HW3-Report

21307347 陈欣宇

1. **环境搭建**

在上一次实验HW2基础上进行本次实验

1. **作业内容**

**Phong shading 与 VBO 绘制三维物体：**

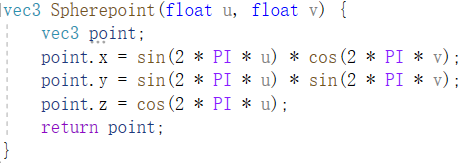
1. 通过fragment shader实现Phong shading：
   1. 在vertex shader中输出法向量：
   2. 在fragment shader中通过Phong shading计算三类反射
2. 使用VBO存储顶点与连接关系：
   1. 通过细分小球产生足够多的三角面片

**实现思路**

先使用小球绘制，便于划分三角面片，实现fragment shader基本绘制后，再进行讨论分析

1. **创建小球**

Spherepoint：得到球面上的点，其中输入u，v范围[0，1]



Sphere：“根据划分遍历点得到三角形，其中lons和lats分别为圆面在xoz和xoy平面划分的段数，将点赋予vertices，用于存入VBO缓存

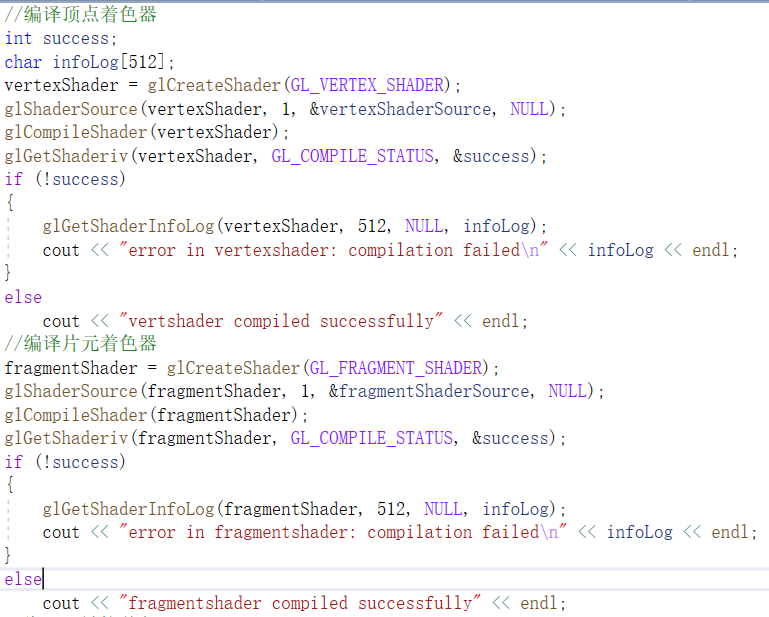


1. **初始化着色器**：

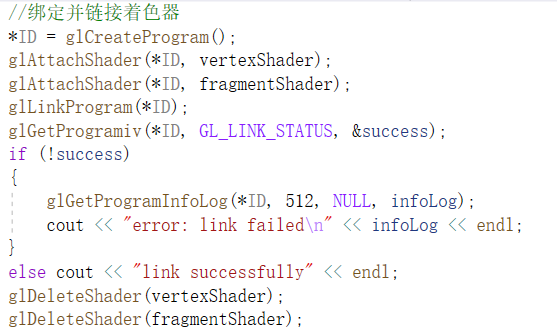
initShader函数：读取vert和frag文件代码



编译顶点着色器和片元着色器代码



绑定着色器和目标程序对象id



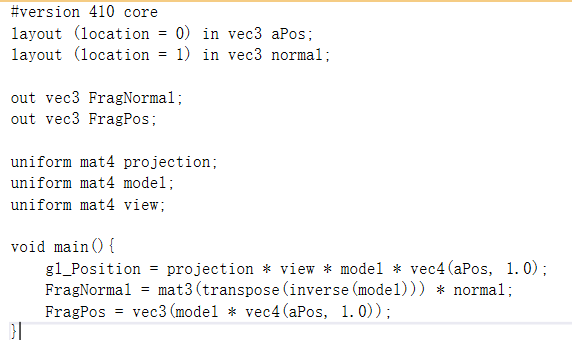
1. **初始化VBO**

vao、vbo生成并绑定，写入顶点数据到缓冲区，因为顶点和法向量属性都位于vertices中一同写入缓冲区，由glVertexAttribPointer设置顶点位置和法向量的读取方式，最后暂时解绑VAO



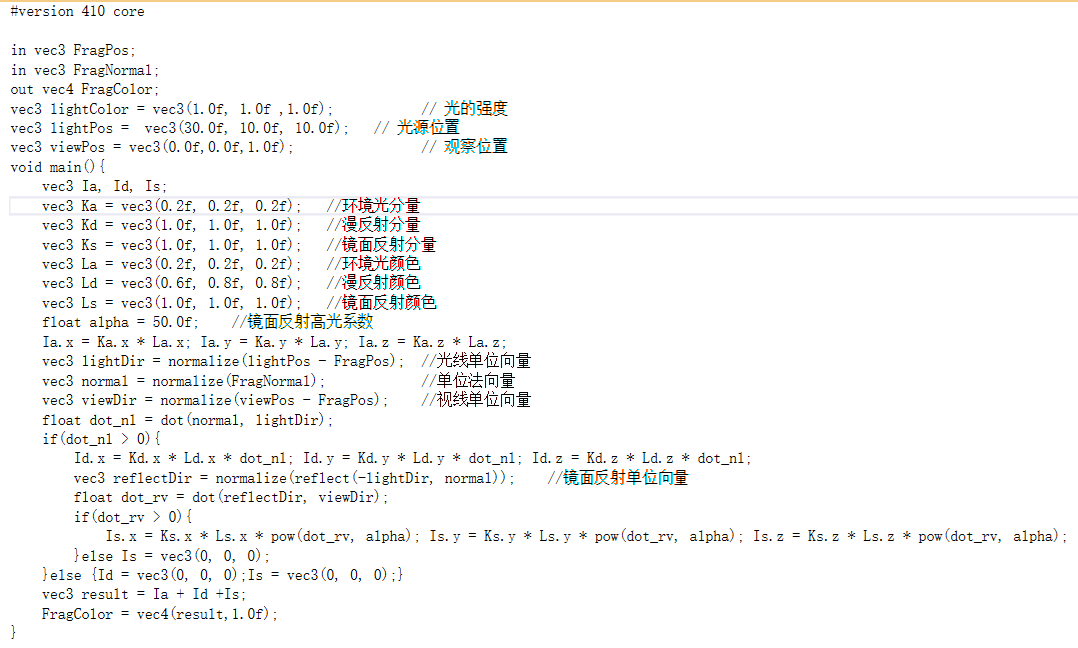
1. **顶点着色器**

Layout中location对应上面glVertexAttribPointer的第一个参数，uniform为绘制函数传入参数，根据模型变换、视图和投影变换对顶点位置和法向量进行变换，通过out传入片元着色器



1. **片元着色器**

设置光源和观察位置，以及顶点着色器传入参数，设置phong shading的三种类型颜色计算，传出最终颜色FragColor到渲染管线的后续阶段，最后显示颜色



1. **绘制图像**

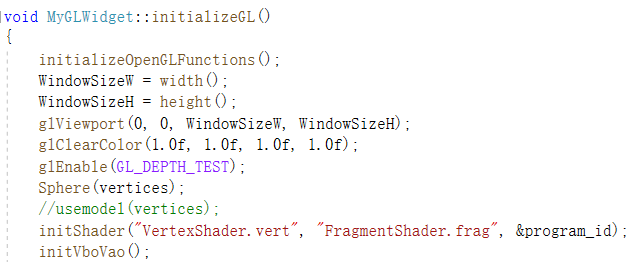
draw函数：

设置模型平移旋转矩阵model，模型视图矩阵view，模型投影矩阵projection，传入shader，之后绑定VAO号绘制图像，解绑VAO

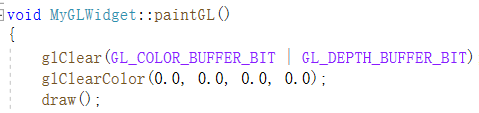


1. **其他工作**

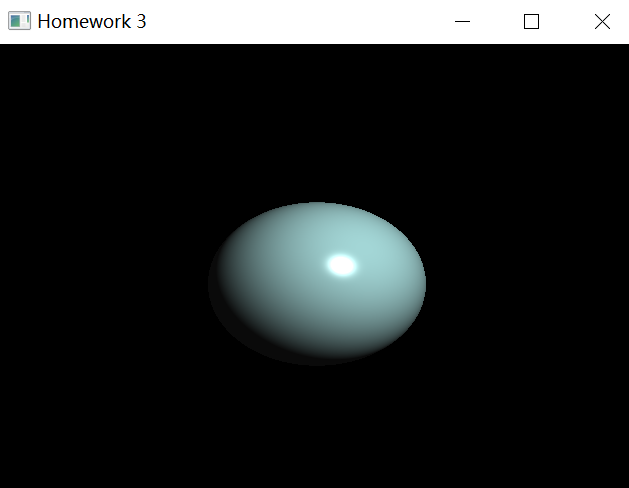
初始化：启动深度检测，创建小球的点存储在vertices中，其中vertices为全局变量，再初始化shader编译代码链接着色器，初始化VBO、VAO，将vertices传入缓冲区



paintGL由update调用，设置背景颜色，调用绘制函数



运行绘制结果：



**使用多个细分迭代次数讨论以下内容：**

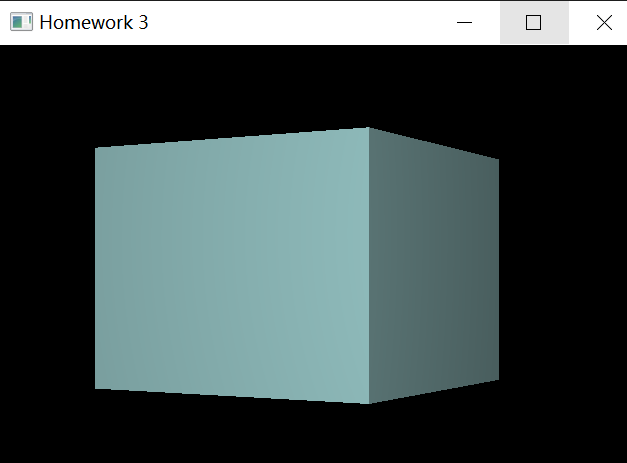
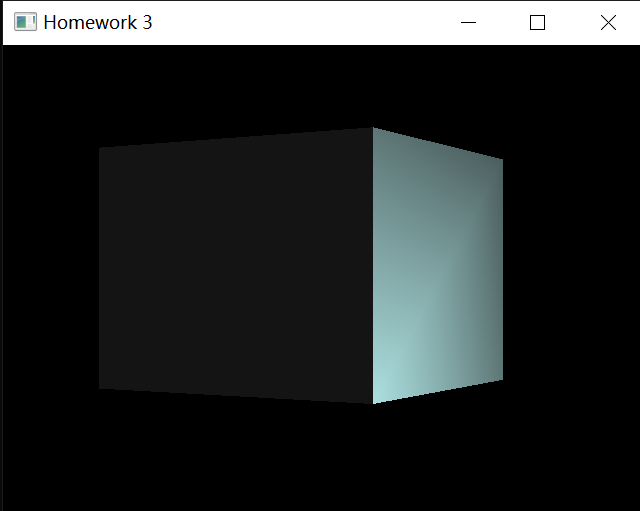
1. **对比Phong shading与OpenGL自带的smoothing shading的区别**

使用Phong shading相比smoothing shading的不同在于，smoothing shading先计算顶点颜色，再对片元颜色直接插值，而Phong shading则是先对三角形顶点的法向量进行插值，再通过插值法向量计算颜色。Smoothing shading的渲染速度快于Phong shading，但Phong shading的渲染质量更高。

用OpenGL自带的smoothing shading进行绘图，使用代码如下，模型视图调至于Phong shading相同。

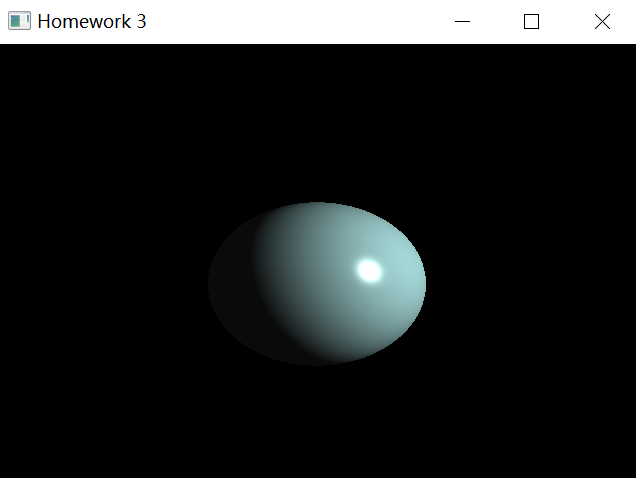
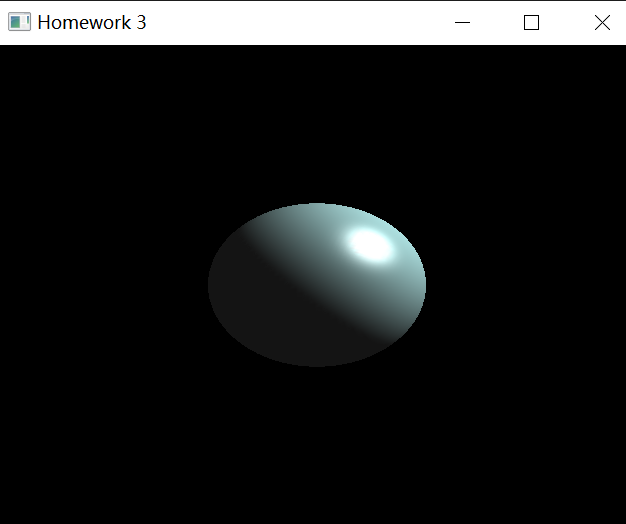


运行结果如下图，左边为使用Phong shading进行渲染，右边使用smoothing shading，因为是对每个三角形进行插值，导致同一个面内部存在一条分界线，而Phong shading明显没有这个问题。

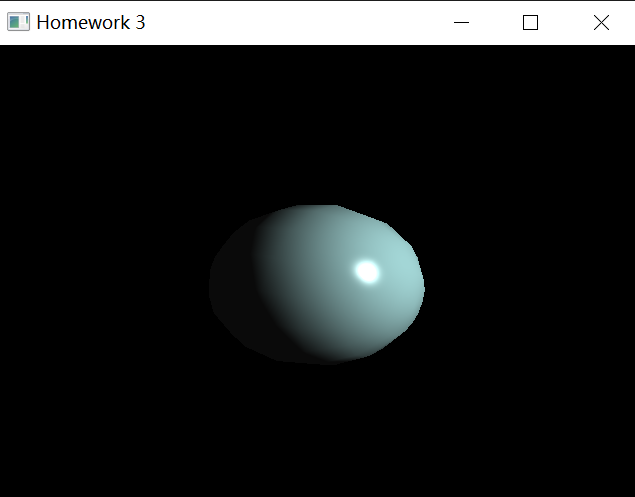
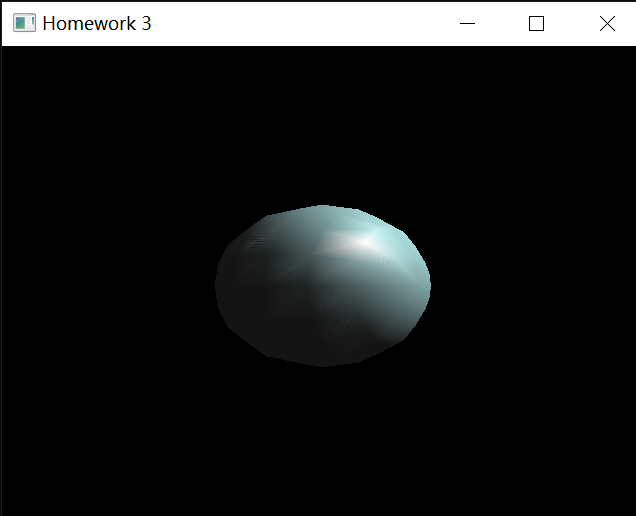
 

以下在小球不同细分程度上比较二者的区别：左为phong shading，右为smoothing shading

细分程度128x128

细分程度：16x16

可以看到使用Phong shading的小球在较小的细分程度下人能看出镜面高光、漫反射和环境光，而使用smoothing shading的镜面高光则因为颜色插值出现了变形的情况。Phong shading更能适应较大块三角片的着色情况。

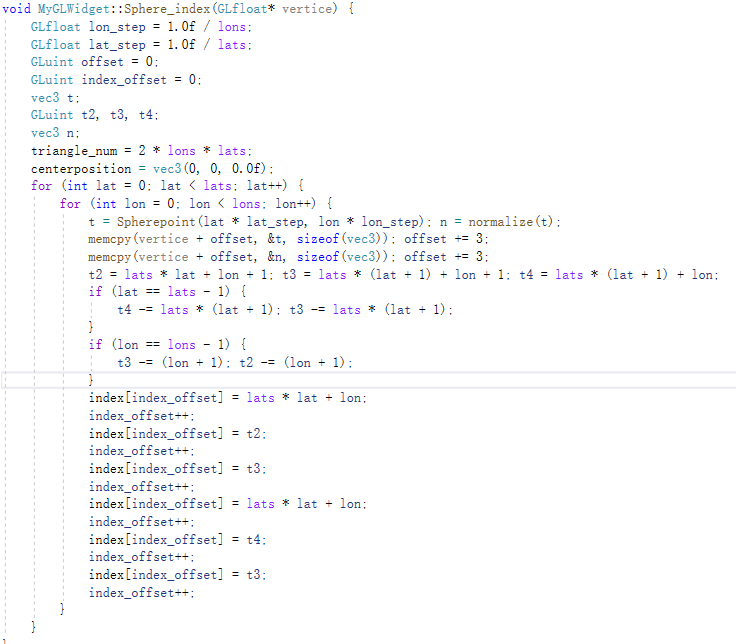
1. **使用VBO进行绘制及通过glVertex进行绘制的区别**

以上已经分别实现VBO和glVertex的绘制，使用VBO能将顶点数据存储在显存中，数据存储在缓冲区对象中，而不必每次绘制都传递数据，数据传输效率较高，特别是在大型场景下的数据运输开销较低。

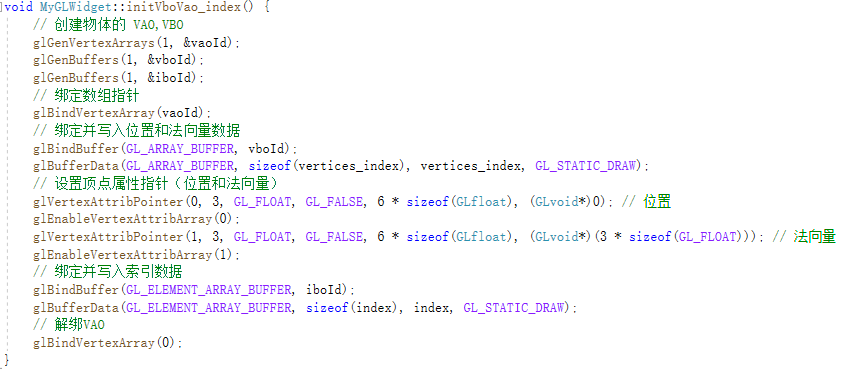
1. **讨论VBO中是否使用index array的效率区别**

将小球细分的点用index array，比较测试效率差异：

重新实现创建小球函数，对于小球上每一个点，只存入缓冲区一次，构造索引如下代码，每三个索引指向的点即为一个三角片。

****

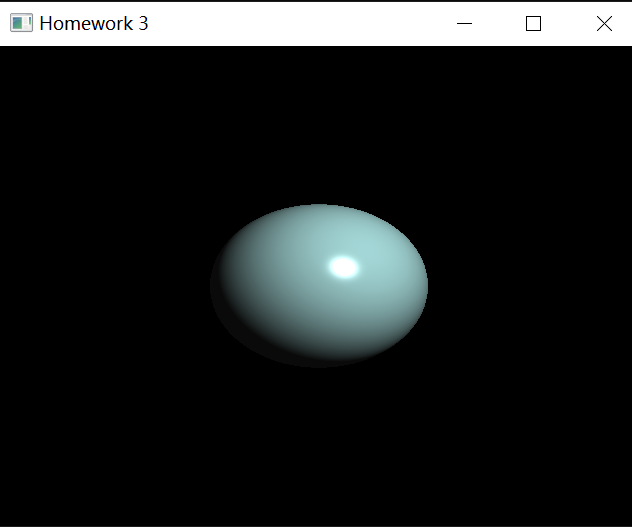
重新创建VBOVAO初始化函数，引入ibo绑定索引数据，使用glDrawElements进行绘制

****

重新创建draw\_index函数，使用glDrawElements进行绘制

****

绘制结果如下：

****

效率比对：

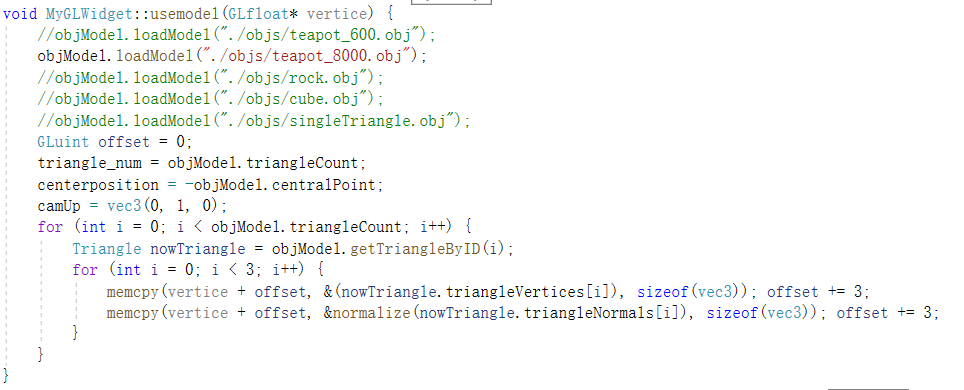
VBO有无使用Index array的区别主要在于存入缓冲区的数据量，也就是初始化的过程，在更新过程都是用了存储在缓冲区的数据，更新速度没有比较空间，因此主要比对初始化过程，渲染同样小球画面，在不同细分程度下，比对初始化到渲染成功时间。

|  |  |  |
| --- | --- | --- |
| VBO | 128x128 | 1024x1024 |
| 不使用index array | 0.0747 s | 0.809 s |
| 使用index array | 0.0653 s | 0.292 s |

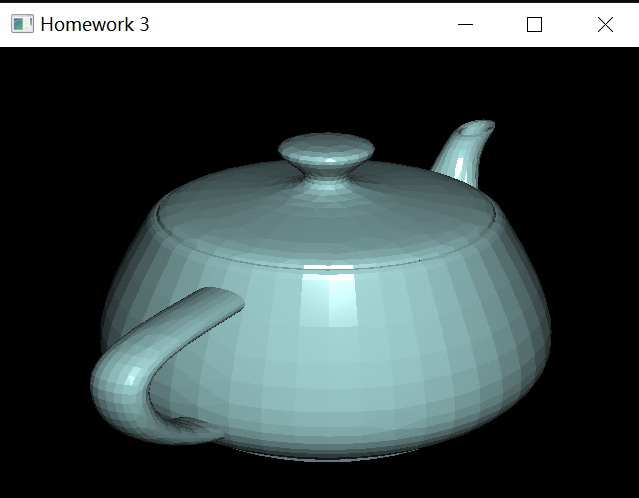
可以看出VBO使用index array后渲染效率有明显提升，特别是在细分程度较大，使用index array减少缓冲区存储量的优势更为明显。

1. **对比、讨论HW3和HW2的渲染结果、效率的差别**

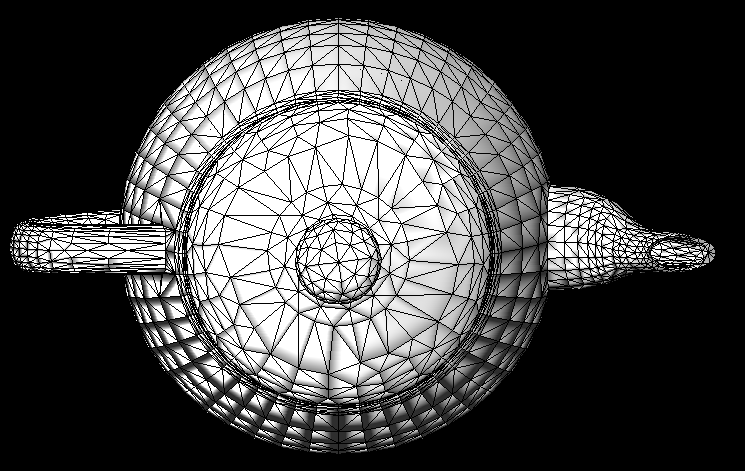
HW3加载模型到VBO方式如下：顶点位置及法向量都已给出，与创建小球类似，赋值到vertice即可

****

使用最大模型进行比较，HW3绘制结果如下，初始化到绘制过程耗时0.346s，



而HW2使用Phong shading需要耗时31s左右，差距显著



（补充：在HW2的实验中无论是法颜色插值还是法向量插值最终的颜色都无法达到很好的效果，后来总结使用模型给定的法向量很难达到理想情况，再次根据顶点所在面的法向量取均值效果会更好，当前比较结果中HW2和HW3都未重新计算顶点法向量，构成比较条件）

可以看出使用VBO具有明显的效率优势，渲染结果的质量虽然未完善，但也取得不错质量。

**三、问题与总结：**

本次作业主要在于学习使用顶点着色器、片元着色器以及VBO的使用，起初对这些模块的使用一知半解，通过查阅资料逐渐了解各个部件具体如何使用，在图形处理这方面的代码很容易因为一点差错而导致整个图像出现很大问题，学习实现fragment shader、使用VBO并不耗时，主要在于寻找bug的时间，因此在实现过程就需要谨慎处理。具体的使用方法还有很多需要学习的地方，比如控制VBO传入顶点着色器的参数，如何单独使用着色器这些方法等，希望之后能够更多了解。