总结

本文主要对OpenGL ES 3.0进行了简单的介绍，并没有深入讨论里面涉及到的一些细节。说明了与之前版本的区别，并用一个简单的例子介绍了OpenGL ES 3.0程序的大体框架。

# 参考文献

[1] DanGinsburg, BudirijantoPurnomo, 金斯伯格,等. OpenGL ES 3.0编程指南[M]. 机械工业出版社, 2015.1-29页

[2] 邹鹤敏. 基于OpenGL ES的3D游戏技术的研究与实现[D]. 西南交通大学, 2014.