**计算机图形学**

**实验报告**

**exp7**

**姓名：宋灵冰**

**学号：24320162202900**

**目录**

[**Task1** 1](#_Toc514846213)

[功能说明： 1](#_Toc514846214)

[实现过程： 1](#_Toc514846215)

[效果预览： 2](#_Toc514846216)

[源代码： 2](#_Toc514846217)

[**Task2** 6](#_Toc514846218)

[功能说明： 6](#_Toc514846219)

[效果预览： 6](#_Toc514846220)

[源代码： 6](#_Toc514846221)

**Task1**

功能说明：

实现为每个顶点计算Phong照明强度的功能。输入为ply模型，例如之前提供的模型，不使用OpenGL的光照功能，自行完成Phong照明模型，具体要求为：

（1） 设置视点、光源，以及各个光源系数（要求漫射、镜射、环境光等），设置物体材质。

（2） 对每个顶点用Phong照明模型计算顶点颜色，用glColor3f上色

（3） 对于三角形内的颜色，用OpenGL的glShadeModel( GL\_SMOOTH) 填色模型填充。

效果预览：

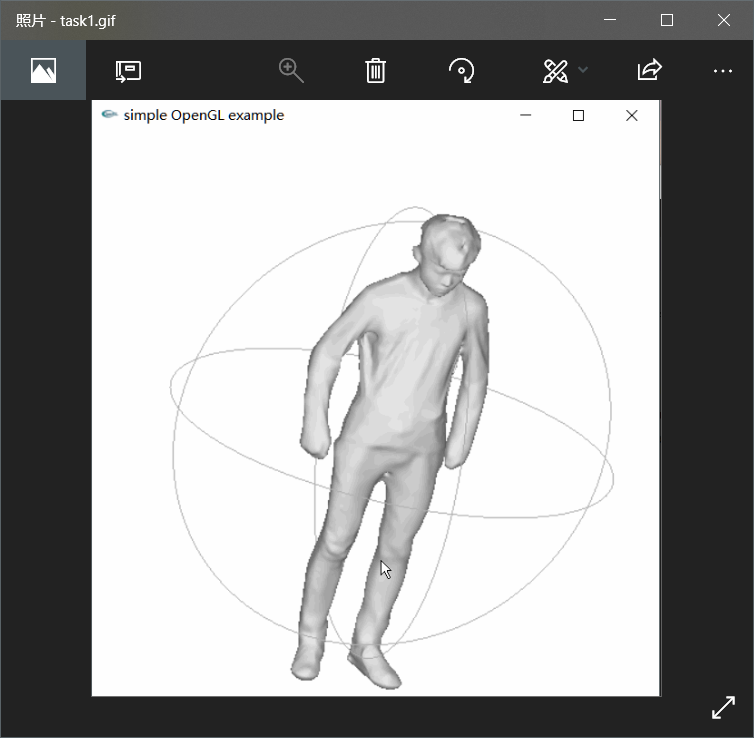


图1.1 Task1程序预览（查看附件动图）

源代码：

//lightManager.h

#ifndef \_LIGHT\_MANAGER\_H\_

#define \_LIGHT\_MANAGER\_H\_

#include <gl/glut.h>

#include <math.h>

class Light {

public:

Light();

void setLightPosition(const GLfloat\* params);

void setLightPosition(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC);

void setAmbientLight(const GLfloat\* params);

void setDiffuseLight(const GLfloat\* params);

void setSpecularLight(const GLfloat\* params);

void setAmbientLight(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC);

void setDiffuseLight(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC);

void setSpecularLight(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC);

void setAmbientMaterial(const GLfloat\* params);

void setDiffuseMaterial(const GLfloat\* params);

void setSpecularMaterial(const GLfloat\* params);

void setAmbientMaterial(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC);

void setDiffuseMaterial(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC);

void setSpecularMaterial(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC);

void setLightColor(const GLfloat\* camPosition, const GLfloat\* pointPosition, const GLfloat\* vector);

private:

GLfloat lightPosition[3];

//光源

GLfloat ambientLight[3];

GLfloat diffuseLight[3];

GLfloat specularLight[3];

//材质

GLfloat ambientMaterial[3];

GLfloat diffuseMaterial[3];

GLfloat specularMaterial[3];

//镜面系数

GLfloat specularStrength;

void setMatrix(GLfloat\* result, const GLfloat\* params);

GLfloat max(GLfloat numA, GLfloat numB);

void normalize(GLfloat\* result, const GLfloat\* origin);

GLfloat dot(const GLfloat\* matA, const GLfloat\* matB);

void reflect(GLfloat\* result, const GLfloat\* matA, const GLfloat\* matB);

};

Light::Light() {

for (int i = 0; i < 3; i++) {

lightPosition[i] = 0.0;

ambientLight[i] = 0.0;

diffuseLight[i] = 1.0;

specularLight[i] = 1.0;

ambientMaterial[i] = 0.2;

diffuseMaterial[i] = 0.8;

specularMaterial[i] = 0.0;

}

specularStrength = 3.0;

}

void Light::setLightColor(const GLfloat\* camPosition, const GLfloat\* pointPosition, const GLfloat\* vector) {

//该顶点在对应光照下的颜色

GLfloat ambientColor[3], diffuseColor[3], specularColor[3];

//求环境光颜色

for (int i = 0; i < 3; i++) {

ambientColor[i] = ambientLight[i] \* ambientMaterial[i];

}

//规范化的法向

GLfloat normal[3];

normalize(normal, vector);

//入射向量，反射向量，反向的入射向量

GLfloat inVector[3], outVector[3], reverseInVector[3];

for (int i = 0; i < 3; i++) {

inVector[i] = pointPosition[i] - lightPosition[i];

outVector[i] = lightPosition[i] - 2 \* normal[i] \* dot(normal, inVector);

reverseInVector[i] = -inVector[i];

}

//漫反射向量

GLfloat diffuseVector[3];

normalize(diffuseVector, reverseInVector);

//漫反射系数

GLfloat diffuseFactor = max(dot(diffuseVector, normal), 0.0);

//求漫反射光颜色

for (int i = 0; i < 3; i++) {

diffuseColor[i] = diffuseFactor \* diffuseLight[i] \* diffuseMaterial[i];

}

//镜面反射向量

GLfloat specularVector[3];

normalize(specularVector, outVector);

//视线方向向量

GLfloat viewVector[3];

for (int i = 0; i < 3; i++) {

viewVector[i] = camPosition[i] - pointPosition[i];

}

//镜面反射系数

GLfloat specularFactor = pow(max(dot(specularVector, viewVector), 0.0), specularStrength);

//求镜面反射光颜色

for (int i = 0; i < 3; i++) {

specularColor[i] = specularFactor \* specularLight[i] \* specularMaterial[i];

}

//最终颜色

GLfloat resultColor[3];

for (int i = 0; i < 3; i++) {

resultColor[i] = ambientColor[i] + diffuseColor[i] + specularColor[i];

}

glColor3fv(resultColor);

}

void Light::setLightPosition(const GLfloat\* params) {

setMatrix(lightPosition, params);

}

void Light::setAmbientLight(const GLfloat\* params) {

setMatrix(ambientLight, params);

}

void Light::setDiffuseLight(const GLfloat\* params) {

setMatrix(diffuseLight, params);

}

void Light::setSpecularLight(const GLfloat\* params) {

setMatrix(specularLight, params);

}

void Light::setAmbientMaterial(const GLfloat\* params) {

setMatrix(ambientMaterial, params);

}

void Light::setDiffuseMaterial(const GLfloat\* params) {

setMatrix(diffuseMaterial, params);

}

void Light::setSpecularMaterial(const GLfloat\* params) {

setMatrix(specularMaterial, params);

}

void Light::setLightPosition(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC) {

lightPosition[0] = paramA;

lightPosition[1] = paramB;

lightPosition[2] = paramC;

}

void Light::setAmbientLight(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC) {

ambientLight[0] = paramA;

ambientLight[1] = paramB;

ambientLight[2] = paramC;

}

void Light::setDiffuseLight(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC) {

diffuseLight[0] = paramA;

diffuseLight[1] = paramB;

diffuseLight[2] = paramC;

}

void Light::setSpecularLight(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC) {

specularLight[0] = paramA;

specularLight[1] = paramB;

specularLight[2] = paramC;

}

void Light::setAmbientMaterial(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC) {

ambientMaterial[0] = paramA;

ambientMaterial[1] = paramB;

ambientMaterial[2] = paramC;

}

void Light::setDiffuseMaterial(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC) {

diffuseMaterial[0] = paramA;

diffuseMaterial[1] = paramB;

diffuseMaterial[2] = paramC;

}

void Light::setSpecularMaterial(const GLfloat paramA, const GLfloat paramB, const GLfloat paramC) {

specularMaterial[0] = paramA;

specularMaterial[1] = paramB;

specularMaterial[2] = paramC;

}

void Light::setMatrix(GLfloat\* result, const GLfloat\* params) {

for (int i = 0; i < 3; i++) {

result[i] = params[i];

}

}

GLfloat Light::max(GLfloat numA, GLfloat numB) {

return (numA > numB) ? numA : numB;

}

void Light::normalize(GLfloat\* result, const GLfloat\* origin) {

float length = sqrt(origin[0] \* origin[0] + origin[1] \* origin[1] + origin[2] \* origin[2]);

for (int i = 0; i < 3; i++) {

result[i] = origin[i] / length;

}

}

GLfloat Light::dot(const GLfloat\* matA, const GLfloat\* matB) {

GLfloat result = 0.0;

for (int i = 0; i < 3; i++) {

result += matA[i] \* matB[i];

}

return result;

}

#endif // !\_LIGHT\_MANAGER\_H\_

//main.cpp

/\*

\* 该程序在轨迹球程序基础上实现以下功能

\* 1. 实现光照的函数，包括漫反射、镜面反射和环境光

\* 2.对每个顶点应用光照效果

\*\*\*\*\*\*\*

\* @author：宋灵冰

\*/

#include <iostream>

#include <vector>

#include <string.h>

#include <stdio.h>

#include <gl/glut.h>

#include "lightManager.h"

using namespace std;

typedef struct {

GLdouble x, y, z;

}Point3d;

typedef struct {

int index[3];

}FaceIndex;

enum mouseButton { LEFT\_CLICK, MIDDLE\_CLICK, NO\_CLICK };

GLfloat scaleLevel = 1.0;

GLfloat moveX = 0.0;

GLfloat moveY = 0.0;

const GLfloat moveZ = 0.0;

GLfloat tempX = 0.0;

GLfloat tempY = 0.0;

GLfloat tempZ = 0.0;

GLfloat normalX = 0.0;

GLfloat normalY = 0.0;

GLfloat normalZ = 0.0;

float theta = 0;

int orgX, orgY;

bool moveState = false;

int width = 500;

int height = 500;

int rangeX = 1000;

int rangeY = 1000;

const double ballRadius = 800;

const double PI = 3.1415926535;

int pointNum, faceNum;

Point3d corePoint = { 0,0,0 };

float rotateR1[16], rotateR2[16], translateR1[16], translateR2[16], tempR[16];

vector<Point3d> pointContainer, normalContainer;

vector<FaceIndex> faceContainer;

GLenum mode = GL\_FILL;

mouseButton mouseMode;

Light light;

GLfloat camPosition[] = { 0, 0, 0, 0, 0,-1, 0, 1, 0 };

GLfloat lightPosition[] = { 0.0f, 0.0f, 5000.0f, 1.0f }; //定义光源位置

void readPly(const char\* fileName) {

FILE\* f = fopen(fileName, "r");

if (!f)

return;

char strBuffer[255];

for (int i = 0; i < 3; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

fscanf(f, "element vertex %d\n", &pointNum);

for (int i = 0; i < 6; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

fscanf(f, "element face %d\n", &faceNum);

for (int i = 0; i < 2; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

Point3d tempPoint;

Point3d tempNormal;

FaceIndex tempFace;

for (int i = 0; i < pointNum; i++) {

fscanf(f, "%lf %lf %lf %lf %lf %lf \n",

&tempPoint.x, &tempPoint.y, &tempPoint.z,

&tempNormal.x, &tempNormal.y, &tempNormal.z);

corePoint.x += tempPoint.x;

corePoint.y += tempPoint.y;

corePoint.z += tempPoint.z;

pointContainer.push\_back(tempPoint);

normalContainer.push\_back(tempNormal);

}

corePoint.x /= (double)pointNum;

corePoint.y /= (double)pointNum;

corePoint.z /= (double)pointNum;

for (int i = 0; i < faceNum; i++) {

fscanf(f, "3 %d %d %d \n",

&tempFace.index[0], &tempFace.index[1], &tempFace.index[2]);

faceContainer.push\_back(tempFace);

}

fclose(f);

}

void menuFunc(int value) {

switch (value) {

case 1:

mode = GL\_POINT;

break;

case 2:

mode = GL\_LINE;

break;

case 3:

mode = GL\_FILL;

break;

default:

break;

}

glutPostRedisplay();

}

//设置初始属性

void myinit() {

//设置背景颜色

glClearColor(1.0, 1.0, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1000.0, 1000.0, -1000.0, 1000.0, -10000.0, 10000.0);

//设置观察参数

glMatrixMode(GL\_MODELVIEW);

//启用深度缓存

glEnable(GL\_DEPTH\_TEST);

glEnable(GLUT\_MULTISAMPLE);

//初始化坐标系

glLoadIdentity();

GLfloat color[3] = {0.5,0.5,0.5};

light.setLightPosition(lightPosition);

light.setAmbientLight(color);

readPly("lizhenxiout-repaired.ply");

glutCreateMenu(menuFunc);

glutAddMenuEntry("Point Mode", 1);

glutAddMenuEntry("Line Mode", 2);

glutAddMenuEntry("Mesh Mode", 3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glShadeModel(GL\_SMOOTH);

glGetFloatv(GL\_MODELVIEW\_MATRIX, rotateR1);

glGetFloatv(GL\_MODELVIEW\_MATRIX, translateR1);

}

void drawModel() {

FaceIndex tempFace;

Point3d tempFaceVertex[3];

Point3d tempFaceNormal[3];

GLfloat color[2][3] = { { 0.6,0.6,0.6 },{ 0.6,0.6,0.6 } };

light.setAmbientMaterial(color[0]);

light.setDiffuseMaterial(color[1]);

glPolygonMode(GL\_FRONT\_AND\_BACK, mode);

glGetFloatv(GL\_MODELVIEW\_MATRIX, tempR);

glBegin(GL\_TRIANGLES);

{

for (int i = 0; i < faceNum; i++) {

tempFace = faceContainer[i];

for (int j = 0; j < 3; j++) {

tempFaceVertex[j] = pointContainer[tempFace.index[j]];

tempFaceNormal[j] = normalContainer[tempFace.index[j]];

GLfloat vertex[3], normal[3];

vertex[0] = tempR[0] \* tempFaceVertex[j].x + tempR[4] \* tempFaceVertex[j].y + tempR[8] \* tempFaceVertex[j].z + tempR[12];

vertex[1] = tempR[1] \* tempFaceVertex[j].x + tempR[5] \* tempFaceVertex[j].y + tempR[9] \* tempFaceVertex[j].z + tempR[13];

vertex[2] = tempR[2] \* tempFaceVertex[j].x + tempR[6] \* tempFaceVertex[j].y + tempR[10] \* tempFaceVertex[j].z + tempR[14];

normal[0] = tempR[0] \* tempFaceNormal[j].x + tempR[4] \* tempFaceNormal[j].y + tempR[8] \* tempFaceNormal[j].z;

normal[1] = tempR[1] \* tempFaceNormal[j].x + tempR[5] \* tempFaceNormal[j].y + tempR[9] \* tempFaceNormal[j].z;

normal[2] = tempR[2] \* tempFaceNormal[j].x + tempR[6] \* tempFaceNormal[j].y + tempR[10] \* tempFaceNormal[j].z;

light.setLightColor(camPosition, vertex, normal);

glVertex3d(tempFaceVertex[j].x, tempFaceVertex[j].y, tempFaceVertex[j].z);

}

}

}

glEnd();

}

void drawTraceBall() {

GLfloat color[3][3] = { { 1,0,0 },{ 0,1,0 },{ 0,0,1 } };

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[0]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[0]);

glTranslatef(corePoint.x - ballRadius, corePoint.y, corePoint.z);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[1]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[1]);

glTranslatef(corePoint.x - ballRadius, corePoint.y, corePoint.z);

glRotatef(90, 1.0, 0.0, 0.0);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[2]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[2]);

glTranslatef(corePoint.x, corePoint.y - ballRadius, corePoint.z);

glRotatef(90, 0.0, 0.0, 1.0);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

}

//用于初始显示图像

void displayScene() {

//清空窗口

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotatef(theta, normalX, normalY, normalZ);

glGetFloatv(GL\_MODELVIEW\_MATRIX, rotateR2);

glLoadMatrixf(rotateR2);

glMultMatrixf(rotateR1);

glGetFloatv(GL\_MODELVIEW\_MATRIX, rotateR1);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(moveX \* 3.0, moveY \* 3.0, moveZ);

glGetFloatv(GL\_MODELVIEW\_MATRIX, translateR2);

glLoadMatrixf(translateR2);

glMultMatrixf(translateR1);

glGetFloatv(GL\_MODELVIEW\_MATRIX, translateR1);

//scale rotate translate

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glScalef(scaleLevel, scaleLevel, scaleLevel);

glTranslatef(corePoint.x, corePoint.y, corePoint.z);

glGetFloatv(GL\_MODELVIEW\_MATRIX, tempR);

glLoadMatrixf(tempR);

glMultMatrixf(translateR1);

glMultMatrixf(rotateR1);

glTranslatef(-corePoint.x, -corePoint.y, -corePoint.z);

drawModel();

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(corePoint.x, corePoint.y, corePoint.z);

glGetFloatv(GL\_MODELVIEW\_MATRIX, tempR);

glLoadMatrixf(tempR);

glMultMatrixf(rotateR1);

glTranslatef(-corePoint.x, -corePoint.y, -corePoint.z);

drawTraceBall();

glFlush();

}

//实现缩放回调函数，使用[]控制缩放

void scaleFunc(unsigned char key, int x, int y) {

switch (key) {

case '[':

if (scaleLevel > 0)

scaleLevel -= 0.1;

break;

case ']':

scaleLevel += 0.1;

break;

case 27:

exit(0);

break;

default:

break;

}

glutPostRedisplay();

}

//设置鼠标左键、中键回调函数，使其分别实现对模型进行旋转、平移

void mouseFunc(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && !moveState) {

mouseMode = LEFT\_CLICK;

orgX = x;

orgY = y;

moveState = true;

}

if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN && !moveState) {

mouseMode = MIDDLE\_CLICK;

orgX = x;

orgY = y;

moveState = true;

}

if (state == GLUT\_UP && moveState) {

moveState = false;

mouseMode = NO\_CLICK;

moveX = moveY = 0;

theta = 0;

}

glutPostRedisplay();

}

void calculateAngle(int x, int y) {

float screenCenterX = width / 2.0;

float screenCenterY = height / 2.0;

float radiusRange = ballRadius / (float)rangeX\*screenCenterX;

float vectorOrgX = orgX - screenCenterX;

float vectorOrgY = height - orgY - screenCenterY;

float orgDistance;

if (vectorOrgX\*vectorOrgX + vectorOrgY \* vectorOrgY > radiusRange\*radiusRange) {

float radius = sqrt(vectorOrgX\*vectorOrgX + vectorOrgY \* vectorOrgY);

vectorOrgX \*= (radiusRange / radius);

vectorOrgY \*= (radiusRange / radius);

orgDistance = radiusRange;

} else {

orgDistance = sqrt(vectorOrgX \* vectorOrgX + vectorOrgY \* vectorOrgY);

}

float vectorCurX = x - screenCenterX;

float vectorCurY = height - y - screenCenterY;

float curDistance;

if (vectorCurX\*vectorCurX + vectorCurY \* vectorCurY > radiusRange\*radiusRange) {

float radius = sqrt(vectorCurX\*vectorCurX + vectorCurY \* vectorCurY);

vectorCurX \*= (radiusRange / radius);

vectorCurY \*= (radiusRange / radius);

curDistance = radiusRange;

} else {

curDistance = sqrt(vectorCurX \* vectorCurX + vectorCurY \* vectorCurY);

}

float vectorOrgZ = sqrt(radiusRange \* radiusRange - orgDistance \* orgDistance);

float vectorCurZ = sqrt(radiusRange \* radiusRange - curDistance \* curDistance);

normalX = vectorOrgY \* vectorCurZ - vectorCurY \* vectorOrgZ;

normalY = vectorOrgZ \* vectorCurX - vectorOrgX \* vectorCurZ;

normalZ = vectorOrgX \* vectorCurY - vectorOrgY \* vectorCurX;

orgX = x;

orgY = y;

theta = asin(sqrt(normalX \* normalX + normalY \* normalY + normalZ \* normalZ) / (sqrt(vectorOrgX\*vectorOrgX + vectorOrgY \* vectorOrgY + vectorOrgZ \* vectorOrgZ)\*sqrt(vectorCurX\*vectorCurX + vectorCurY \* vectorCurY + vectorCurZ \* vectorCurZ))) / PI \* 180;

}

void moveFunc(int x, int y) {

switch (mouseMode) {

case LEFT\_CLICK:

calculateAngle(x, y);

break;

case MIDDLE\_CLICK:

moveX = x - orgX;

moveY = -y + orgY;

orgX = x;

orgY = y;

break;

}

glutPostRedisplay();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

width = w;

height = h;

glMatrixMode(GL\_PROJECTION);

//初始化坐标系

glLoadIdentity();

if (w <= h) {

glOrtho(-1000, 1000,

-1000.0\*(GLfloat)h / (GLfloat)w, 1000.0\*(GLfloat)h / (GLfloat)w,

-10000, 10000);

rangeX = 1000;

rangeY = (int)(1000.0 \* (GLfloat)h / (GLfloat)w);

} else {

glOrtho(-1000.0\*(GLfloat)w / (GLfloat)h, 1000.0\*(GLfloat)w / (GLfloat)h,

-1000, 1000,

-10000, 10000);

rangeX = (int)(1000.0 \* (GLfloat)w / (GLfloat)h);

rangeY = 1000;

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glutPostRedisplay();

}

int main(int argc, char \*\*argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(width, height);

glutInitWindowPosition(100, 100);

glutCreateWindow("simple OpenGL example");

myinit();

glutKeyboardFunc(scaleFunc);

glutMouseFunc(mouseFunc);

glutMotionFunc(moveFunc);

glutDisplayFunc(displayScene);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}

**Task2\_1**

功能说明：

光源相对模型不动，视点移动

效果预览：

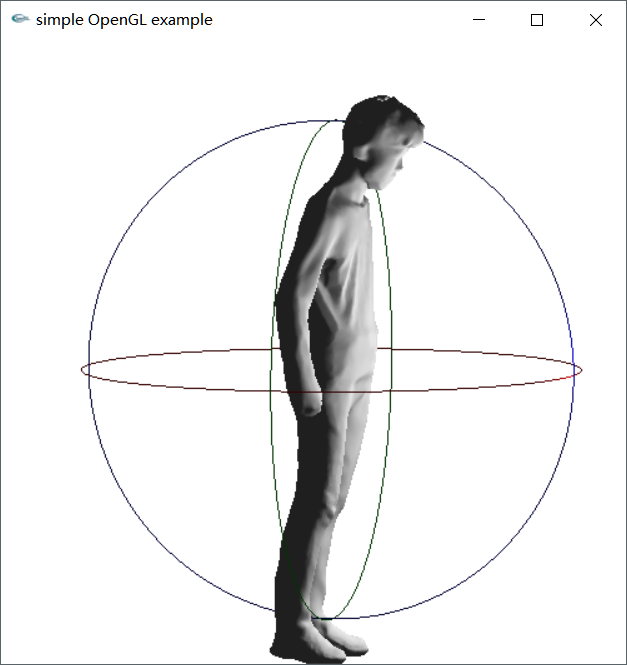


图2.1 Task2\_1 程序预览（查看附件动图）

源代码：

/\*

\* 该程序实现以下功能

\* 1. 读入ply文件内容，将模型显示出来

\* 2. 添加菜单，允许选择画点、画线、或者画多边形着色。

\* 3. 设置鼠标左键、中键回调函数，使其分别实现对模型进行旋转、平移。

\* 4. 由于右键已被菜单占用，故缩放功能实用[]键完成，注意切换为英文模式

\* 5. 设置光照。

\* 6. 以模型重心为旋转中心，旋转模型

\* 7. 光源相对模型不动，视点移动

\*\*\*\*\*\*\*

\* @author：宋灵冰

\*/

#include <iostream>

#include <vector>

#include <string.h>

#include <stdio.h>

#include <gl/glut.h>

using namespace std;

typedef struct {

GLdouble x, y, z;

}Point3d;

typedef struct {

int index[3];

}FaceIndex;

enum mouseButton { LEFT\_CLICK, MIDDLE\_CLICK, NO\_CLICK };

GLfloat angleX = 0.0;

GLfloat angleY = 0.0;

GLfloat scaleLevel = 1.0;

GLfloat moveX = 0.0;

GLfloat moveY = 0.0;

const GLfloat moveZ = 0.0;

GLfloat tempX = 0.0;

GLfloat tempY = 0.0;

int orgX, orgY;

bool moveState = false;

int pointNum, faceNum;

Point3d corePoint = { 0,0,0 };

vector<Point3d> pointContainer, normalContainer;

vector<FaceIndex> faceContainer;

GLenum mode = GL\_FILL;

mouseButton mouseMode;

GLfloat lightPosition[] = { 0.0f, 0.0f, 0.0f, 1.0f }; //定义光源位置

void readPly(const char\* fileName) {

FILE\* f = fopen(fileName, "r");

if (!f)

return;

char strBuffer[255];

for (int i = 0; i < 3; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

fscanf(f, "element vertex %d\n", &pointNum);

for (int i = 0; i < 6; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

fscanf(f, "element face %d\n", &faceNum);

for (int i = 0; i < 2; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

Point3d tempPoint;

Point3d tempNormal;

FaceIndex tempFace;

for (int i = 0; i < pointNum; i++) {

fscanf(f, "%lf %lf %lf %lf %lf %lf \n",

&tempPoint.x, &tempPoint.y, &tempPoint.z,

&tempNormal.x, &tempNormal.y, &tempNormal.z);

corePoint.x += tempPoint.x;

corePoint.y += tempPoint.y;

corePoint.z += tempPoint.z;

pointContainer.push\_back(tempPoint);

normalContainer.push\_back(tempNormal);

}

corePoint.x /= (double)pointNum;

corePoint.y /= (double)pointNum;

corePoint.z /= (double)pointNum;

for (int i = 0; i < faceNum; i++) {

fscanf(f, "3 %d %d %d \n",

&tempFace.index[0], &tempFace.index[1], &tempFace.index[2]);

faceContainer.push\_back(tempFace);

}

fclose(f);

}

void menuFunc(int value) {

switch (value) {

case 1:

mode = GL\_POINT;

break;

case 2:

mode = GL\_LINE;

break;

case 3:

mode = GL\_FILL;

break;

default:

break;

}

glutPostRedisplay();

}

//设置初始属性

void myinit() {

//设置背景颜色

glClearColor(1.0, 1.0, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1000.0, 1000.0, -1000.0, 1000.0, -10000.0, 10000.0);

//设置观察参数

glMatrixMode(GL\_MODELVIEW);

//启用深度缓存

glEnable(GL\_DEPTH\_TEST);

glEnable(GLUT\_MULTISAMPLE);

//初始化坐标系

glLoadIdentity();

readPly("lizhenxiout-repaired.ply");

glutCreateMenu(menuFunc);

glutAddMenuEntry("Point Mode", 1);

glutAddMenuEntry("Line Mode", 2);

glutAddMenuEntry("Mesh Mode", 3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glEnable(GL\_LIGHT0);

//开启灯光

glEnable(GL\_LIGHTING);

}

void drawModel() {

FaceIndex tempFace;

Point3d tempFaceVertex[3];

Point3d tempFaceNormal[3];

GLfloat color[2][3] = { { 0.6,0.6,0.6 },{ 0.8,0.8,0.8 } };

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[0]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[1]);

glPolygonMode(GL\_FRONT\_AND\_BACK, mode);

glBegin(GL\_TRIANGLES);

{

for (int i = 0; i < faceNum; i++) {

tempFace = faceContainer[i];

for (int j = 0; j < 3; j++) {

tempFaceVertex[j] = pointContainer[tempFace.index[j]];

tempFaceNormal[j] = normalContainer[tempFace.index[j]];

glNormal3d(tempFaceNormal[j].x, tempFaceNormal[j].y, tempFaceNormal[j].z);

glVertex3d(tempFaceVertex[j].x, tempFaceVertex[j].y, tempFaceVertex[j].z);

}

}

}

glEnd();

}

void drawTraceBall() {

const double ballRadius = 800;

const double PI = 3.1415926535;

GLfloat color[3][3] = { { 1,0,0 },{ 0,1,0 },{ 0,0,1 } };

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[0]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[0]);

glTranslatef(corePoint.x - ballRadius, corePoint.y, corePoint.z);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[1]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[1]);

glTranslatef(corePoint.x - ballRadius, corePoint.y, corePoint.z);

glRotatef(90, 1.0, 0.0, 0.0);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[2]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[2]);

glTranslatef(corePoint.x, corePoint.y - ballRadius, corePoint.z);

glRotatef(90, 0.0, 0.0, 1.0);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

}

//用于初始显示图像

void displayScene() {

//清空窗口

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

{

glPushMatrix();

glScalef(scaleLevel, scaleLevel, scaleLevel);

glTranslatef(moveX \* 3.0, moveY \* 3.0, moveZ);

glTranslatef(corePoint.x, corePoint.y, corePoint.z);

glRotatef(angleX, 1.0, 0.0, 0.0);

glRotatef(angleY, 0.0, 1.0, 0.0);

glTranslatef(-corePoint.x, -corePoint.y, -corePoint.z);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);

drawModel();

glPopMatrix();

glPushMatrix();

glTranslatef(corePoint.x, corePoint.y, corePoint.z);

glRotatef(angleX, 1.0, 0.0, 0.0);

glRotatef(angleY, 0.0, 1.0, 0.0);

glTranslatef(-corePoint.x, -corePoint.y, -corePoint.z);

drawTraceBall();

glPopMatrix();

}

glPopMatrix();

glFlush();

}

//实现缩放回调函数，使用[]控制缩放

void scaleFunc(unsigned char key, int x, int y) {

switch (key) {

case '[':

if (scaleLevel > 0)

scaleLevel -= 0.1;

break;

case ']':

scaleLevel += 0.1;

break;

case 27:

exit(0);

break;

default:

break;

}

glutPostRedisplay();

}

//设置鼠标左键、中键回调函数，使其分别实现对模型进行旋转、平移

void mouseFunc(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && !moveState) {

mouseMode = LEFT\_CLICK;

orgX = x;

orgY = y;

tempX = angleY;

tempY = angleX;

moveState = true;

}

if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN && !moveState) {

mouseMode = MIDDLE\_CLICK;

orgX = x;

orgY = y;

tempX = moveX;

tempY = moveY;

moveState = true;

}

if (state == GLUT\_UP && moveState) {

moveState = false;

mouseMode = NO\_CLICK;

}

glutPostRedisplay();

}

void moveFunc(int x, int y) {

switch (mouseMode) {

case LEFT\_CLICK:

angleY = tempX + (x - orgX);

angleX = tempY - (y - orgY);

break;

case MIDDLE\_CLICK:

moveX = tempX + x - orgX;

moveY = tempY - y + orgY;

break;

}

glutPostRedisplay();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

//初始化坐标系

glLoadIdentity();

if (w <= h)

glOrtho(-1000, 1000,

-1000.0\*(GLfloat)h / (GLfloat)w, 1000.0\*(GLfloat)h / (GLfloat)w,

-10000, 10000);

else

glOrtho(-1000.0\*(GLfloat)w / (GLfloat)h, 1000.0\*(GLfloat)w / (GLfloat)h,

-1000, 1000,

-10000, 10000);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glutPostRedisplay();

}

int main(int argc, char \*\*argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("simple OpenGL example");

myinit();

glutKeyboardFunc(scaleFunc);

glutMouseFunc(mouseFunc);

glutMotionFunc(moveFunc);

glutDisplayFunc(displayScene);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}

**Task2\_2**

功能说明：

模型不动，光源随视点一起移动

效果预览：

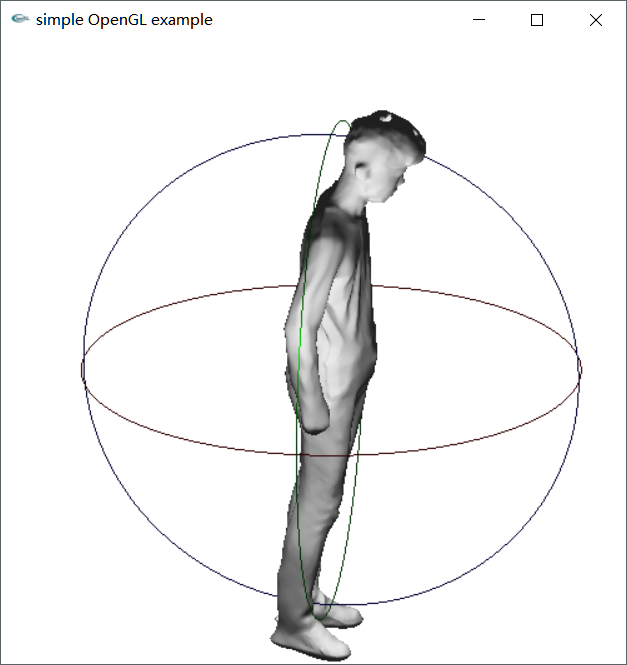


图2.2 Task2\_2 程序预览（查看附件动图）

源代码：

/\*

\* 该程序实现以下功能

\* 1. 读入ply文件内容，将模型显示出来

\* 2. 添加菜单，允许选择画点、画线、或者画多边形着色。

\* 3. 设置鼠标左键、中键回调函数，使其分别实现对模型进行旋转、平移。

\* 4. 由于右键已被菜单占用，故缩放功能实用[]键完成，注意切换为英文模式

\* 5. 设置光照。

\* 6. 以模型重心为旋转中心，旋转模型

\* 7. 模型不动，光源随视点一起移动

\*\*\*\*\*\*\*

\* @author：宋灵冰

\*/

#include <iostream>

#include <vector>

#include <string.h>

#include <stdio.h>

#include <gl/glut.h>

using namespace std;

typedef struct {

GLdouble x, y, z;

}Point3d;

typedef struct {

int index[3];

}FaceIndex;

enum mouseButton { LEFT\_CLICK, MIDDLE\_CLICK, NO\_CLICK };

GLfloat angleX = 0.0;

GLfloat angleY = 0.0;

GLfloat scaleLevel = 1.0;

GLfloat moveX = 0.0;

GLfloat moveY = 0.0;

const GLfloat moveZ = 0.0;

GLfloat tempX = 0.0;

GLfloat tempY = 0.0;

int orgX, orgY;

bool moveState = false;

int pointNum, faceNum;

Point3d corePoint = { 0,0,0 };

vector<Point3d> pointContainer, normalContainer;

vector<FaceIndex> faceContainer;

GLenum mode = GL\_FILL;

mouseButton mouseMode;

GLfloat lightPosition[] = { 0.0f, 0.0f, 0.0f, 1.0f }; //定义光源位置

void readPly(const char\* fileName) {

FILE\* f = fopen(fileName, "r");

if (!f)

return;

char strBuffer[255];

for (int i = 0; i < 3; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

fscanf(f, "element vertex %d\n", &pointNum);

for (int i = 0; i < 6; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

fscanf(f, "element face %d\n", &faceNum);

for (int i = 0; i < 2; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

Point3d tempPoint;

Point3d tempNormal;

FaceIndex tempFace;

for (int i = 0; i < pointNum; i++) {

fscanf(f, "%lf %lf %lf %lf %lf %lf \n",

&tempPoint.x, &tempPoint.y, &tempPoint.z,

&tempNormal.x, &tempNormal.y, &tempNormal.z);

corePoint.x += tempPoint.x;

corePoint.y += tempPoint.y;

corePoint.z += tempPoint.z;

pointContainer.push\_back(tempPoint);

normalContainer.push\_back(tempNormal);

}

corePoint.x /= (double)pointNum;

corePoint.y /= (double)pointNum;

corePoint.z /= (double)pointNum;

for (int i = 0; i < faceNum; i++) {

fscanf(f, "3 %d %d %d \n",

&tempFace.index[0], &tempFace.index[1], &tempFace.index[2]);

faceContainer.push\_back(tempFace);

}

fclose(f);

}

void menuFunc(int value) {

switch (value) {

case 1:

mode = GL\_POINT;

break;

case 2:

mode = GL\_LINE;

break;

case 3:

mode = GL\_FILL;

break;

default:

break;

}

glutPostRedisplay();

}

//设置初始属性

void myinit() {

//设置背景颜色

glClearColor(1.0, 1.0, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1000.0, 1000.0, -1000.0, 1000.0, -10000.0, 10000.0);

//设置观察参数

glMatrixMode(GL\_MODELVIEW);

//启用深度缓存

glEnable(GL\_DEPTH\_TEST);

glEnable(GLUT\_MULTISAMPLE);

//初始化坐标系

glLoadIdentity();

readPly("lizhenxiout-repaired.ply");

glutCreateMenu(menuFunc);

glutAddMenuEntry("Point Mode", 1);

glutAddMenuEntry("Line Mode", 2);

glutAddMenuEntry("Mesh Mode", 3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glEnable(GL\_LIGHT0);

//开启灯光

glEnable(GL\_LIGHTING);

}

void drawModel() {

FaceIndex tempFace;

Point3d tempFaceVertex[3];

Point3d tempFaceNormal[3];

GLfloat color[2][3] = { { 0.6,0.6,0.6 },{ 0.8,0.8,0.8 } };

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[0]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[1]);

glPolygonMode(GL\_FRONT\_AND\_BACK, mode);

glBegin(GL\_TRIANGLES);

{

for (int i = 0; i < faceNum; i++) {

tempFace = faceContainer[i];

for (int j = 0; j < 3; j++) {

tempFaceVertex[j] = pointContainer[tempFace.index[j]];

tempFaceNormal[j] = normalContainer[tempFace.index[j]];

glNormal3d(tempFaceNormal[j].x, tempFaceNormal[j].y, tempFaceNormal[j].z);

glVertex3d(tempFaceVertex[j].x, tempFaceVertex[j].y, tempFaceVertex[j].z);

}

}

}

glEnd();

}

void drawTraceBall() {

const double ballRadius = 800;

const double PI = 3.1415926535;

GLfloat color[3][3] = { { 1,0,0 },{ 0,1,0 },{ 0,0,1 } };

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[0]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[0]);

glTranslatef(corePoint.x - ballRadius, corePoint.y, corePoint.z);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[1]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[1]);

glTranslatef(corePoint.x - ballRadius, corePoint.y, corePoint.z);

glRotatef(90, 1.0, 0.0, 0.0);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[2]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[2]);

glTranslatef(corePoint.x, corePoint.y - ballRadius, corePoint.z);

glRotatef(90, 0.0, 0.0, 1.0);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

}

//用于初始显示图像

void displayScene() {

//清空窗口

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

{

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);

glPushMatrix();

glScalef(scaleLevel, scaleLevel, scaleLevel);

glTranslatef(moveX \* 3.0, moveY \* 3.0, moveZ);

glTranslatef(corePoint.x, corePoint.y, corePoint.z);

glRotatef(angleX, 1.0, 0.0, 0.0);

glRotatef(angleY, 0.0, 1.0, 0.0);

glTranslatef(-corePoint.x, -corePoint.y, -corePoint.z);

drawModel();

glPopMatrix();

glPushMatrix();

glTranslatef(corePoint.x, corePoint.y, corePoint.z);

glRotatef(angleX, 1.0, 0.0, 0.0);

glRotatef(angleY, 0.0, 1.0, 0.0);

glTranslatef(-corePoint.x, -corePoint.y, -corePoint.z);

drawTraceBall();

glPopMatrix();

}

glPopMatrix();

glFlush();

}

//实现缩放回调函数，使用[]控制缩放

void scaleFunc(unsigned char key, int x, int y) {

switch (key) {

case '[':

if (scaleLevel > 0)

scaleLevel -= 0.1;

break;

case ']':

scaleLevel += 0.1;

break;

case 27:

exit(0);

break;

default:

break;

}

glutPostRedisplay();

}

//设置鼠标左键、中键回调函数，使其分别实现对模型进行旋转、平移

void mouseFunc(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && !moveState) {

mouseMode = LEFT\_CLICK;

orgX = x;

orgY = y;

tempX = angleY;

tempY = angleX;

moveState = true;

}

if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN && !moveState) {

mouseMode = MIDDLE\_CLICK;

orgX = x;

orgY = y;

tempX = moveX;

tempY = moveY;

moveState = true;

}

if (state == GLUT\_UP && moveState) {

moveState = false;

mouseMode = NO\_CLICK;

}

glutPostRedisplay();

}

void moveFunc(int x, int y) {

switch (mouseMode) {

case LEFT\_CLICK:

angleY = tempX + (x - orgX);

angleX = tempY - (y - orgY);

break;

case MIDDLE\_CLICK:

moveX = tempX + x - orgX;

moveY = tempY - y + orgY;

break;

}

glutPostRedisplay();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

//初始化坐标系

glLoadIdentity();

if (w <= h)

glOrtho(-1000, 1000,

-1000.0\*(GLfloat)h / (GLfloat)w, 1000.0\*(GLfloat)h / (GLfloat)w,

-10000, 10000);

else

glOrtho(-1000.0\*(GLfloat)w / (GLfloat)h, 1000.0\*(GLfloat)w / (GLfloat)h,

-1000, 1000,

-10000, 10000);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glutPostRedisplay();

}

int main(int argc, char \*\*argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("simple OpenGL example");

myinit();

glutKeyboardFunc(scaleFunc);

glutMouseFunc(mouseFunc);

glutMotionFunc(moveFunc);

glutDisplayFunc(displayScene);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}

**Task2\_3**

功能说明：

模型和视点均不动，光源绕模型旋转

效果预览：

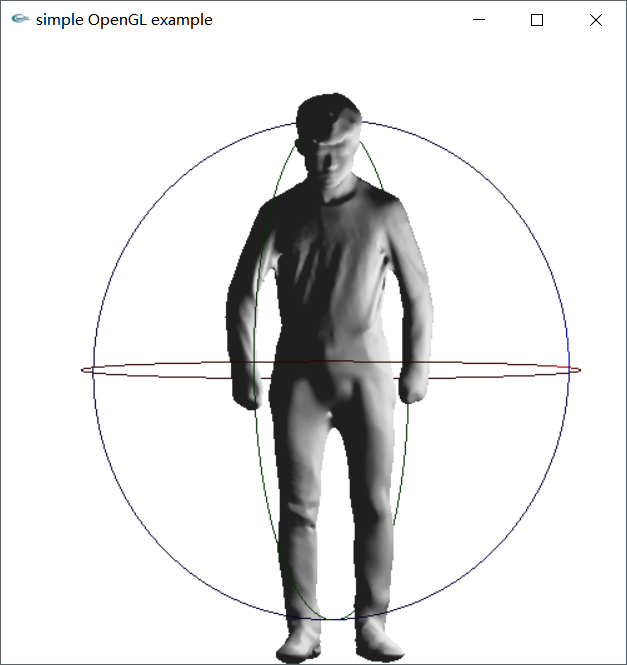


图2.3 Task2\_3 程序预览（查看附件动图）

源代码：

/\*

\* 该程序实现以下功能

\* 1. 读入ply文件内容，将模型显示出来

\* 2. 添加菜单，允许选择画点、画线、或者画多边形着色。

\* 3. 设置鼠标左键、中键回调函数，使其分别实现对模型进行旋转、平移。

\* 4. 由于右键已被菜单占用，故缩放功能实用[]键完成，注意切换为英文模式

\* 5. 设置光照。

\* 6. 以模型重心为旋转中心，旋转模型

\* 7. 模型和视点均不动，光源绕模型旋转

\*\*\*\*\*\*\*

\* @author：宋灵冰

\*/

#include <iostream>

#include <vector>

#include <string.h>

#include <stdio.h>

#include <gl/glut.h>

using namespace std;

typedef struct {

GLdouble x, y, z;

}Point3d;

typedef struct {

int index[3];

}FaceIndex;

enum mouseButton { LEFT\_CLICK, MIDDLE\_CLICK, NO\_CLICK };

GLfloat angleX = 0.0;

GLfloat angleY = 0.0;

GLfloat scaleLevel = 1.0;

GLfloat moveX = 0.0;

GLfloat moveY = 0.0;

const GLfloat moveZ = 0.0;

GLfloat tempX = 0.0;

GLfloat tempY = 0.0;

int orgX, orgY;

bool moveState = false;

int pointNum, faceNum;

Point3d corePoint = { 0,0,0 };

vector<Point3d> pointContainer, normalContainer;

vector<FaceIndex> faceContainer;

GLenum mode = GL\_FILL;

mouseButton mouseMode;

GLfloat lightPosition[] = { 0.0f, 0.0f, 0.0f, 1.0f }; //定义光源位置

void readPly(const char\* fileName) {

FILE\* f = fopen(fileName, "r");

if (!f)

return;

char strBuffer[255];

for (int i = 0; i < 3; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

fscanf(f, "element vertex %d\n", &pointNum);

for (int i = 0; i < 6; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

fscanf(f, "element face %d\n", &faceNum);

for (int i = 0; i < 2; i++) {

fgets(strBuffer, 255, f);

memset(strBuffer, '\0', 255);

}

Point3d tempPoint;

Point3d tempNormal;

FaceIndex tempFace;

for (int i = 0; i < pointNum; i++) {

fscanf(f, "%lf %lf %lf %lf %lf %lf \n",

&tempPoint.x, &tempPoint.y, &tempPoint.z,

&tempNormal.x, &tempNormal.y, &tempNormal.z);

corePoint.x += tempPoint.x;

corePoint.y += tempPoint.y;

corePoint.z += tempPoint.z;

pointContainer.push\_back(tempPoint);

normalContainer.push\_back(tempNormal);

}

corePoint.x /= (double)pointNum;

corePoint.y /= (double)pointNum;

corePoint.z /= (double)pointNum;

for (int i = 0; i < faceNum; i++) {

fscanf(f, "3 %d %d %d \n",

&tempFace.index[0], &tempFace.index[1], &tempFace.index[2]);

faceContainer.push\_back(tempFace);

}

fclose(f);

}

void menuFunc(int value) {

switch (value) {

case 1:

mode = GL\_POINT;

break;

case 2:

mode = GL\_LINE;

break;

case 3:

mode = GL\_FILL;

break;

default:

break;

}

glutPostRedisplay();

}

//设置初始属性

void myinit() {

//设置背景颜色

glClearColor(1.0, 1.0, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1000.0, 1000.0, -1000.0, 1000.0, -10000.0, 10000.0);

//设置观察参数

glMatrixMode(GL\_MODELVIEW);

//启用深度缓存

glEnable(GL\_DEPTH\_TEST);

glEnable(GLUT\_MULTISAMPLE);

//初始化坐标系

glLoadIdentity();

readPly("lizhenxiout-repaired.ply");

glutCreateMenu(menuFunc);

glutAddMenuEntry("Point Mode", 1);

glutAddMenuEntry("Line Mode", 2);

glutAddMenuEntry("Mesh Mode", 3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glEnable(GL\_LIGHT0);

//开启灯光

glEnable(GL\_LIGHTING);

}

void drawModel() {

FaceIndex tempFace;

Point3d tempFaceVertex[3];

Point3d tempFaceNormal[3];

GLfloat color[2][3] = { { 0.6,0.6,0.6 },{ 0.8,0.8,0.8 } };

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[0]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[1]);

glPolygonMode(GL\_FRONT\_AND\_BACK, mode);

glBegin(GL\_TRIANGLES);

{

for (int i = 0; i < faceNum; i++) {

tempFace = faceContainer[i];

for (int j = 0; j < 3; j++) {

tempFaceVertex[j] = pointContainer[tempFace.index[j]];

tempFaceNormal[j] = normalContainer[tempFace.index[j]];

glNormal3d(tempFaceNormal[j].x, tempFaceNormal[j].y, tempFaceNormal[j].z);

glVertex3d(tempFaceVertex[j].x, tempFaceVertex[j].y, tempFaceVertex[j].z);

}

}

}

glEnd();

}

void drawTraceBall() {

const double ballRadius = 800;

const double PI = 3.1415926535;

GLfloat color[3][3] = { { 1,0,0 },{ 0,1,0 },{ 0,0,1 } };

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[0]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[0]);

glTranslatef(corePoint.x - ballRadius, corePoint.y, corePoint.z);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[1]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[1]);

glTranslatef(corePoint.x - ballRadius, corePoint.y, corePoint.z);

glRotatef(90, 1.0, 0.0, 0.0);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[2]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[2]);

glTranslatef(corePoint.x, corePoint.y - ballRadius, corePoint.z);

glRotatef(90, 0.0, 0.0, 1.0);

glutWireTorus(ballRadius, ballRadius + 1, 1000, 1);

glPopMatrix();

}

//用于初始显示图像

void displayScene() {

//清空窗口

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

{

drawModel();

glPushMatrix();

glScalef(scaleLevel, scaleLevel, scaleLevel);

glTranslatef(moveX \* 3.0, moveY \* 3.0, moveZ);

glTranslatef(corePoint.x, corePoint.y, corePoint.z);

glRotatef(angleX, 1.0, 0.0, 0.0);

glRotatef(angleY, 0.0, 1.0, 0.0);

glTranslatef(-corePoint.x, -corePoint.y, -corePoint.z);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);

glPopMatrix();

glPushMatrix();

glTranslatef(corePoint.x, corePoint.y, corePoint.z);

glRotatef(angleX, 1.0, 0.0, 0.0);

glRotatef(angleY, 0.0, 1.0, 0.0);

glTranslatef(-corePoint.x, -corePoint.y, -corePoint.z);

drawTraceBall();

glPopMatrix();

}

glPopMatrix();

glFlush();

}

//实现缩放回调函数，使用[]控制缩放

void scaleFunc(unsigned char key, int x, int y) {

switch (key) {

case '[':

if (scaleLevel > 0)

scaleLevel -= 0.1;

break;

case ']':

scaleLevel += 0.1;

break;

case 27:

exit(0);

break;

default:

break;

}

glutPostRedisplay();

}

//设置鼠标左键、中键回调函数，使其分别实现对模型进行旋转、平移

void mouseFunc(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && !moveState) {

mouseMode = LEFT\_CLICK;

orgX = x;

orgY = y;

tempX = angleY;

tempY = angleX;

moveState = true;

}

if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN && !moveState) {

mouseMode = MIDDLE\_CLICK;

orgX = x;

orgY = y;

tempX = moveX;

tempY = moveY;

moveState = true;

}

if (state == GLUT\_UP && moveState) {

moveState = false;

mouseMode = NO\_CLICK;

}

glutPostRedisplay();

}

void moveFunc(int x, int y) {

switch (mouseMode) {

case LEFT\_CLICK:

angleY = tempX + (x - orgX);

angleX = tempY - (y - orgY);

break;

case MIDDLE\_CLICK:

moveX = tempX + x - orgX;

moveY = tempY - y + orgY;

break;

}

glutPostRedisplay();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

//初始化坐标系

glLoadIdentity();

if (w <= h)

glOrtho(-1000, 1000,

-1000.0\*(GLfloat)h / (GLfloat)w, 1000.0\*(GLfloat)h / (GLfloat)w,

-10000, 10000);

else

glOrtho(-1000.0\*(GLfloat)w / (GLfloat)h, 1000.0\*(GLfloat)w / (GLfloat)h,

-1000, 1000,

-10000, 10000);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glutPostRedisplay();

}

int main(int argc, char \*\*argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("simple OpenGL example");

myinit();

glutKeyboardFunc(scaleFunc);

glutMouseFunc(mouseFunc);

glutMotionFunc(moveFunc);

glutDisplayFunc(displayScene);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}