上机作业3

1.利用半边数据结构读入OBJ文件

首先，创建一个名为ptr_mesh_的Mesh3D对象。

```
Mesh3D *ptr_mesh_ = new Mesh3D();
```

使用LoadFromOBJFile函数从"gourd.obj"文件中加载模型。

```
bool is_open = ptr_mesh_->LoadFromOBJFile("gourd.obj");
```

使用for循环遍历所有的面，对每个面进行绘制。对于每个面，首先获取该面的三个顶点的位置。

如果change变量的值为0，则使用平面光照进行处理；如果change变量的值为1，则使用平滑光照进行处理。使用glNormal3fv函数指定该面的法向量，并使用glVertex3f函数指 定三角形的三个顶点的位置。

这段代码可以用于加载和显示OBJ格式的3D模型，并且支持平面光照和平滑光照两种处理方式。

```
for(int i=0; i < ptr_mesh_->num_of_face_list(); ++i){
    point first, second, third;
    first = ptr_mesh_->get_face(i)->pedge_->pvert_->position();  //第一个点
    second = ptr_mesh_->get_face(i)->pedge_->pnext_->pvert_->position(); //第二个
点
    third = ptr_mesh_->get_face(i)->pedge_->pnext_->pnext_->pvert_->position();
//第三个点
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glBegin(GL_TRIANGLES);  //画三角形
    if(change == 0){
        glNormal3fv(ptr_mesh_->get_face(i)->facevector); //平面光照
        glVertex3f(first[0], first[1], first[2]);
        glVertex3f(second[0], second[1], second[2]);
        glVertex3f(third[0], third[1], third[2]);
    }
    else if(change == 1){  //平滑光照处理
        glNormal3fv(ptr_mesh_->get_face(i)->pedge_->pvert_->pointvector);
        glVertex3f(first[0], first[1], first[2]);
        glNormal3fv(ptr_mesh_->get_face(i)->pedge_->pnext_->pvert_-
>pointvector);
        glVertex3f(second[0], second[1], second[2]);
        glNormal3fv(ptr_mesh_->get_face(i)->pedge_->pnext_->pnext_->pvert_-
>pointvector);
        glVertex3f(third[0], third[1], third[2]);
    }
    glEnd();
}
```

2.计算网格面法向。

首先查看面的半边数据结构的存储方式。

```
class HE_face
{
public:
    int         id_;
    HE_edge     *pedge_;        //!< one of the half-edges_list bordering the
face
    Vec3f       normal_;        //!< face normal
    int         valence_;       //!< the number of edges_list
    int         selected_;      //!< a tag: whether the face is selected
    Vec4f       color_;         //!< the color of this face
    BoundaryTag boundary_flag_; //!< this flag is used to split the mesh
```

由面指向边再指向点和点的位置。

由三角形三个点得到两条边的向量，再将两条边叉乘得到面法向量，并将其单位化。

```
void Mesh3D::ComputePerFaceNormal(HE_face* hf){
    //图形学课程上机作业
    //请在此处添加计算面法向量代码
    point a, b, c;
    a = hf->pedge_->pvert_->position();  //得到三角形的三个点
    b = hf->pedge_->pnext_->pvert_->position();
    c = hf->pedge_->pnext_->pnext_->pvert_->position();
    point m1, m2;  //得到两条边向量
    m1 = b-a;
    m2 = c-a;

    double x = m1[1]*m2[2]-m1[2]*m2[1]; //叉乘得到法向量
    double y = m1[2]*m2[0]-m1[0]*m2[2];
    double z = m1[0]*m2[1]-m1[1]*m2[0];

    double norm = sqrt(x*x+y*y+z*z);

    hf->facevector[0] = x/norm;
    hf->facevector[1] = y/norm;
    hf->facevector[2] = z/norm;
}
```

3.计算网格顶点法向。

首先查看顶点的半边数据结构的存储方式。

```
class HE_vert
{
public:
    int         id_;
    point   position_;      //!< vertex position
    Vec3f   normal_;        //!< vertex normal
    Vec3f   texCoord_;      //!< texture coord
```

```
    Vec4f   color_;
    HE_edge     *pedge_;        //!< one of the half-edges_list emanating from
the vertex
    int         degree_;
    BoundaryTag boundary_flag_; //!< boundary flag
    int         selected_;      //!< a tag: whether the vertex is selected

    /*--------------------add by wang kang at 2013-10-12----------------------*/
    std::vector<size_t> neighborIdx;
```

网格顶点法向量为与顶点相连的各面法向量和的平均值。因此需要求出相邻的面法向量与总面数。通过nextedge->ppair_->pnext_指向下一条边；通过nextedge与firstedge是否相等判断是否结束。

```
void Mesh3D::ComputePerVertexNormal(HE_vert* hv)
{
    //图形学课程上机作业
    //请在此处添加计算面法向代码
    HE_edge *firstedge = hv->pedge_;
    HE_face *face_ = firstedge->pface_;
    Mesh3D::ComputeFaceNormal(face_);  //调用函数计算面法向量

    hv->pointvector[0] = 0;  //点向量初始化为0
    hv->pointvector[1] = 0;
    hv->pointvector[2] = 0;

    float total = 0;  //与该顶点相邻的面的数量
    for(int i=0; i<3; ++ i){
        hv->pointvector[i] += face->facevector[i];  //顶点相邻的面法向量之和
    }

    ++ total;
    HE_edge *nextedge = firstedge->ppair_->pnext_;  //下一条边
    while(nextedge != firstedge){
        face_ = nextedge->pface_;
        Mesh3D::ComputePerFaceNormal(face_);
        for(int i=0; i<3; ++ i){
            hv->pointvector[i] += face->facevector[i];
        }
        ++ total;
        nextedge = nextedge->ppair_->pnext_;
    }
    for(int i=0; i<3; ++ i){
        hv->pointvector[i] = hv->pointvector[i]/total;
    }
}
```

4.使用OpenGL光照：

(1) 设置固定光源和可移动光源。

固定光源：

```
//固定光源
GLfloat sun_light_position[] = {0.0f, 0.0f, 4.0f, 1.0f};
GLfloat sun_light_ambient[] = {0.0f, 0.0f, 0.0f, 1.0f};
GLfloat sun_light_diffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat sun_light_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};

glLightfv(GL_LIGHT0, GL_POSITION, sun_light_position);
glLightfv(GL_LIGHT0, GL_AMBIENT, sun_light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, sun_light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, sun_light_specular);

//开启灯光
glEnable(GL_LIGHT0);
glEnable(GL_LIGHTING);
glEnable(GL_DEPTH_TEST);
```

可移动光源：

可以通过xyz旋转来实现物体的旋转，相当于将光源绕着物体旋转。

首先，定义一个名为light_pos的GLfloat数组，用于存储光源的位置。位置设置为(0.0, 2.0, 0.0)，最后一个元素设置为1.0，表示点光源。使用glViewport函数设置了视口的左下角坐标和视口的宽度和高度。

使用glMatrixMode和glLoadIdentity函数设置了投影矩阵。使用了透视投影，调用gluPerspective函数来设置透视投影的参数。使用glMatrixMode和glLoadIdentity函数设置了模型视图矩阵。

使用glLightfv函数将light_pos数组设置为光源的位置，并使用glEnable函数函数启用了一个名为GL_LIGHT1的光源。使用glutSwapBuffers函数切换缓冲区，完成场景的绘制。

```
//可移动光源
GLfloat light_pos[] = {0.0,2.0,0.0,1.0};
glViewport(0, 0, (GLsizei)w, (GLsizei)h);
glMxtrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(40.0, (GLfloat)w/h, 1.0, 100.0);
glMatrixMode(GL_MODEVIEW);
glLoadIdentity();
glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glEnable(GL_LIGHT1);

glutSwapBuffers();
```

(2) 切换平面明暗处理和平滑明暗处理。

平面明暗处理时，利用上面第2步中计算的面法向；平滑明暗处理时，利用第3步中计算的顶点法向。

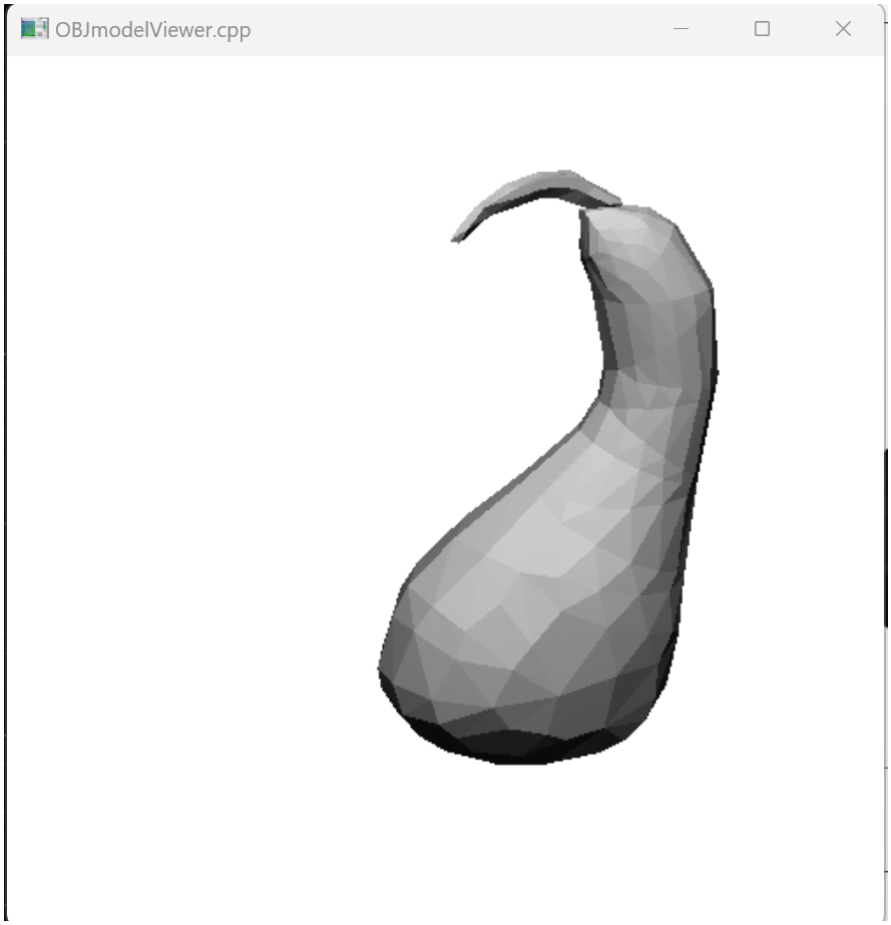
切换方式：键盘输入小写字母c，从平面切换到平滑光照；输入大写字母C从平滑切换到平面光照。

```
if(change == 0){
        glNormal3fv(ptr_mesh_->get_face(i)->facevector); //平面光照
```

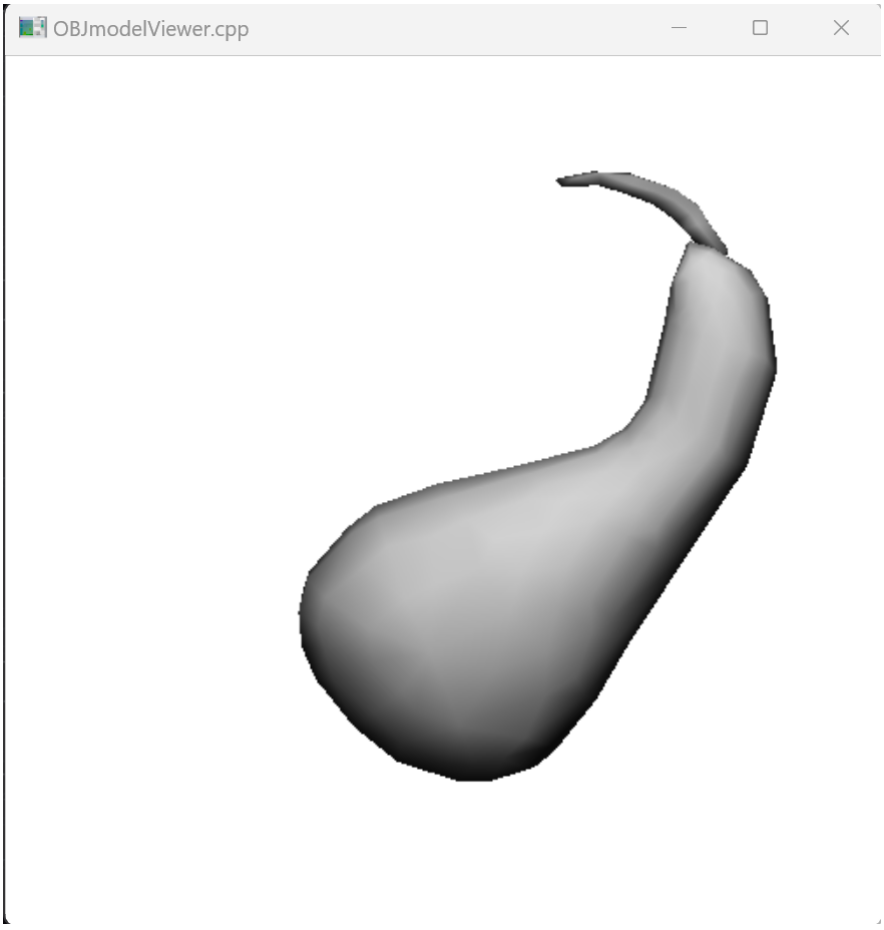
```
        glVertex3f(first[0], first[1], first[2]);
        glVertex3f(second[0], second[1], second[2]);
        glVertex3f(third[0], third[1], third[2]);
    }
else if(change == 1){  //平滑光照处理
        glNormal3fv(ptr_mesh_->get_face(i)->pedge_->pvert_->pointvector);
        glVertex3f(first[0], first[1], first[2]);
        glNormal3fv(ptr_mesh_->get_face(i)->pedge_->pnext_->pvert_-
>pointvector);
        glVertex3f(second[0], second[1], second[2]);
        glNormal3fv(ptr_mesh_->get_face(i)->pedge_->pnext_->pnext_->pvert_-
>pointvector);
        glVertex3f(third[0], third[1], third[2]);
    }
    glEnd();
```

5.运行效果

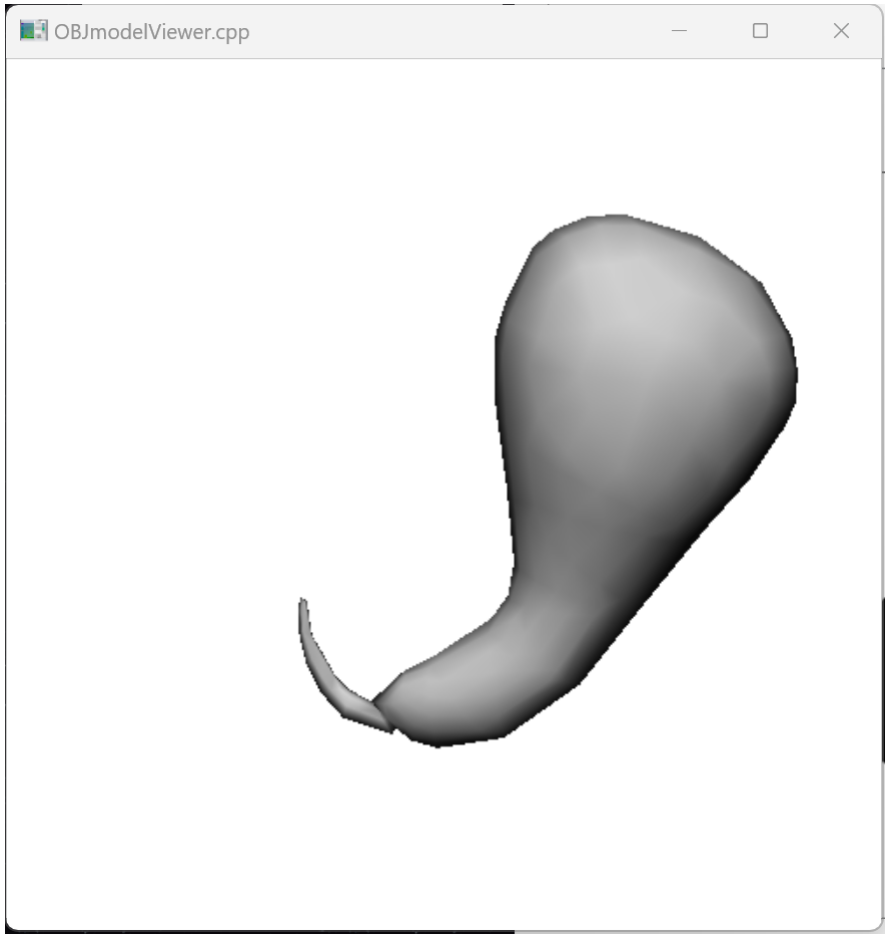
固定光源：

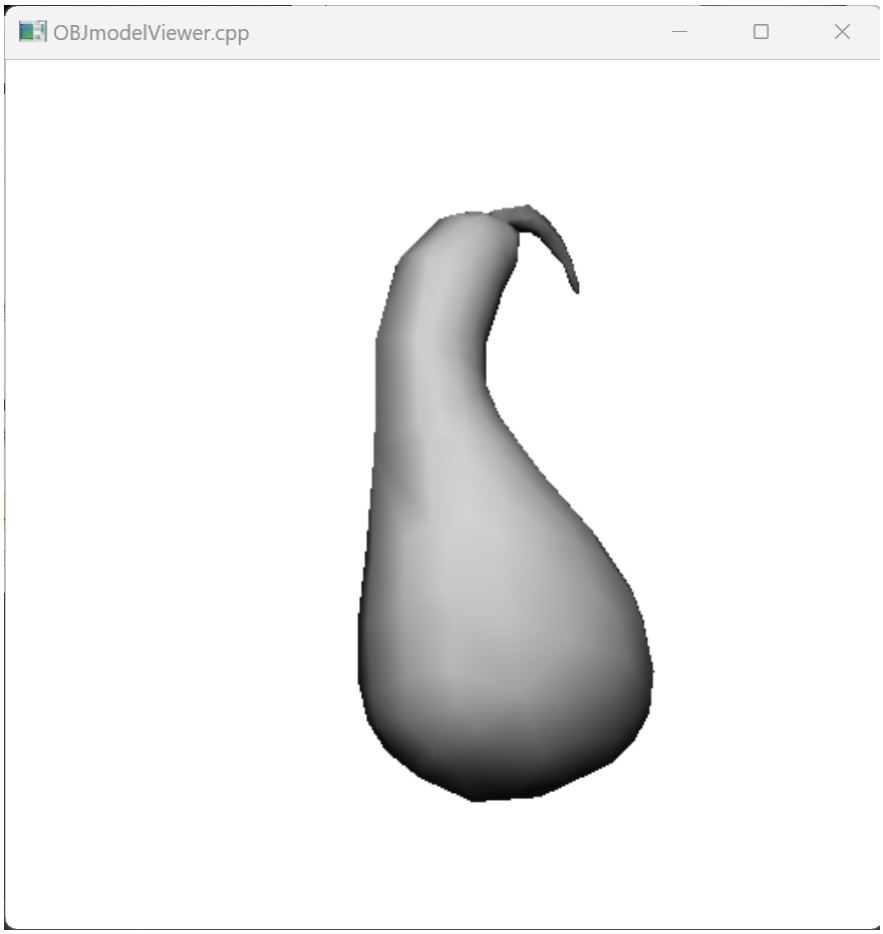


平滑处理后：



可移动光源：





通过观察可以看出，物体正前方有一盏白灯。物体进行旋转时，正前方的地方会被照亮，也就相当于物体固定不动，人的视角和白灯一样绕着物体旋转所观察到的现象，达成移动光源的效果。