MyArrow的专栏

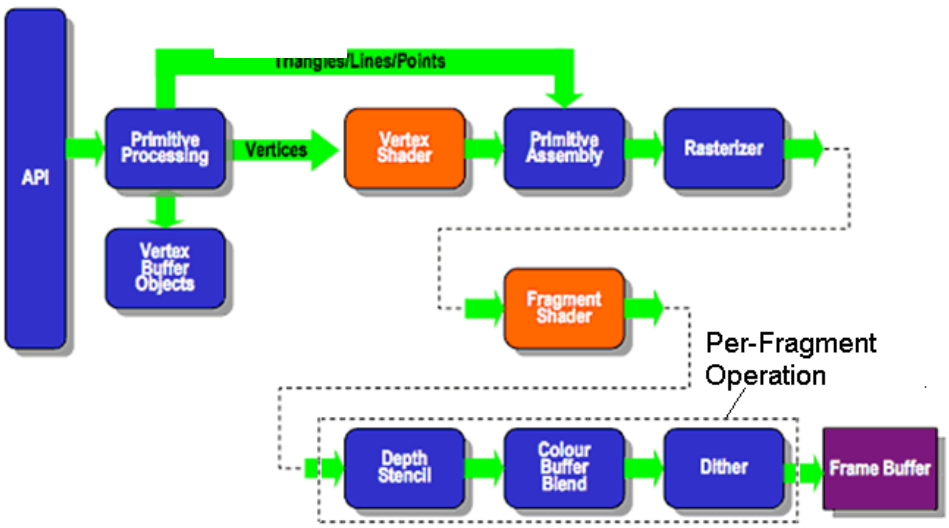
OpenGL ES 2.0基本概念

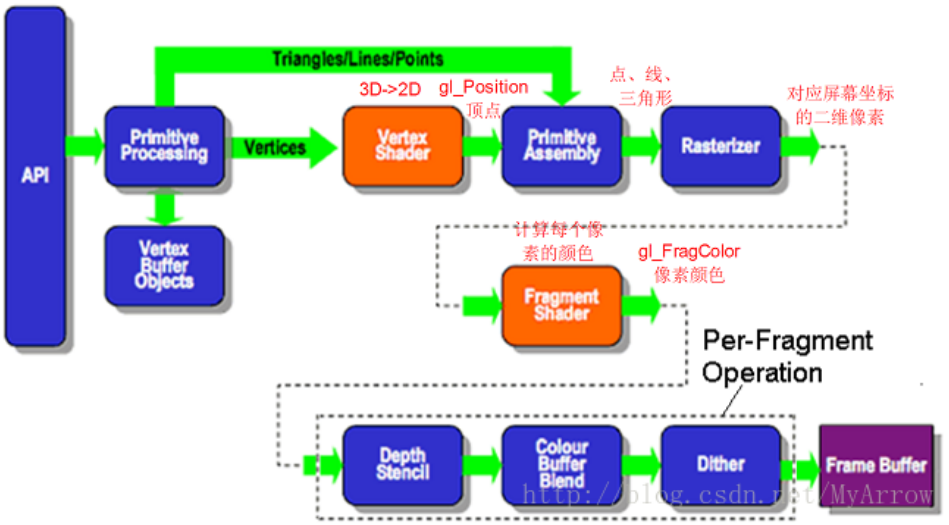
分类： OpenGL ES 2012-06-26 10:53 9640人阅读 评论(0) 收藏 举报

shader primitive variables attributes input

1. OpenGL ES 2.0可编程管道

OpenGL负责把三维空间中的对象通过投影、光栅化转换为二维图像，然后呈现到屏幕上。



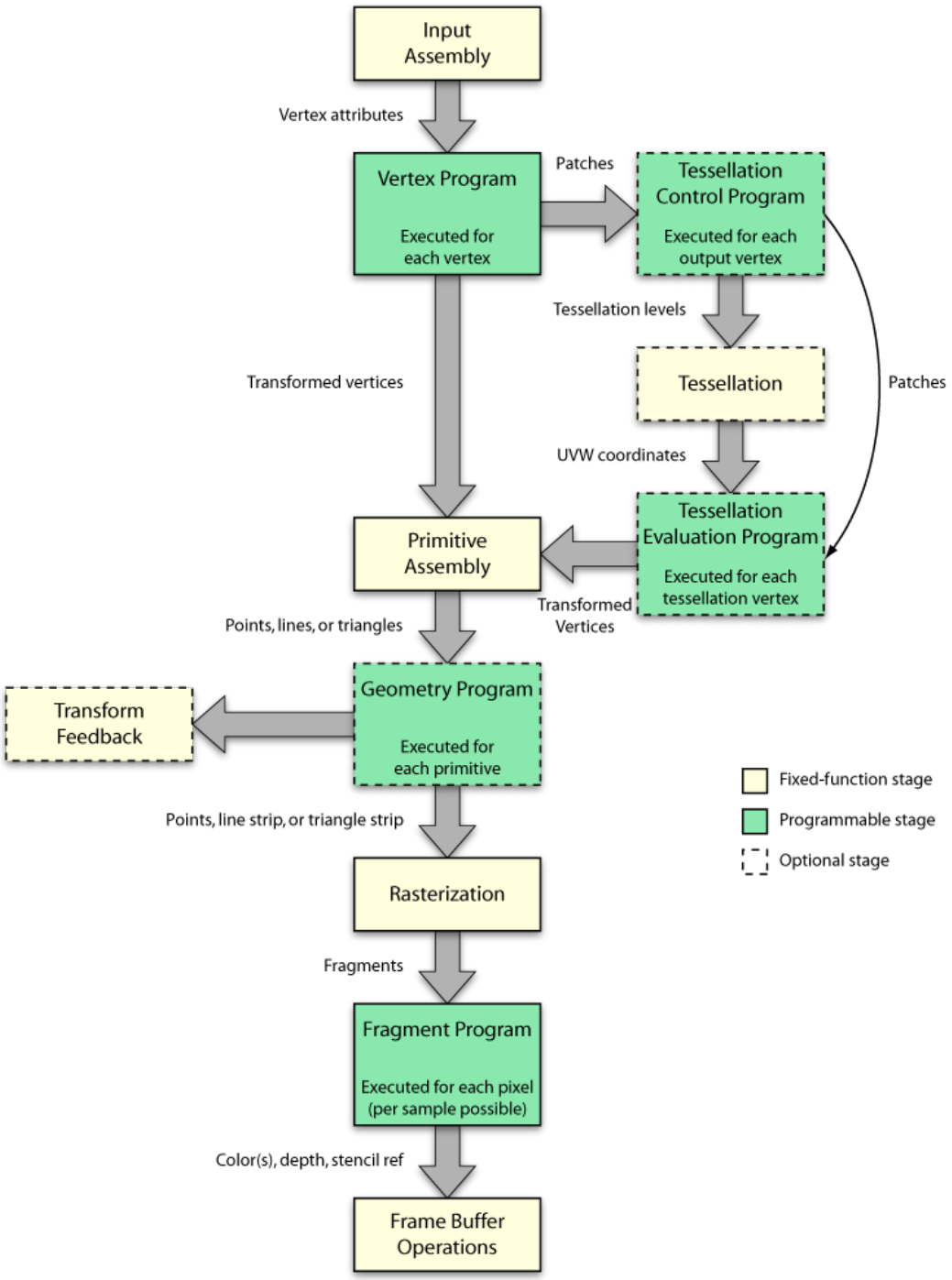


上图黄色部分(Vertex Shader和Fragment Shader)为此管道的可编程部分。整个管道包含以下两个规范：

1) OpenGL ES 2.0 API specification

2) OpenGL ES Shading Language Specification (OpenGL ES SL)

详细流程图如下：



此流程把三维数据变换为可以显示的二维数据。

2. Shader(就是一段程序Program)

Shader是一段执行在GPU上的程序(所以Shader也可以叫做**Program**)，此程序使用OpenGL ES SL语言来编写。它是一个描述顶点或像素特性的简单程序。

2.1 Pixel Shader

Pixel Shader(像素着色器)就是众所周知的Fragment Shader，它计算每个像素的颜色和其它属性。它通过应用光照值、凹凸贴图，阴影，镜面高光，半透明等处理来计算像素的颜色并输出。它也可改变像素的深度(z-buffering)或在多个渲染目标被激活的状态下输出多种颜色。一个Pixel Shader不能产生复杂的效果，因为它只在一个像素上进行操作，而不知道场景的几何形状。

2.2 Vertex Shader

对于发送给GPU的每一个Vertex(顶点)，都要执行一次Vertex Shader(顶点着色器)。

其功能是把每个顶点在虚拟空间中的三维坐标变换为可以在屏幕上显示的二维坐标，并带有用于z-buffer的深度信息。Vertex Shader可以操作的属性有：位置、颜色、纹理坐标，但是不能创建新的顶点。

Vertex Shader输入数据如下：

1)**Attributes:** 通过顶点数组(vertex arrays)提供的每个顶点数据

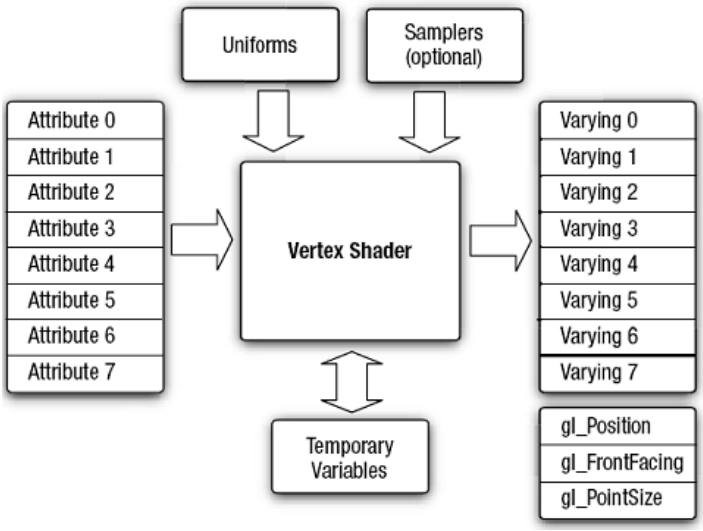
2)**Uniforms:** Vertex Shader使用的常量数据

3)**Samplers:** Vertex Shader使用的一个表示纹理类型的Uniforms，在Vertex Shader中是可选的

4)**Shader program:** Vertex Shader程序源码或可执行文件，它描述了将在顶点上执行的操作

Vertex Shader输出为：varying variables（Vertex Shader的输出变量，作为Fragment Shader的输入。必须与Fragment Shader中的输入Varying一一对应。）

gl_Position： 是每个点固有的Varying，表示点的空间位置。



A Vertex Shader Example：

```
// uniforms used by the vertex shader
uniform mat4 u_mvpMatrix; // matrix to convert P from model
                          // space to normalized device space.
// attributes input to the vertex shader
attribute vec4 a_position;   // position value
attribute vec4 a_color;      // input vertex color
// varying variables – input to the fragment shader
varying vec4 v_color;        // output vertex color
 void  main()
{
    v_color = a_color;
    gl_Position = u_mvpMatrix * a_position;
}
```

2.3 Geometry Shader

Geometry Shader是一个相对新的Shader类型。它在Driect3D 10和OpenGL 3.2中开始引入，在OpengGL 2.0+中作为扩展使用。它的输入为：点、线和三角形；其输出为点、线带和三角形带。

Geometry Shader程序在Vertex Shader程序执行之后执行。

3. Shading Language

GLSL : Shading language developed for use with OpenGL
HLSL : Shading language developed for use with the Direct3D API

Shading Language仅适合于为GPU编程，编程工具为Cg(Nvidia开发)，Cg编译器可以输出Direct3X或OpenGL Shader程序。

4. Shading 算法

4.1 插值技术(可以和任何照明模型组合):

- Flat shading
- Gouraud shading
- Phong shading

4.2 照明模型 (可以和任何插值技术组合):

- Blinn–Phong
- Cook–Torrance (microfacets)
- Lambert
- Minnaert
- Oren–Nayar (Rough opaque diffuse surfaces)
- Phong
- Ward anisotropic

5. 原语汇编(Primitive Assembly)

在管道(pipeline)中，执行完Vertex Shader之后，就执行原语汇编。一个原语(primitive)就是一个可以使用OpenGL ES画图命令进行绘制的几何对象。

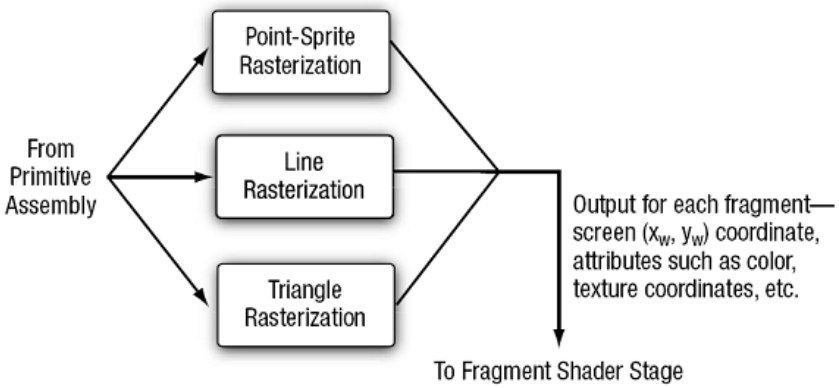
在原语汇编阶段，被Vertex Shader处理输出的顶点被组合进行一个单独可被绘制的**几何原语**(如：点、线和三角形)。然后对几何原语执行以下两种操作:

- Clipping(裁剪): 只保留在可视空间范围内的原语
- Culling(选择): 基于面向前或后，进行原语的选择

在裁剪和选择之后，其下一个阶段为进行Geometry Shader(如果存在)或光栅化处理。

6. 光栅(Rasterization)

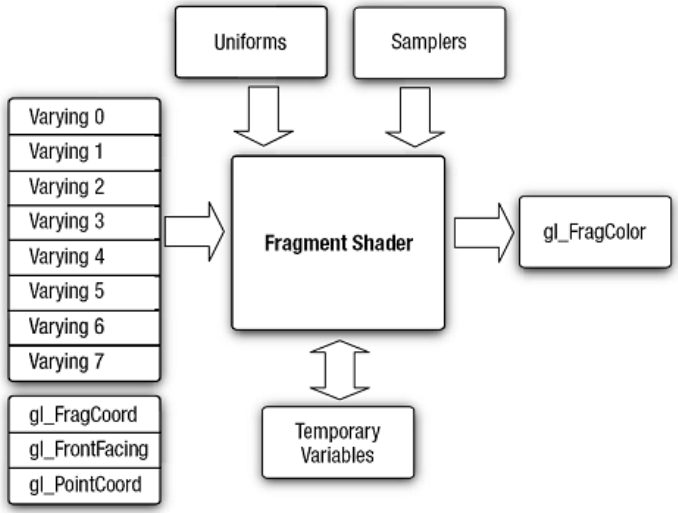
光栅处理的目标：把原语转换为一系列二维fragments，这些二维fragments将被Fragment Shader处理，且这些二维fragments表示可被绘制在屏幕上的像素。



7. Fragment Shader

对每个光栅化产生的fragment执行一次Fragment Shader，其输入内容如下:

- Varying variables：Vertex Shader的输出，且由光栅化单元对每个fragment使用插值方法产生
- Uniforms：Fragment Shader使用的常量数据
- Samplers：一个表示纹理的常量数据
- Shader program：Fragment shader程序源码或可执行文件，它描述了将在fragment上执行的操作。



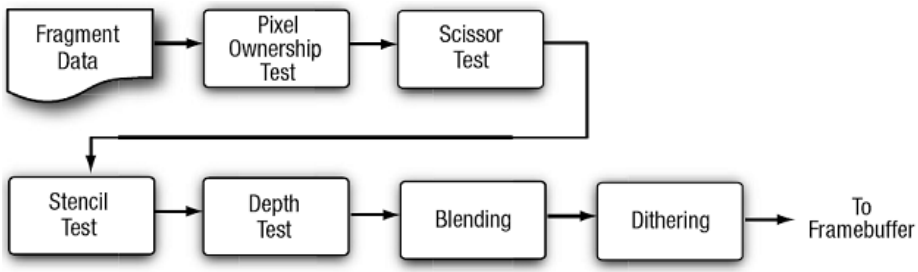
Fragment Shader要么丢弃此fragment，要么为此fragment产生对应的color，并保存在gl_FragColor中.

A Fragment Shader Example:

```
precision mediump float;
varying vec4 v_color; // input vertex color from vertex shader
void
main(void)
{
    gl_FragColor = v_color;
}
```

8. Per-Fragment Operations

在执行Fragment Shader之后，其下一步处理为Per-Fragment Operation。其处理流程如下:



执行Per-Fragment Operation的结果为此Fragment被丢弃，或在framebuffer对应位置(Xw,Yw)写入fragment**颜色值**，**深度值**或**stencil值**。