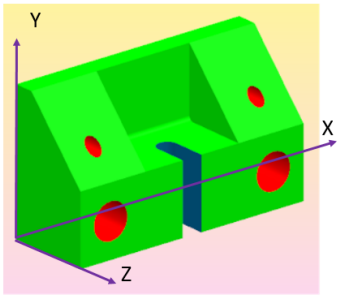
高斌_大作业 2 设计说明书

一、 作业要求

大作业2：零件显示

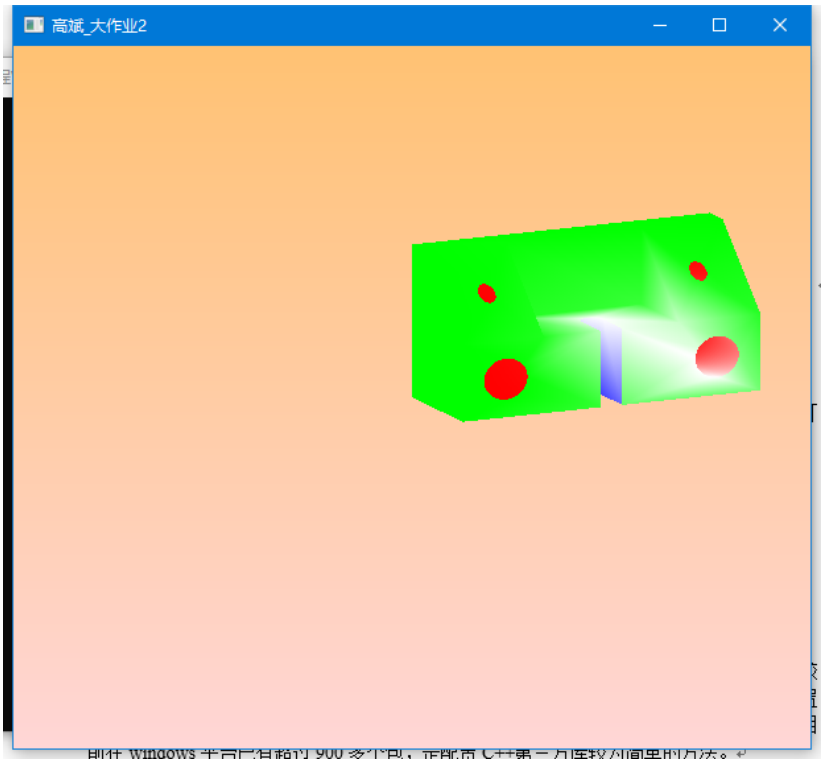
- 按照图中的颜色显示零件以及背景色，并用键盘控制零件的旋转
- 四个孔均为通孔
- 背景色均匀过渡
- 上下左右键和其他键分别控制旋转：
  - 上↑：绕X轴逆时针旋转
  - 下↓：绕X轴顺时针旋转
  - 左←：绕Y轴逆时针旋转
  - 右→：绕Y轴顺时针旋转
  - F5:绕Z轴逆时针旋转
  - 空格键：绕Z轴顺时针旋转



二、操作

1、打开“发布版_直接运行”文件夹，双击 exe 文件；

2、键盘方向键，↑：绕 X 轴逆时针旋转下，↓：绕 X 轴顺时针旋转，左←：绕 Y 轴逆时针旋转，右→：绕 Y 轴顺时针旋转，空格键：绕 Z 轴顺时针旋转，F5:绕 Z 轴逆时针旋转



三、方法和流程（具体代码都有注释）

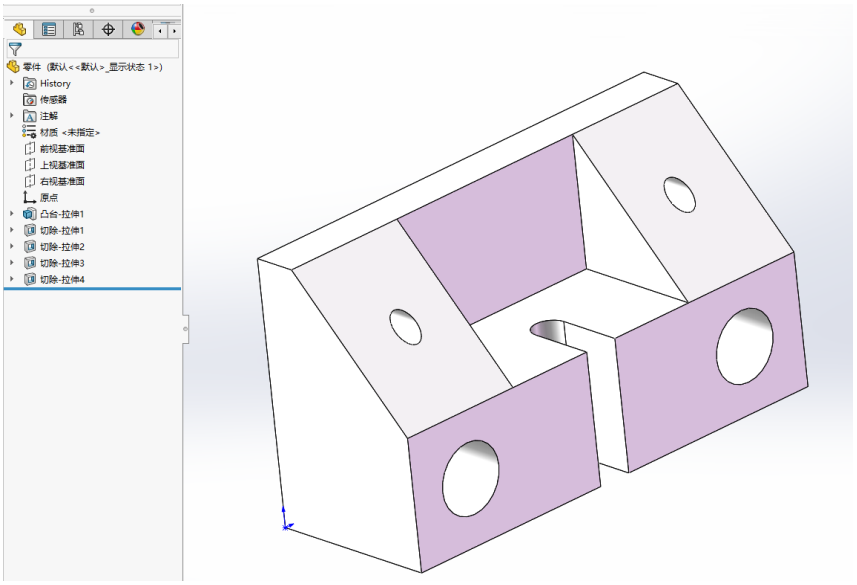
1、数据获取

（1）所用工具：SolidWorks2018，Blender。

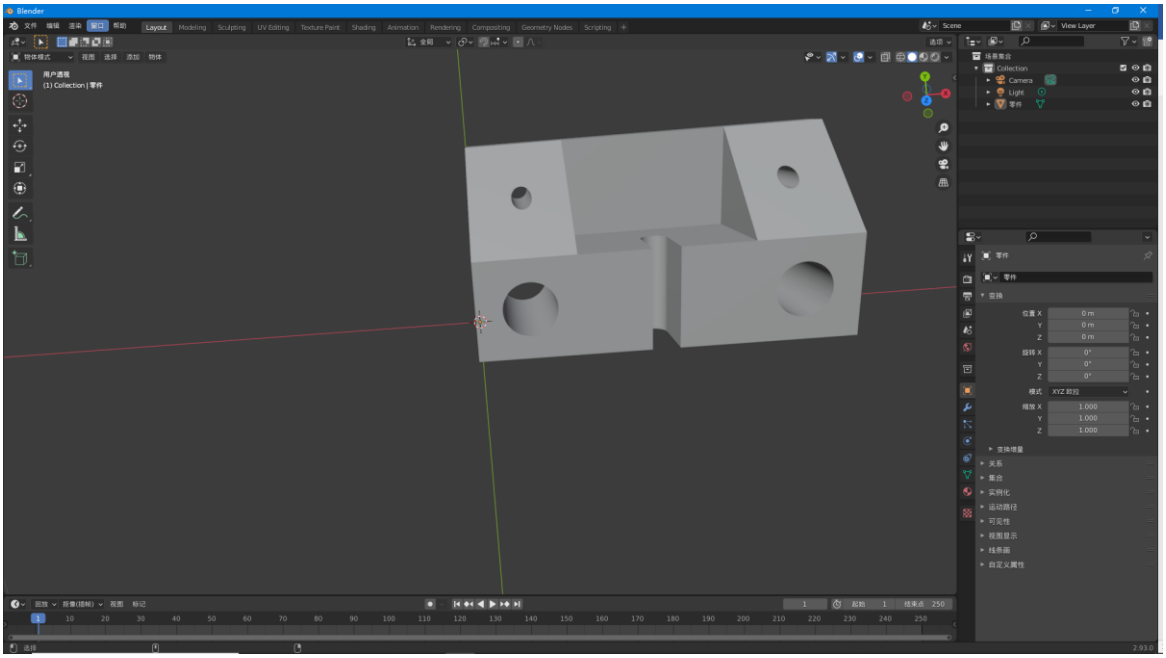
因为把每个表面上面的多边形或者三角形的位置算出来找规律比较困难，我们可以直接利用外部的三维建模软件，建立零件的三维模型，导出来 OBJ 格式的文件，这种文件包含零件表面的多变形顶点数据。然后 opengl 通过读取这些数据来绘制三角面，最终显示零件。

（2）具体过程

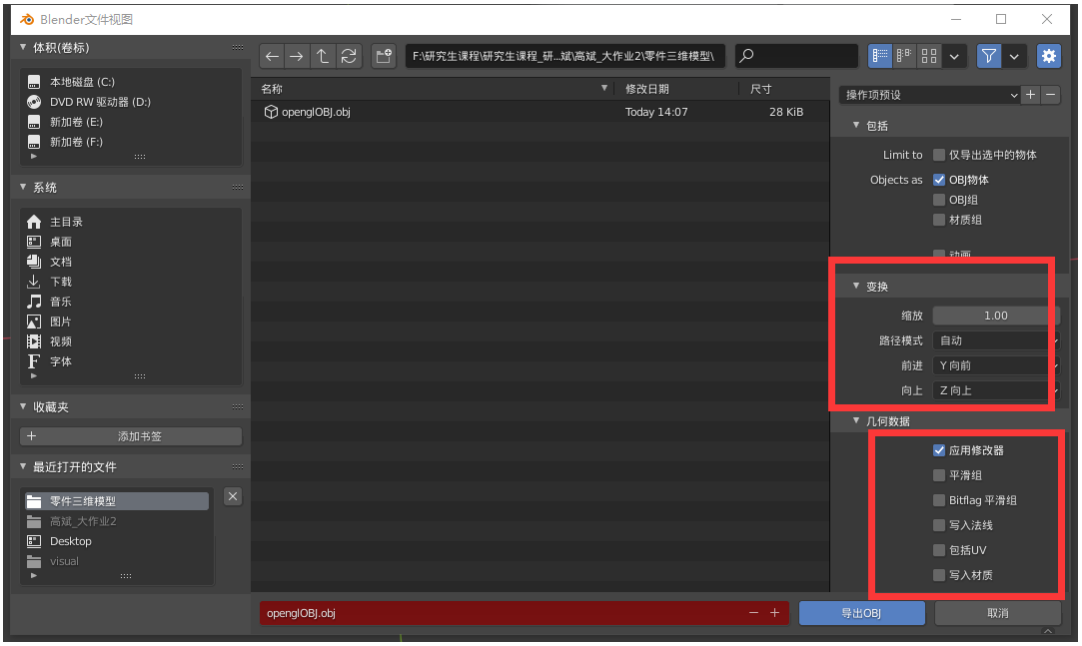
1、 首先在 SolidWorks2018 中建立零件的模型。



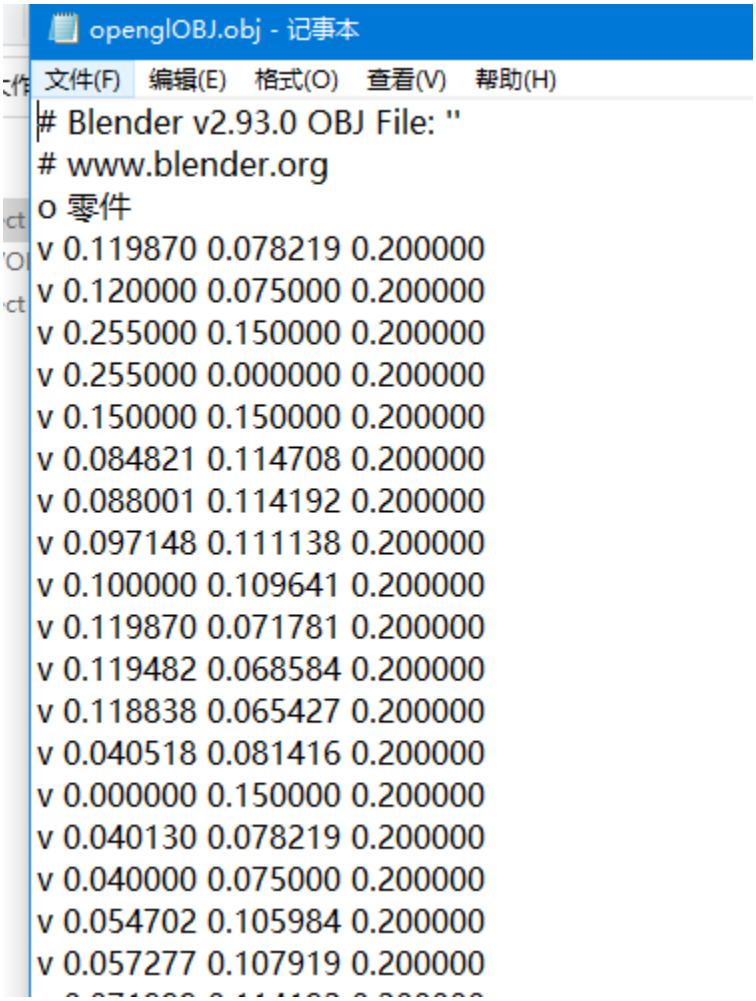
2 将 SolidWorks 建立的三维模型导出为 stl 格式的文件。因为 SolidWorks 本身无法直接导出 obj 格式，所以先转化为 stl，然后用三维建模软件 Blender 将 stl 转为 OBJ, Blender 是一个跨平台三维软件，擅长于各种三维文件格式的转换。



在这里要注意，在导出 obj 时，因为坐标系定义的原因，在 opengl 中显示的坐标系和 blender 中不一样，所以在导出时，需设置坐标系方向，同时顶点文件还需要是单独的数字（在 C++文件读取中会出问题），所以要设置导出的几何数据只包含顶点的坐标数据。



得到的 obj 文件用文本打开是这样：其实就是点数据



3、 在 Visual studio2017 中新建项目，在 C++源文件中加入文件读写的头文件，然后创建一个读取 OBJ 文件的类，其中包含读取函数和绘制函数的方法，以及顶点数据的存储.

```
/*定义一个读取三维obj文件的类*/
class ObjLoader
{
public:
    struct vertex
    {
        float x;
        float y;
        float z;
    };
    ObjLoader(std::string filename);//读取函数
    void Draw();//绘制函数
private:
    std::vector<std::vector<GLfloat>> v;//存放顶点(x,y,z)坐标
    std::vector<std::vector<GLint>> f;//存放面的三个顶点索引
};
```

4、 文件读取的方法实现如下图，具体思路是通过一行一行的读取，判断首字母是 v 还是 f，v 代表的是几何体顶点，f 代表的是面（存的前面的点所在的面，这里是在一个面的三个点的索引组成一个 f）。

```
ObjLoader::ObjLoader(std::string filename)
{
    std::ifstream file(filename);
    std::string line;
    while (getline(file, line))
    {
        if (line.substr(0, 1) == "v")
        {
            std::vector<GLfloat> Point;
            GLfloat x, y, z;
            std::istringstream s(line.substr(2));
            s >> x;
            s >> y;
            s >> z;
            Point.push_back(x);
            Point.push_back(y);
            Point.push_back(z);
            v.push_back(Point);
        }
        else if (line.substr(0, 1) == "f")
        {
            std::vector<GLint> vIndexSets;
            GLint u, v, w;
            std::istringstream vtns(line.substr(2));
            vtns >> u; vtns >> v; vtns >> w;
            vIndexSets.push_back(u - 1);
            vIndexSets.push_back(v - 1);
            vIndexSets.push_back(w - 1);
            f.push_back(vIndexSets);
        }
    }
    file.close();
```

5、 读取的方法实现之后就是实现了，通过读取存储下来的每三个点，绘制一个三角面，进而实现零件的绘制。

```
void ObjLoader::Draw()
{
    glBegin(GL_TRIANGLES);//开始绘制
    for (int i = 0; i < f.size(); i++)
    {
        vertex a, b, c; //三个顶点
        if ((f[i]).size() != 3)
        {
            std::cout << "ERRER::THE SIZE OF f IS NOT 3!" << std::endl;
        }
        else {
            GLint firstVertexIndex = (f[i])[0];//取出顶点索引
            GLint secondVertexIndex = (f[i])[1];
            GLint thirdVertexIndex = (f[i])[2];

            a.x = (v[firstVertexIndex])[0];//第一个顶点
            a.y = (v[firstVertexIndex])[1];
            a.z = (v[firstVertexIndex])[2];

            b.x = (v[secondVertexIndex])[0]; //第二个顶点
            b.y = (v[secondVertexIndex])[1];
            b.z = (v[secondVertexIndex])[2];

            c.x = (v[thirdVertexIndex])[0]; //第三个顶点
            c.y = (v[thirdVertexIndex])[1];
            c.z = (v[thirdVertexIndex])[2];
```

```
            glVertex3f(a.x*1.5, a.y*1.5, -a.z*1.5);//绘制三角面
            glVertex3f(b.x*1.5, b.y*1.5, -b.z*1.5);
            glVertex3f(c.x*1.5, c.y*1.5, -c.z*1.5);
        }
    }
    glEnd();
}
```

2、项目实现流程:

（1）首先初始化窗口。设置一些读取键盘的操作和初始化光照等。

```c
int main(int argc, char* argv[])
{
    /*初始化，创建窗口*/
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(400, 150);
    glutInitWindowSize(800, 800);
    glutCreateWindow("高斌_大作业2");
    /*键盘命令*/
    glutSpecialFunc(SpecialKeys);
    glutKeyboardFunc(KeyboardKeys);
    /*调用绘制*/
    SetupRC();
    glutDisplayFunc(&gaobin_Object);
    glutMainLoop();      //消息循环（处理操作系统等的消息，例如键盘、鼠标事件等）
    return 0;
}
```

（2）键盘命令的实现，因为所需的键盘指令除了空格之外都是特殊按键，所以需要查找空格的 ASCII 码来读取空格键。这里需要两个函数：

```c
/*普通按键*/
void KeyboardKeys(unsigned char key, int x, int y)
{
    if (key == 32)
        zRot -= 5.0f;
}
```

```c
/*特殊按键*/
void SpecialKeys(int key, int x, int y)
{
    if (key == GLUT_KEY_UP)
        xRot -= 5.0f;
    if (key == GLUT_KEY_DOWN)
        xRot += 5.0f;
    if (key == GLUT_KEY_LEFT)
        yRot += 5.0f;
    if (key == GLUT_KEY_RIGHT)
        yRot -= 5.0f;
    if (key == GLUT_KEY_F5)
        zRot += 5.0f;
    glutPostRedisplay();
    glutSwapBuffers();
}
```

（3）键盘指令设置完成后需要写绘制的函数 `void gaobin_Object(void)`，在经过一些初始设置后，首先定义背景，因为背景需要渐变色，这里我们在背景上画一个四边形，然后设置过度颜色；

```c
/*背景*/
glBegin(GL_POLYGON);                //多边形
    glColor3f(1.0f, 0.84f, 0.84f);
    glVertex3f(-1.0f, -1.0f, 0.6f); //左下
    glVertex3f(1.0f, -1.0f,0.6f);   //右下
    glColor3f(1.0f, 0.76f, 0.45f);
    glVertex3f(1.0f, 1.0f,  0.6f);  //右上
    glVertex3f(-1.0f, 1.0f, 0.6f);  //左上
glEnd();
```

（4）接下来就是绘制零件，首先通过事先定义好的类中的读文件方法，读取存在项目同目录中的 obj 文件，`ObjLoader monkey = ObjLoader("openglOBJ.obj");` monkey 就是类的一个对象，存着零件的绘制数据，然后设置旋转参数。在设置旋转参数之前需要用 `glPushMatrix();` 命令将这次操作与其他绘制独立，因为背景也是绘制的，旋转的时候零件动，背景要固定。

```
    /*模型变换和视图变换*/
    glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();                          //保证每一次操
        /*分别绕x,y,z旋转*/
        glRotatef(xRot, 1.0f, 0.0f, 0.0f);
        glRotatef(yRot, 0.0f, 1.0f, 0.0f);
        glRotatef(zRot, 0.0f, 0.0f, 1.0f);
        /*绘制物体*/
        monkey.Draw();
    glPopMatrix();                           //保证每一
```

（5）颜色设置，我们在得到顶点数据后，就可以知道几个孔和 U 型槽的面上点所在的范围，在类的绘制方法中，我们加一个判断，将点在这些范围内的绘制设置为一个颜色，其他的面设置为别的颜色。下面是一个小例子：

```
/*左下角孔颜色*/
if ((a.x >= 0.03&&a.x <= 0.12&&a.y>= 0.03&&a.y<= 0.12)/*第一个顶点在孔内*/
    &&
    (b.x >= 0.03&&b.x <= 0.12&&b.y >= 0.03&&b.y<= 0.12)/*第二个顶点在孔内*/
    &&
    (c.x >= 0.03&&c.x <= 0.12&&c.y >= 0.03&&c.y <= 0.12))/*第三个顶点在孔内*/
{
    glColor3f(1.0, 0.0, 0.0);
}
```

----------其余判断----------

```
/*中间圆槽颜色*/
else if ((a.x >= 0.25&&a.x <= 0.30&&a.z >= 0.1&&a.z<= 0.3)/*第一个顶点在孔内*/
    &&
    (b.x >= 0.25&&b.x <= 0.30&&b.z >= 0.1 && b.z <= 0.3)/*第二个顶点在孔内*/
    &&
    (c.x >= 0.25&&c.x <= 0.30&&c.z >= 0.1 && c.z <= 0.3))/*第三个顶点在孔内*/
{
    glColor3f(0.0, 0.0, 1.0);
}
/*剩余颜色*/
else glColor3f(0.0, 1.0, 0.0);
glVertex3f(a.x*1.5, a.y*1.5, -a.z*1.5);//绘制三角面
glVertex3f(b.x*1.5, b.y*1.5, -b.z*1.5);
glVertex3f(c.x*1.5, c.y*1.5, -c.z*1.5);
```

（6）设置完成后，零件的显示会有颜色重叠，立体感不强，我们在初始化 void SetupRC(void) 函数中加入深度判断函数 glEnable(GL_DEPTH_TEST)，将重叠的部分哪个在前显示哪个。然后加入光照和材质颜色反射。首先是设一个全局的环境光，然后我又加了三个围绕零件的灯。参数实现如下：

灯的位置和照射反向参数：

```
/*灯颜色材质*/
GLfloat lightPos1[] = { 0.28f,0.23f,0.3f,1.0f };      /*第一个灯*/
GLfloat lightDirection1[] = { 0.0,-1.0,-1.0 };

GLfloat lightPos2[] = { 0.0f,0.5f,0.30f,1.0f };      /*第二个灯*/
GLfloat lightDirection2[] = { 1.0,-1.0,-1.0 };

GLfloat lightPos3[] = { -0.5f,-0.0f,0.1f,1.0f };     /*第三个灯*/
GLfloat lightDirection3[] = { 1.0,0.0,0.0 };

GLfloat specular[] = { 1.0f,1.0f,1.0f,1.0f };
GLfloat specrof[] = { 1.0f,1.0f,1.0f,1.0f };
GLfloat ambientLight_whole[] = { 1.0f,1.0f,1.0f,1.0f };
GLfloat ambientLight[] = { 0.2f,0.2f,0.2f,1.0f };
```

光照的设置：

```
/*环境光*/
glEnable(GL_LIGHTING);                                  //开灯
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLight_whole);   //全局环境光，提供轻微的环境光，以便可以看到物
/*1号灯*/
glLightfv(GL_LIGHT0, GL_DIFFUSE, ambientLight);     //光源中的散射光强度（光仅由漫反射和镜面反射组成）
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);        //光源中的镜面反射光强度
glLightfv(GL_LIGHT0, GL_POSITION, lightPos1);        //指定光源位置
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, lightDirection1);  // 聚光灯主轴方向  spot direction
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF,180.0f);          //指定光源的最大散布角 （创建聚光灯，v表示参数是向量（数
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT,500.0f);       //指定聚焦光源指数
glEnable(GL_LIGHT0);                                 //启动0号光源
/*2号灯*/
glLightfv(GL_LIGHT1, GL_DIFFUSE, ambientLight);     //光源中的散射光强度（光仅由漫反射和镜面反射组成）
glLightfv(GL_LIGHT1, GL_SPECULAR, specular);        //光源中的镜面反射光强度
glLightfv(GL_LIGHT1, GL_POSITION, lightPos2);        //指定光源位置
glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, lightDirection2);  // 聚光灯主轴方向  spot direction
glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 180.0f);         //指定光源的最大散布角 （创建聚光灯，v表示参数是向量（数
glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 500.0f);       //指定聚焦光源指数
glEnable(GL_LIGHT1);                                 //启动1号光源
/*3号灯*/
glLightfv(GL_LIGHT2, GL_DIFFUSE, ambientLight);     //光源中的散射光强度（光仅由漫反射和镜面反射组成）
glLightfv(GL_LIGHT2, GL_SPECULAR, specular);        //光源中的镜面反射光强度
glLightfv(GL_LIGHT2, GL_POSITION, lightPos3);        //指定光源位置
glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, lightDirection3);  // 聚光灯主轴方向  spot direction
glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, 60.0f);          //指定光源的最大散布角 （创建聚光灯，v表示参数是向量（数
glLightf(GL_LIGHT2, GL_SPOT_EXPONENT, 500.0f);       //指定聚焦光源指数
glEnable(GL_LIGHT2);                                 //启动2号光源
```

材质的反射设置：

```
glEnable(GL_COLOR_MATERIAL);                          //使用颜色材质，颜色追踪。激活光照的情
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);   //来决定对物体的正面还是反面，对环境光、镜

glMaterialfv(GL_FRONT, GL_SPECULAR, specrof);        //材质的镜面反射颜色
glMateriali(GL_FRONT, GL_SHININESS, 128);            //镜面反射指数
glDepthFunc(GL_LEQUAL);
```

（7）最终效果