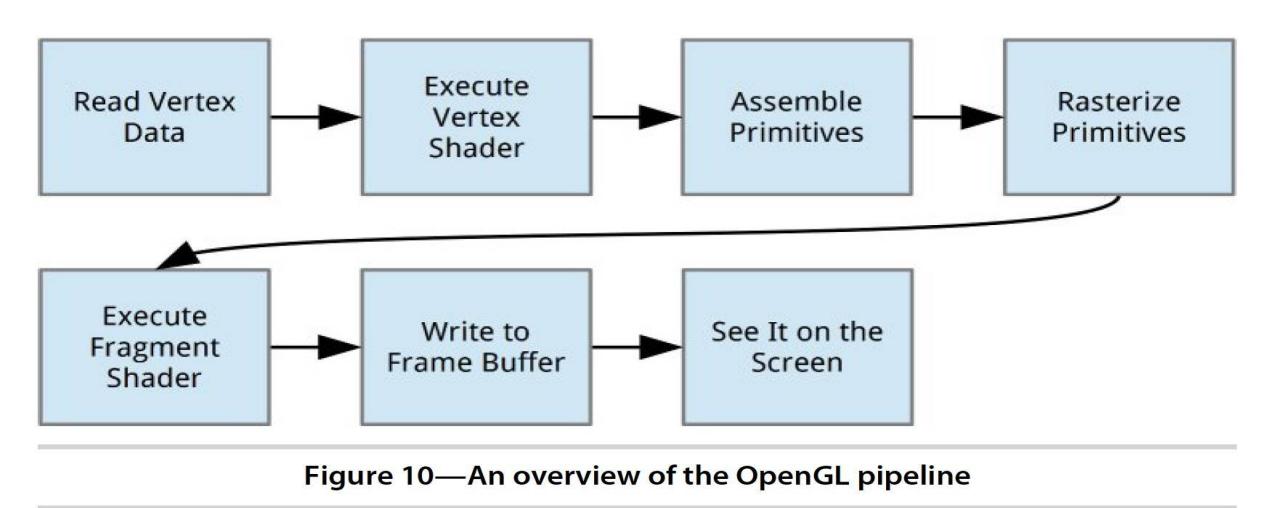
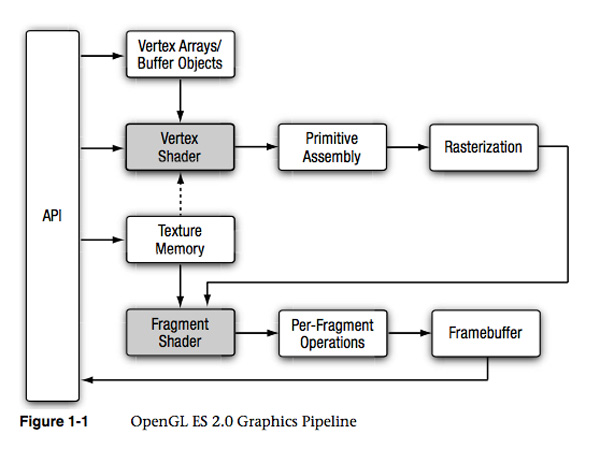
**Opengles**

一、pipeline

在 OpenGL ES 1.0 版本中，支持固定管线，而 OpenGL ES 2.0 版本不再支持固定管线，只支持可编程管线。2.0版本的pipeline如下：





一张图像要显示在屏幕上需要知道每一个像素点的显示位置和像素值，opengles 2.0中有两个可编程着色器（shader）来计算这两个变量，分别是1、使用Vertex Shader确定每一个点的具体显示的位置；2、使用Fragment Shader确定每一个点的具体颜色值。

Pipeline就从顶点着色器的数据读取开始：

1. read vertex data：顶点数据来源，这是渲染管线的顶点输入，一般一张图片需要四个点来描述，所以顶点数据是一个容量为8（2d）或12（3d）的浮点型数组，表示图片的四个顶点，把该数组传入顶点着色器。
2. Vertex Shader：顶点着色器通过可编程的方式实现对顶点的操作，如进行坐标空间转换，计算纹理坐标，顶点处理所做的工作包括：顶点变换、光照计算、纹理坐标变换。输出变换后的数据。
3. Primitive Assembly：图元装配。顶点着色器处理之后的数据经过图元装配阶段变成基本图元，opengl支持点、线、三角形三种。装配好的图元还要进行裁剪和剔除，裁剪掉视锥体之外的图元，剔除则有正面、背面和全部三种方式。
4. Rasterization：光栅化。图元装配后得到要显示的图像轮廓，光栅化就是利用插值算法计算出二维fragment，fragment包含图像轮廓内的每一个像素点信息，包含位置、颜色、纹理坐标。
5. Fragment Shader：片元着色器。可编程，接受光栅化处理之后的fragment，在着色器内部计算出每一个像素点的颜色、深度值。
6. Per-Fragment Operation：在这一阶段对片元着色器输出的每一个片元进行一系列测试与处理，从而决定最终用于渲染的像素。

Pixel ownership test：该测试决定像素在framebuffer中的位置是不是为当前OpenGL ES所有。也就是说测试某个像素是否对用户可见或者被重叠窗口所阻挡。

Scissor Test：剪裁测试，判断像素是否在由glScissor定义的剪裁矩形内，不在该剪裁区域内的像素就会被剪裁掉。

Stencil Test：模版测试，将模版缓存中的值与一个参考值进行比较，从而进行相应的处理。

Depth Test：深度测试，比较下一个片段与帧缓冲区中的片段的深度，从而决定哪一个像素在前面，哪一个像素被遮挡。

Blending：混合，混合是将片段的颜色和帧缓冲区中已有的颜色值进行混合，并将混合所得的新值写入帧缓冲。

Dithering：抖动，抖动是使用有限的色彩让你看到比实际图象更多色彩的显示方式，以缓解表示颜色的值的精度不够大而导致的颜色剧变的问题。

1. FrameBuffer。Pipeline的最后一步，这里存放了最终要显示在屏幕上的颜色、深度、模板值等。Client端（cpu）也可以通过PBO、glReadPixels等api从fbo读取颜色信息。

pipeline结束后把fbo中数据交换到屏幕上显示。

两个着色器的一些具体操作细节：

1、顶点着色器：顶点着色器对每个顶点执行一次运算，在渲染管线中每个顶点都是独立地被执行。在顶点着色器中最重要的任务是执行顶点坐标变换，坐标系变换。

顶点着色器的另一个功能是向后面的片段着色器提供一组易变变量（varying）。易变变量会在图元装配阶段之后被执行插值计算，如果是单重采样，其插值点为片段的中心，如果多重采样，其插值点可能为多个采样片段中的任意一个位置。易变变量可以用来保存插值计算片段的颜色，纹理坐标等信息。所以纹理坐标也是经过光栅化处理之后才传给片元着色器的。顶点着色器会给顶点绑定一个纹理坐标，在光栅化时通过插值使其一一对应。

1. 片元着色器。它的输入是每一个像素点的位置和纹理坐标（都是插值得到）。根据纹理坐标获取纹理像素值然后赋值给fragment作为它的颜色。所以可编程的片段着色器是实现一些高级特效如纹理贴图，光照，环境光，阴影等功能的基础。片段着色器的主要作用是计算每一个片段最终的颜色值（或者丢弃该片段）。

二、opengles调用流程

1、EGL

EGLDisplay display = eglGetDisplay(EGL\_DEFAULT\_DISPLAY)得到默认的egl display。

eglInitialize(display, &major, &minor)初始化display。

static const EGLint configAttribs[] = {

EGL\_RENDERABLE\_TYPE, EGL\_OPENGL\_ES2\_BIT,

EGL\_SURFACE\_TYPE, EGL\_WINDOW\_BIT,

EGL\_BLUE\_SIZE, 8,

EGL\_GREEN\_SIZE, 8,

EGL\_RED\_SIZE, 8,

EGL\_NONE

};

eglChooseConfig(display, configAttribs, &config, 1, &numConfig)，找到符合configAttribs的display配置config。

EGLSurface surface = eglCreateWindowSurface(display, config, window, NULL);根据window创建surface。

static const EGLint contextAttribs[] = {

EGL\_CONTEXT\_CLIENT\_VERSION, 2,

EGL\_NONE

};

EGLSurface context = eglCreateContext(display, config, EGL\_NO\_CONTEXT, contextAttribs);根据contextAttribs创建context。

eglMakeCurrent(display, surface, surface, context)绑定display surface context生效。

销毁时使用eglDestroyContext、eglDestroySurface、eglTerminate、eglMakeCurrent(egl->display, EGL\_NO\_SURFACE, EGL\_NO\_SURFACE, EGL\_NO\_CONTEXT);

Egl的作用主要是给opengles绑定一个surface。

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);设置默认的颜色值

glClear() 使上面的默认颜色值生效

glEnable(GL\_CULL\_FACE);图元装配时，保留正面的三角形

glCullFace(GL\_BACK);丢掉背面朝向的三角形

glDisable(GL\_DEPTH\_TEST);关掉深度测试

2、opengles操作

A、创建顶点和片元着色器并link到program

GLuint shader = glCreateShader(shader\_type)

glShaderSource(shader, 1, &shader\_source, NULL)

glCompileShader(shader)

glGetShaderiv

创建shader，load shader程序，编译shader，检测shader状态。创建Vertex shader和Fragment shader。

glCreateProgram

glAttachShader

glLinkProgram

glGetProgramiv

创建program，attach vertex和fragment shader，link program，检测program状态。

B、获取着色器中变量handle

glGetAttribLocation 得到shader中attribute属性变量的location（类似于handle），attribute只存在于顶点着色器，用来传输顶点坐标和纹理坐标等。

glGetUniformLocation 得到shader中uniform属性变量的location（类似于handle），uniform在图元处理过程中其值保持不变，由应用程序传输给着色器。

C、创建纹理并激活绑定到着色器sampler2D

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);解包时字节对齐，解包就是从cpu到gpu的数据传输。PACK是从gpu到cpu。

glUseProgram使用哪个program

glGenTextures生成纹理name

glActiveTexture激活纹理

glBindTexture绑定纹理对象和name

glTexParameteri

glTexParameterf给纹理设置参数：

GL\_TEXTURE\_MIN\_FILTER、GL\_TEXTURE\_MAG\_FILTER、GL\_TEXTURE\_WRAP\_S、GL\_TEXTURE\_WRAP\_T

GL\_TEXTURE\_MIN\_FILTER：当纹理需要缩小时，使用算法类型如下可选：

第一种是GL\_NEAREST，就是直接取原始纹理限定区域中最接近的一个像素的信息，作为映射点的信息。

第二种是GL\_LINEAR，就是根据原始纹理限定区域中最接近的四个像素的信息，计算出来一个加权平均值，作为映射点的信息。

第三种是GL\_NEAREST\_MIPMAP\_NEAREST

第四种是GL\_LINEAR\_MIPMAP\_NEAREST

第五种是GL\_NEAREST\_MIPMAP\_LINEAR

第六种是GL\_LINEAR\_MIPMAP\_LINEAR

GL\_TEXTURE\_MAG\_FILTER：当纹理需要放大时，使用的算法有GL\_NEAREST和GL\_LINEAR。

GL\_TEXTURE\_WRAP\_S和GL\_TEXTURE\_WRAP\_T表示S和T两个方向怎么填充颜色。

GL\_CLAMP\_TO\_EDGE使用纹理最后一行或一列的颜色值填充直到铺满

GL\_REPEAT重复纹理像素值直到铺满

GL\_MIRRORED\_REPEAT以镜像的形式重复

glUniform1i把创建的纹理与shader中的sampler2D变量绑定。

D、给着色器变量赋值

glUniformMatrix3fv给Uniform类型3x3矩阵变量赋值。

glUniformMatrix4fv给Uniform类型4x4矩阵变量赋值。

glVertexAttribPointer、glEnableVertexAttribArray给Vertex attribute变量赋值，使能。用来传输顶点和纹理坐标。

varying由顶点着色器传输给片段着色器中的插值数据。

E、绘制

glViewport设定绘制区域，指定绘制区域的左下角坐标和绘制宽高

glBindTexture、glTexImage2D给纹理对象赋值。

glDrawArrays这前面都是参数配置，这个动作才会真正开始整个opengles的pipeline，最终绘制到framebuffer（fbo）

eglSwapBuffers交换帧缓冲区，使绘制好的fbo交换到屏幕上。

三、glsl特性

glsl是着色器编程的脚本语言。

1、预定义

precision highp float;指定float类型默认精度为highp

#error指令会触发编译错误并向日志中写入内容。

#pragma指令可以向编译器明确与实现相关的指令。

#version指定了编译着色器的GLSL对应版本，对于OpenGL ES 2.0的着色器应将此值设置为100。默认是100

#extension，用来控制是否启用某些扩展的功能，当供应商扩展GLSL时，会增加新的语言扩展明细，如GL\_OES\_texture\_3D、GL\_OES\_EGL\_image\_external等。使用模板：

#extension extension\_name : behavior

behavior的选项有：

require指明扩展是必须的

enable指明扩展是启用的

warn

disable

例如，shader使用外部纹理时的写法如下：

#extension GL\_OES\_EGL\_image\_external : require

uniform samplerExternalOES inputImageTexture

2、attribute：

OpenGL ES 2.0实现支持的最少attribute个数是8个，注意attribute是用来存储单个顶点信息，应用程序的顶点数组在使用glVertexAttribPointer传递顶点时是一个一个传输的。

3、uniform

顶点着色器和片段着色器共享了uniform变量的命名空间，同一个uniform变量在顶点和片段着色器都可以获取到，uniform变量通常是存储在硬件中的“常量区”，所以数量也是有限制的，OpenGL ES 2.0的实现必须提供至少128个顶点uniform矢量及16个片段uniform矢量。

4、varying

OpenGL ES 2.0实现中的varying变量最小支持数为8，varying变量是vertex shader输出 fragment shader的输入，中间会经过光栅化插值处理，纹理坐标一般用varying的形式传递给片元着色器。

使用glsl语言实现特效