<http://www.tuicool.com/articles/qEri6nq>

**OpenGL中glVertex、显示列表(glCallList)、顶点数组(Vertex array)、VBO及VAO区别**

**实践证明，不管是哪一种方式都需要调用glSwapBuffer或者glFlush（单缓冲与双缓冲都已证明）**

**1.glVertex**

最原始的设置顶点方法，在glBegin和glEnd之间使用。OpenGL3.0已经废弃此方法。每个glVertex与GPU进行一次通信，十分低效。

gl**Begin**(GL\_TRIANGLES);

glVertex(0, 0);

glVertex(1, 1);

glVertex(2, 2);

glEnd();

**2.显示列表(glCallList)**

每个glVertex调用都与GPU进行一次通信，显示列表是收集好所有的顶点，一次性的发送给GPU。缺点是在绘制之前就要把要传给GPU的顶点准备好，传后就不能修改了。

1 GLuint glassList;

2 glNewList(glassList, GL\_COMPILE);

3 DrawGlass();

4 glEndList();

5

6 glCallList(glassList); *//DrawGlass();*

**3.顶点数组(Vertex Array)**

顶点数组也是收集好所有的顶点，一次性发送给GPU。不过数据不是存储于GPU中的，绘制速度上没有显示列表快，优点是可以修改数据。

显示列表和顶点数组都是过时的东西了， 下面的VBO和VAO才是重点 ！

**#define MEDIUM\_STARS 40**

M3DVector2f vMediumStars[MEDIUM\_STARS];

*//在这做点vMediumStars的设置//*

glVertexPointer(2, GL\_FLOAT, 0, vMediumStars);

glDrawArrays(GL\_POINTS, 0, MEDIUM\_STARS);

**4.VBO（Vertex Buffer Object）顶点缓冲区对象**

VBO，全称为Vertex Buffer Object，与FBO，PBO并称，但它实际上老不少。就某种意义来说，他就是VA（Vertex Array）的升级版。VBO出现的背景是人们发现VA和显示列表还有让人不满足的地方。一般，在OpenGL里，提高顶点绘制的办法：

（1）显示列表：把常规的glBegin()-glEnd()中的代码放到一个显示列表中（通常在初始化阶段完成），然后每遍渲染都调用这个显示列表。

（2）VA：使用顶点数组，把顶点以及顶点属性数据作为数组，渲染的时候直接用一个或几个函数调动这些数组里的数据进行绘制，形式上是减少函数调用的次数（告别glVertex），提高绘制效率。

但是，这两种方法都有缺点。 VA是在客户端设置的，所以执行这类函数（glDrawArray或glDrawElement）后，客户端还得把得到的顶点数据向服务端传输一次（所谓的“二次处理”），这样一来就有了不必要的动作了，降低了效率 ——如果我们写的函数能直接把顶点数据发送给服务端就好了——这正是VBO的特性之一。 显示列表的缺点在于它的古板，一旦设定就不容许修改，所以它只适合对一些“固定”的东西的绘制进行包装 。（我们无办法直接在硬件层改顶点数据，因为这是脱离了流水线的事物）。而VBO直接把顶点数据交到流水线的第一步，与显示列表的效率还是有差距，但它这样就得到了操作数据的弹性——渲染阶段，我们的VBO绘制函数持续把顶点数据交给流水线，在某一刻我们可以把该帧到达了流水线的顶点数据取回客户端修改（Vertex mapping），再提交回流水线（Vertex unmapping），或者用glBufferData或glBufferSubData重新全部或buffer提交修改了的顶点数据，这是VBO的另一个特性。

VBO结合了VA和显示列表这个说法不太妥当，应该说它结合了两者的一些特性，绘制效率在两者之间，且拥有良好的数据更改弹性。这种折衷造就了它一直为目前最高的地位。

*//创建VBO及VBO赋值*

glGenBuffers(1, &m\_nPositionVBO);

glBufferData(GL\_ARRAY\_BUFFER,

sizeof(posData), posData, GL\_STREAM\_DRAW);

glGenBuffers(1, &m\_nTexcoordVBO);

glBufferData(GL\_ARRAY\_BUFFER,

sizeof(texData), texData, GL\_STREAM\_DRAW);

glGenBuffers(1, &m\_nIndexVBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER,

sizeof(indexData), indexData, GL\_STATIC\_DRAW);

*//代码一，不使用shader VBO已经创建好了*

glBindBuffer(GL\_ARRAY\_BUFFER, m\_nPositionVBO);

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(2, GL\_FLOAT, 0, **NULL**);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_nTexcoordVBO);

glEnableClientState(GL\_TEXTURE\_COORD\_ARRAY);

glTexCoordPointer(2, GL\_FLOAT, 0, **NULL**);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, m\_nIndexVBO);

glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_SHORT, **NULL**);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, **NULL**);

glDisableClientState(GL\_TEXTURE\_COORD\_ARRAY);

glDisableClientState(GL\_VERTEX\_ARRAY);

glBindBuffer(GL\_ARRAY\_BUFFER, **NULL**);

*//代码二，使用shader*

glBindBuffer(GL\_ARRAY\_BUFFER, m\_nPositionVBO);

glEnableVertexAttribArray(VAT\_POSITION);

glVertexAttribPointer(VAT\_POSITION, 2, GL\_INT, GL\_FALSE, 0, **NULL**);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_nTexcoordVBO);

glEnableVertexAttribArray(VAT\_TEXCOORD);

glVertexAttribPointer(VAT\_TEXCOORD, 2, GL\_INT, GL\_FALSE, 0, **NULL**);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, m\_nIndexVBO);

glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_SHORT, **NULL**);

glDisableVertexAttribArray(VAT\_POSITION);

glDisableVertexAttribArray(VAT\_TEXCOORD);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, **NULL**);

glBindBuffer(GL\_ARRAY\_BUFFER, **NULL**);

**5.VAO（Vertex Array Object）顶点数组对象**

VBO将顶点信息放到GPU中，GPU在渲染时去缓存中取数据，二者中间的桥梁是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切换。这也是为什么要在渲染过程中，在每份绘制代码之中会有glBindbuffer、glEnableVertexAttribArray、glVertexAttribPointer。那么优化的方法来了，把这些都放到初始化时候完成吧！ VAO记录该次绘制所需要的所有VBO所需信息，把它保存到VBO特定位置，绘制的时候直接在这个位置取信息绘制。

VAO的全名是Vertex Array Object，首先，它不是Buffer-Object，所以不用作存储数据；其次，它针对“顶点”而言，也就是说它跟“顶点的绘制”息息相关。（VAO和VA没有任何关系）

VAO记录的是一次绘制中所需要的信息，这包括“数据在哪里glBindBuffer”、“数据的格式是怎么样的glVertexAttribPointer”、shader-attribute的location的启用glEnableVertexAttribArray。

glGenBuffers(1, &m\_nQuadPositionVBO);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_nQuadPositionVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(fQuadPos), fQuadPos, GL\_STREAM\_DRAW);

glGenBuffers(1, &m\_nQuadTexcoordVBO);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_nQuadTexcoordVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(fQuadTexcoord), fQuadTexcoord, GL\_STREAM\_DRAW);

glGenBuffers(1, &m\_nQuadIndexVBO);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, m\_nQuadIndexVBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(nQuadIndex), nQuadIndex, GL\_STREAM\_DRAW);

*//VAO 初始化部分*

glGenVertexArrays(1, &m\_nQuadVAO);

glBindVertexArray(m\_nQuadVAO);

*//开始保存状态*

glBindBuffer(GL\_ARRAY\_BUFFER, m\_nQuadPositionVBO);

glEnableVertexAttribArray(VAT\_POSITION);

glVertexAttribPointer(VAT\_POSITION, 2, GL\_INT, GL\_FALSE, 0, **NULL**);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_nQuadTexcoordVBO);

glEnableVertexAttribArray(VAT\_TEXCOORD);

glVertexAttribPointer(VAT\_TEXCOORD, 2, GL\_INT, GL\_FALSE, 0, **NULL**);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, m\_nQuadIndexVBO);

*//保存结束*

glBindVertexArray(**NULL**);

glBindBuffer(GL\_ARRAY\_BUFFER, **NULL**);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, **NULL**);

以上就是VAO的使用方法了。VAO可以理解为一个状态容器，记录VBO的状态。