9.2OBJ 格式的多边形表示模型.cpp

```
#define GLUT_DISABLE_ATEXIT_HACK
#include "ObjLoader.h"
#include "GLUT.H"

int nearplane_width = 600; //视景体宽度
int nearplane_height = 600; //视景体高度
int nearplane_distance = 500; //视景体近平面与视点距离
int farplane_distance = nearplane_distance + 300; //视景体远平面与视点距离

float eye_x = 20;
float eye_z = 20;
float theta = 0.1;

my_triangle_3DModel cur3DModel; //场景中的3D模型

//模型加载
void init(void)
{
	ObjLoader objModel = ObjLoader("lily-impeller.obj");
	cur3DModel = objModel.my_3DModel;
}

//绘制内容
void display(void)
{
	glClearColor(1.f, 1.f, 1.f, 0.f);
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glColor3f(157.0 / 256, 195.0 / 256, 230.0 / 256);

	//开始绘制
	glBegin(GL_TRIANGLES);
	for (int i = 0; i < cur3DModel.faceSets.size(); i++)
	{
		//取出顶点序号获得相应顶点坐标
		my_3D_point_coord point1, point2, point3;
		int firstPointIndex = cur3DModel.faceSets[i].first_point_index;
		int secondPointIndex = cur3DModel.faceSets[i].second_point_index;
		int thirdPointIndex = cur3DModel.faceSets[i].third_point_index;

		point1 = cur3DModel.pointSets[firstPointIndex]; //第一个顶点
		point2 = cur3DModel.pointSets[secondPointIndex]; //第二个顶点
		point3 = cur3DModel.pointSets[thirdPointIndex]; //第三个顶点
```

```
            //取出法向序号获得相应法向
            my_3Dvector vector1, vector2, vector3;
            int firstNormalIndex = cur3DModel.faceSets[i].first_point_normal_index;
            int secondNormalIndex = cur3DModel.faceSets[i].second_point_normal_index;
            int thirdNormalIndex = cur3DModel.faceSets[i].third_point_normal_index;

            vector1 = cur3DModel.pointNormalSets[firstNormalIndex]; //第一个点的法向量
            vector2 = cur3DModel.pointNormalSets[secondNormalIndex]; //第二个点的法向量
            vector3 = cur3DModel.pointNormalSets[thirdNormalIndex]; //第三个点的法向量

            //取出纹理坐标序号获得相应纹理坐标
            my_2D_Texture_coord texture1, texture2, texture3;
            int firstTextureIndex = cur3DModel.faceSets[i].first_point_texture_index;
            int secondTextureIndex = cur3DModel.faceSets[i].second_point_texture_index;
            int thirdTextureIndex = cur3DModel.faceSets[i].third_point_texture_index;

            texture1 = cur3DModel.pointTextureSets[firstTextureIndex]; //第一个点的纹理
            texture2 = cur3DModel.pointTextureSets[secondTextureIndex]; //第二个点的纹理
            texture3 = cur3DModel.pointTextureSets[thirdTextureIndex]; //第三个点的纹理

            //绘制三角网格面
            glNormal3f(vector1.dx, vector1.dy, vector1.dz); //绑定顶点的法向
            glTexCoord2f(texture1.u, texture1.v); //绑定顶点的纹理
            glVertex3f(point1.x, point1.y, point1.z); //绘制顶点

            glNormal3f(vector2.dx, vector2.dy, vector2.dz);
            glTexCoord2f(texture2.u, texture2.v);
            glVertex3f(point2.x, point2.y, point2.z);

            glNormal3f(vector3.dx, vector3.dy, vector3.dz);
            glTexCoord2f(texture3.u, texture3.v);
            glVertex3f(point3.x, point3.y, point3.z);
        }
        glEnd();

        glutSwapBuffers();
}

//键盘交互事件
void keyboard(unsigned char key, int x, int y)
{
        switch (key)
        {
```

```
        case 'z': //打开深度缓冲测试
        case 'Z':
        {
            glEnable(GL_DEPTH_TEST); //打开深度缓冲测试
            glDepthFunc(GL_LEQUAL); //判断遮挡关系时，离视点近的物体遮挡离视点远的
物体
            glutPostRedisplay();
            break;
        }
        case 'c': //关闭深度缓冲测试
        case 'C':
        {
            glDisable(GL_DEPTH_TEST); //关闭深度缓冲测试
            glutPostRedisplay();
            break;
        }
        case 'l': //打开灯光
        case 'L':
        {
            GLfloat light_position[] = { 100.0, 100.0, 100.0, 0.0 };
            GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
            glLightfv(GL_LIGHT0, GL_POSITION, light_position);
            glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
            glEnable(GL_LIGHTING);
            glEnable(GL_LIGHT0);
            GLfloat diffuse[] = { 1, 0, 0, 1.0 };
            glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);

            glutPostRedisplay();
            break;
        }
        case 27:
            exit(0);
            break;
        }
}

//投影方式、modelview方式设置
void reshape(int w, int h)
{
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
```

```
    if (w <= h)
        glOrtho(-0.5 * nearplane_width, 0.5 * nearplane_width, -0.5 * nearplane_height *
(GLfloat)nearplane_height / (GLfloat)nearplane_width, 0.5 * nearplane_height *
(GLfloat)nearplane_height / (GLfloat)nearplane_width,
            -nearplane_distance, farplane_distance); //相对于视点
    else
        glOrtho(-0.5 * nearplane_width, 0.5 * nearplane_width, -0.5 * nearplane_height *
(GLfloat)nearplane_width / (GLfloat)nearplane_height, 0.5 * nearplane_height *
(GLfloat)nearplane_width / (GLfloat)nearplane_height,
            -nearplane_distance, farplane_distance);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    gluLookAt(eye_x, 10, eye_z, 0, 0, 0, 0, 1, 0);
}

//鼠标交互事件
void mouse(int button, int state, int x, int y)
{
    switch (button)
    {
    case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN)
        {
            eye_x = eye_x * cosf(-theta) + eye_z * sinf(-theta);
            eye_z = eye_z * cosf(-theta) - eye_x * sinf(-theta);
            reshape(nearplane_width, nearplane_height);
            glShadeModel(GL_FLAT);
            glutPostRedisplay();
        }
        break;
    case GLUT_RIGHT_BUTTON:
        if (state == GLUT_DOWN)
        {
            eye_x = eye_x * cosf(theta) + eye_z * sinf(theta);
            eye_z = eye_z * cosf(theta) - eye_x * sinf(theta);
            reshape(nearplane_width, nearplane_height);
            glutPostRedisplay();
        }
        break;
    default:
        break;
    }
```

```
}

//主调函数
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(nearplane_width, nearplane_height);
	glutInitWindowPosition(100, 100);
	glutCreateWindow("三维模型加载");

	init();

	glutReshapeFunc(reshape);
	glutDisplayFunc(display);
	glutKeyboardFunc(keyboard);
	glutMouseFunc(mouse);
	glutMainLoop();
	return 0;
}
```

ObjLoader.h

```
#pragma once
#include <vector>
#include <string>
#include <iostream>
#include <fstream>

using namespace std;

//纹理坐标数据结构
struct my_2D_Texture_coord
{
	float u;//u方向上的坐标
	float v;//v方向上的坐标

	my_2D_Texture_coord() {}

	~my_2D_Texture_coord() {}

	my_2D_Texture_coord(float ui, float vi)
	{
		u = ui;
		v = vi;
```

```
    }
};

//顶点数据结构
struct my_3D_point_coord
{
    float x;//x轴坐标值
    float y;//y轴坐标值
    float z;//z轴坐标值

    my_3D_point_coord() {}

    ~my_3D_point_coord() {}

    my_3D_point_coord(float xi, float yi, float zi)
    {
        x = xi;
        y = yi;
        z = zi;
    }
};

//面数据结构，即构成三维图形的一个三角网格面
struct my_triangle_indices
{
    int first_point_index;//第一个点序号
    int first_point_texture_index;//第一个纹理坐标序号
    int first_point_normal_index;//第一个点法向序号

    int second_point_index;//第二个点序号
    int second_point_texture_index;//第二个纹理坐标序号
    int second_point_normal_index;//第二个点法向序号

    int third_point_index;//第三个点序号
    int third_point_texture_index;//第三个纹理坐标序号
    int third_point_normal_index;//第三个点法向序号
};

class my_3Dvector
{
public:
    float dx;
    float dy;
    float dz;
```

```
        float len;
public:
        my_3Dvector() {}
        ~my_3Dvector() {}

        my_3Dvector(float x, float y, float z)
        {
                dx = x;
                dy = y;
                dz = z;

                len = sqrtf(powf(dx, 2) + powf(dy, 2) + powf(dz, 2));
        }
};

//三维空间中的三角网格模型
struct my_triangle_3DModel
{
        float transparency = 0;
        float reflection = 0;

        //以下用于计算直接光照能量
        float material_ambient_rgb_reflection[3] = { 0,0,0 }; //环境光反射系数，初始不反射
        float material_diffuse_rgb_reflection[3] = { 0,0,0 };//漫反射系数，初始不反射
        float material_specular_rgb_reflection[3] = { 0,0,0 };//镜面光反射系数，初始不反射
        float ns = 0;//聚光指数，初始不聚光

        vector<my_3D_point_coord> pointSets;//存放模型所有顶点
        vector<my_3Dvector> pointNormalSets;//存放模型所有顶点的法向
        vector<my_2D_Texture_coord> pointTextureSets;//存放模型所有纹理坐标
        vector<my_triangle_indices> faceSets;//存放模型所有三角网格面

        //模型尺寸
        float max_x = -1e8, min_x = 1e8;
        float max_y = -1e8, min_y = 1e8;
        float max_z = -1e8, min_z = 1e8;

        void modify_color_configuration(float transparency, float reflection, float
ambient_reflection[], float diffuse_reflection[], float specular_reflection[], float ns)
        {
                this->transparency = transparency;
                this->reflection = reflection;

                //以下用于计算直接光照能量       memcpy用于把资源内存拷到目标内存中
```

```
            memcpy(this->material_ambient_rgb_reflection, ambient_reflection, 3 * sizeof(float));
            memcpy(this->material_diffuse_rgb_reflection, diffuse_reflection, 3 * sizeof(float));
            memcpy(this->material_specular_rgb_reflection, specular_reflection, 3 * sizeof(float));
            this->ns = ns;
        }
};

//模型加载器
class ObjLoader
{
public:
    my_triangle_3DModel my_3DModel;

public:
    ObjLoader() {}
    ObjLoader(string filename) //构造函数
    {
        string line;
        fstream f;
        f.open(filename, ios::in);
        if (!f.is_open()) {
            cout << "Something Went Wrong When Opening Objfiles" << endl;
        }

        while (!f.eof())
        {
            getline(f, line);//拿到obj文件中一行，作为一个字符串
            if (line.find("#") != -1)
                continue;

            line.append(" ");
            vector<string> parameters;
            string ans = "";

            for (unsigned int i = 0; i < line.length(); i++)
            {
                char ch = line[i];
                if (ch != ' ')
                {
                    ans += ch;
                }
                else if (ans != "")
                {
                    parameters.push_back(ans); //取出字符串中的元素，以空格切分
```

```
                ans = "";
            }
        }

        if (parameters.size() == 4 || parameters.size() == 3)
        {
            if (parameters[0] == "v")   //顶点
            {
                my_3D_point_coord curPoint;
                curPoint.x = atof(parameters[1].c_str());
                my_3DModel.max_x = my_3DModel.max_x < curPoint.x ? curPoint.x :
my_3DModel.max_x;
                my_3DModel.min_x = my_3DModel.min_x > curPoint.x ? curPoint.x :
my_3DModel.min_x;

                curPoint.y = atof(parameters[2].c_str());
                my_3DModel.max_y = my_3DModel.max_y < curPoint.y ? curPoint.y :
my_3DModel.max_y;
                my_3DModel.min_y = my_3DModel.min_y > curPoint.y ? curPoint.y :
my_3DModel.min_y;

                curPoint.z = atof(parameters[3].c_str());
                my_3DModel.max_z = my_3DModel.max_z < curPoint.z ? curPoint.z :
my_3DModel.max_z;
                my_3DModel.min_z = my_3DModel.min_z > curPoint.z ? curPoint.z :
my_3DModel.min_z;

                my_3DModel.pointSets.push_back(curPoint);
            }
            else if (parameters[0] == "vn") //顶点的法向量
            {
                my_3Dvector curPointNormal;
                curPointNormal.dx = atof(parameters[1].c_str());
                curPointNormal.dy = atof(parameters[2].c_str());
                curPointNormal.dz = atof(parameters[3].c_str());
                my_3DModel.pointNormalSets.push_back(curPointNormal);
            }
            else if (parameters[0] == "vt") //纹理坐标
            {
                my_2D_Texture_coord curTextureCoord;
                curTextureCoord.u = atof(parameters[1].c_str());
                curTextureCoord.v = atof(parameters[2].c_str());
                my_3DModel.pointTextureSets.push_back(curTextureCoord);
            }
```

```
                else if (parameters[0] == "f") //面，顶点索引/纹理 uv 索引/法向索引
                {
                    //因为顶点索引在obj文件中是从1开始的，而我们存放的顶点vector是从0开始的，因此要减1
                    my_triangle_indices curTri;
                    curTri.first_point_index = atoi(parameters[1].substr(0, parameters[1].find_first_of('/')).c_str()) - 1;
                    parameters[1] = parameters[1].substr(parameters[1].find_first_of('/') + 1);
                    curTri.first_point_texture_index = atoi(parameters[1].substr(0, parameters[1].find_first_of('/')).c_str()) - 1;
                    parameters[1] = parameters[1].substr(parameters[1].find_first_of('/') + 1);
                    curTri.first_point_normal_index = atoi(parameters[1].substr(0, parameters[1].find_first_of('/')).c_str()) - 1;

                    curTri.second_point_index = atoi(parameters[2].substr(0, parameters[2].find_first_of('/')).c_str()) - 1;
                    parameters[2] = parameters[2].substr(parameters[2].find_first_of('/') + 1);
                    curTri.second_point_texture_index = atoi(parameters[2].substr(0, parameters[2].find_first_of('/')).c_str()) - 1;
                    parameters[2] = parameters[2].substr(parameters[2].find_first_of('/') + 1);
                    curTri.second_point_normal_index = atoi(parameters[2].substr(0, parameters[2].find_first_of('/')).c_str()) - 1;

                    curTri.third_point_index = atoi(parameters[3].substr(0, parameters[3].find_first_of('/')).c_str()) - 1;
                    parameters[3] = parameters[3].substr(parameters[3].find_first_of('/') + 1);
                    curTri.third_point_texture_index = atoi(parameters[3].substr(0, parameters[3].find_first_of('/')).c_str()) - 1;
                    parameters[3] = parameters[3].substr(parameters[3].find_first_of('/') + 1);
                    curTri.third_point_normal_index = atoi(parameters[3].substr(0, parameters[3].find_first_of('/')).c_str()) - 1;

                    my_3DModel.faceSets.push_back(curTri);
                }
            }
        }
        f.close();
    }
};
```