**Opengl的顶点数组**

按照正常的顶点依次绘制，需要调用的函数太多，同时另外一个问题是相邻多边形的共享顶点的冗余处理

Opengl提供了一些顶点数组函数，允许只用少数几个数组指定大量与顶点相关的数据，并用少量函数调用（与顶点数组的数量相仿）访问这些数据。使用顶点数组，一个拥有20条边的多边形的20个顶点可以放在一个数组中，并且只通过1个函数进行调用。如果每个顶点还有一条法线向量，所有20条法线向量可以放在另一个数组中，也可以只通过一个函数进行调用。

**把数据放在顶点数组中，可以提高应用程序的性能。使用顶点数组可以减少函数调用的次数，从而提高性能。另外，使用顶点数组还可以避免共享顶点的冗余处理。**

使用顶点数组对几何图形进行渲染需要3个步骤：

1 激活（启用）最多可达8个数组，每个数组用于存储不同的类型的数据：顶点坐标、表面法线、RGBA颜色、辅助颜色、颜色索引、雾坐标、纹理坐标以及多边形的边界标志

2 把数据放入数组中，这些数组是通过他们的内存位置的地址（即指针）进行访问的

3 用这些数据绘制几何图形，opengl通过指针从所有的被激活数组中获取数据

**步骤一：启动数组**

调用glEnableClientState函数（使用一个枚举参数），激活选择的数组，从理论上来说，最多可能调用这个函数8次，激活8个可用数组。但在实践中，可以激活的数组最多只有6个，这个因为有些数组不能同时激活，例如GL\_COLOR\_ARRAY和GL\_INDEX\_ARRAY。应用程序的显示模式可以支持RGBA模式，也可以支持颜色索引模式，但不能同时支持者两种模式

该函数的参数可以为下列选项：

GL\_VERTEX\_ARRAY、GL\_COLOR\_ARRAY、GL\_SECONDARY\_COLOR\_ARRAY、GL\_INDEX\_ARRAY、GL\_NORMAL\_ARRAY、GL\_FOG\_COORDINATE\_ARRAY、GL\_TEXTURE\_COORD\_ARRAY和GL\_EDGE\_FLAG\_ARRAY

如果需要使用光照，可能需要为每个顶点定义一条法线向量，在这种情况下使用顶点数组时，需要同时激活表面法线数组和顶点坐标数组

glEnableClientState(GL\_NORMAL\_ARRAY);

glEnableClientState(GL\_VERTEX\_ARRAY);

为什么opengl的设计者要添加一个新的glEnableClientState函数，而不是沿用原来的glEnable函数？

Opengl被设计为客户端/服务器模式，服务器就是图形硬件，客户端就是cpu和内存，由于这对设置功能启用/禁用状态的函数专门用于图形的客户端，因此需要专门设计一堆用于客户端的函数

**步骤二：指定数组的数据**

glVertexPointer(Glint size, GLenum type, GLsizei stride, const GLvoid\* pointer)

指定了需要访问的空间坐标数据。pointer是数组包含的第一个顶点的第一个坐标的内存地址。type指定了数组中每个坐标的数据类型（GL\_SHORT、GL\_INT、GL\_FLOAT、GL\_DOUBLE）。size是每个顶点的坐标数量，它必须是2、3或4，stride是连续顶点之间的字节偏移量，如果stride是0，数组中的顶点便是紧密相邻的

为了访问其他的几个数组，可以使用下面这些类似的函数：

Void glColorPointer(Glint size, GLenum type, GLsizei stride, const GLvoid\* pointer)

Void glSecondaryColorPointer(Glint size, GLenum type, GLsizei stride, const GLvoid\* pointer)

在涉及纹理时，还有一个小小的问题，使用glBegin/glEnd模式时。我们知道有一个称为glMultiTexCoord的新函数负责为每个纹理单位发送纹理坐标，使用顶点数组时，我们可以用下面的这个函数更改glTexCoordPointer的目标纹理

glClientActiveTexture(GLenum texture)

其中texture参数是GL\_TEXTURE0、GL\_TEXTURE1等

启动多个顶点数组例子：

void testVertexArray()

{

GLshort vertices[] = {25, 25, 100, 100, 120, 120, 200, 200};

GLfloat colors[] = {1.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f};

glEnableClientState(GL\_VERTEX\_ARRAY); // 启动顶点数组

glEnableClientState(GL\_COLOR\_ARRAY);

glVertexPointer(2, GL\_SHORT, 0, vertices);

glColorPointer(3, GL\_FLOAT, 0, colors);

//glShadeModel(GL\_FLAT);

glBegin(GL\_LINES);

glArrayElement(0); // 一次只能操作一个顶点

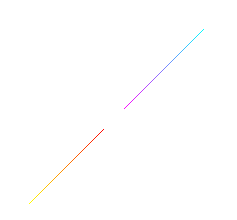
glArrayElement(1);

glArrayElement(2);

glArrayElement(3);

glEnd();

}



glArrayElement函数从任何已经glEnableClientState启用的数组中查找对应的数组数据，如果一个数组被启用，并且还没有指定一个对应的数组（glColorPointer、glVertexPointer等），就可能出现非法的内存访问，可能导致程序崩溃。使用glArrayElement的优点就是只要用一个函数调用就可以代替一个特定顶点指定所有数据所需要的几个函数调用（glNormal、glColor、glVertex等）

通过跨距将多个数组放置在一个数组

void testVertexArray()

{

GLfloat pointers[] = {25, 25, 1.0f, 1.0f, 0.0f,

100, 100, 1.0f, 0.0f, 0.0f,

120, 120, 1.0f, 0.0f, 1.0f,

200, 200, 0.0f, 1.0f, 1.0f};

glEnableClientState(GL\_VERTEX\_ARRAY); // 启动顶点数组

glEnableClientState(GL\_COLOR\_ARRAY);

glVertexPointer(2, GL\_FLOAT, 5 \* sizeof(GLfloat), &pointers[0]);

glColorPointer(3, GL\_FLOAT, 5 \* sizeof(GLfloat), &pointers[2]);

glBegin(GL\_LINES);

glArrayElement(0);

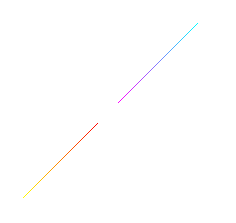
glArrayElement(1);

glArrayElement(2);

glArrayElement(3);

glEnd();

}



解引用顶点数组列表

Opengl提供了一个glDrawElements函数用于循环调用glArrayElement，但是需要调用一个索引数组。（与上面的等价）

void testVertexArray()

{

GLfloat pointers[] = {25, 25, 1.0f, 1.0f, 0.0f,

100, 100, 1.0f, 0.0f, 0.0f,

120, 120, 1.0f, 0.0f, 1.0f,

200, 200, 0.0f, 1.0f, 1.0f};

GLubyte index[] = {0, 1, 2, 3};

glEnableClientState(GL\_VERTEX\_ARRAY); // 启动顶点数组

glEnableClientState(GL\_COLOR\_ARRAY);

glVertexPointer(2, GL\_FLOAT, 5 \* sizeof(GLfloat), &pointers[0]);

glColorPointer(3, GL\_FLOAT, 5 \* sizeof(GLfloat), &pointers[2]);

glDrawElements(GL\_LINES, 4, GL\_UNSIGNED\_BYTE, index);

}

甚至可以直接用glDrawArrays(GL\_LINES, 0, 4)代替glDrawElements

混合数组：

glInterleavedArrays也可以同时指定几个顶点数组，同时该函数还能启用和禁用适当的数组（因此，它组合了步骤一的启动数组和步骤二的指定数据数据）

GLfloat pointers[] = {1.0f, 1.0f, 0.0f, 25, 25, 0,

1.0f, 0.0f, 0.0f, 100, 100, 0,

1.0f, 0.0f, 1.0f, 120, 120, 0,

0.0f, 1.0f, 1.0f, 200, 200, 0};

glInterleavedArrays(GL\_C3F\_V3F, 0, pointers);

glDrawArrays(GL\_LINES, 0, 4);

**opengl的索引顶点数组**

索引顶点数组就是并不按顺序从头遍历到尾的顶点数组，它的访问顺序是由一个单独的索引数组指定的。索引顶点数组可以节省内存，并减少变换开销，在理想状态下，它们可能比显示列表更快。

使用普通的渲染方式或顶点数组，不存在其他机制让两个相邻的三角形共享一组顶点，如下图显示了共享一条边的三角形带，尽管三角形带可以很好地处理各种三角形共享顶点，但是我们没有办法避免两个三角形带所共享的顶点的变换开销，因为每个三角形带都必须单独指定。



例子：一个简单的立方体

如果我们在一个顶点数组中对法线或顶点进行复用，而不是多次存储它，将可以节省大量的内存，除了节省内存外，优秀的opengl实现还可以对它进行优化，将这些顶点只变换一次，从而节省宝贵的变换时间。

static GLfloat corners[] = {

-25.0f, 25.0f, 25.0f, // 0 // 立方体的前面

25.0f, 25.0f, 25.0f, // 1

25.0f, -25.0f, 25.0f, // 2

-25.0f, -25.0f, 25.0f, // 3

-25.0f, 25.0f, -25.0f, // 4 // 立方体的后面

25.0f, 25.0f, -25.0f, // 5

25.0f, -25.0f, -25.0f, // 6

-25.0f, -25.0f, -25.0f // 7

};

static GLubyte indexes[] = {

0, 1, 2, 3, // 前面

4, 5, 1, 0, // 顶面

3, 2, 6, 7, // 底面

5, 4, 7, 6, // 后面

1, 5, 6, 2, // 右面

4, 0, 3, 7 // 左面

};

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glPushMatrix();

glTranslatef(0.0f, 0.0f, -50.0f);

glRotatef(20.0f, 1.0f, 0.0f, 0.0f);

glRotatef(20.0f, 0.0f, 0.0f, 1.0f);

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 0, corners);

**glDrawElements(GL\_QUADS, 24, GL\_UNSIGNED\_BYTE, indexes);**

glDisableClientState(GL\_VERTEX\_ARRAY);

glPopMatrix();

注意：具有锐利边缘和叫的模型所拥有的可共享顶点往往较少，但是，那些具有较大平滑表面区域的模型在使用索引顶点数组时常常能节省更多的**内存和转换开销**，通过内存传输的集合图形越少，意味着对应的数学操作就越少。即使是在静态图形中，索引顶点数组有时候也可以取得比显式列表高得多的效果。对于许多实时渲染程序来说，索引顶点数组常常是进行几何图形渲染的理想选择，很快就可以看到，在顶点数组中更进一步，实现更快的性能，把显示列表远远抛开一边