A3: OpenGL & Phong Shading

一、概述

**1.目标：**

（1）用OpenGL实现场景的窗口交互（已提供glCanvas类和Camera的控制方法）

（2）在ray caster中实现Blinn-Phong光照模型

**2.知识点介绍：**

**要使用OpenGL来绘制，需要把之前的primitives重新用OpenGL的API实现。**

*（1）画Plane：*OpenGL只能画一个很大的矩形来代替无边际的Plane。平面的方

程为Ax+By+Cz+D=0。已知法线Normal(A,B,C)和D，平面到原点的距离。可以在空间

中任取一个向量v（通常取v(0,0,1），如果v和法线平行，则取v（0,1,0）)，和Normal

做叉乘归一化之后可以得到平面正交基的一个向量b1，再用Normal和b2做叉乘并归

一化，可以得到另一个向量b2。取一个10^6以下的数字作为边长，根据平面正交基，

就可以算出这个矩形的四个顶点。

*（2）画球体：*通过在水平方向和垂直方向细分，用四边形或三角形拼接而成。y是垂直方向，在空间中球上某点的坐标表示为：

x= radius \* sin (angle\_y) \* cos (angle\_xz)

z= radius \* sin (angle\_y) \* sin (angle\_xz);

y= radius \* cos (angle\_y);

其中angle\_xz的变化范围是0-2π，angle\_y的变化范围是0-π。

**着色方法：**

*（1）球体的两种法线表示方式：*

Flat Shading:球的法线是画出的每个QUAD的法线。

Gouraud Shading:球心到构成每个QUAD的顶点的向量是法线，每个顶点单独设置

一次发现，OpenGL会将顶点间的法线进行插值。

*（2）Blinn-Phong：*包含diffuseColor和specular color。本次在上次任务的基础上添加了specular color，公式为：

myspecularColor = specularColor \* pow (nDoth, exponent)\*lightColor \* nDotl;

其中nDoth=dot(normal, (normal+dirToLight).Normalize())。

*（3）specular lobe fix ：*

当nDotHh小于0时直接截断到0会导致高光和阴影的分界处出现hard edge。解决方法是，不直接Clamp，而是再乘上dot(normal,dirToLight)。由于本次不要求掌握用OpenGL的三个Pass方式进行修补，我只在ray caster中加上了修补效果。

二、实现细节

**1. 用OpenGL画图元：**

**（1）Sphere：**

|  |
| --- |
| void Sphere::paint () {  material->glSetMaterial ();  int M = n\_phi; int N = n\_theta;  float step\_y = PI / M; float step\_xz = 2 \* PI / N;//每次步进的角度  float x[4], y[4], z[4];//四个顶点坐标 float angle\_y = 0.0;//起始角度  float angle\_xz = 0.0; int i = 0, j = 0;  glBegin (GL\_QUADS);  for (i = 0; i < M; i++){  angle\_y = i \* step\_y; //每次步进step\_y  for (j = 0; j < N; j++){  angle\_xz = j \* step\_xz; //每次步进step\_xz  x[0] = radius \* sin (angle\_y) \* cos (angle\_xz);//: :左上角  z[0] = radius \* sin (angle\_y) \* sin (angle\_xz);  y[0] = radius \* cos (angle\_y);  x[1] = radius \* sin (angle\_y + step\_y) \* cos (angle\_xz); //: : 左下角  z[1] = radius \* sin (angle\_y + step\_y) \* sin (angle\_xz);  y[1] = radius \* cos (angle\_y + step\_y);  x[2] = radius\*sin(angle\_y+step\_y)\*cos(angle\_xz+step\_xz);// : : 右下角  z[2] = radius \* sin (angle\_y + step\_y)\*sin (angle\_xz + step\_xz);  y[2] = radius \* cos (angle\_y + step\_y);  x[3] = radius \* sin (angle\_y) \* cos (angle\_xz + step\_xz);//: : 右上角  z[3] = radius \* sin (angle\_y) \* sin (angle\_xz + step\_xz);  y[3] = radius \* cos (angle\_y);  Vec3f a(center.x () + x[0], center.y () + y[0], center.z () + z[0]);  Vec3f b (center.x () + x[1], center.y () + y[1], center.z () + z[1]);  Vec3f c (center.x () + x[2], center.y () + y[2], center.z () + z[2]);  Vec3f d (center.x () + x[3], center.y () + y[3], center.z () + z[3]);  Vec3f ab = a - b; Vec3f bc = b - c;  Vec3f normal; Vec3f::Cross3 (normal, bc, ab); normal.Normalize ();  Vec3f n\_a = a - center; Vec3f n\_b = b - center;  Vec3f n\_c = c - center; Vec3f n\_d = d - center;  //画出这个平面  if (!gouraud) {//flat shading  glNormal3f (normal.x (), normal.y (), normal.z ());  glVertex3f (a.x(),a.y(),a.z()); glVertex3f (b.x (), b.y (), b.z ());  glVertex3f (c.x(),c.y(),c.z()); glVertex3f (d.x (), d.y (), d.z ());  }else {//gouraud shading  glNormal3f (n\_a.x(),n\_a.y(),n\_a.z());glVertex3f (a.x(),a.y(),a.z ());  glNormal3f (n\_b.x(),n\_b.y(),n\_b.z());glVertex3f (b.x(),b.y (),b.z ());  glNormal3f (n\_c.x(),n\_c.y(),n\_c.z());glVertex3f (c.x(),c.y (),c.z ());  glNormal3f (n\_d.x (),n\_d.y(),n\_d.z());glVertex3f (d.x(),d.y(), d.z());  }  }//循环画出这一层的平面  }画出剩余层  glEnd ();  } |

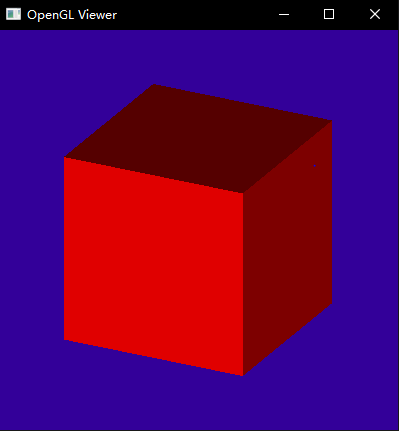
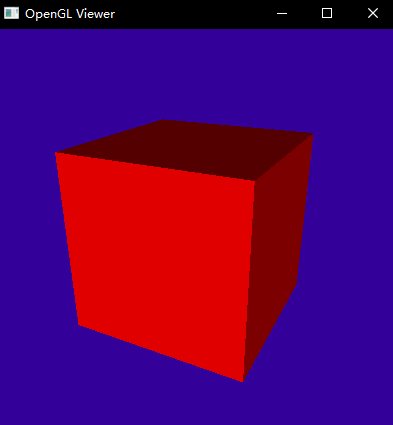
**（2）Plane：**

|  |
| --- |
| void Plane::paint () {  mat->glSetMaterial ();  Vec3f v(0,0,1);  if (normal.x () == 0 && normal.y () == 0 && normal.z () == 1) {  v = Vec3f (0,1,0);  }  Vec3f b1; Vec3f::Cross3 (b1, normal, v); b1.Normalize ();//x轴  Vec3f b2; Vec3f::Cross3 (b2,normal, b1); b2.Normalize ();//y轴  float big=100;  Vec3f a = b1 \* big + b2 \* big + this->d\*normal;  Vec3f b = b1\*big - 1 \* b2\*big + this->d \* normal;  Vec3f c = -1\*b1 \* big - 1 \* b2\*big + this->d \* normal;  Vec3f d = -1 \* b1 \* big + 1 \* b2\*big + this->d \* normal;  glBegin (GL\_QUADS);  glNormal3f (normal.x(), normal.y(), normal.z());  glVertex3f (a.x (), a.y (), a.z ());  glVertex3f (b.x (), b.y (), b.z ());  glVertex3f (c.x (), c.y (), c.z ());  glVertex3f (d.x (), d.y (), d.z ());  glEnd ();  } |

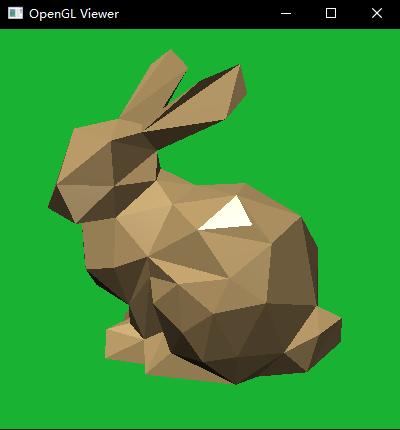
**（3）Blinn-Phong&** **specular lobe fix：**

|  |
| --- |
| //myspecularColor  Vec3f v = ray.getDirection ()\*-1.0f;  Vec3f h = v + dirToLight; h.Normalize ();  float nDoth = n.Dot3 (h);  bool isFix = true;  if (nDoth < 0) {  if (!isFix) { nDoth = 0;}  else {nDoth\*=nDotl;if (nDoth < 0) nDoth = 0;}  }  myspecularColor = specularColor \* pow (nDoth, exponent)\*lightColor \* nDotl;  return mydiffuseColor + myspecularColor; |

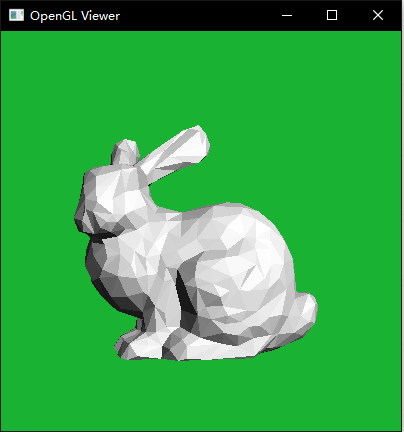
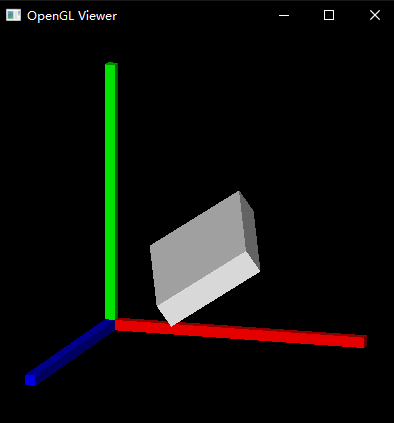
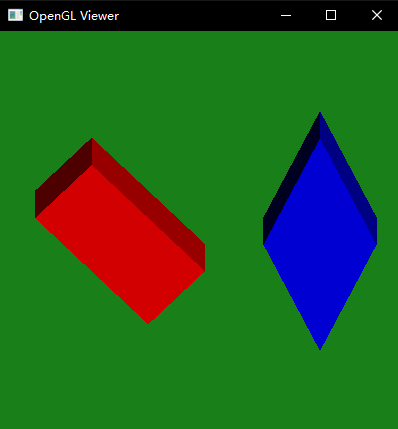
三、结果展示

1.glCanvas测试：

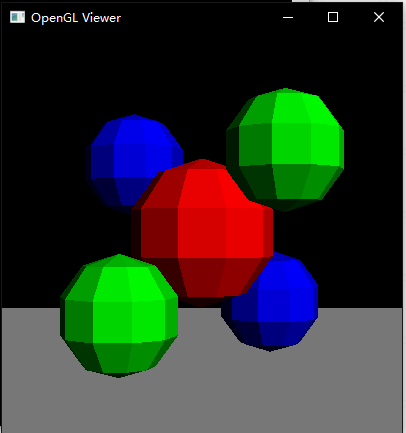
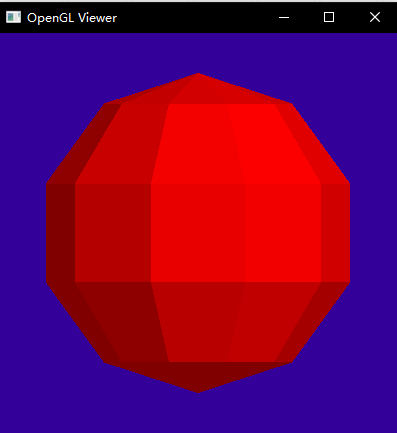
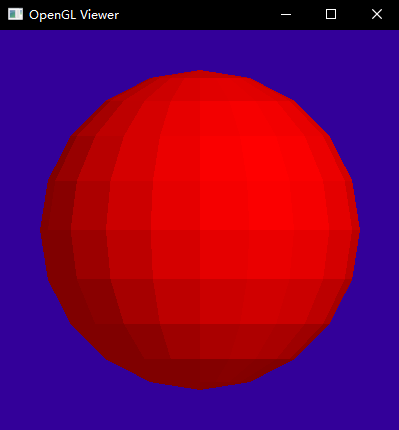
2. Blinn-Phong测试：



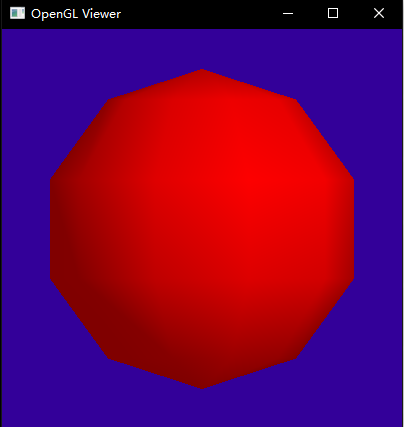
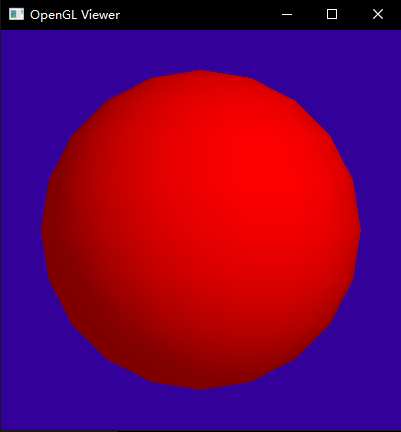
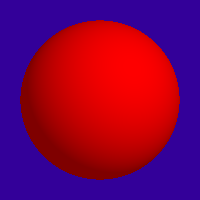
3.OpenGL Transform测试：

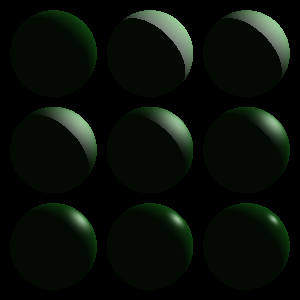
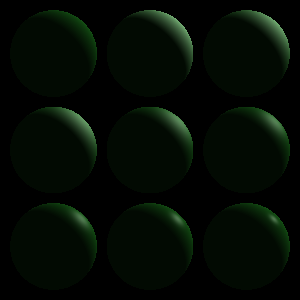


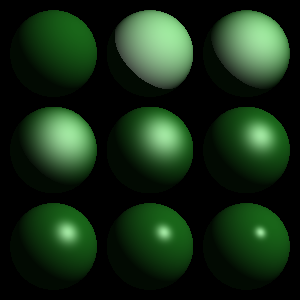
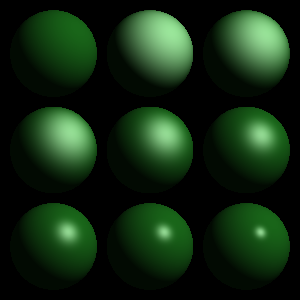
4.Plane&Sphere测试：

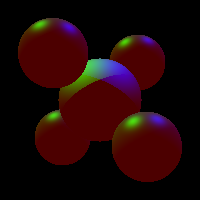
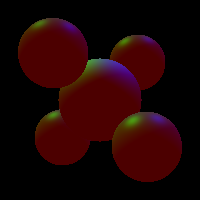
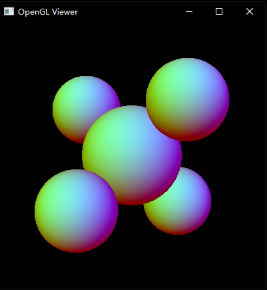
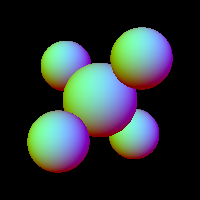


5.Gouraud测试：



6. specular lobe fix测试：



7. weird\_lighting测试：

四、心得体会

使用OpenGL后，三角形绘制是十分方便的，但是球体就比较麻烦，因为不能用数值定义确定点的位置而必须使用四边形或三角形拼接。但是使用OpenGL后渲染速度变得很快很快，同时也封装了很多内容，使用起来比较方便。通过前面ray caster的实现，可以说对OpenGL的一些内部工作方式有了理解。