OpenGL状态量

1. Buffer

glBindBuffer(GL\_ARRAY\_BUFFER,VBO);

1. ShaderProgram

glUseProgram(shaderProgram);

1. VertexAttributeArray

glEnableVertexAttributeArray(0);

1. VertexArray

glBindVertexArray(VAO);

1. Buffer操作

buffer是在内存中已有绘制相关的数据，将这些数据拷贝到gpu存储(显存)中，之后的渲染流程中直接从显存中取数据，而不是再通过从内存中取数据(较慢)。

1. 创建buffer

unsigned int VBO;

glGenBuffers(1,&VBO);

通过带入取值的方式获得buffer id。

1. 操作buffer

glBindBuffer(GL\_ARRAY\_BUFFER,VBO);

1. 指定buffer数据

glBufferData(GL\_ARRAY\_BUFFER,size\_byte,content\_array,draw\_mode);

1. Shader
2. 创建shader

unsigned int vertexShader;

vertexShader=glCreateShader(GL\_VERTEX\_SHADER);

1. 指定shader内容及编译

glShaderSource(vertexShader,1,&vertexShaderSource,NULL);

glCompileShader(vertexShader);

1. 获取编译错误

int success;

char infoLog[512];

glGetShaderiv(vertexShader,GL\_COMPILE\_STATUS,&sucess);

if(!sucess)

{

glGetShaderInfoLog(vertexShader,512,NULL,infoLog);

}

1. shader program

//创建

unsigned int shaderProgram;

shaderProgram=glCreateProgram();

//链接

glAttachShader(shaderProgram,vertexShader);

glAttachShader(shaderProgram,fragmentShader);

glLinkProgram(shaderProgram);

//获取错误

glGetProgramiv(shaderProgram,GL\_LINK\_STATUS,&sucess);

if(!sucess)

{

glGetProgramInfoLog(shaderProgram,512,NULL,infoLog);

}

1. 使用shader

glUseProgram(shaderProgram);

1. 删除shader

glDeleteShader(vertexShader);

glDeleteShader(fragmentShader);

1. Objects
2. VBO(Vertex Buffer Object)

vertex数据的存储，通过Buffer相关操作创建和指定数据内容。

1. VAO(Vertex Array Object)

与VBO的VertexAttribute(指明buffer的内容结构)和EBO进行关联，当有多个物体进行渲染时，只需初始时通过bind VAO配置一次，之后再次需渲染时，切换到相关联的VAO即可。

1. EBO(Element Array Object)

使用原生的VBO存储可能会产生大量的数据冗余，所以在VBO中存储的是内容不相同的vertex,EBO存储的是原数据内容的索引。通过Buffer相关操作创建和指定数据内容。

四、绘画

glDrawArrays(GL\_TRIANGLES,0,3);

glDrawElements(GL\_TRIANGLES,6,GL\_UNSIGNED\_INT,0);

**Advance GLSL**

1. 内置变量
2. **VertexShader**

gl\_Position:输出的NDC坐标的位置

gl\_PointSize:改变点绘制的大小，但需要在opengl程序中启用:

glEnable(GL\_PROGRAM\_POINT\_SIZE);

gl\_VertexID:只读，当采用索引绘制的方式，该值表示索引的值；当采用数组绘制的方式，该值表示正在处理的顶点序号。

1. **FragementShader**

gl\_FragCoord:x,y值表示屏幕空间中的坐标，z值表示depth值。

gl\_FrontFacing:表示当前渲染的图元是物体是否是物体的正面

gl\_FragDepth:由于gl\_FragCoord只是可读的，所以其中的深度值不能进行修改。在FragmentShader中可通过gl\_FragDepth修改深度值，但是这会使得前期深度测试失效。但在gl4.2以后，可通过layout(depth\_[condition]) out float gl\_FragDepth 来使得部分前期深度测试有效。

1. Interface blocks

在shader的流水线中，由前一个shader向之后的shader传递变量的做法目前是采用对应的in、out声明并且类型和变量名一致。但需要传递的变量比较多时，就会显得比较繁琐。可以采用声明统一的接口块进行管理:

在VertexShader中:

out VS\_OUT

{

vec2 TexCoord;

} vs\_out;

在FragmentShader中对应:

in VS\_OUT

{

vec2 TexCoord;

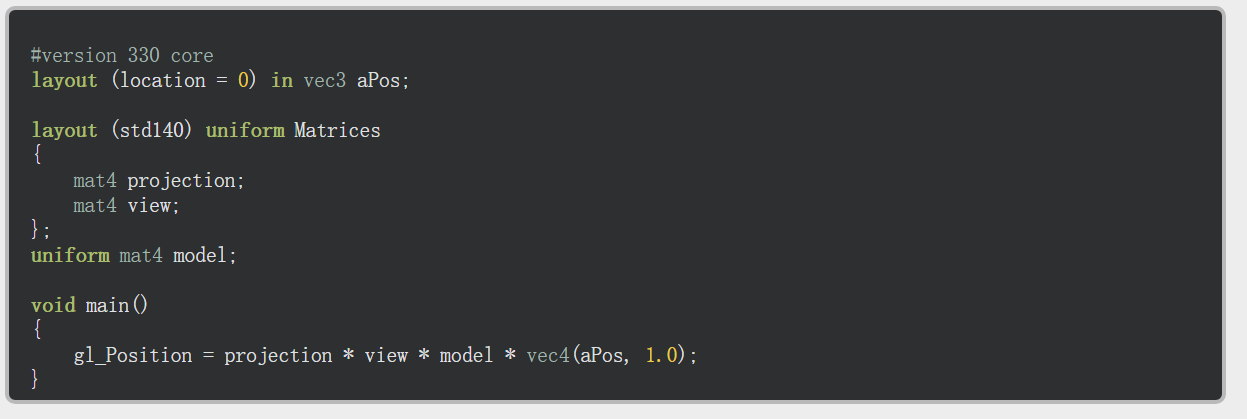
} frag\_in;

1. Uniform buffer objects
2. uniform 块布局

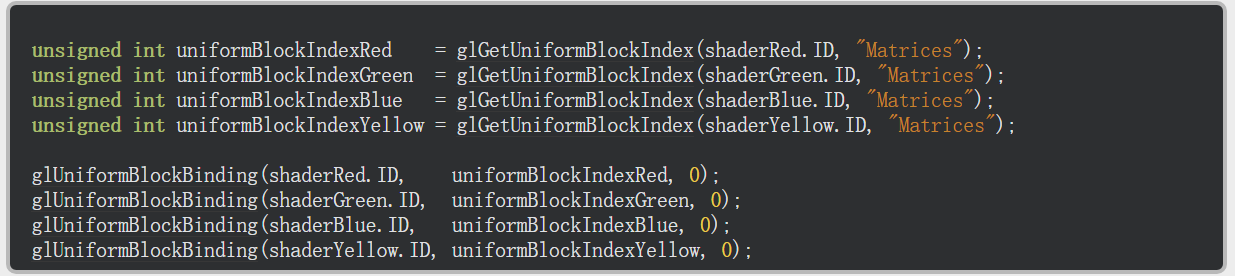
std140、shared、packed

1. 使用unifrom 块

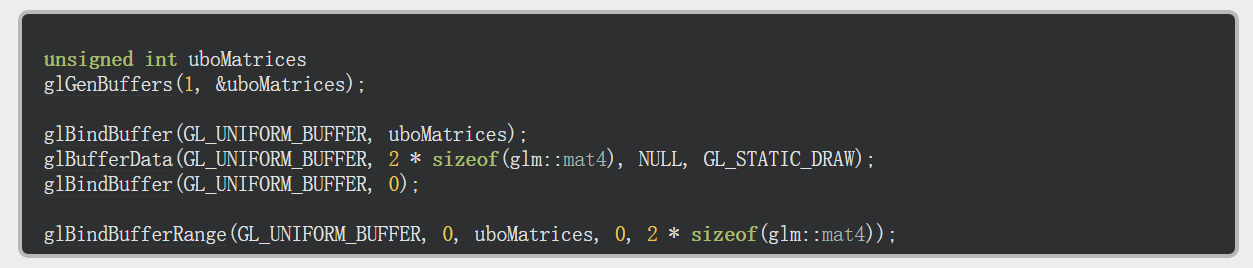
* 在shader中声明uniform块



* 设置绑点



* 创建buffer object



* 改变buffer 数据

