OPENGL 中的vertex buffer是gpu中的buffer

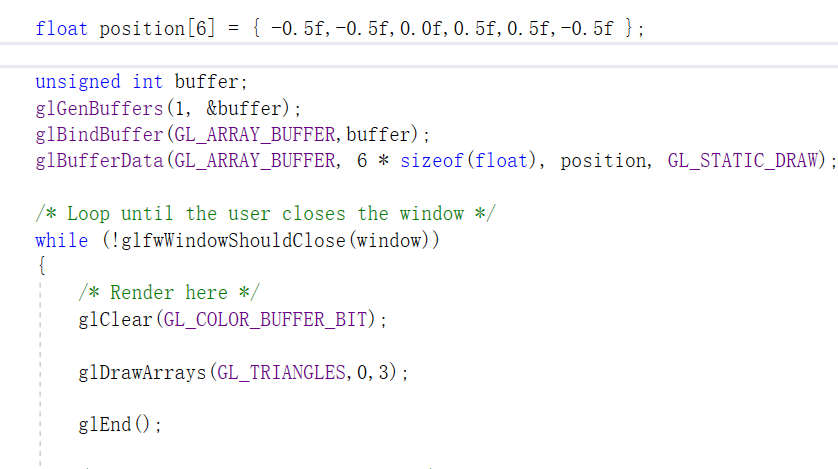
目的就是存储我们想要描述的数据然后告诉gpu这些就是我要的数据，你来显示他们。

然后GPU说，OK，我来阅读他们然后展示

总的来说就是collect data， then draw them

OPENGL 是一个state machine,

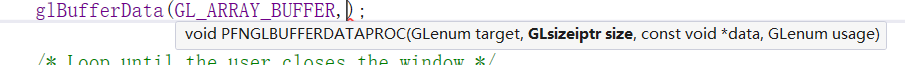
我们要做的是选择一个buffer,选择一个shader，然后告诉opengl draw a triangle



1代表着我们要生成一个buffer

&buffer代表着buffer所在的地址也就是Pointer，这个Pointer指着这个buffer，而buffer是整数只是因为opengl是一个state machine，所以自然就是state1 2 3 ….他是这个buffer 的ID，我们给每一个generate的东西一个ID， 0就是Invalid ，比如buffer1 buffer2….

glBindBuffer(GL\_ARRAY\_BUFFER,buffer); 就是把我们生成的buffer bind到buffer array上。然后这个buffer就work了。



SIZE就是我想让这个buffer多大

然而这个Position本质上是6个float array，opengl不知道如何切割，也许不是三角形，也许是4D等等

我们需要一个IndexBuffer,第一个GL\_TRIANGLES指的是我们要画三角形，0指的是我们从array里的index0开始统计，也就是-0.5f of position，然后3就是vertice的数量

然而这些实际上只是position，一个vertex需要包含的东西远不止position,

Vertex buffer是在gpu的memory存数据，这些数据关于我们要画的东西，shader是一个executable program在gpu上运行，读取数据并且display在屏幕上，所以shader实际上是在读vertex buffer的数据，但这些其实只是一组byte，我们需要告诉opengl，前12byte是float，用来存储position，后来的是其他信息，颜色啥的。 这就是vertex attribute的作用，tell shader我的数据具体意思

Vertex与Position的区别，vertices通常包含了position信息，但包含的更多data

例如 position坐标

Texture coordinate纹理坐标

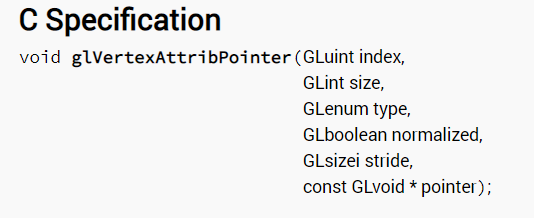
Normal



这一步在你描述完buffer想要的data之前没必要写

我们用

glVertexAttriPointer 来描述vertex attribute



第一个是index,指定我们所要描写的属性所在的Index

例子：我们有三个属性position, texture coordinate, normal，第一个在index0,第二个在index1,第三个在index2,这样shader在read bufffer的时候，看见是哪个Index,就分辨了是什么属性

第二个是size，值只能是1,/2/3/4 他不是真实的byte或我要占多少memory，它是与size连接起来的



以这个为例，我们per vertex要用两个float作为 position，这里size就是2

第三个就是type ，这里就是gl\_float

第四个基本设置成gl\_false

第五个stride，也就是the size of each vertex，例如3D 坐标，两个texture coordinate，1个normal，那么他们占得总容量也就是32BYTE,这个属性用来把每个vertex切割

第六个 pointer,第一个vertex的切割 ，描述position, offset就是0，因为第一个就是position

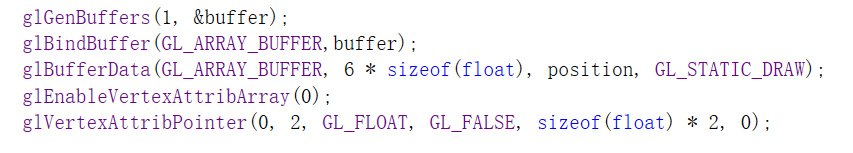
对于texture coordinate，这个值就是12，对于normal,这个值就是20，他不是index，不然这个值就是0,11,19，他是距离beginning的bytes，



这并不是最后一步，我们必须要应用这个vertexattribpointer，也就是enable

只要actural buffer is bound,我们就可以用enable，所以在定义pointer之前还是之后写都行，最好写在之前





0就是enable

Shader章节

基本上就是a program runs on your gpu

Program就是 a block of code

在gpu而不是cpu上运行的原因就是为了告诉显卡what to do, what to display on the screen

Shader有很多种

最基本的就是vertex shader 和fragment shader

一个最粗略的Pipeline:

我们在CPU写了一大堆data，并像GPU传输一些draw code以及与他们绑定的states,最后我们到了shader这一步，GPU开始真正处理这些数据并开始生成图片

Shader

先vertex shader再fragment shader

Vertex shader，我们每试图去render一个vertex就会去call vertex shader。我们这里有三个vertices，也就需要call 3 times

Vertex shader基本目标：告诉opengl 你想让你的vertex出现在screen里的哪个地方， 然后也把其他信息颜色啥的传递给next stage.

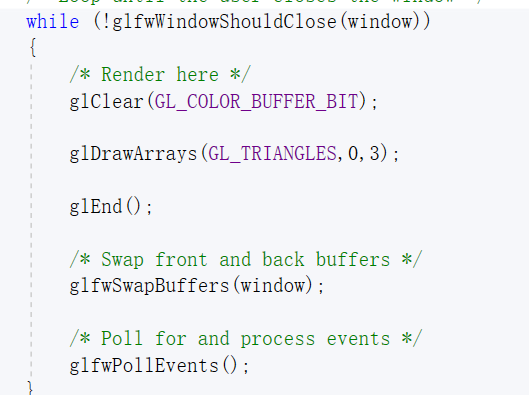
作用就是你移动camera，也改变的对应的screen坐标

只要所有vertex都被定位好，我们就进入fragment shader

Fragment shader, 每当我们试图去rasterize一个Pixel，就call fragment shader一次

Rasterize：光栅化，也就是给每个pixel上色，上透明度啥的，换句话说，也就是给vertices填满,一个Pixel一个Pixel的填，每一个call一次fragment shader。

他的基本目标：决定这个Pixel什么颜色

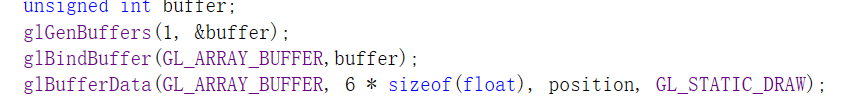


这一段什么意思

第一行，只要window没关掉就会一直循环，

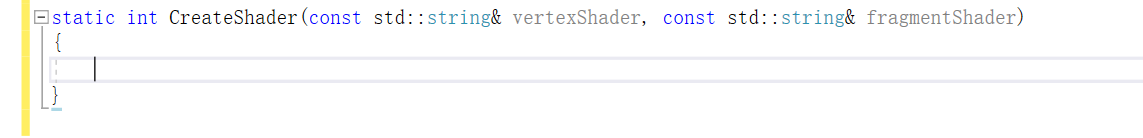
第二行，glClear， clear the screen

第三行glDrawArrays，draw the currently bound buffer

就是这些

后面两个暂时不用管

基本上就是clearing and draw

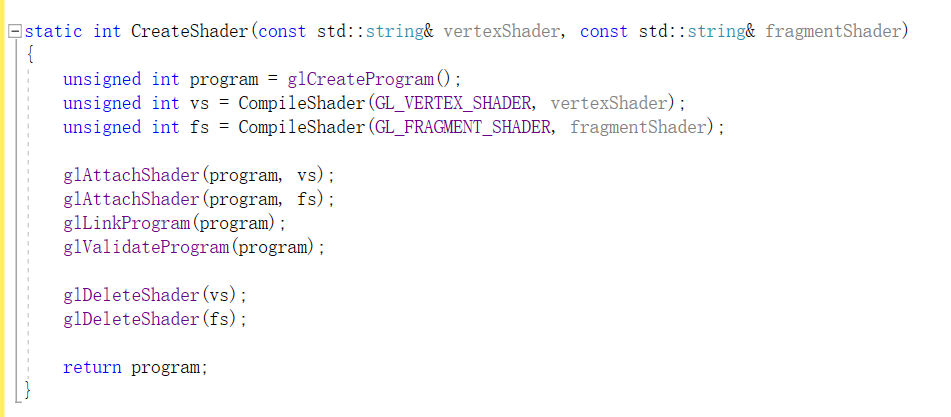


这个function功能：

1. 给opengl提供我们实际shader的源代码
2. 把vertexShader 与fragmentShader绑定在一个shader program
3. 给我们一个ID 标识来bind，所以return int

有点像genBuffer，会get my id back,（1是我们要生产成一个buffer p）

具体代码



第一个参数，用string vertexShader 来reference 我们实际的vertexShader Soucre Code,

第二个参数同理。

第一行，openGL惯例,创造一个空program object，并且return一个可以作为reference的int

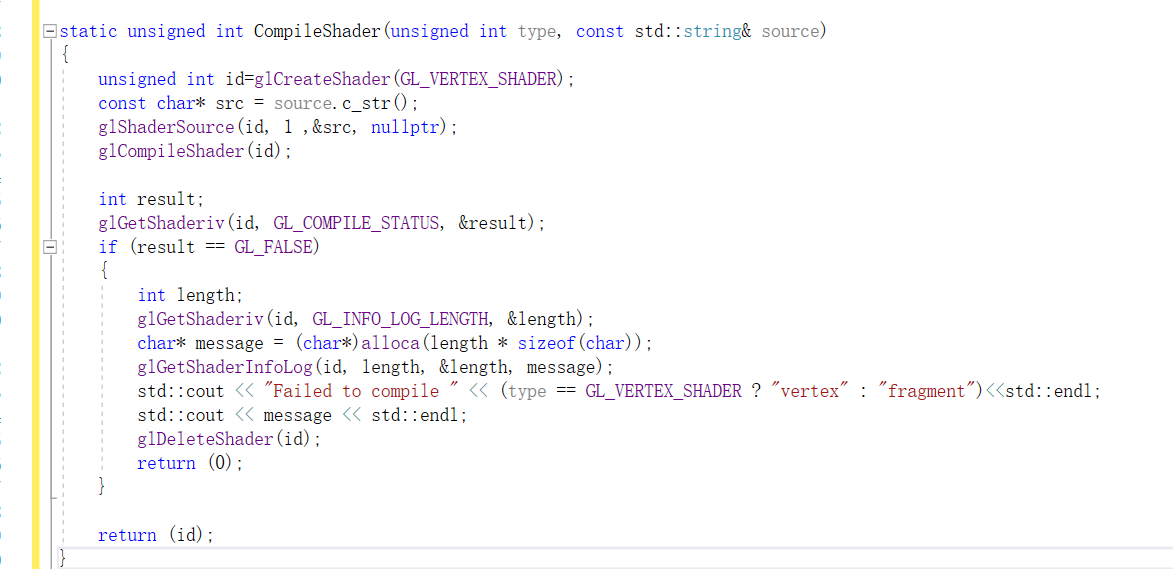
第二行创造一个vertex shader，并且用这个shader compile我们的vertexShader(一个referencce，指向我们实际的要被compile的内容) ，然后return一个id来作为reference，我们用vs 来指代我们的shader

第三行同理

四五行没啥好说的，把我们的shader连接到program上，

Link validate就是字面意思，以后也不用管，

Compile shader具体代码



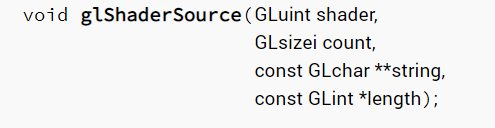
第一个参数type :GL\_VERTEX\_SHADER或者GL\_FRAGMENT\_SHADER,实际上都是一个整数,第二个是一个reference，对应fragmentshader or vertexShader//而fragmentShader或vertexShader实际上也是对shader 数据的reference。 换句话说source就是源代码

第一行glCreateShader，创造一个指定类型的空shader object，并return一个reference来指代他

第二行，我们的源代码实际上是一串不可改变的array，

这一行你也可以写成，创造一个Pointer src，指向源代码的开头第一个char

第三行,

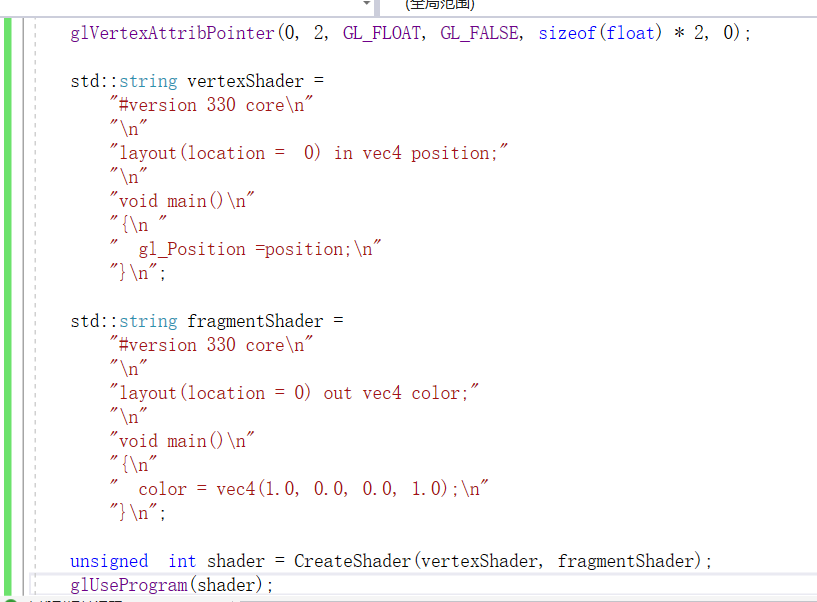
第一个参数就是id，注意这时id所指带的shader是完全空的，第二个参数是1，意思是我们指定多少source code，我们这里只要一段source code，所以是1，第三个参数是是一个char，意思是我们内容的第一个字母，他是一个双重指针，第一重指向原source code，第二层指向第一层第一个字母，第四个参数可以默认填nullPointer

这一行的目的是把原来ID内的shader中的内容完全由STRING（我们想要的新source code）替换掉，而char就指向了我们的string

第四行glCompileShader，compile指定id的shader

再然后的就是报错系统

创建



Version 330就是我们所用shader版本

把glVertexAttribPointer也就是location0开始的vertex转化成vec4 4D OPENGL标准形式

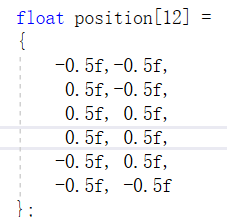
Position就是Normal position

Fragment是上色过程， 所以改成out ccolor

Index buffer

Opengl里所有东西都是由三角形构成的

例如正方形，就可以由两个三角形构成

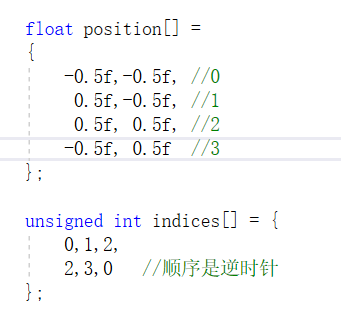




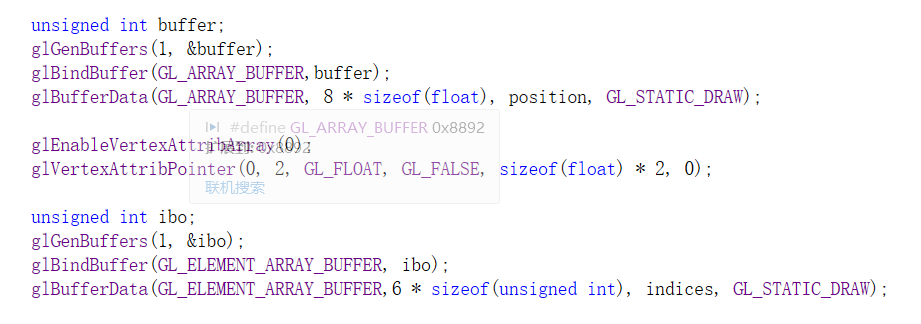


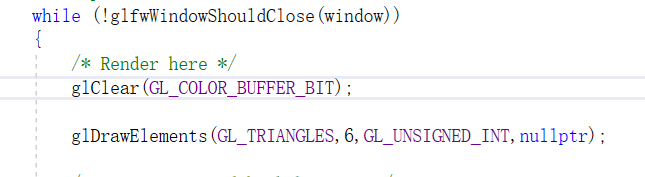
但实际上我们有两个vertex是重复的，画一个正方形可以接受，但如果画一个具体模型，具体大型项目，会造成大量浪费

所以我们要加一个index buffer



Indices就是我们的index buffer ，





和buffer大同小异

最后一个Drawelements

6是我们的indices数量，

第三个参数是我们INDICE的数据类型，一般是gl\_uninsigned\_int

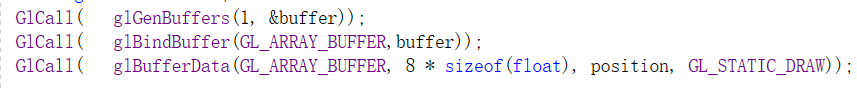
最后一个是nullptr

GlGeterror:会告诉你error种类

如果没检测到error,会return GL\_NO\_ERROR,GL\_NO\_ERROR的实际值是0

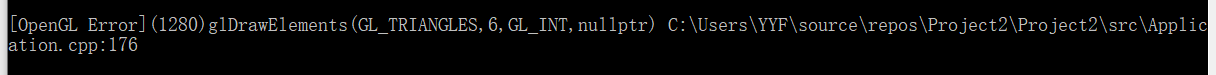
这里实际是叫你写自己的一个DEBUGGER,我们只管写就完事儿了

然后在可能出现bug的地方加入debugger(实际上最好每一个function都加一个，没多麻烦)





怎么多行一起加glcall alt shift 左键

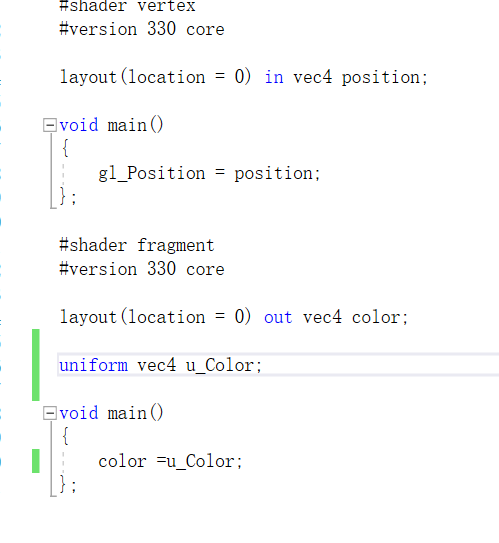


然后会告诉你ERROR，1280是error信息， 下面一行是具体位置

主要用了macros

Uniform是一个概念：你可以从 CPU SIDE OF THINGS 例如C++得到东西并放进shader

为什么：因为shader颜色很难写

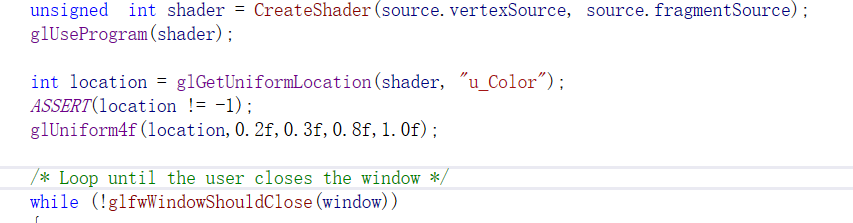


首先修改我们的shader

我们要在actual drawing之前就写Uniform



同时要在我们的shader被bound之后写Uniform



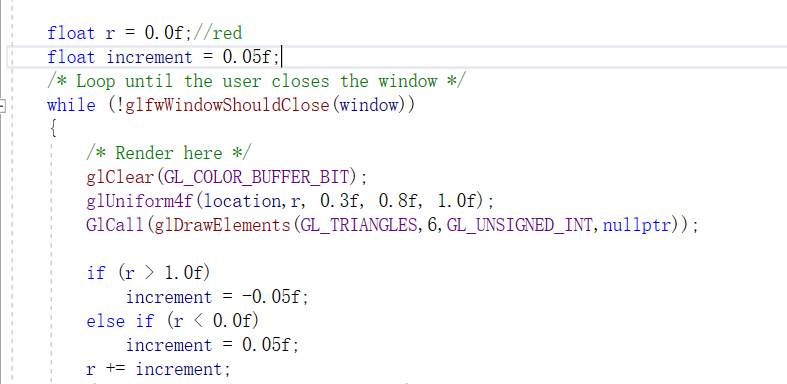
所以最好在shader创建之后写

GlUniform4f, 4f代表4float，意思是我们这里实际代替shader的就4个float

这个function用来描述uniform variable的值， 我们用location来指代我们具体描述哪个uniform variable

而这个Location是由glGetUniformLocation return的

接下来的就是实际应用,来展示Uniform的方便性



可以看到我们能用r来代替具体颜色

然后再画了一次以后我们还能改变r,因为是while循环，会想弱智一样乱闪，r一直在+-0.5

他为啥叫uniform，因为他只能描述一种，他在GLDRAW之前，不可能用这个颜色画完一个三角形，又用另一个颜色画另一个三角形。