opengl的VAO

VBO在渲染阶段才指定数据位置和顶点信息，然后根据此信息去解析缓冲区里的数据，联系着两者中间的桥梁是GL-Context。GL-Context整个程序一般只有一个，所以如果一个渲染流程里有两份不同的绘制代码，GL-Context就负责在他们之间进行状态切换、这也是为什么在渲染过程中，在每份绘制代码之中有glBindBuffer/glEnableVertexAttribArray/glVertexAttribPointer。那么优化方法就来了---把这些都放在初始化时候完成吧！--这样做的限制条件是“负责记录状态的GL-Context一般只有一个”，那么就不直接用GL-Context记录，用别的东西做记录吧--这个东西针对“每份绘制代码”有一个，记录该次绘制所需要的所有VBO所需信息，把它保存到GPU的特定位置，绘制的时候直接在这个位置取信息绘制。于是，VAO诞生了。

VAO的全名是Vertex Array Object，首先，它不是Buffer-Object，所以不用做存储数据；其次，它针对“顶点”而言，也就是说它跟"顶点的绘制"息息相关。它记录数据的存储和如何使用的细节信息。使用VAO的优势在于，如果有多个物体需要绘制，那么设置一次绘制物体需要的顶点数据、数据解析方式等信息，然后通过VAO保存起来后，后续的操作不再需要重复这一过程，只需要将VAO设定为当前的VAO，那么opengl则会使用这些状态信息，当场景中物体较多时，优势十分明显。

使用VAO的步骤：

1）产生VAO

void glGenVertexArrays(GLsizei n, GLuint \*arrays)

n：要产生的VAO对象的数量

arrays：存放产生的VAO对象的名称

2）绑定VAO

void glBindVertexArray(GLuint array)

array：要绑定的顶点数组的名字

3）产生VBOs

void glGenBuffers(GLsizei n, GLuint \*buffers)

4）绑定VBOs

void glBindBuffer(GLenum target, GLuint buffer)

绑定一个缓冲区对象

target可能取值：GL\_ARRAY\_BUFFER、GL\_ELEMENT\_ARRAY\_BUFFER、GL\_PIXEL\_PACK\_BUFFER、GL\_PIXEL\_UNPACK\_BUFFER

5）给VBO分配数据

void glBufferData(GLenum target, GLsizeiptr size, const GLvoid \*data, GLenum usage)

target可能取值为：GL\_ARRAY\_BUFFER（表示顶点数据）、GL\_ELEMENT\_ARRAY\_BUFFER（表示索引数据）、GL\_PIXEL\_PACK\_BUFFER（表示从opengl获取的像素数据）、GL\_PIXEL\_UNPACK\_BUFFER（表示传递给opengl的像素数据）

size：缓冲区对象的字节数

data：指针，指向用于拷贝到缓冲区对象的数据，或者是NULL，表示暂时不分配数据

6）定义存放顶点属性数据的数组

void glEnableVertexArribArray(GLuint index)

index：指定了需要启动的顶点属性数组的索引

注意：它只在opengl 2.0及以上版本才有

7）给对应的顶点属性数组指定数据

void glVertexAttribPoint(GLuint index, GLint size, GLenum type, GLboolean normalized, GLsizei stride, const GLvoid\* pointer)

index：要指定数据的顶点属性数组的索引。

size：每个顶点属性的数据个数。可能的取值是1、2、3或者4.初始值是4.

type：数组中每个数据的类型。可能的取值是：GL\_BYTE, GL\_UNSIGNED\_BYTE, GL\_SHORT, GL\_UNSIGNED\_SHORT, GL\_INT, GL\_UNSIGNED\_INT, GL\_FLOAT, or GL\_DOUBLE。初始值是GL\_FLOAT。

normalized：指定顶点数在被访问的时候是否需要被归一化。

注意：如果有个非零的缓冲对象绑定到GL\_ARRAY\_BUFFER，那么pointer就是对应的缓冲区对象的偏移量。

 stride：两个连续顶点的属性之间的偏移量。

pointer：指向数组中的第一个顶点属性的第一个数据。

8)渲染时，只需要绑定对应的VAO即可

glBindVertexArray(vaoHandle)

9）使用完毕后清除绑定

glBindVertexArray(0)

1. void initVBO()
2. {
3. // Create and populate the buffer objects
4. GLuint vboHandles[2];
5. glGenBuffers(2, vboHandles);
6. GLuint positionBufferHandle = vboHandles[0];
7. GLuint colorBufferHandle = vboHandles[1];
9. //绑定VBO以供使用
10. glBindBuffer(GL\_ARRAY\_BUFFER,positionBufferHandle);
11. //加载数据到VBO
12. glBufferData(GL\_ARRAY\_BUFFER,9 \* sizeof(float),
13. positionData,GL\_STATIC\_DRAW);
15. //绑定VBO以供使用
16. glBindBuffer(GL\_ARRAY\_BUFFER,colorBufferHandle);
17. //加载数据到VBO
18. glBufferData(GL\_ARRAY\_BUFFER,9 \* sizeof(float),
19. colorData,GL\_STATIC\_DRAW);
21. glGenVertexArrays(1,&vaoHandle);
22. glBindVertexArray(vaoHandle);
24. glEnableVertexAttribArray(0);//顶点坐标
25. glEnableVertexAttribArray(1);//顶点颜色
27. //调用glVertexAttribPointer之前需要进行绑定操作
28. glBindBuffer(GL\_ARRAY\_BUFFER, positionBufferHandle);
29. glVertexAttribPointer( 0, 3, GL\_FLOAT, GL\_FALSE, 0, (GLubyte \*)NULL );
31. glBindBuffer(GL\_ARRAY\_BUFFER, colorBufferHandle);
32. glVertexAttribPointer( 1, 3, GL\_FLOAT, GL\_FALSE, 0, (GLubyte \*)NULL );
33. }
34. //使用VAO、VBO绘制
35. glBindVertexArray(vaoHandle);
36. glDrawArrays(GL\_TRIANGLES,0,3);
37. glBindVertexArray(0);