

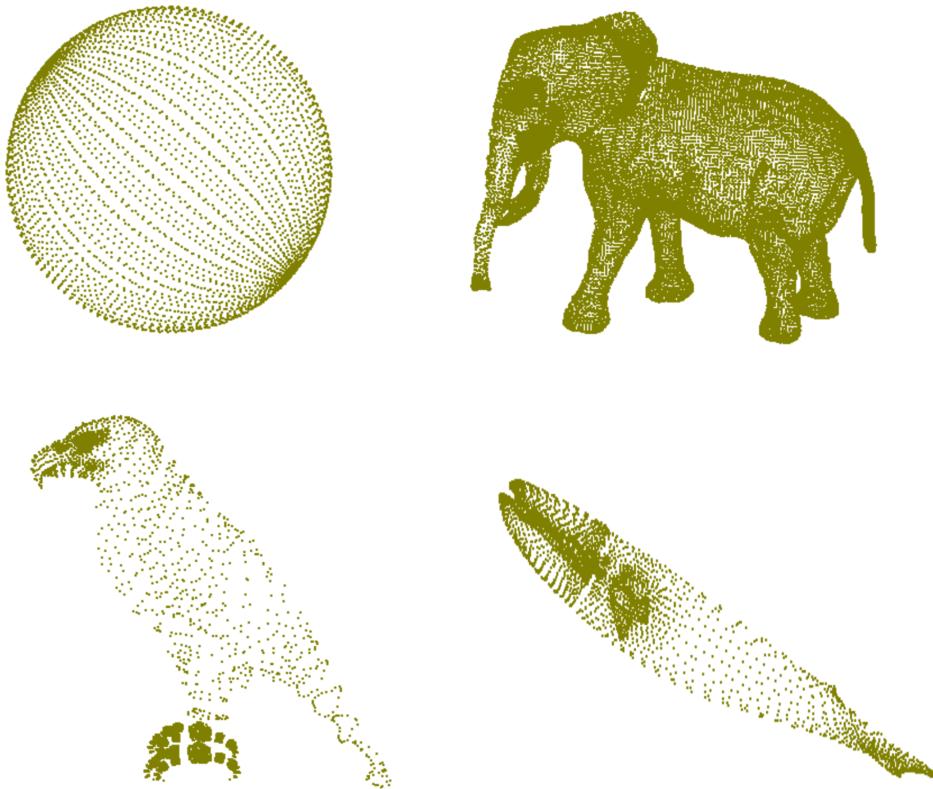
计算机网络 课程实验报告

学号:201700130011	姓名: 刘建东	班级: 17 级菁英班
实验题目: 使用 OpenGL 显示 OBJ 模型		
实验内容:		
1. 使用 OpenGL 显示 OBJ 模型 2. 程序支持分别显示面片和线框 3. 程序支持对 OBJ 模型显示贴图		
实验过程:		
一、构建 mesh 来存取 OBJ 文件		
(1) 点类		
<pre>struct Vertex { float x, y, z; //定义三维图形的点 };</pre>		
(2) 面类		
<pre>struct Face { Face(void) : verts(0) {}; vector<Vertex> verts; //记录面的所有顶点 float normal[3]; //记录面的法向量，分别是x, y, z三个方向 vector<pair<float, float> > texture; };</pre>		
(3) Mesh		
<pre>struct myMesh { myMesh(void) : verts(0), faces(0) {}; vector<Vertex> verts; vector<Face> faces; vector<pair<float, float> > texture; }mesh;</pre>		
存储过程中全部使用了 <code>vector</code> 来对图中元素进行存取，没有使用半边结构。因为半边结构与前向星邻接表类似，实际使用中与 <code>vector</code> 的时间空间复杂度无较大差别。		
二、读取 OBJ 文件		
此处主要是一个模拟过程，将 OBJ 文件中的顶点信息（以 <code>v</code> 开头行）、纹理坐标信息（以 <code>vt</code> 开头行）、面信息（以 <code>f</code> 开头行）进行读取处理，并存入 <code>mesh</code> 中。		
三、显示 OBJ 文件的点阵图		
将 <code>mesh</code> 中点的三维坐标直接用 <code>opengl</code> 绘出即可。		

```

void draw_points(){
    glPushMatrix();
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
    //绘制点的信息
    glPointSize(2);
    for (int j = 0 ; j < (int)mesh.verts.size(); j++) {
        glBegin(GL_POINTS);
        glColor3f(0.5, 0.5, 0.0);
        glVertex3f( (float)(mesh.verts[j].x), (float)(mesh.verts[j].y), (float)(mesh.verts[j].z));
        glEnd();
    }
    glDisable(GL_COLOR_MATERIAL);
    glPopMatrix();
}

```



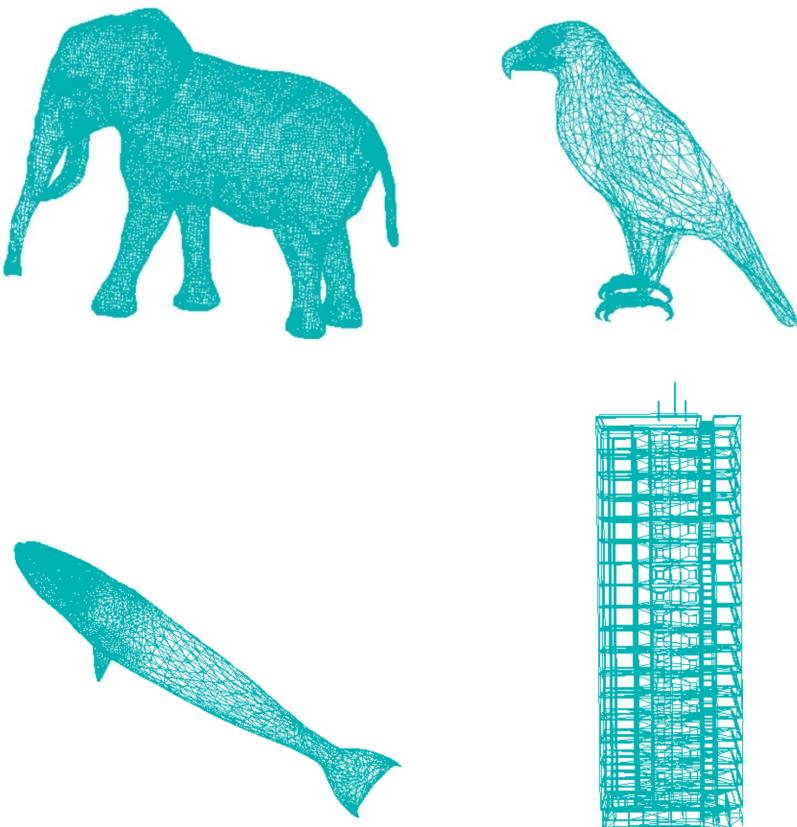
四、显示 OBJ 文件的线框

将 mesh 中面的每个三角形的三条边连起来，即可显示线框。

```

void draw_lines(){
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, my_set_material);
    double temp_x = 0, temp_y = 0, temp_z = 0;
    for (int i = 0; i < (int)mesh.faces.size(); i++) {
        Face& face = mesh.faces[i];
        glColor3f(0.0, 0.7, 0.7);
        glBegin(GL_LINES);
        for (int j = 0; j < (int)face.verts.size(); j++) {
            Vertex vert = face.verts[j];
            if(j==0){
                temp_x = vert.x;
                temp_y = vert.y;
                temp_z = vert.z;
                continue;
            }
            glVertex3f(temp_x, temp_y, temp_z);
            glVertex3f(vert.x, vert.y, vert.z);
            temp_x = vert.x;
            temp_y = vert.y;
            temp_z = vert.z;
        }
        glEnd();
    }
    glDisable(GL_LIGHT0);
    glDisable(GL_NORMALIZE);
    glDisable(GL_LIGHTING);
    glDisable(GL_DEPTH_TEST);
}

```



五、显示 OBJ 文件的面片

将 mesh 中每个面的信息读取出来，利用面上的顶点以及面的法向量直接绘制每个面即可。

```
glPushMatrix();
glEnable(GL_COLOR_MATERIAL);
	glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
 glBegin(GL_POINTS);
 for (int i = 0; i < (int)mesh.faces.size(); i++) {
     Face& face = mesh.faces[i];
     glBegin(GL_POLYGON);           //绘制多边形
     //在绘制面的过程中载入我们已经计算好的法线量信息
     glColor3f(0.5, 0.5, 0.5);
     glNormal3fv(face.normal);    //在绘制面的时候同时载入法向量信息
     for (int j = 0; j < (int)face.verts.size(); j++) {
         Vertex vert = face.verts[j];
         glVertex3f(vert.x, vert.y, vert.z);
     }
     glEnd();
 }
 glDisable(GL_COLOR_MATERIAL);
 glPopMatrix();
```



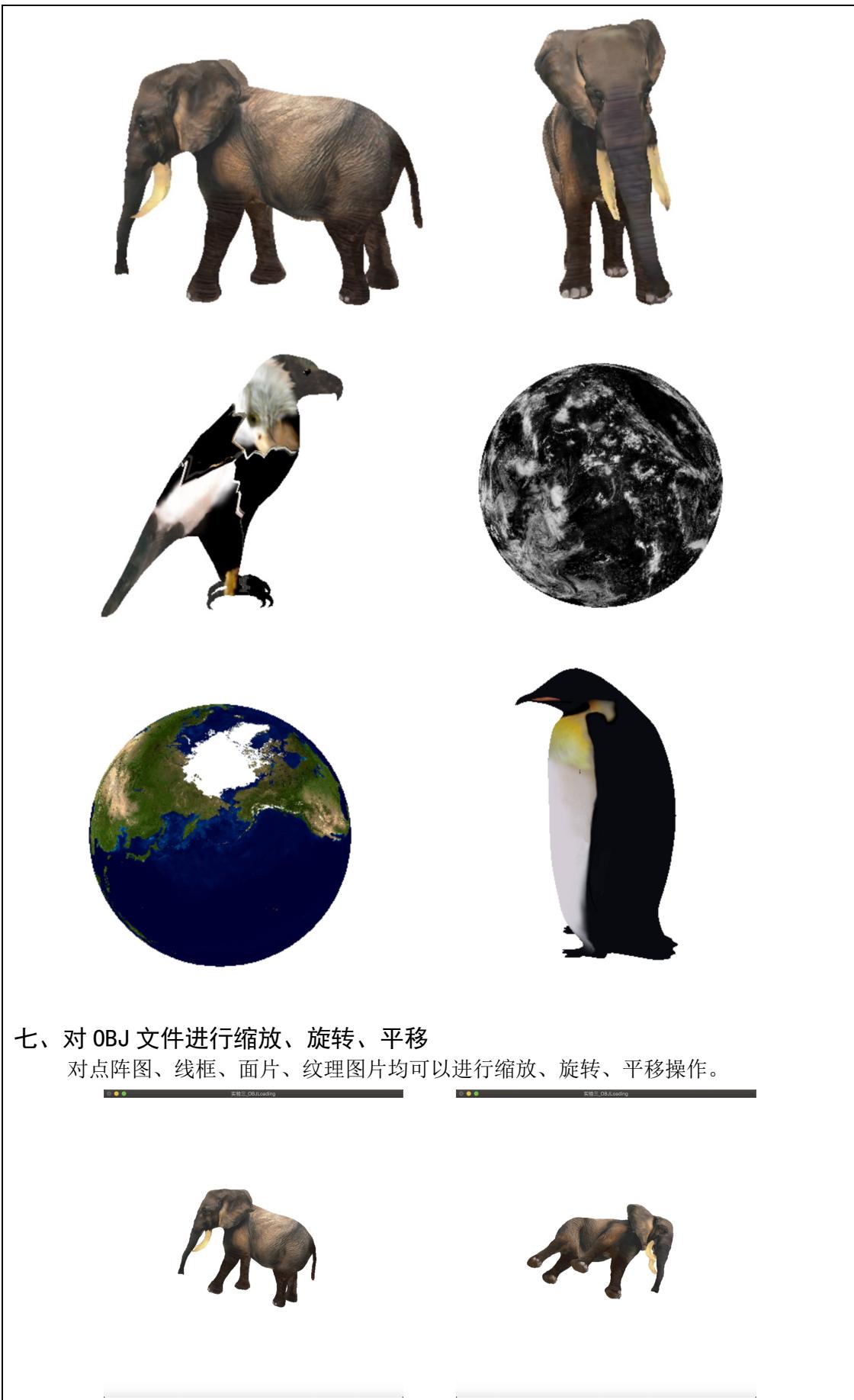
六、对 OBJ 文件进行贴图

只有有纹理坐标的 OBJ 文件才可以加载贴图文件，需要读入 OBJ 文件中的 vt 信息，然后再绘制面的时候开启纹理，并加载纹理坐标即可实现贴图。

```
void display()
{
    static GLfloat light0_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);

    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texGround);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

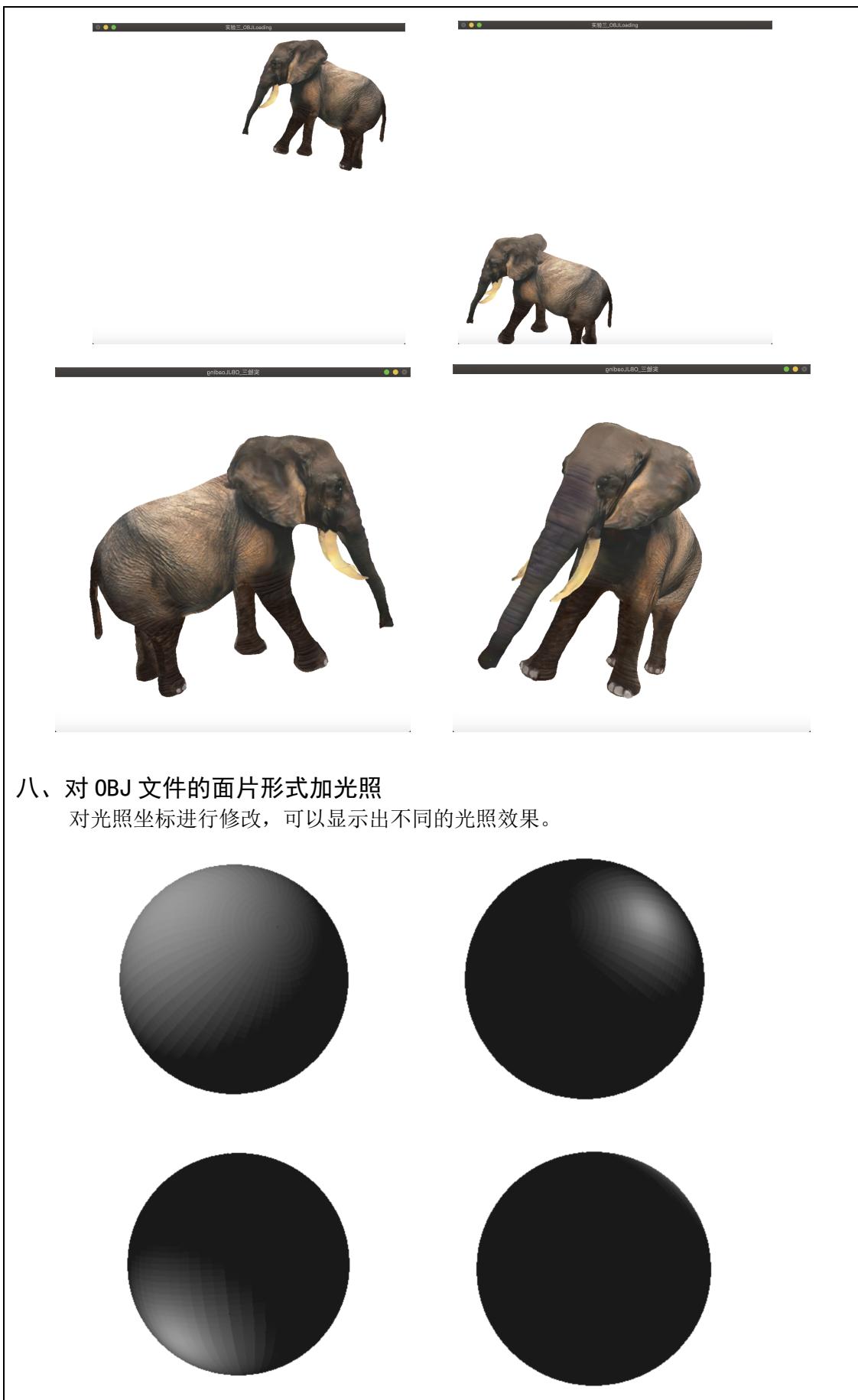
    for (int i = 0; i < (int)mesh.faces.size(); i++) {
        Face& face = mesh.faces[i];
        glBegin(GL_POLYGON); //绘制多边形
        //在绘制面的过程中载入我们已经计算好的法线量信息
        glNormal3fv(face.normal); //在绘制面的时候同时载入法向量信息
        for (int j = 0; j < (int)face.verts.size(); j++) {
            glTexCoord2f(face.texture[j].first, face.texture[j].second);
            glVertex3f(face.verts[j].x, face.verts[j].y, face.verts[j].z);
        }
        glEnd();
    }
    glDisable(GL_TEXTURE_2D);
}
```



七、对 OBJ 文件进行缩放、旋转、平移

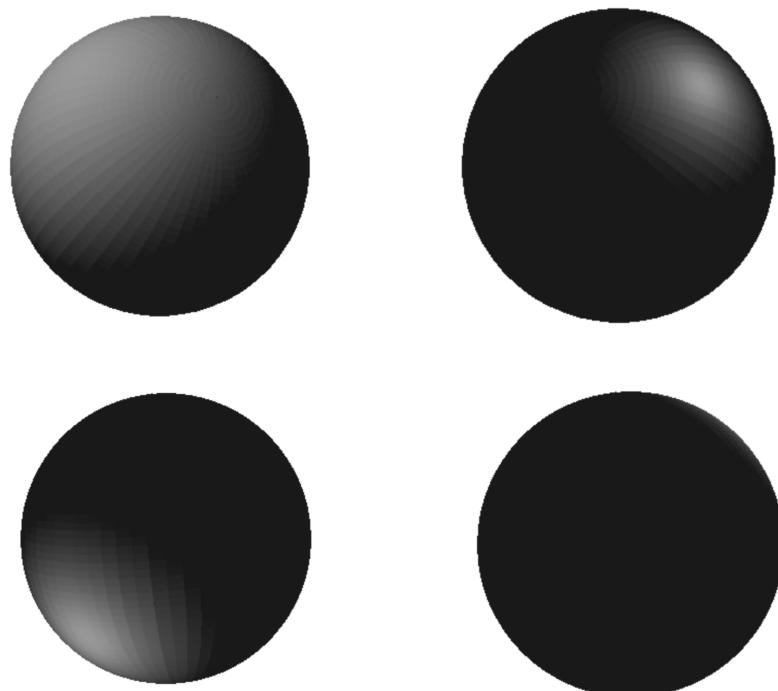
对点阵图、线框、面片、纹理图片均可以进行缩放、旋转、平移操作。





八、对 OBJ 文件的面片形式加光照

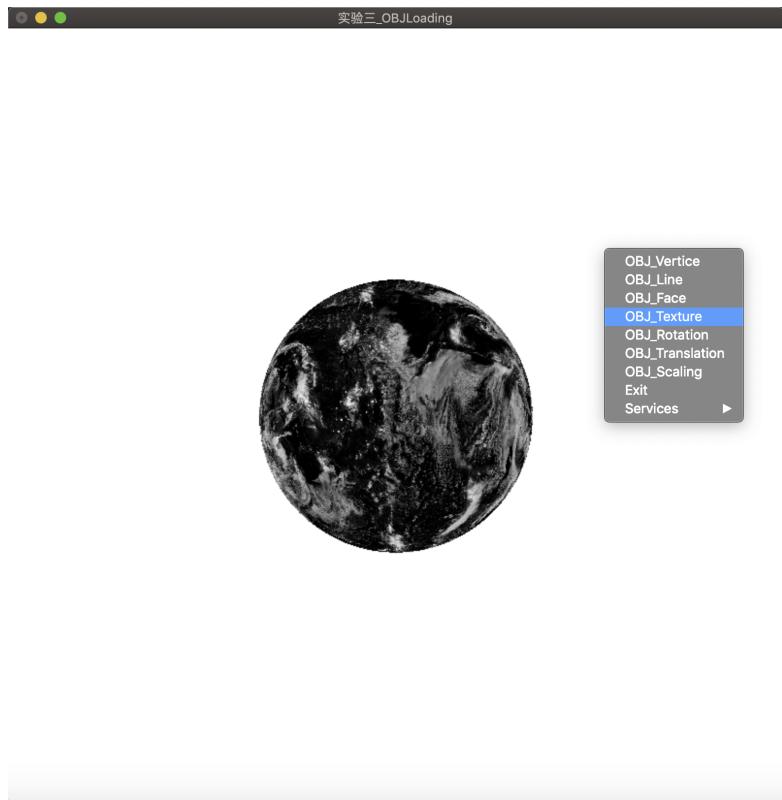
对光照坐标进行修改，可以显示出不同的光照效果。



九、添加交互方式

① 鼠标

鼠标右键即会出现菜单，从上往下分别是点阵图、线框图、面片、纹理图、旋转、平移、缩放、退出程序，一共有 7 个功能。



```
//注册菜单
glutCreateMenu(MyMenu); //注册菜单回调函数
glutAddMenuEntry("OBJ_Vertex", 1); //添加菜单项
glutAddMenuEntry("OBJ_Line", 2);
glutAddMenuEntry("OBJ_Face", 3);
glutAddMenuEntry("OBJ_Texture", 4);
glutAddMenuEntry("OBJ_Rotation", 5);
glutAddMenuEntry("OBJ_Translation", 6);
glutAddMenuEntry("OBJ_Scaling", 7);
glutAddMenuEntry("Exit", 8);
glutAttachMenu(GLUT_RIGHT_BUTTON); //把当前菜单注册到指定的鼠标键
```

② 键盘

键盘则绑定了数字键'1'和'2'，用于移动光照位置。因此键盘绑定了一个功能。

```
void GLUTKeyboard(unsigned char key, int x, int y)
{
    if(key == '1'){
        for(int i = 0 ; i < 4 ;i++)
            light0_position[i] += 0.1;
    }
    else if(key == '2'){
        for(int i = 0 ; i < 4 ;i++)
            light0_position[i] -= 0.1;
    }
    GLUTmouse[0] = x;
    GLUTmouse[1] = GLUTwindow_height - y;
    GLUTmodifiers = glutGetModifiers();
}
```

实验总结：

1. 实验中存取 mesh 信息时，对于点、面、纹理信息均采取了 vector 进行存储，替代了半边结构。面的法向量信息直接根据面上的点进行确定，即面上两个向量叉乘即可得到。

```
struct Vertex {
    float x, y, z; //定义三维图形的点
};

struct Face {
    Face(void) : verts(0) {};
    vector<Vertex> verts; //记录面的所有顶点
    float normal[3]; //记录面的法向量，分别是x, y, z三个方向
    vector<pair<float, float> > texture;
};

struct myMesh {
    myMesh(void) : verts(0), faces(0) {};
    vector<Vertex> verts;
    vector<Face> faces;
    vector<pair<float, float> > texture;
}mesh;
```

2. 实验中对于 OBJ 文件与 BMP 文件均进行了读取，两个文件都有自己对应的读取函数。其中 OBJ 文件仅支持 v、vt、f 信息的读取，而 BMP 文件在像素均为 2 的幂次以及位数为 24 位时显示效果最佳。
3. 此实验一共有 8 个功能，分别是加载点阵图、加载线框图、加载面片图、加载纹理图、旋转、平移、缩放以及光照。