**计算机图形学**

**实验报告**

**exp5**

**目录**

**[Task1\_2](#_Toc512692068)** [1](#_Toc512692068)

[功能说明： 1](#_Toc512692069)

[效果预览： 1](#_Toc512692070)

[源代码： 1](#_Toc512692071)

[**Task2** 10](#_Toc512692072)

[功能说明： 10](#_Toc512692073)

[效果预览： 10](#_Toc512692074)

[源代码： 10](#_Toc512692075)

**Task1\_2**

功能说明：

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. 设置了轨迹球

效果预览：

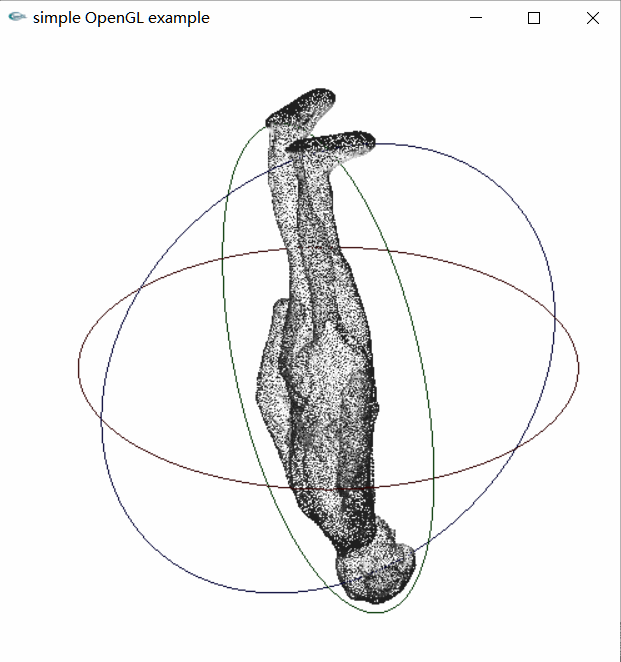


图1.1 Task1\_2程序预览（查看附件动图）

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. 设置了轨迹球

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

\* @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\_POINT;

mouseButton mouseMode;

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

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

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);

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**

功能说明：

绘制太阳、地球、月球三者运动动画

实现了以下功能

1. 用不同大小的球代表太阳、地球、月球

2. 地球和月球共同绕太阳旋转，月球同时还绕地球旋转

3. 以太阳作为发光体，实现地球和月球对太阳光的反射效果

4. 为了更加真实，使用随机数生成星星作为背景

效果预览：

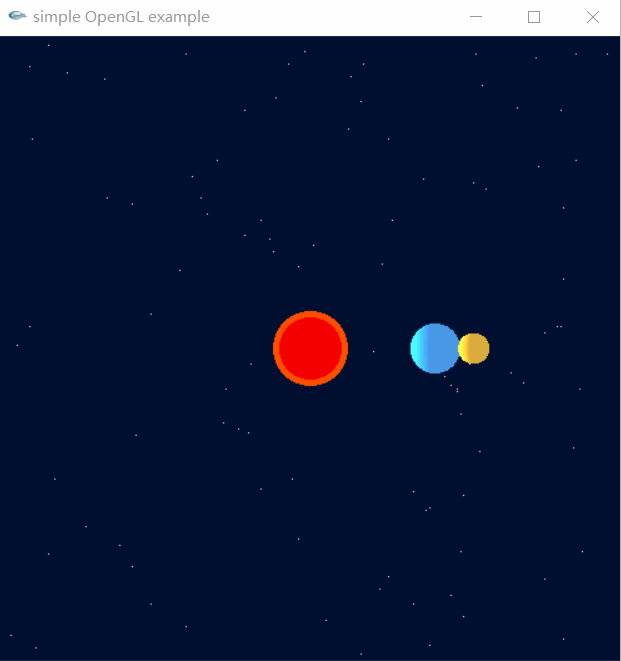


图2.1 Task2 程序预览（查看附件动图）

源代码：

/\*

\* 绘制太阳、地球、月球三者运动动画

\* 实现了以下功能

\* 1. 用不同大小的球代表太阳、地球、月球

\* 2. 地球和月球共同绕太阳旋转，月球同时还绕地球旋转

\* 3. 以太阳作为发光体，实现地球和月球对太阳光的反射效果

\* 4. 为了更加真实，使用随机数生成星星作为背景

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

\* @author：宋灵冰

\*/

#include <iostream>

#include <vector>

#include <string.h>

#include <stdio.h>

#include <time.h>

#include <gl/glut.h>

using namespace std;

const GLdouble SunRadius = 10;

const GLdouble HaloRadius = 12;

const GLdouble EarthRadius = 8;

const GLdouble MoonRadius = 5;

const GLdouble EarthToSun = 40;

const GLdouble MoonToEarth = 15;

int earthAngle = 0;

int moonAngle = 0;

int countNum = 0;

GLfloat width = 100, height = 100;

//设置初始属性

void myinit() {

//设置背景颜色

glClearColor(0.0, 15.0/255.0, 48.0/255.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-100.0, 100.0, -100.0, 100.0, -100.0, 100.0);

//设置观察参数

glMatrixMode(GL\_MODELVIEW);

//初始化坐标系

glLoadIdentity();

//启用深度缓存

glEnable(GL\_DEPTH\_TEST);

GLfloat ambient[] = { 0.5f, 0.5f, 0.5f, 1.0f }; //定义环境光颜色

GLfloat diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f }; //定义漫反射光颜色

GLfloat specular[] = { 0.0f, 0.0f, 0.0f, 1.0f }; //定义镜面反射光颜色

glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuse);

glLightfv(GL\_LIGHT1, GL\_SPECULAR, specular);

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

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

glEnable(GL\_LIGHT0);

//开启灯光

glEnable(GL\_LIGHTING);

}

void drawSun() {

GLfloat color[2][3][3] = { { { 0.8,0.0,0.0 },{ 1.0,0.0,0.0 },{ 0.4,0.0,0.0 } },

{ { 0.9,113.0 / 255.0, 0.0 }, { 1.0,113.0 / 255.0,0.0 }, { 0.4,0.0,0.0 } } };

glPushMatrix();

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

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, color[0][1]);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_EMISSION, color[0][2]);

glutSolidSphere(SunRadius, 100, 100);

glPopMatrix();

glPushMatrix();

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, color[1][0]);

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

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_EMISSION, color[1][2]);

glTranslatef(0, 0, -HaloRadius);

glutSolidSphere(HaloRadius, 100, 100);

glPopMatrix();

}

void drawEarth() {

GLfloat color[3][3] = { { 30.0 / 255.0,144.0 / 255.0,1.0 },{ 0.0,191.0 / 255.0,1.0 },{ 0.2,0.2,0.2 } };

glPushMatrix();

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

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

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

glRotatef(earthAngle, 0, 1, 0);

glTranslatef(EarthToSun, 0, 0);

glutSolidSphere(EarthRadius, 100, 100);

glPopMatrix();

}

void drawMoon() {

GLfloat color[3][3] = { { 238.0 / 255.0,173.0 / 255.0,14.0 / 255.0 },{ 1.0,185.0 / 255.0,15.0 / 255.0 },{ 0.2,0.2,0.2 } };

glPushMatrix();

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

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

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

glRotatef(earthAngle, 0, 1, 0);

glTranslatef(EarthToSun, 0, 0);

glRotatef(moonAngle, 0, 1, 0);

glTranslatef(MoonToEarth, 0, 0);

glutSolidSphere(MoonRadius, 100, 100);

glPopMatrix();

}

void drawStar() {

GLfloat color[3][3] = { { 1.0, 1.0, 1.0 },{ 1.0, 1.0, 1.0 },{ 0.2,0.2,0.2 } };

srand((unsigned)time(NULL));

glPushMatrix();

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

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

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

glBegin(GL\_POINTS);

for (int i = 0; i < 100; i++) {

int x = rand() % (int)(2 \* width) - (int)width;

int y = rand() % (int)(2 \* height) - (int)height;

int z = rand() % 100 - 100;

glVertex3i(x, y, z);

}

glEnd();

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();

{

drawSun();

drawEarth();

drawMoon();

drawStar();

}

glPopMatrix();

glFlush();

//交换通道

glutSwapBuffers();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

//初始化坐标系

glLoadIdentity();

if (w <= h) {

height = 100.0\*(GLfloat)h / (GLfloat)w;

glOrtho(-100, 100,

-height, height,

-100, 100);

}

else {

width = 100.0\*(GLfloat)w / (GLfloat)h;

glOrtho(-width, width,

-100, 100,

-100, 100);

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glutPostRedisplay();

}

void rotatePlanet(int value) {

if (countNum == 0) {

earthAngle = (earthAngle + 1) % 360;

}

moonAngle = (moonAngle + 1) % 360;

countNum = (countNum + 1) % 10;

glutPostRedisplay();

glutTimerFunc(5, rotatePlanet, 1);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DEPTH | GLUT\_DOUBLE);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("simple OpenGL example");

myinit();

glutDisplayFunc(displayScene);

glutReshapeFunc(reshape);

glutTimerFunc(5, rotatePlanet, 1);

glutMainLoop();

return 0;

}