逻辑教育
Logic education

# Hello CC

OpenGL ES 主题[1]

## 视觉班—OpenGL ES GLSL 之旅

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

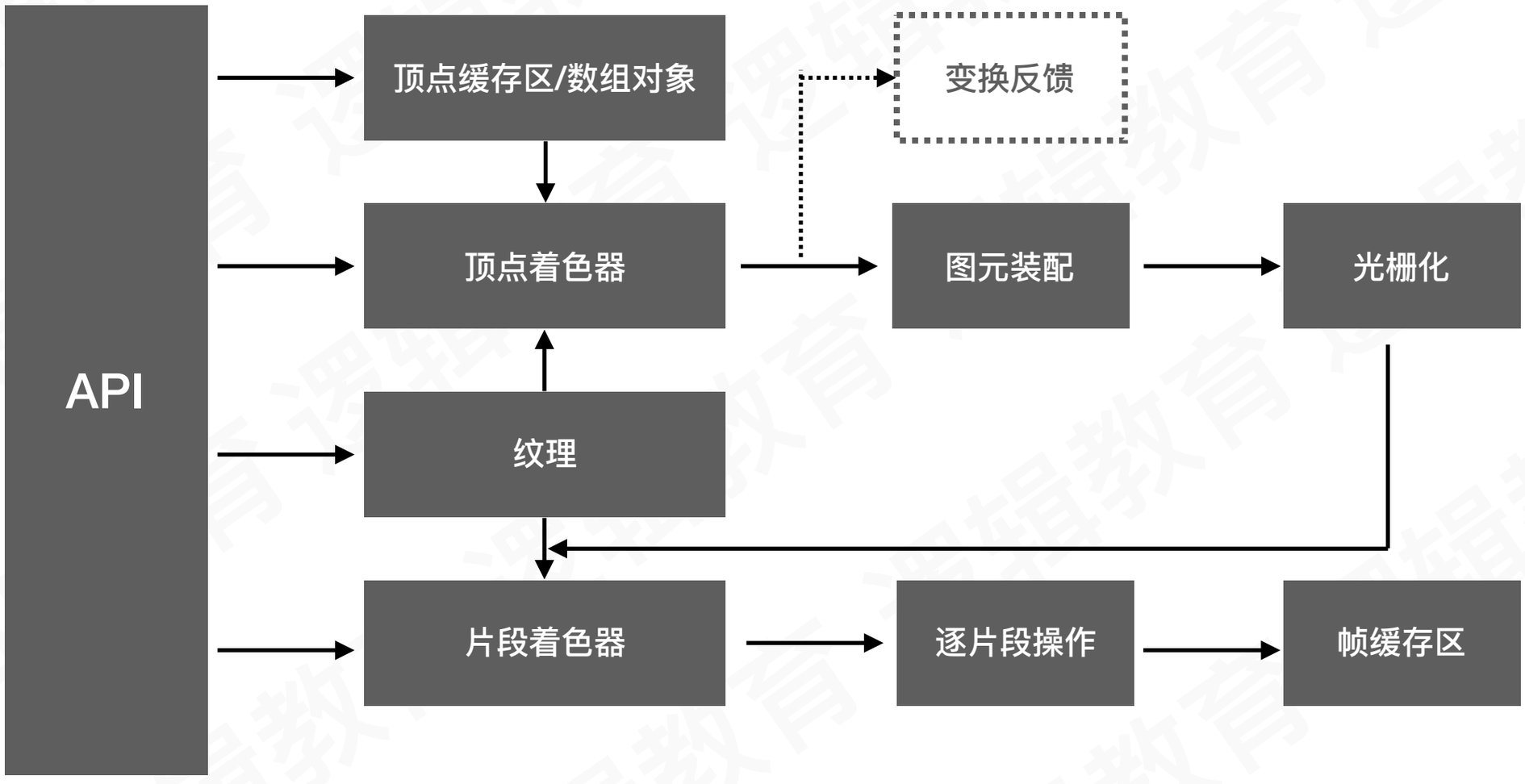
# OpenGL ES 简介

OpenGL ES (OpenGL for Embedded Systems) 是以手持和嵌入式为目标的高级3D图形应用程序编程接口(API). OpenGL ES 是目前智能手机中占据统治地位的图形API.支持的平台: iOS, Andriod , BlackBerry ,bada ,Linux ,Windows.

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# OpenGL ES 3.0 图形管线

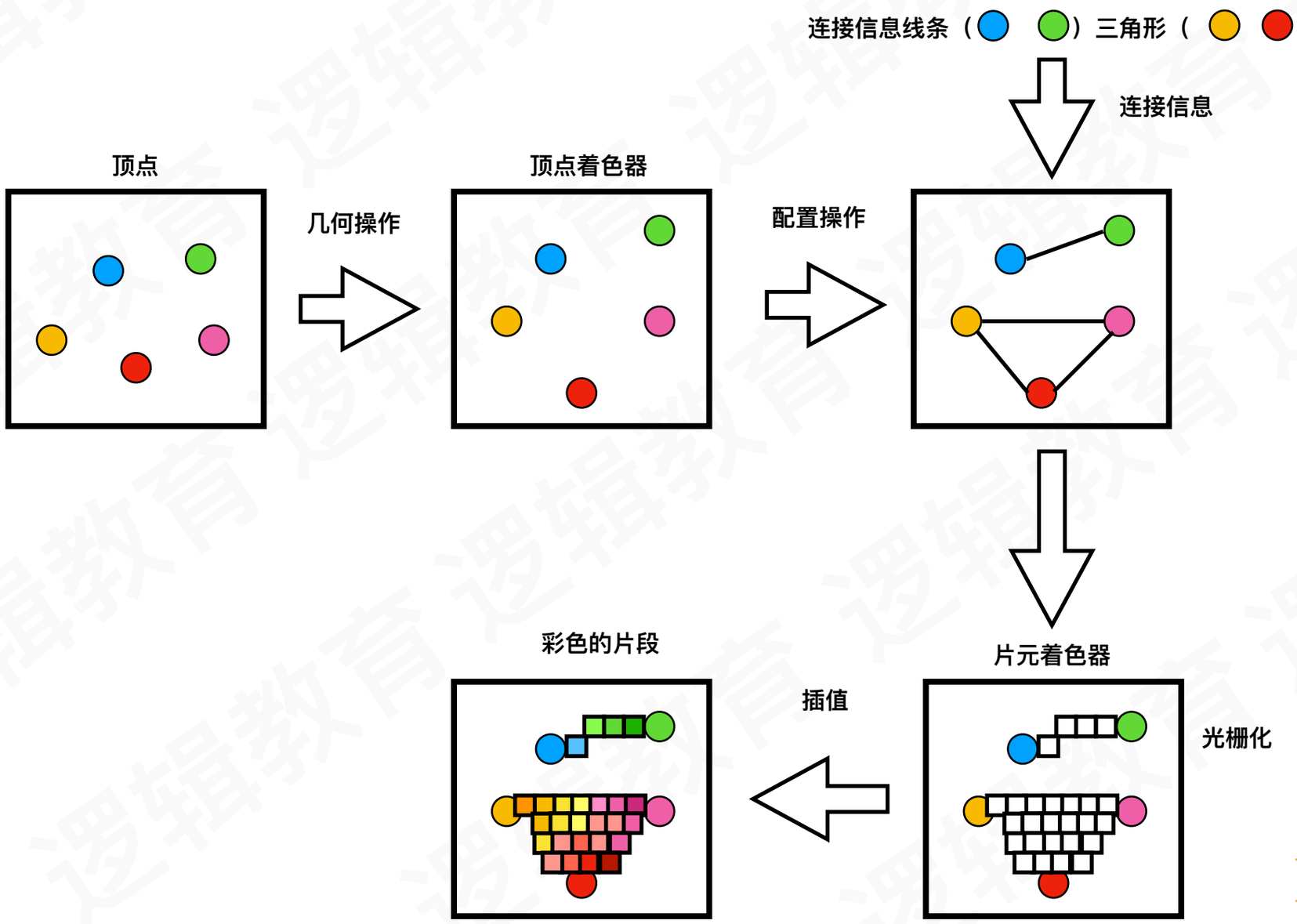



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 着色器渲染过程

在渲染过程中，必须存储2种着色器，分别是顶点着色器、片元着色器。顶点着色器是第一个着色器、片元着色器是最后一个。顶点着色器中处理顶点、片元着色器处理像素点颜色。



逻辑教育
Logic education

# 顶点着色器

顶点着色器 输入:

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/可执行文件

2.顶点着色器输入(属性) — 用顶点数组提供每个顶点的数据

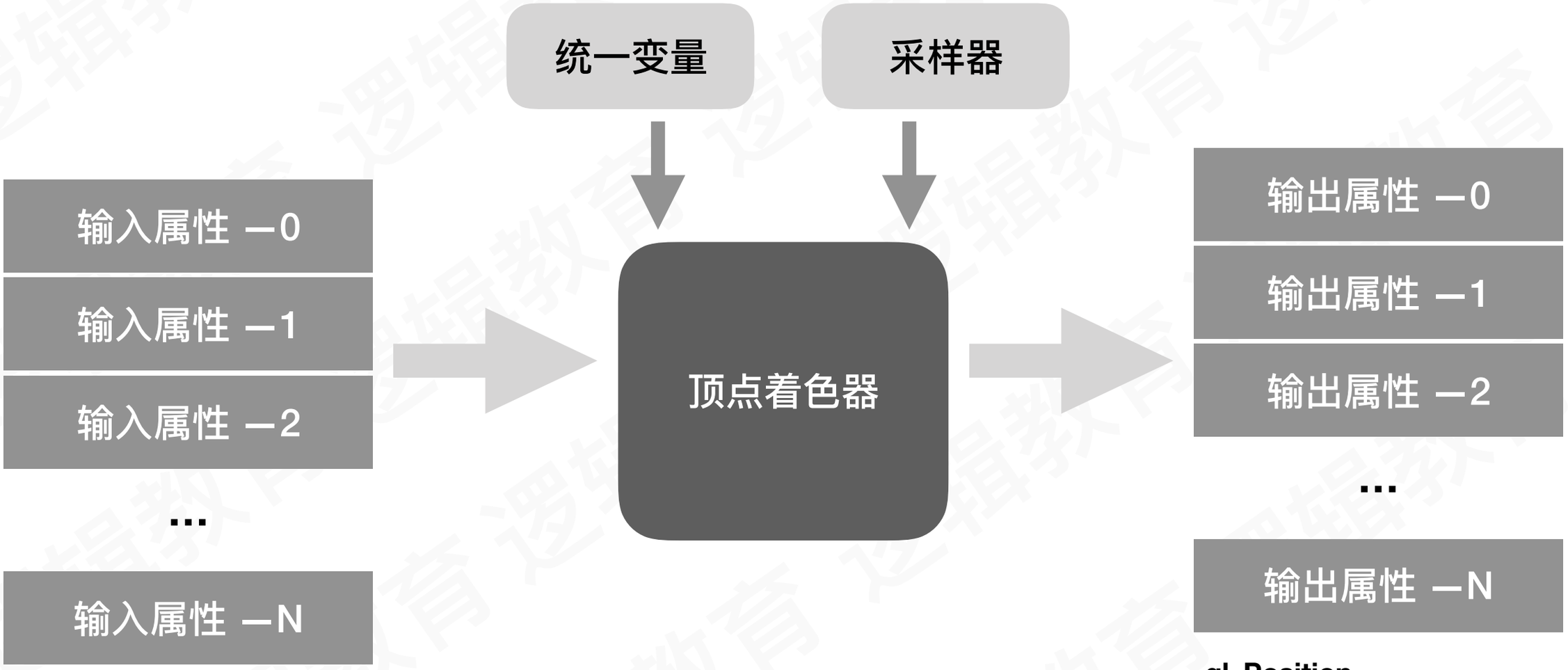
3.统一变量(uniform)—顶点/片元着色器使用的不变数据

4. 采样器—代表顶点着色器使用纹理的特殊统一变量类型.

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# OpenGL ES 3.0 顶点着色器




课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 顶点着色业务:

顶点着色器 业务:

1. 矩阵变换位置

2.计算光照公式生成逐顶点颜色

3.生成/变换纹理坐标

总结: 它可以用于执行自定义计算,实施新的变换,照明或者传统的固定功能所不允许的基于顶点的效果.

课程研发:CC老师
课程授课:CC老师

# 顶点着色代码案例:

```
attribute vec4 position;
attribute vec2 textCoordinate;
uniform mat4 rotateMatrix;
varying lowp vec2 varyTextCoord;
void main()
{
    varyTextCoord = textCoordinate;
    vec4 vPos = position;
    vPos = vPos * rotateMatrix;
    gl_Position = vPos;
}
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 图元装配

顶点着色器之后,下一个阶段就是图元装配.

图元(Primitive): 点,线,三角形等.

图元装配: 将顶点数据计算成一个个图元.在这个阶段会执行裁剪、透视分割和Viewport变换操作。

图元类型和顶点索确定将被渲染的单独图元。对于每个单独图元及其对应的顶点，图元装配阶段执行的操作包括：将顶点着色器的输出值执行裁剪、透视分割、视口变换后进入光栅化阶段。

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 光栅化

在这个阶段绘制对应的图元(点/线/三角形). 光栅化就是将图元转化成一组二维片段的过程.而这些转化的片段将由片元着色器处理.这些二维片段就是屏幕上可绘制的像素.


课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 片段着色器/片元着色器

片元着色器/片段着色器 输入:

1. 着色器程序—描述片段上执行操作的顶点着色器程序源代码/可执行文件

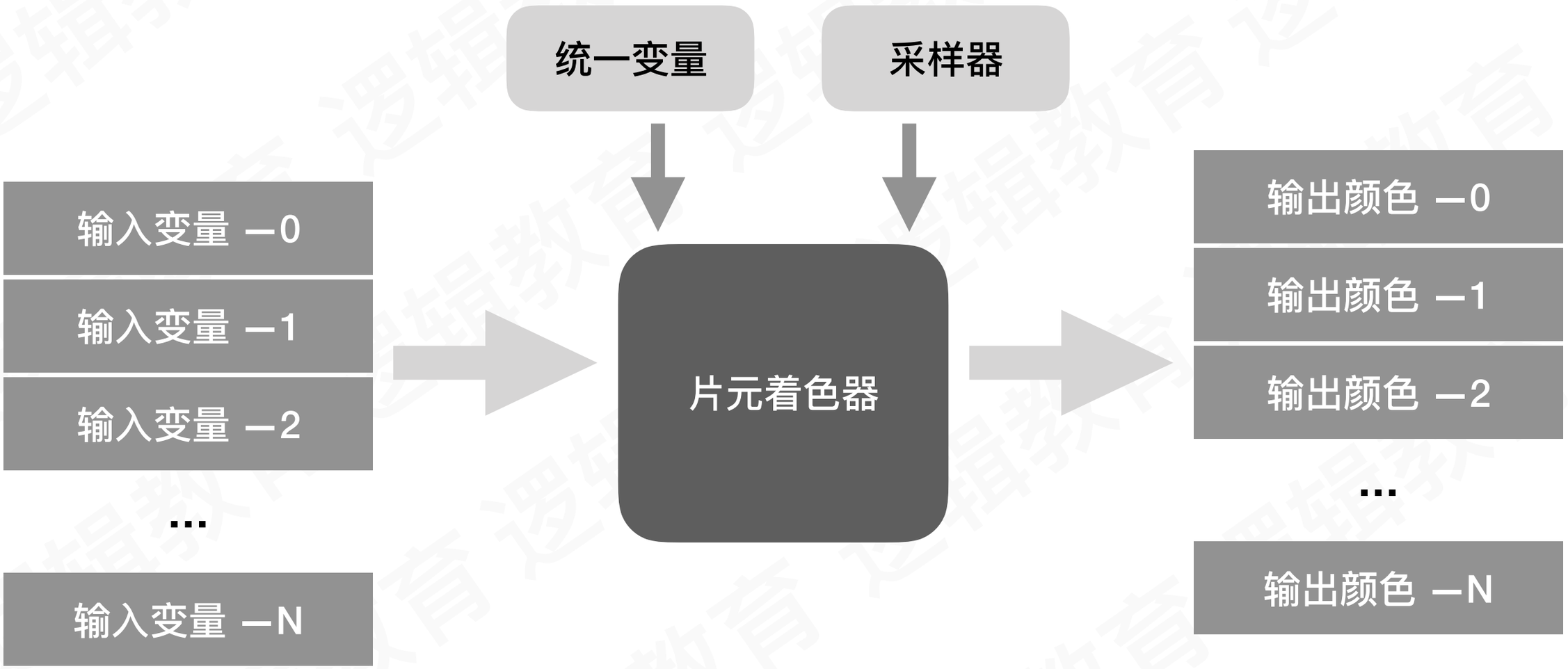
2.输入变量— 光栅化单元用插值为每个片段生成的顶点着色器输出

3.统一变量(uniform)—顶点/片元着色器使用的不变数据

4. 采样器—代表片元着色器使用纹理的特殊统一变量类型.

课程研发:CC老师
课程授课:CC老师

# OpenGL ES 3.0 片段着色器/片元着色器




**gl_FragColor**

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 片元着色器业务:

片元着色器 业务:

1. 计算颜色

2. 获取纹理值

3. 往像素点中填充颜色值(纹理值/颜色值);

总结: 它可以用于图片/视频/图形中每个像素的颜色填充(比如给视频添加滤镜,实际上就是将视频中每个图片的像素点颜色填充进行修改.)

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

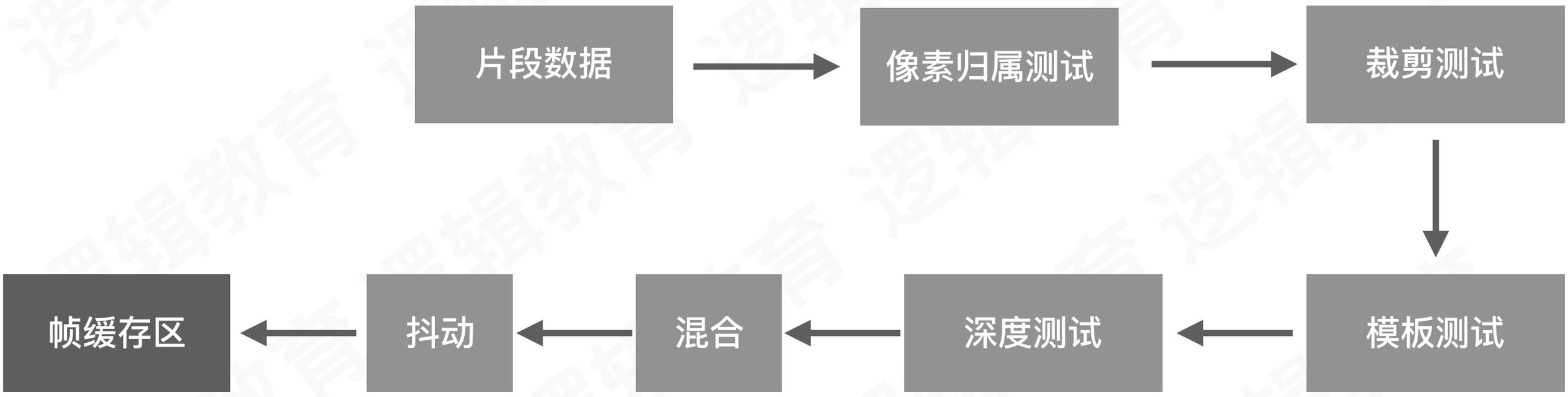
# 片元着色代码案例:

```
varying lowp vec2 varyTextCoord;
uniform sampler2D colorMap;
void main()
{
    gl_FragColor = texture2D(colorMap, varyTextCoord);
}
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 逐片段操作




课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 逐片段操作

- 像素归属测试: 确定帧缓存区中位置$(X_W,Y_W)$的像素目前是不是归属于OpenGL ES所有.例如,如果一个显示OpenGL ES帧缓存区View被另外一个View 所遮蔽.则窗口系统可以确定被遮蔽的像素不属于OpenGL ES 上下文.从而不全显示这些像素. 而像素归属测试是OpenGL ES 的一部分,它不由开发者开人为控制,而是由OpenGL ES 内部进行.
- 裁剪测试: 裁剪测试确定$(X_W,Y_W)$是否位于作为OpenGL ES状态的一部分裁剪矩形范围内.如果该片段位于裁剪区域之外,则被抛弃.
- 模板和深度测试: 输入片段的模板/深度值进步比较,确定片段是否拒绝测试
- 混合: 混合将新生成的片段颜色与保存在帧缓存的位置的颜色值组合起来.
- 抖动: 抖动可用于最小化因为使用有限精度在帧缓存区中保存颜色值而产生的伪像.

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# EGL (Embedded Graphics Library )

- OpenGL ES 命令需要渲染上下文和绘制表面才能完成图形图像的绘制.
- 渲染上下文: 存储相关OpenGL ES 状态.
- 绘制表面: 是用于绘制图元的表面,它指定渲染所需要的缓存区类型,例如颜色缓存区,深度缓冲区和模板缓存区.
- OpenGL ES API 并没有提供如何创建渲染上下文或者上下文如何连接到原生窗口系统. EGL 是Khronos 渲染API(如OpenGL ES) 和原生窗口系统之间的接口. 唯一支持OpenGL ES 却不支持EGL 的平台是iOS

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# EGL (Embedded Graphics Library )

**EGL的主要功能如下：**

1. 和本地窗口系统（native windowing system）通讯；
2. 查询可用的配置；
3. 创建OpenGL ES可用的“绘图表面”（drawing surface）；
4. 同步不同类别的API之间的渲染，比如在OpenGL ES和OpenVG之间同步，或者在OpenGL和本地窗口的绘图命令之间；
5. 管理“渲染资源”，比如纹理映射（rendering map）。

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# OpenGL ES 错误处理

如果不正确使用OpenGL ES 命令,应用程序就会产生一个错误编码. 这个错误编码将被记录,可以用glGetError查询. 在应用程序用glGetError查询第一个错误代码之前,不会记录其他错误代码. 一旦查询到错误代码,当前错误代码便复位为GL_NO_ERROR.

## GLenum glGetError(void)

| 错误代码 | 描述 |
| --- | --- |
| GL_NO_ERROR | 从上一次调用glGetError 以来没有生成任何错误 |
| GL_INVALID_ENUM | GLenum 参数超出范围,忽略生成错误命令 |
| GL_INVALID_VALUE | 数值型 参数超出范围,忽略生成错误命令 |
| GL_INVALID_OPERATION | 特定命令在当前OpenGL ES 状态无法执行 |
| GL_OUT_OF_MEMORY | 内存不足时执行该命令,如果遇到这个错误,除非当前错误代码,否则OpenGL ES 管线的状态被认为未定义 |

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# OpenGL 与OpenGL ES的基本概念与历史

## 五、OpenGL ES 的版本

**OpenGL ES 1.X :针对固定功能流水管线硬件**
**OpenGL ES 2.X :针对可编程流水管线硬件**
**OpenGL ES 3.X :OpenGL ES 2.0的扩展**

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 案例目标

- 用EAGL 创建屏幕上的渲染表面
- 加载顶点/片元着色器
- 创建一个程序对象,并链接顶点/片元着色器,并链接程序对象
- 设置视口
- 清除颜色缓存区
- 渲染简单图元
- 使颜色缓存区的内容在EAGL 窗口表现呈现

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 着色器与程序

- 着色器与程序对象
- 创建和编译着色器
- 创建并链接程序
- 获取和设置统一变量
- 获取和设置属性
- 着色器编译器与程序二进制代码

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 着色器与程序

- 需要创建2个基本对象才能用着色器进行渲染: 着色器对象和程序对象.
- 获取链接后着色器对象的一般过程包括6个步骤:
  1. 创建一个顶点着色器对象和一个片段着色器对象
  2. 将源代码链接到每个着色器对象
  3. 编译着色器对象
  4. 创建一个程序对象
  5. 将编译后的着色器对象连接到程序对象
  6. 链接程序对象

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 创建与编译一个着色器

```
GLuint glCreateShader(GLenum type);
type    — 创建着色器的类型,GL_VERTEX_SHADER 或者GL_FRAGMENT_SHADER
返回值 — 是指向新着色器对象的句柄.可以调用glDeleteShader 删除

void glDeleteShader(GLuint shader);
shader — 要删除的着色器对象句柄


void glShaderSource(GLuint shader , GLSizei count ,const GLChar * const *string, const GLint
*length);
shader — 指向着色器对象的句柄
count   — 着色器源字符串的数量,着色器可以由多个源字符串组成,但是每个着色器只有一个main函数
string   — 指向保存数量的count 的着色器源字符串的数组指针
length  — 指向保存每个着色器字符串大小且元素数量为count 的整数数组指针.
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 创建与编译一个着色器

```
void glCompileShader(GLuint shader);

shader — 需要编译的着色器对象句柄

void glGetShaderiv(GLuint shader , GLenum pname , GLint *params );
shader — 需要编译的着色器对象句柄
pname — 获取的信息参数,可以为 GL_COMPILE_STATUS/GL_DELETE_STATUS/
GL_INFO_LOG_LENGTH/GL_SHADER_SOURCE_LENGTH/ GL_SHADER_TYPE

params — 指向查询结果的整数存储位置的指针.


void glGetShaderInfolog(GLuint shader , GLSizei maxLength, GLSizei *length , GLChar *infoLog);
shader — 需要获取信息日志的着色器对象句柄
maxLength — 保存信息日志的缓存区大小
length — 写入的信息日志的长度(减去null 终止符); 如果不需要知道长度. 这个参数可以为Null
infoLog — 指向保存信息日志的字符缓存区的指针.
```

# 创建与链接程序

**GLUint glCreateProgram( )**
**创建一个程序对象**
**返回值: 返回一个执行新程序对象的句柄**

**void glDeleteProgram( GLuint program )**
**program： 指向需要删除的程序对象句柄**

**//着色器与程序连接/附着**
**void glAttachShader( GLuint program , GLuint shader );**
**program： 指向程序对象的句柄**
**shader： 指向程序连接的着色器对象的句柄**

**//断开连接**
**void glDetachShader(GLuint program);**
**program : 指向程序对象的句柄**
**shader： 指向程序断开连接的着色器对象句柄**

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 创建与链接程序

glLinkProgram(GLuint program)
program: 指向程序对象句柄

链接程序之后, 需要检查链接是否成功. 你可以使用glGetProgramiv 检查链接状态:
void glGetProgramiv (GLuint program,GLenum pname, GLint *params);
program: 需要获取信息的程序对象句柄
pname : 获取信息的参数,可以是:

GL_ACTIVE_ATTRIBUTES
GL_ACTIVE_ATTRIBUTES_MAX_LENGTH
GL_ACTIVE_UNIFORM_BLOCK
GL_ACTIVE_UNIFORM_BLOCK_MAX_LENGTH
GL_ACTIVE_UNIFROMS
GL_ACTIVE_UNIFORM_MAX_LENGTH
GL_ATTACHED_SHADERS
GL_DELETE_STATUS
GL_INFO_LOG_LENGTH
GL_LINK_STATUS
GL_PROGRAM_BINARY_RETRIEVABLE_HINT
GL_TRANSFORM_FEEDBACK_BUFFER_MODE
GL_TRANSFORM_FEEDBACK_VARYINGS
GL_TRANSFORM_FEEDBACK_VARYING_MAX_LENGTH
GL_VALIDATE_STATUS

params : 指向查询结果整数存储位置的指针

逻辑教育
Logic education

# 创建与链接程序

从程序信息日志中获取信息
void glGetPorgramInfoLog(  GLuint program ,GLSizei maxLength, GLSizei *length , GLChar *infoLog )

program：指向需要获取信息的程序对象句柄
maxLength：存储信息日志的缓存区大小
length：写入的信息日志长度(减去null 终止符),如果不需要知道长度,这个参数可以为Null.
infoLog：指向存储信息日志的字符缓存区的指针

void glUseProgram(GLuint program)
program:  设置为活动程序的程序对象句柄.

逻辑教育
Logic education

# 顶点着色程序与片元着色程序如何实现编译、绑定和连接的?

## 1.指定属性

关于 gltLoadShaderPairWithAttributes 函数的底层实现分析

```
    //参数第一部分: 着色器程序名称
    //参数第二部分: 参数数量
    //参数第三部分: 索引于参数名称

    myIdentityShader = gltLoadShaderPairWithAttributes("ShadedIdentity.vp", "ShadedIdentity.fp", 2,
                                                       GLT_ATTRIBUTE_VERTEX, "vVertex", GLT_ATTRIBUTE_COLOR, "vColor");
}

GLuint gltLoadShaderPairWithAttributes(const char *szVertexProg,const char *szFramgmentProg)
{
```

## 2.设置源代码

创建2个着色器ID来加载着色器源码

```
    //零时着色器对象
    GLuint hVertextShader;
    GLuint hFragmentShader;
    GLuint hReturn = 0;
    GLint testVal;

    //创建着色器对象
    hVertextShader = glCreateShader(GL_VERTEX_SHADER);
    hFragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
```

顶点着色程序与片元着色程序如何实现编译、绑定和连接的?

将着色器源文件送入着色器对象中

```
//加载着色器程序
//顶点程序加载
if (gltLoadShaderFile(szVertexProg, hVertextShader) == false) {
    glDeleteShader(hVertextShader);
    glDeleteShader(hFragmentShader);

    printf("Not Found Shader");
    return 0;
}

//片段程序加载
if (gltLoadShaderFile(szFramgmentProg, hFragmentShader) == false) {
    glDeleteShader(hFragmentShader);
    glDeleteShader(hVertextShader);

    printf("Not Found Shader");
    return 0;

}
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

顶点着色程序与片元着色程序如何实现编译、绑定和连接的?

编译着色器，然后判断是否有错误

```
//编译顶点着色器程序和片元着色器程序
glCompileShader(hVertextShader);
glCompileShader(hFragmentShader);

//检查顶点着色器程序错误
glGetShaderiv(hVertextShader,GL_COMPILE_STATUS,&testVal);
if (testVal == GL_FALSE) {
    char infolog[1024];
    glGetShaderInfoLog(hVertextShader,1024,NULL,infolog);
    printf("The shader at %s failde",infolog);
    glDeleteShader(hVertextShader);
    glDeleteShader(hFragmentShader);
    return 0;

}
//检查片元着色器程序错误
glGetShaderiv(hFragmentShader,GL_COMPILE_STATUS,&testVal);
if (testVal == GL_FALSE) {
    char infolog[1024];
    glGetShaderInfoLog(hFragmentShader,1024,NULL,infolog);
    printf("The shader at %s failde",infolog);
    glDeleteShader(hVertextShader);
    glDeleteShader(hFragmentShader);
    return 0;

}
```

课程研发:CC老师
课程授课:CC老师

顶点着色程序与片元着色程序如何实现编译、绑定和连接的?

进行连接&绑定

```
//创建最终的程序对象，连接着色器
hReturn = glCreateProgram();
glAttachShader(hReturn,hVertextShader);
glAttachShader(hReturn,hFragmentShader);

//将参数名绑定指定的参数位置列表上
va_list attributeList;
va_start(attributeList,szFramgmentProg);

//重复迭代参数列表
char *szNextArg;
int iArgCount = va_arg(attributeList,int);
for (int i = 0; i < iArgCount; i++) {
    int index = va_arg(attributeList,int);
    szNextArg = va_arg(attributeList,char *);
    glBindAttribLocation(hReturn,index,szNextArg);
}

va_end(attributeList);
```

1.创建最终程序
2.绑定顶点着色器
3.绑定片元着色器

1.建立参数列表
2.开始绑定

通过循环绑定参数列表。
1.获取属性个数
2.遍历属性列表
3.绑定属性

顶点着色程序与片元着色程序如何实现编译、绑定和连接的?

连接着色器

```
    //尝试连接
    glLinkProgram(hReturn);

    //生成运行文件后可以删除
    glDeleteShader(hVertextShader);
    glDeleteShader(hFragmentShader);

    //确认连接有效
    glGetProgramiv(hReturn,GL_LINK_STATUS,&testVal);

    if (testVal == GL_FALSE) {
        char infolog[1024];
        glGetShaderInfoLog(hReturn,1024,NULL,infolog);
        printf("The shader at %s failde",infolog);
        glDeleteProgram(hReturn);
        return 0;

    }

    return 0;
}
```

课程研发:CC老师
课程授课:CC老师

## 向量数据类型

| 类型 | 描述 |
| --- | --- |
| vec2,vec3,vec4 | 2分量、3分量、4分量浮点向量 |
| ivec2,ivec3,ivec4 | 2分量、3分量、4分量整型向量 |
| uvec2,uvec3,uvec4 | 2分量、3分量、4分量无符号整型向量 |
| bvec2,bvec3,bvec4 | 2分量、3分量、4分量bool型向量 |

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

## 矩阵数据类型

| 类型 | 描述 |
| --- | --- |
| mat2,mat2x2 | 两行两列 |
| mat3,mat3x3 | 三行三列 |
| mat4,mat4x4 | 四行四列 |
| mat2x3 | 三行两列 |
| mat2x4 | 四行两列 |
| mat3x2 | 两行三列 |
| mat3x4 | 四行三列 |
| mat4x2 | 两行四列 |
| mat4x3 | 三行四列 |

## mat列x行

# 变量存储限定符

| 限定符 | 描述 |
| --- | --- |
| <none> | 只是普通的本地变量，外部不见，外部不可访问 |
| const | 一个编译常量，或者说是一个对函数来说为只读的参数 |
| in/varying | 从以前阶段传递过来的变量 |
| in/varying centroid | 一个从以前的阶段传递过来的变量，使用质心插值 |
| out/attribute | 传递到下一个处理阶段或者在一个函数中指定一个返回值 |
| out/attribute centroid | 传递到下一个处理阶段，质心插值 |
| uniform | 一个从客户端代码传递过来的变量，在顶点之间不做改变 |

**质心插值学术文献：http://www.ixueshu.com/document/f2f5be57efaaad68.html**

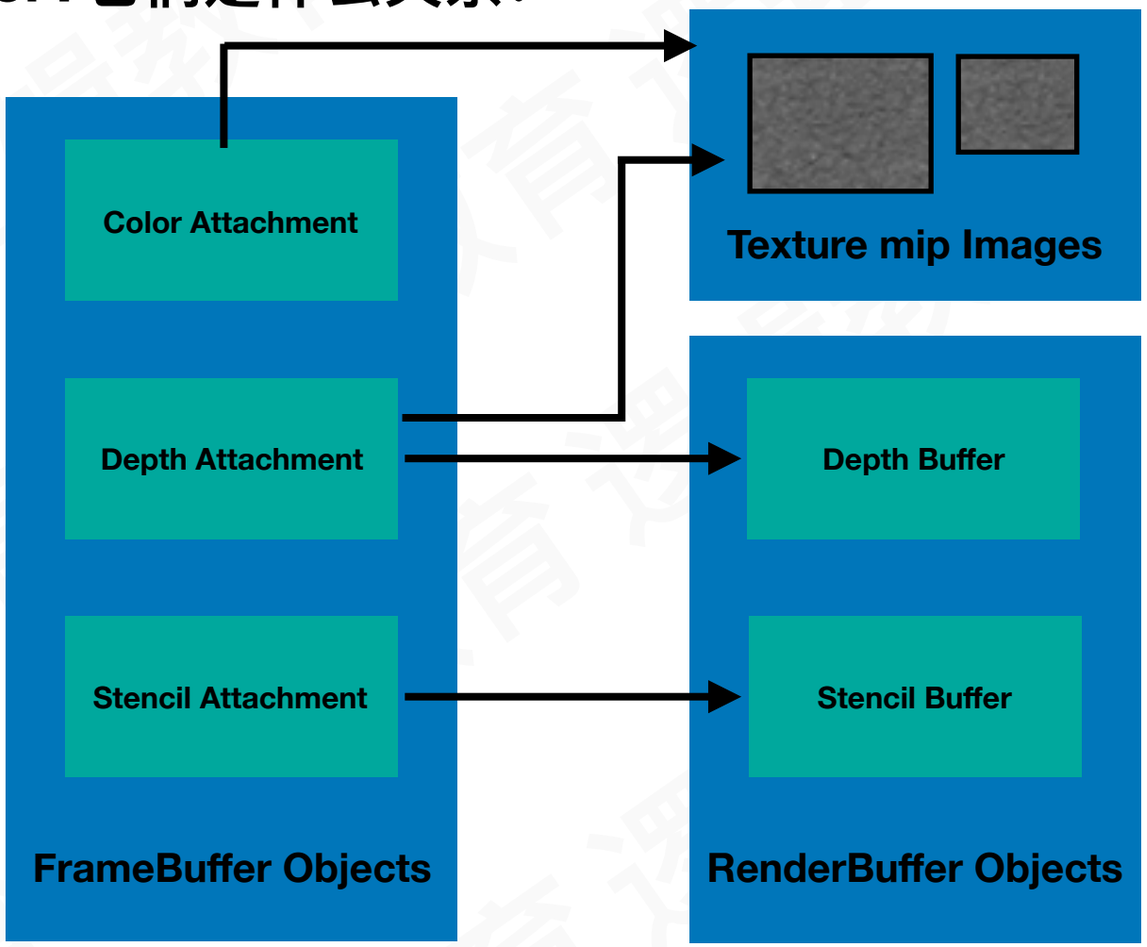
逻辑教育
Logic education

**思考题:**

# 为什么要用FrameBuffer 和 RenderBuffer?它们是什么关系?

A renderbuffer object is a 2D image buffer allocated by the application. The renderbuffer can be used to allocate and store color, depth, or stencil values and can be used as a color, depth, or stencil attachment in a framebuffer object. A renderbuffer is similar to an off–screen window system provided drawable surface, such as a pbuffer. A renderbuffer, however, cannot be directly used as a GL texture.

一个renderbuffer 对象是通过应用分配的一个2D图像缓存区。renderbuffer 能够被用来分配和存储颜色、深度或者模板值。也能够在一个framebuffer被用作颜色、深度、模板的附件。一个renderbuffer是一个类似于屏幕窗口系统提供可绘制的表面。比如pBuffer。一个renderbuffer,然后它并不能直接的使用像一个GL 纹理。



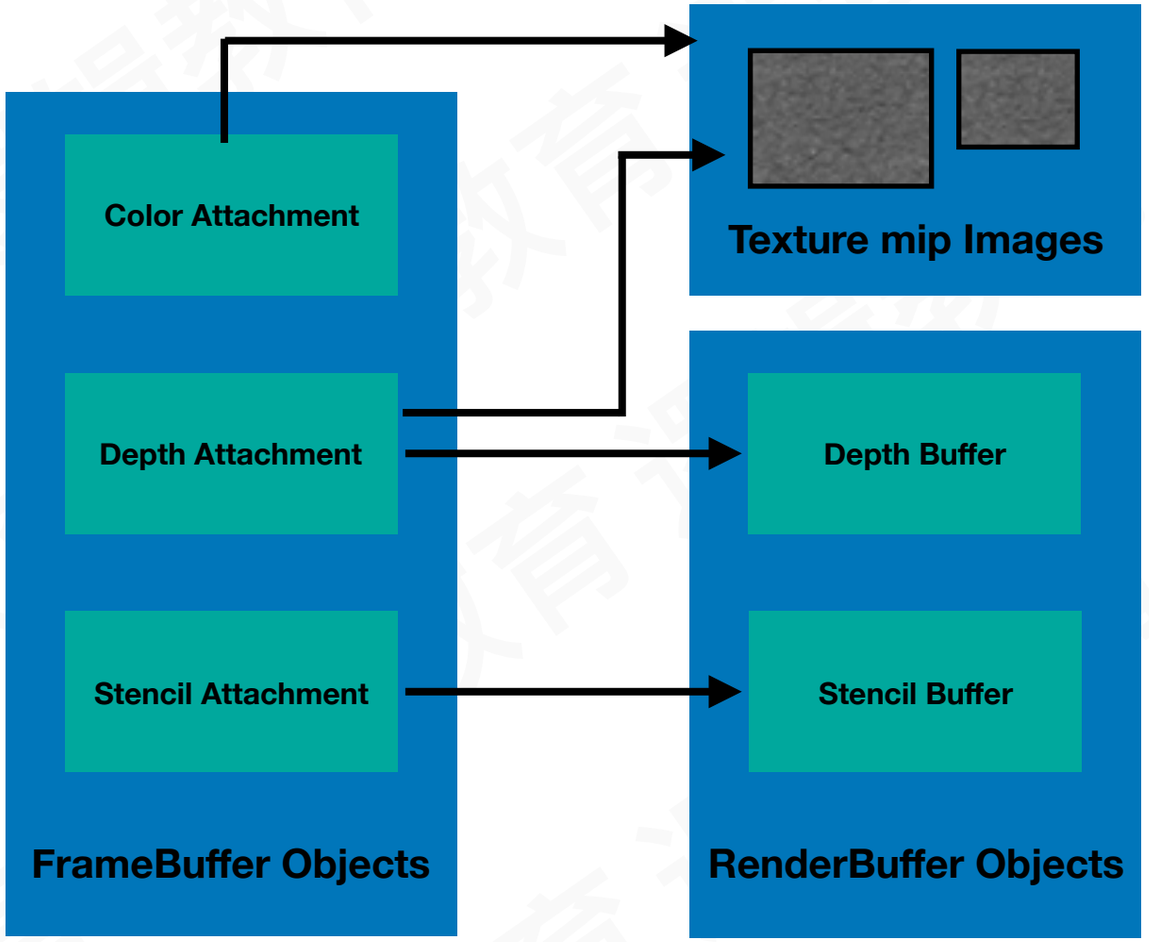

**FrameBuffer Objects,RenderBuffer Objects and Textures**

课程研发:CC老师
课程授课:CC老师

# 为什么要用FrameBuffer 和 RenderBuffer?它们是什么关系?

A framebuffer object (often referred to as an FBO) is a collection of color, depth, and stencil buffer attachment points; state that describes properties such as the size and format of the color, depth, and stencil buffers attached to the FBO; and the names of the texture and renderbuffer objects attached to the FBO. Various 2D images can be attached to the color attachment point in the framebuffer object. These include a renderbuffer object that stores color values, a mip–level of a 2D texture or a cube map face, or even a mip–level of a 2D slice in a 3D texture. Similarly, various 2D images contain– ing depth values can be attached to the depth attachment point of an FBO. These can include a renderbuffer, a mip–level of a 2D texture or a cubemap face that stores depth values. The only 2D image that can be attached to the stencil attachment point of an FBO is a renderbuffer object that stores stencil values.

一个 frameBuffer 对象(通常被称为一个FBO)。是一个收集颜色、深度和模板缓存区的附着点。描述属性的状态，例如颜色、深度和模板缓存区的大小和格式，都关联到FBO(Frame Buffer Object)。并且纹理的名字和renderBuffer 对象也都是关联于FBO。各种各样的2D图形能够被附着framebuffer对象的颜色附着点。它们包含了renderbuffer对象存储的颜色值、一个2D纹理或立方体贴图。或者一个mip–level的二维切面在3D纹理。同样，各种各样的2D图形包含了当时的深度值可以附加到一个FBO的深度附着点钟去。唯一的二维图像，能够附着在FBO的模板附着点，是一个renderbuffer对象存储模板值。



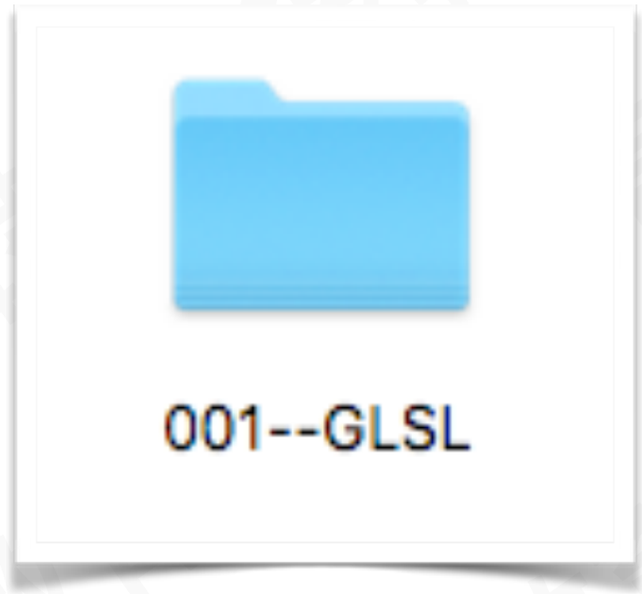

**FrameBuffer Objects,RenderBuffer Objects and Textures**

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 案例实战

- 使用OpenGL  ES GLSL 渲染一张图片到屏幕.




课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

# 思考题:

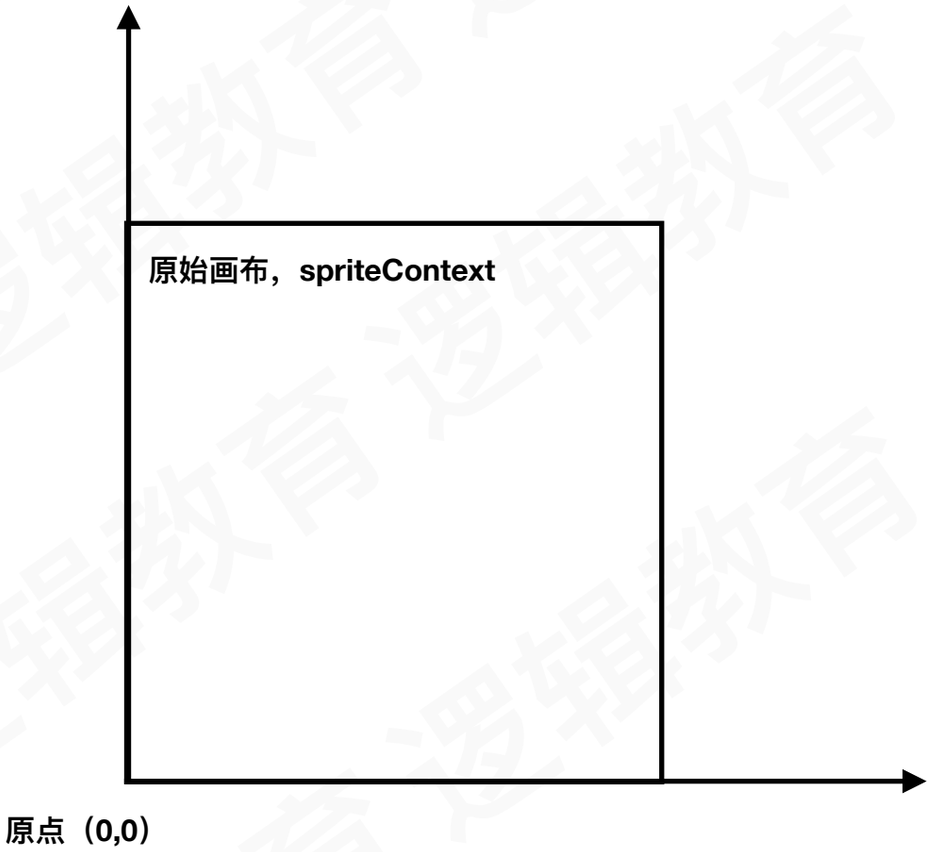
1、课堂案例中 001-GLSL 实现的渲染图片是倒置的，如果解决?

课程研发:CC老师
课程授课:CC老师

# 设备空间转用户空间

## 1、创建原始画布spriteContext

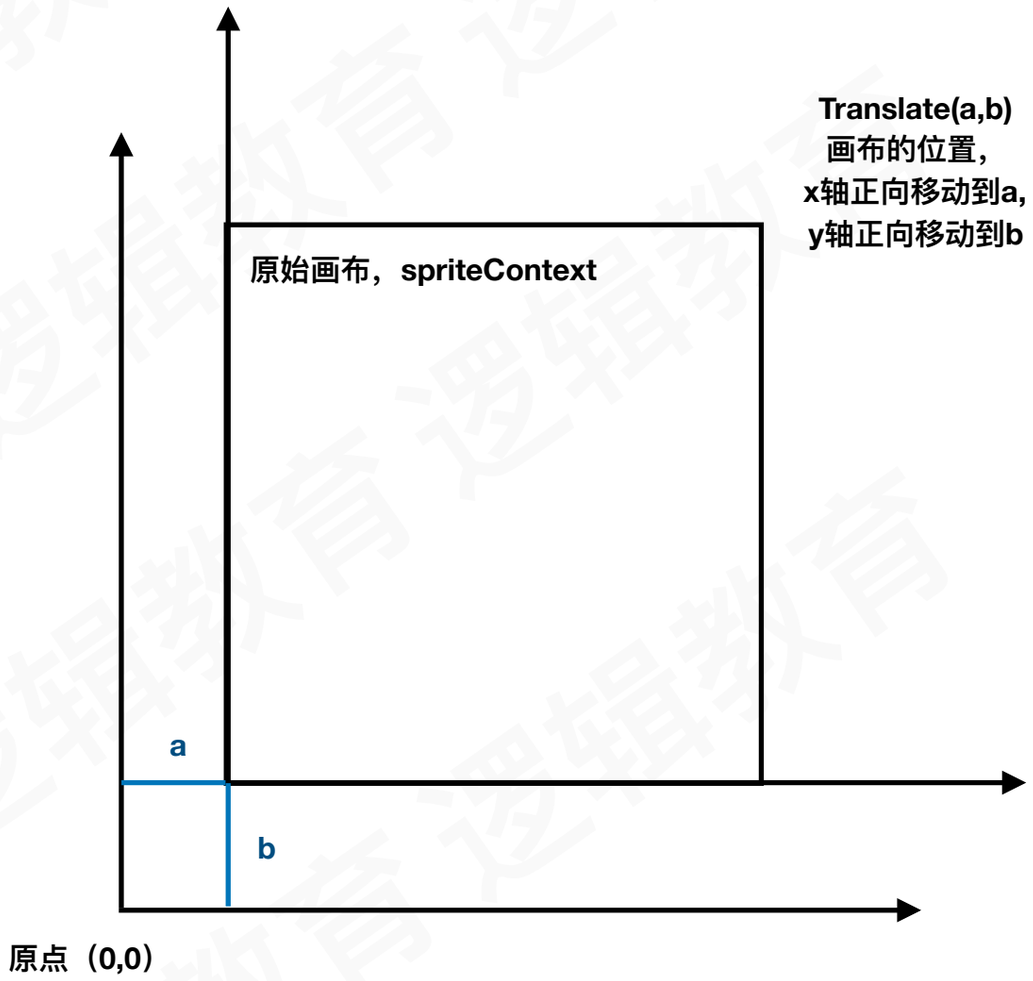
```
367     //4.创建上下文
368     /*
369      参数1: data,指向要渲染的绘制图像的内存地址
370      参数2: width,bitmap的宽度，单位为像素
371      参数3: height,bitmap的高度，单位为像素
372      参数4: bitPerComponent,内存中像素的每个组件的位数，比如32位RGBA，就设置为8
373      参数5: bytesPerRow,bitmap的没一行的内存所占的比特数
374      参数6: colorSpace,bitmap上使用的颜色空间  kCGImageAlphaPremultipliedLast: RGBA
375      */
376     CGContextRef spriteContext = CGBitmapContextCreate(spriteData, width, height,
            8, width*4,CGImageGetColorSpace(spriteImage),
            kCGImageAlphaPremultipliedLast);
377
```



## 2、平移画布CGContextTranslateCTM(spriteContext, rect.origin.x, rect.origin.y)

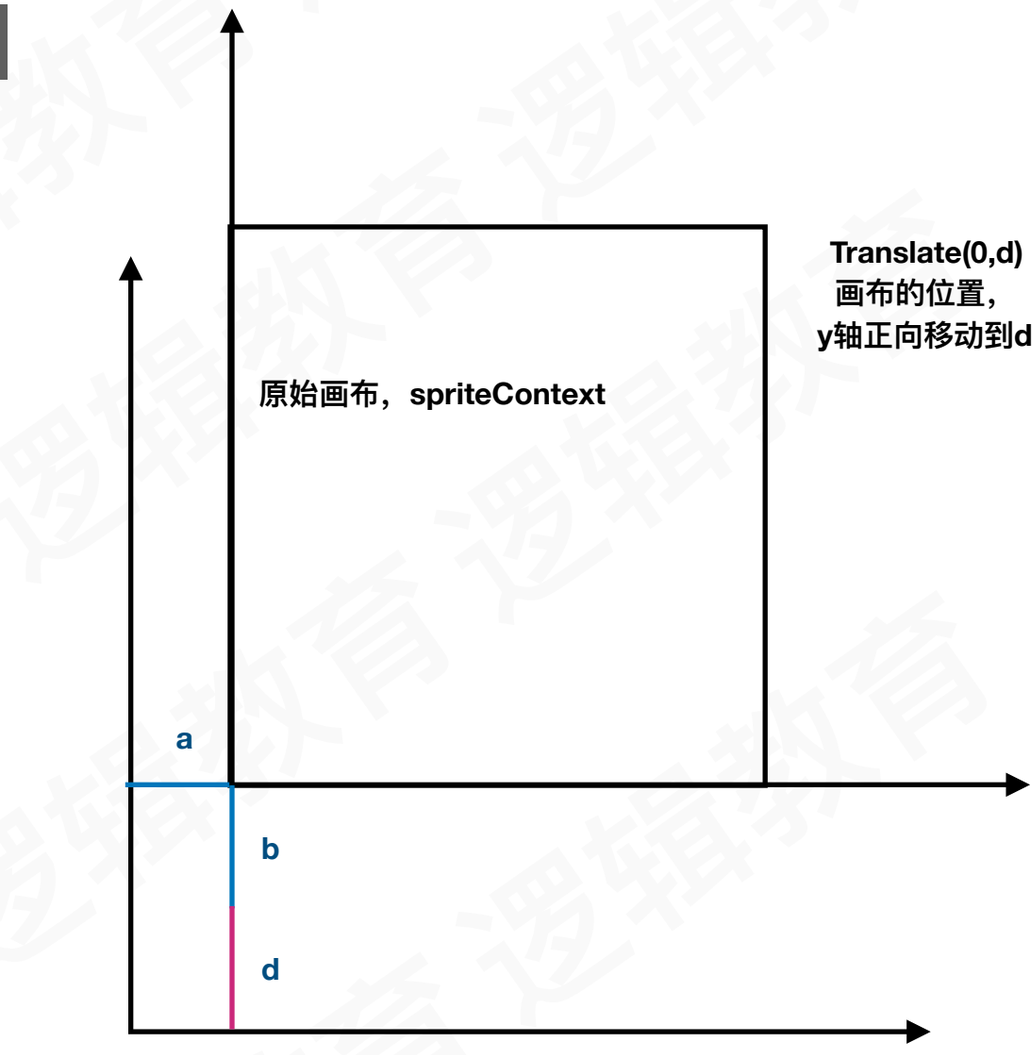
```
        spriteImage);
    /*
     解决图片倒置的方法(2):

     */
    CGContextTranslateCTM(spriteContext, rect.origin.x, rect.origin.y);
    CGContextTranslateCTM(spriteContext, 0, rect.size.height);
    CGContextScaleCTM(spriteContext, 1.0, -1.0);
    CGContextTranslateCTM(spriteContext, -rect.origin.x, -rect.origin.y);
    CGContextDrawImage(spriteContext, rect, spriteImage);
```



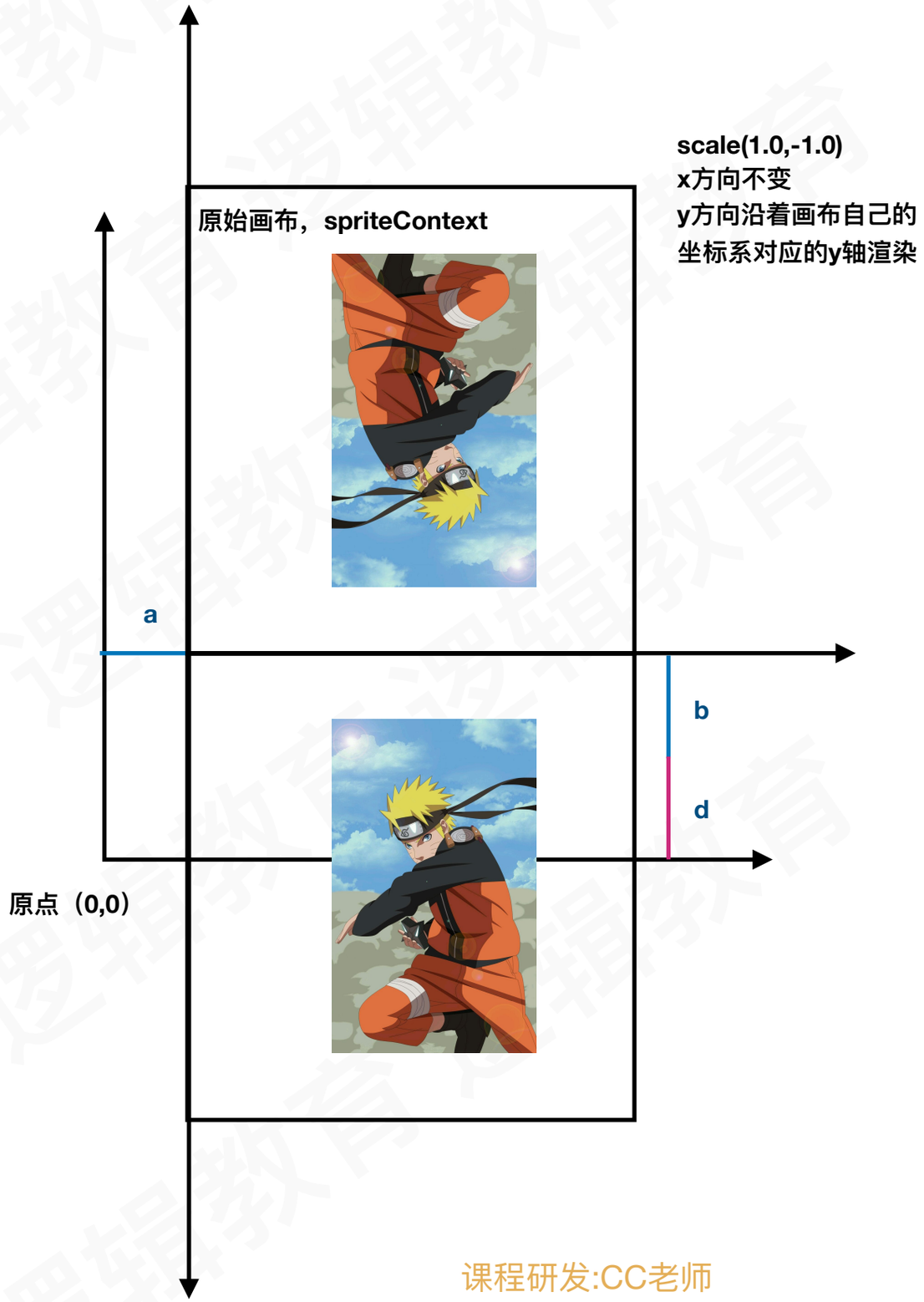
## 3、平移画布CGContextTranslateCTM(spriteContext,0, rect.size.height);

```
391     /*
392      解决图片倒置的方法(2):
393
394      */
395     CGContextTranslateCTM(spriteContext, rect.origin.x, rect.origin.y);
396     CGContextTranslateCTM(spriteContext, 0, rect.size.height);
397     CGContextScaleCTM(spriteContext, 1.0, -1.0);
398     CGContextTranslateCTM(spriteContext, -rect.origin.x, -rect.origin.y);
399     CGContextDrawImage(spriteContext, rect, spriteImage);
400
```



## 4、翻转画布CGContextScaleCTM(spriteContext,1.0, -1.0);

```
391     /*
392      解决图片倒置的方法(2):
393
394      */
395     CGContextTranslateCTM(spriteContext, rect.origin.x, rect.origin.y);
396     CGContextTranslateCTM(spriteContext, 0, rect.size.height);
397     CGContextScaleCTM(spriteContext, 1.0, -1.0);
398     CGContextTranslateCTM(spriteContext, -rect.origin.x, -rect.origin.y);
399     CGContextDrawImage(spriteContext, rect, spriteImage);
400
401
402     CGContextRelease(spriteContext);
403
```

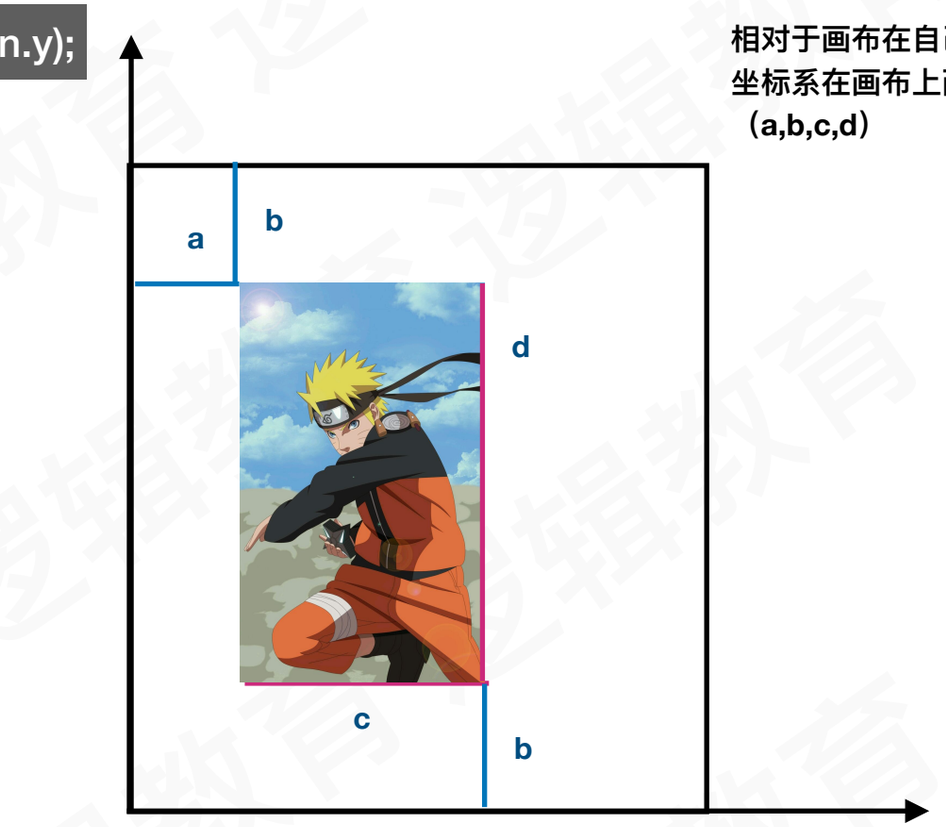


## 4、平移画布：CGContextTranslateCTM(spriteContext, -rect.origin.x, -rect.origin.y);

```
    spriteImage);
/*
 解决图片倒置的方法(2):

 */
CGContextTranslateCTM(spriteContext, rect.origin.x, rect.origin.y);
CGContextTranslateCTM(spriteContext, 0, rect.size.height);
CGContextScaleCTM(spriteContext, 1.0, -1.0);
CGContextTranslateCTM(spriteContext, -rect.origin.x, -rect.origin.y);
CGContextDrawImage(spriteContext, rect, spriteImage);
```

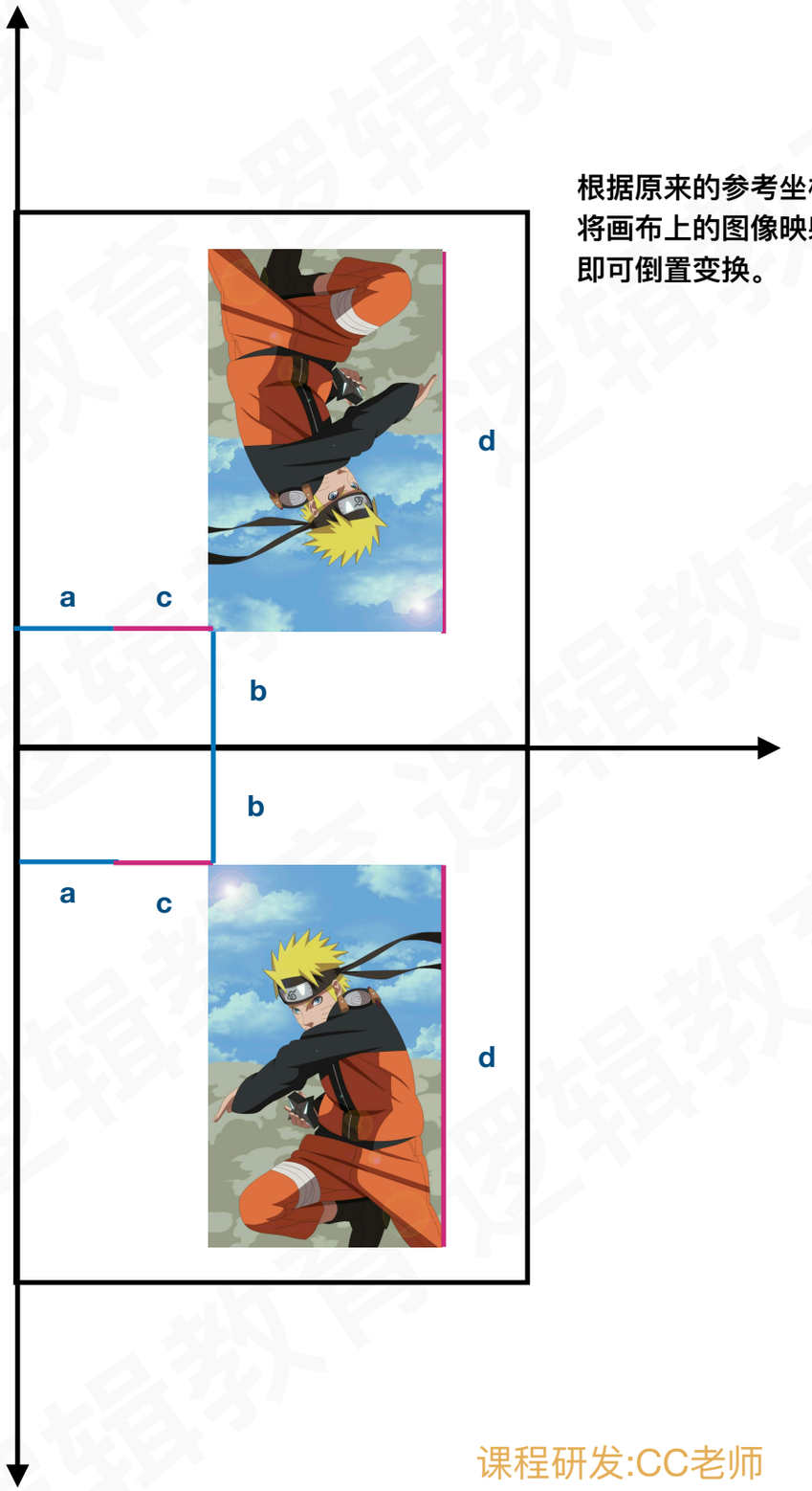
相对于画布在自己的坐标系在画布上画图（a,b,c,d）




原点（0,0）

逻辑教育
Logic education

## 5、倒置变换




根据原来的参考坐标系，
将画布上的图像映射到设备上
即可倒置变换。

逻辑教育
Logic education

**思考题:**

**案例中，顶点着色器调用次数与片元着色器调用次数与什么有关？谁比较多?**

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic ed
答案

片元着色器次数比较多!
顶点着色器调用次数与顶点数量相关,
片元着色器调用与像素多少相关

课程研发:CC老师
课程授课:CC老师