**计算机图形学**

**实验报告**

**exp6**

**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**

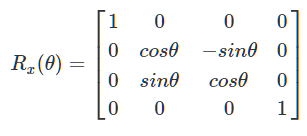
功能说明：

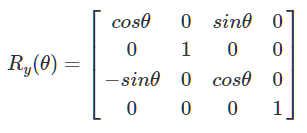
1. 定义了myRotateX、myRotateY、myRotateZ函数，分别实现绕X、Y、Z轴旋转的旋转矩阵计算并应用到当前模型视图矩阵

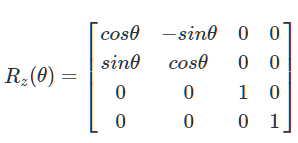
2. 分别调用自定义函数与opengl内置函数，输出旋转矩阵结果，进行验证

实现过程：

已知绕X、Y、Z轴旋转的旋转矩阵为







根据定义进行计算相应的旋转矩阵，并且切换至模型视图矩阵下使用glLoadMatrixf函数将矩阵载入即可实现对应功能

效果预览：

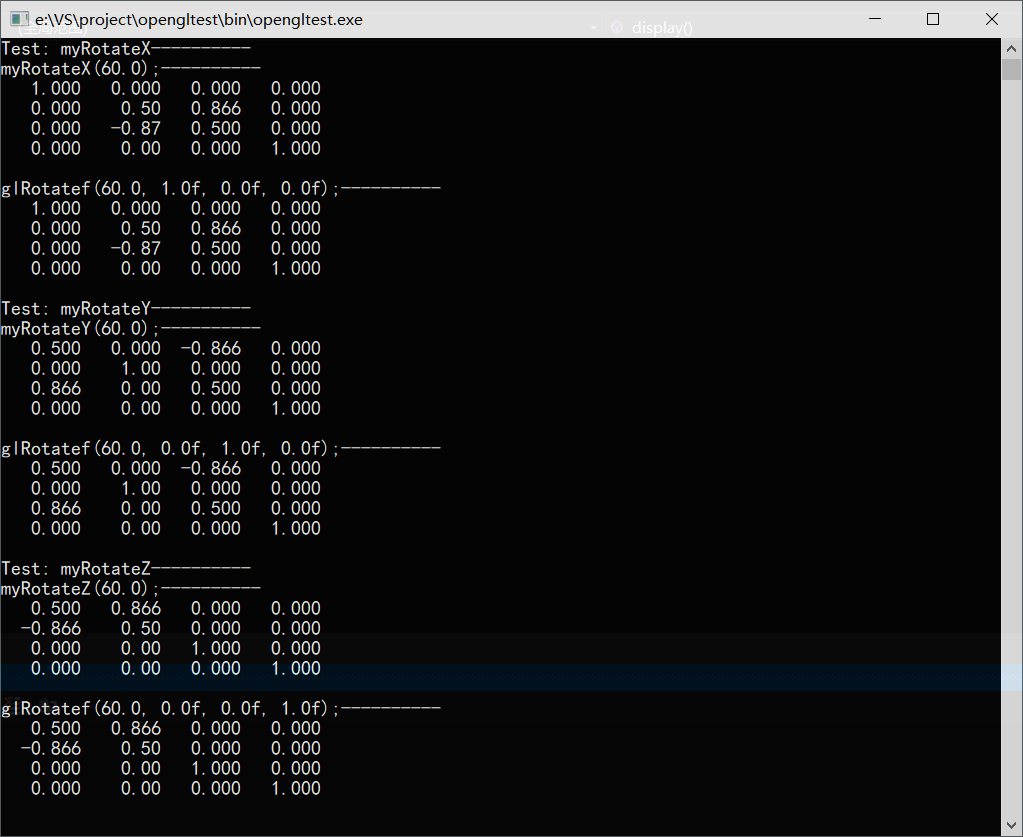


图1.1 Task1程序预览

源代码：

/\*

\* 该程序实现以下功能

\* 1. 定义了myRotateX、myRotateY、myRotateZ函数，

\* 分别实现绕X、Y、Z轴旋转的旋转矩阵计算

\* 2. 分别调用自定义函数与opengl内置函数，输出旋转矩阵结果，进行验证

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

\* @author：宋灵冰

\*/

#include <stdio.h>

#include <math.h>

#include <GL/glut.h>

void PrintMatrix(float matrix[16])

{

printf("%8.3f%8.3f%8.3f%8.3f\n"

"%8.3f%8.2f%8.3f%8.3f\n"

"%8.3f%8.2f%8.3f%8.3f\n"

"%8.3f%8.2f%8.3f%8.3f\n\n",

matrix[0], matrix[1], matrix[2], matrix[3],

matrix[4], matrix[5], matrix[6], matrix[7],

matrix[8], matrix[9], matrix[10], matrix[11],

matrix[12], matrix[13], matrix[14], matrix[15]);

}

void myRotateZ(float alpha)

{

float a = alpha / 180 \* 3.14159;

float R[16] = { 0 };

R[0] = cos(a);

R[1] = sin(a);

R[4] = -sin(a);

R[5] = cos(a);

R[10] = 1;

R[15] = 1;

glMatrixMode(GL\_MODELVIEW);

glLoadMatrixf(R);

}

void myRotateX(float alpha)

{

float a = alpha / 180 \* 3.14159;

float R[16] = { 0 };

R[0] = 1;

R[5] = cos(a);

R[6] = sin(a);

R[9] = -sin(a);

R[10] = cos(a);

R[15] = 1;

glMatrixMode(GL\_MODELVIEW);

glLoadMatrixf(R);

}

void myRotateY(float alpha)

{

float a = alpha / 180 \* 3.14159;

float R[16] = { 0 };

R[0] = cos(a);

R[2] = -sin(a);

R[5] = 1;

R[8] = sin(a);

R[10] = cos(a);

R[15] = 1;

glMatrixMode(GL\_MODELVIEW);

glLoadMatrixf(R);

}

void MyOrtho(float left, float right, float bottom, float top, float near0, float far0)

{

float M[16] = { 0 };

M[0] = 2 / (right - left);

M[5] = 2 / (top - bottom);

M[10] = -2 / (far0 - near0);

M[12] = -(right + left) / (right - left);

M[13] = -(top + bottom) / (top - bottom);

M[14] = -(far0 + near0) / (far0 - near0);

M[15] = 1;

glMatrixMode(GL\_PROJECTION);

glLoadMatrixf(M);

}

void display()

{

float R[16];

glMatrixMode(GL\_MODELVIEW);

printf("Test: myRotateX----------\n");

printf("myRotateX(60.0);----------\n");

glLoadIdentity();

myRotateX(60.0);

glGetFloatv(GL\_MODELVIEW\_MATRIX, R);

PrintMatrix(R);

printf("glRotatef(60.0, 1.0f, 0.0f, 0.0f);----------\n");

glLoadIdentity();

glRotatef(60.0, 1.0f, 0.0f, 0.0f);

glGetFloatv(GL\_MODELVIEW\_MATRIX, R);

PrintMatrix(R);

printf("Test: myRotateY----------\n");

printf("myRotateY(60.0);----------\n");

glLoadIdentity();

myRotateY(60.0);

glGetFloatv(GL\_MODELVIEW\_MATRIX, R);

PrintMatrix(R);

printf("glRotatef(60.0, 0.0f, 1.0f, 0.0f);----------\n");

glLoadIdentity();

glRotatef(60.0, 0.0f, 1.0f, 0.0f);

glGetFloatv(GL\_MODELVIEW\_MATRIX, R);

PrintMatrix(R);

printf("Test: myRotateZ----------\n");

printf("myRotateZ(60.0);----------\n");

glLoadIdentity();

myRotateZ(60.0);

glGetFloatv(GL\_MODELVIEW\_MATRIX, R);

PrintMatrix(R);

printf("glRotatef(60.0, 0.0f, 0.0f, 1.0f);----------\n");

glLoadIdentity();

glRotatef(60.0, 0.0f, 0.0f, 1.0f);

glGetFloatv(GL\_MODELVIEW\_MATRIX, R);

PrintMatrix(R);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(10, 10);

glutInitWindowPosition(0, 0);

glutCreateWindow("Test");

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**Task2**

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

不足：没有实现绕中轴旋转，只是绕重心旋转

效果预览：

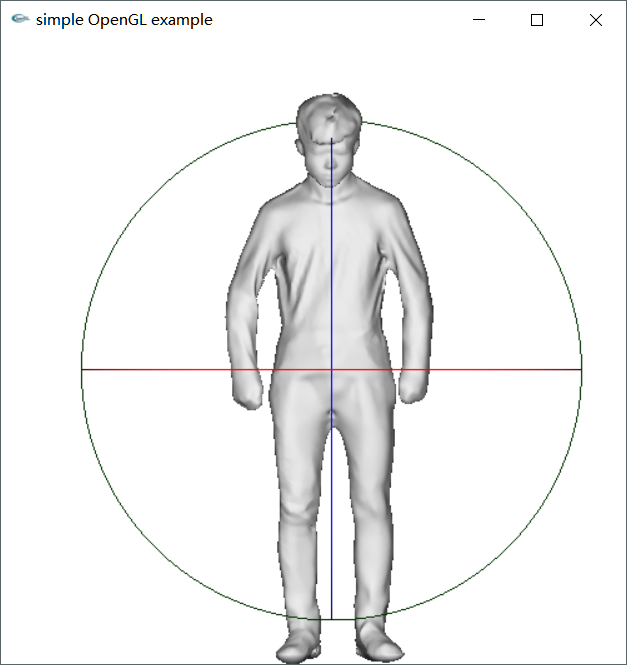


图2.1 Task2 程序预览（查看附件动图）

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>

#pragma warning(disable:4996)

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;

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, 5000.0f, 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);

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 },{ 0.8,0.8,0.8 } };

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() {

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;

}