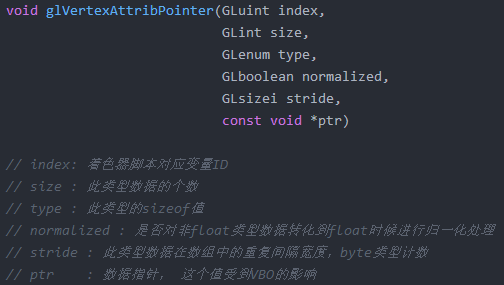
<https://blog.csdn.net/xiajun07061225/article/details/7628146>

<https://blog.csdn.net/ylbs110/article/details/52074826>

首先看看glEnableVertexAtrribArray的作用及调用位置问题

对于OpenGL ES3.0，可以使用至少16个顶点属性，默认情况下，出于性能考虑，所有顶点着色器的属性（Attribute）变量都是关闭的，意味着数据在着色器端是不可见的，哪怕数据已经上传到GPU，由glEnableVertexAttribArray启动指定属性，才可在顶点着色器中方为逐顶点的属性数据。glVertexAttribPointer或者VBO只是建立CPU和GPU之间的逻辑连接，从而实现了CPU数据上传至GPU，但是，数据在GPU中是否可见，即着色器能否读取到数据，由是否启用了对应的属性决定，这就是glEnableVertexAttribArray的功能，允许顶点着色器读取GPU（服务器端）数据。

关于glVertexAttribPointer



简单来说，它和glVertexPointer是同一个东西，只是如果用了glVertexAttribPointer，那么我们就不需要死板地只制定vertices为顶点位置，而是可以创建一个包含了所有顶点位置+顶点颜色+顶点法线+顶点XX的数组，通过一次将其送入显存，然后调用glVertexAttribPointer索引其属性就可以实现了，比如，有一个attrib[]数组，前3位位位置，后面一次为颜色、法线，这样就有3个index，依次为0，1,2，在调用glVertexAttribPointer(0,3,GL\_FLOAT,GL\_FALSE,0,0)；就表示指定了buffer中第一个索引为pos，然后用颜色和法线以此类推

通常我们可以通过glBindAttribLocation绑定一个自定义的属性名字到一个index

**特别注意：该glBindAttribLocation的调用必须在link之前（已经验证）**

glProgram->init("sample.vert", "sample.frag");

glProgram->bindAttrib(1, "a\_position");

glProgram->link();

在link program时，opengl es 2.0对每个顶点属性执行如下操作：

首先检查属性变量是否被通过glBindAttribLocation绑定了属性，如果是，则使用此绑定的属性索引；否则，位置分配一个属性索引。在应用程序中，一般使用函数glBindAttribLocation来绑定每个attribute变量的位置，然后用函数glVertexAttribPointer为每个attribute变量赋值。

**如果在某个着色器中没有使用变量，该变量就会被编译器舍弃掉**

**在查询一个Uniform变量时候，必须在link之后（已经验证）**：

glProgram->init("sample.vert", "sample.frag");

glProgram->**bindAttrib**(1, "a\_position");

cout << "1 index = " << glProgram->getUniformLocation("out\_color") << endl;

glProgram->**link**();

cout << "2 index = " << glProgram->getUniformLocation("out\_color") << endl;



**在给Uniform变量赋值的时候，必须在use之后，因为它是在当前激活的着色器中设置的uniform（已经验证）：**

glProgram->use();

glProgram->setUniformValue4f("out\_color", 0.0f, 1.0f, 0.0f, 1.0f);

**顶点着色器：**

in vec4 a\_position;

in vec4 a\_color;

in vec2 a\_textureCoord;

out vec4 out\_color;

out vec2 out\_textureCoord;

void main(void)

{

gl\_Position = gl\_ModelViewProjectionMatrix \* a\_position;

out\_color = a\_color;

out\_textureCoord = a\_textureCoord;

}

**片元着色器：**

in vec4 out\_color;

in vec2 out\_textureCoord;

**uniform sampler2D our\_texture;**

void main(void)

{

gl\_FragColor = out\_color \* **texture2D**(our\_texture, out\_textureCoord);

}



注意：绑定纹理之后，它会自动地将纹理赋值给片元着色器的采用器，如果我们始终没有调用texture2D函数，会发现获取不到a\_textureCoord属性的索引值