OpenGL实验文档——A类第3题

结31 陈星雨 2013011727 chenxingyustar@gmail.com

**实验要求**

3、 魔方（OpenGL） （15 分）

1) 魔方可以利用鼠标旋转任意角度

2) 魔方贴图要不同（各个小立方体）

3) 用鼠标实现放大缩小功能

4) 魔方可以实现按层次的转动（比如点击一层则旋转一层）

5) 魔方可以使用鼠标右键弹出式菜单选择各种不同的灯光效果

6) 改变魔方背景

7) 可以生成 2-5 阶的魔方，使用弹出式菜单选择

**实验环境**

Visual Studio 2012（Windows8.1 64bit）

**实验原理**

此次实验代码核心部分有以下几块：

1、定义魔方数据结构，并以此为基础绘制2到5阶的魔方。这一步是后面所有操作的基础，因此如何设计能够使后面的操作更便捷显得至关重要。

2、魔方能够实现按层次的转动。这一部分是此次实验的难点。利用OpenGL的选择、拾取功能可以拾取到鼠标点击处的小块的ID，从而进行进一步的计算和判断，使该魔方面旋转。

3、魔方能够利用鼠标右键拖动旋转任意角度，并利用鼠标滚轮实现放大缩小功能。这一部分以B类作业的第4题为基础，使用轨迹球实现。

4、由于左键控制魔方面的旋转，右键控制魔方的整体旋转，滚轮控制缩放，因此题目中要求的“弹出式菜单”不再方便使用，改为采用添加的顶栏菜单来控制魔方的阶数、贴图效果、灯光效果、背景。添加菜单资源即可。

**实验步骤**

建立工程CAD\_MagicCube，使用Windows下OpenGL程序的基本框架，包括窗口的搭建等初始化工作，在框架基础上加上自己编写的代码。

下面针对几个主要功能对一些重要代码作解释：

**1、魔方块的类定义以及魔方的绘制**

定义魔方块的类：

class Cube

{

public:

int posID; //每个方块唯一对应的ID

float points[9][3]; //方块的中心和八个顶点的坐标

bool outerSurface[6]; //每个面是否是外表面（是否需要贴图)

int surfaceType[6]; //每个面对应的贴图样式

};

开始时，对魔方每个块做初始化：

void InitializeCubes(Cube \* cubes) //初始化每个方块的坐标

{

int num = 0;

float x,y,z;

float startPoint;

if(order%2 == 1) startPoint = (1-order)/2.0f;

else startPoint = 0.5f-order/2;

z = startPoint;

for(int i = 0;i < order;i++)

{

y = startPoint;

for(int j = 0; j < order;j++)

{

x = startPoint;

for(int k = 0; k < order; k ++)

{

cubes[num].posID = num;

//中心点

cubes[num].points[0][0] = x;

cubes[num].points[0][1] = y;

cubes[num].points[0][2] = z;

//顶点

cubes[num].points[1][0] = x-0.5f;

cubes[num].points[1][1] = y-0.5f;

cubes[num].points[1][2] = z-0.5f;

cubes[num].points[2][0] = x+0.5f;

cubes[num].points[2][1] = y-0.5f;

cubes[num].points[2][2] = z-0.5f;

cubes[num].points[3][0] = x+0.5f;

cubes[num].points[3][1] = y+0.5f;

cubes[num].points[3][2] = z-0.5f;

cubes[num].points[4][0] = x-0.5f;

cubes[num].points[4][1] = y+0.5f;

cubes[num].points[4][2] = z-0.5f;

cubes[num].points[5][0] = x-0.5f;

cubes[num].points[5][1] = y-0.5f;

cubes[num].points[5][2] = z+0.5f;

cubes[num].points[6][0] = x+0.5f;

cubes[num].points[6][1] = y-0.5f;

cubes[num].points[6][2] = z+0.5f;

cubes[num].points[7][0] = x+0.5f;

cubes[num].points[7][1] = y+0.5f;

cubes[num].points[7][2] = z+0.5f;

cubes[num].points[8][0] = x-0.5f;

cubes[num].points[8][1] = y+0.5f;

cubes[num].points[8][2] = z+0.5f;

InitializeSurfaces(cubes[num],startPoint);

num++;

x += 1.0;

}

y += 1.0;

}

z += 1.0;

}

}

判断和标记每个面的状态（是否位于外表面）

void InitializeSurfaces(Cube &cube,float startPoint)

{

distance = startPoint - 0.5f;

bool outside[9] = {0,0,0,0,0,0,0,0,0};

for(int i = 1;i<9;i++) //判断点是否位于外表面

{

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

{

if(cube.points[i][j]==distance||cube.points[i][j]==-distance)

{

outside[i] = true;

}

}

}

for(int k = 0;k < 6;k++) //标记好每个表面的状态

{

if(outside[eachSurface[k][0]]&&outside[eachSurface[k][1]]&&outside[eachSurface[k][2]]&&outside[eachSurface[k][3]])

{

cube.outerSurface[k] = true;

cube.surfaceType[k] = k + 1;

}

else

{

cube.outerSurface[k] = false;

cube.surfaceType[k] = 0;

}

}

}

下面是每一个魔方块的绘制函数，综合考虑了贴图效果、旋转角度、对灯光效果设置的法线等各个方面的因素，因此代码比较长：

void DrawCube(Cube &cube) // Draw A Cube

{

if(isRotating)

{

if(cube.points[0][rotAxis] == layer)

{

if(rot90)//旋转了90度后，更新每个转完的方块的坐标

{

float temp;

for(int k = 0; k < 9;k++)

{

switch(rotAxis)

{

case 0:

temp = cube.points[k][1];

cube.points[k][1] = -cube.points[k][2]\*((rotAngle>0)?1:-1);

cube.points[k][2] = temp\*((rotAngle>0)?1:-1);

break;

case 1:

temp = cube.points[k][2];

cube.points[k][2] = -cube.points[k][0]\*((rotAngle>0)?1:-1);

cube.points[k][0] = temp\*((rotAngle>0)?1:-1);

break;

case 2:

temp = cube.points[k][0];

cube.points[k][0] = -cube.points[k][1]\*((rotAngle>0)?1:-1);

cube.points[k][1] = temp\*((rotAngle>0)?1:-1);

break;

default:break;

}

}

}

else //若没到90度，则继续旋转

{

switch(rotAxis)

{

case 0:glRotatef(rotAngle,1,0,0);break;

case 1:glRotatef(rotAngle,0,1,0);break;

case 2:glRotatef(rotAngle,0,0,1);break;

default:break;

}

}

}

}

for(int i = 0; i < 6;i++) //分别绘制一个块的六个面

{

if(isSelected)

{

glPushName(i+cube.posID\*6);

}

if(textureType >0)

{

glBindTexture(GL\_TEXTURE\_2D, texture[cube.surfaceType[i]-7 + textureType\*6]);

}

if(textureType == 0)

{

glColor3f(colors[cube.surfaceType[i]][0],colors[cube.surfaceType[i]][1],colors[cube.surfaceType[i]][2]);

}

else

{

glColor3f(1,1,1);

}

//下面针对光照模型设置法线方向

float direction = 1;

int dir = 0;

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

{

if(cube.points[eachSurface[i][0]][j]==cube.points[eachSurface[i][2]][j])

{

dir = j;

direction = cube.points[eachSurface[i][0]][j]/fabs(cube.points[eachSurface[i][0]][j]);

}

}

switch(dir)

{

case 0:

glNormal3f(direction, 0.0f, 0.0f);

break;

case 1:

glNormal3f( 0.0f, direction,0.0f);

break;

case 2:

glNormal3f(0.0f, 0.0f,direction);

break;

}

// 开始绘制面

glBegin(GL\_QUADS);

if(textureType >0)glTexCoord2f(0.0f, 0.0f);

glVertex3f(cube.points[eachSurface[i][0]][0],cube.points[eachSurface[i][0]][1],cube.points[eachSurface[i][0]][2]);

if(textureType >0)glTexCoord2f(1.0f, 0.0f);

glVertex3f(cube.points[eachSurface[i][1]][0],cube.points[eachSurface[i][1]][1],cube.points[eachSurface[i][1]][2]);

if(textureType >0)glTexCoord2f(1.0f, 1.0f);

glVertex3f(cube.points[eachSurface[i][2]][0],cube.points[eachSurface[i][2]][1],cube.points[eachSurface[i][2]][2]);

if(textureType >0)glTexCoord2f(0.0f, 1.0f);

glVertex3f(cube.points[eachSurface[i][3]][0],cube.points[eachSurface[i][3]][1],cube.points[eachSurface[i][3]][2]);

glEnd();

//画边框线

glLineWidth(3);

glColor3f(0,0,0);

glBegin(GL\_LINES); //画边框线

glVertex3f(cube.points[eachSurface[i][0]][0],cube.points[eachSurface[i][0]][1],cube.points[eachSurface[i][0]][2]);

glVertex3f(cube.points[eachSurface[i][1]][0],cube.points[eachSurface[i][1]][1],cube.points[eachSurface[i][1]][2]);

glVertex3f(cube.points[eachSurface[i][1]][0],cube.points[eachSurface[i][1]][1],cube.points[eachSurface[i][1]][2]);

glVertex3f(cube.points[eachSurface[i][2]][0],cube.points[eachSurface[i][2]][1],cube.points[eachSurface[i][2]][2]);

glVertex3f(cube.points[eachSurface[i][2]][0],cube.points[eachSurface[i][2]][1],cube.points[eachSurface[i][2]][2]);

glVertex3f(cube.points[eachSurface[i][3]][0],cube.points[eachSurface[i][3]][1],cube.points[eachSurface[i][3]][2]);

glVertex3f(cube.points[eachSurface[i][3]][0],cube.points[eachSurface[i][3]][1],cube.points[eachSurface[i][3]][2]);

glVertex3f(cube.points[eachSurface[i][0]][0],cube.points[eachSurface[i][0]][1],cube.points[eachSurface[i][0]][2]);

glEnd();

if(isSelected)

{

glPopName();

}

}

if(isRotating)

{

if(!rot90)//画完之后旋转回来

{

if(cube.points[0][rotAxis] == layer)

{

switch(rotAxis)

{

case 0:glRotatef(-rotAngle,1,0,0);break;

case 1:glRotatef(-rotAngle,0,1,0);break;

case 2:glRotatef(-rotAngle,0,0,1);break;

default:break;

}

}

}

}

}

**2、轨迹球部分**

使用ArcBall.h中定义的ArcBall\_t类，类中的一些重要函数：

当点击鼠标时，记录点击的位置：

void ArcBall\_t::click(const Point2fT\* NewPt)

当拖动鼠标时，记录当前鼠标的位置，并计算出旋转的量：

void ArcBall\_t::drag(const Point2fT\* NewPt, Quat4fT\* NewRot)

如果窗口大小改变，设置鼠标移动的范围：

void setBounds(GLfloat NewWidth, GLfloat NewHeight)

下面是完成计算需要的数据结果，都是一些矩阵和向量：

Matrix4fT Transform = { 1.0f, 0.0f, 0.0f, 0.0f,  
 0.0f, 1.0f, 0.0f, 0.0f,  
 0.0f, 0.0f, 1.0f, 0.0f,  
 0.0f, 0.0f, 0.0f, 1.0f };

Matrix3fT LastRot = { 1.0f, 0.0f, 0.0f,   
 0.0f, 1.0f, 0.0f,  
 0.0f, 0.0f, 1.0f };

Matrix3fT ThisRot = { 1.0f, 0.0f, 0.0f,   
 0.0f, 1.0f, 0.0f,  
 0.0f, 0.0f, 1.0f };

ArcBallT ArcBall(640.0f, 480.0f);   
Point2fT MousePt;   
bool isClicked = false; // 是否点击鼠标  
bool isRClicked = false; // 是否右击鼠标  
bool isDragging = false; // 是否拖动

在WindowProc函数中处理鼠标的按键操作：

LRESULT CALLBACK WindowProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

……

switch (uMsg){

case WM\_MOUSEMOVE:

MousePt.s.X = (GLfloat)LOWORD(lParam);

MousePt.s.Y = (GLfloat)HIWORD(lParam);

isClicked = (LOWORD(wParam) & MK\_LBUTTON) ? true : false;

isRClicked = (LOWORD(wParam) & MK\_RBUTTON) ? true : false;

break;

case WM\_LBUTTONUP:

isClicked = false;

break;

case WM\_RBUTTONUP:

isRClicked = false;

break;

case WM\_LBUTTONDOWN:

isClicked = true;

break;

case WM\_RBUTTONDOWN:

isRClicked = true;

break;

……

……

在Update函数中对参数做更新：

if (!isDragging)

{

if (isClicked)

{

……

}

else if(isRClicked)

{

isDragging = true; LastRot = ThisRot; ArcBall.click(&MousePt);

}

}

else

{

if (isClicked)

{

……

}

else if(isRClicked)

{

Quat4fT ThisQuat;

ArcBall.drag(&MousePt, &ThisQuat); Matrix3fSetRotationFromQuat4f(&ThisRot, &ThisQuat); Matrix3fMulMatrix3f(&ThisRot, &LastRot); Matrix4fSetRotationFromMatrix3f(&Transform, &ThisRot);

}

else ……

}

**3、鼠标滚轮缩放部分**

有了前面的基础，这一部分比较简单，只需对鼠标滚轮消息作出响应即可，这里把图形距离屏幕的深度限制在4到20.5之间：

case WM\_MOUSEWHEEL:

if( HIWORD(wParam)> WHEEL\_DELTA )

{

if(zMovement <= 20) zMovement += 0.5;

}

else if( HIWORD(wParam) == WHEEL\_DELTA )

{

if(zMovement >= 4.5) zMovement -= 0.5;

}

break;

**4、对鼠标选中魔方的拾取和魔方面旋转控制**

下面是绘制魔方的代码，分为两种情况进行绘制：拾取模式和普通渲染模式：

int Draw() //绘制魔方

{

GetWindowRect(g\_window->hWnd,&rcWin);

WndHeight = float(rcWin.bottom - rcWin.top);

int id = -1;

if(isRotating)

{

rotAngle += delta;

if(fabs(rotAngle) >= 90.0f)

{

rot90 = true;

}

}

if(isSelected)

{

GLuint selectBuf[512],hits = 0;

GLint viewport[4];//存放可视区参数

glGetIntegerv(GL\_VIEWPORT,viewport);//获取当前视口坐标参数

glSelectBuffer(512,selectBuf); //指定将“图元列表”（点击记录）返回到selectBuf数组中

glRenderMode (GL\_SELECT);

glInitNames();

glMatrixMode(GL\_PROJECTION);

glPushMatrix();

glLoadIdentity();

gluPickMatrix((GLdouble)MousePt.s.X,(GLdouble)(viewport[3]-MousePt.s.Y),2,2,viewport);//创建用于选择的投影矩阵栈

gluPerspective(45.0f, (GLfloat) (viewport[2]-viewport[0])/(GLfloat) (viewport[3]-viewport[1]), 0.1f, 1000.0f);

glMatrixMode(GL\_MODELVIEW); // 选择模型变换矩阵

// 选择绘制魔方

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

glLoadIdentity();

glPushMatrix();

glTranslatef(xMovement\*zMovement/(1.25f\*WndHeight),yMovement\*zMovement/(1.25f\*WndHeight),-zMovement);

glMultMatrixf(Transform.M);

//绘制每个块

for(int i = 0;i<order\*order\*order;i++)

{

DrawCube(cubes[i]);

}

glPopMatrix();

glMatrixMode(GL\_PROJECTION);

glPopMatrix();

glMatrixMode(GL\_MODELVIEW);

hits = glRenderMode (GL\_RENDER);

id = processHits (hits, selectBuf);

glFlush();

}

else

{

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

glLoadIdentity();

glTranslatef(xMovement\*zMovement/(1.25f\*WndHeight),yMovement\*zMovement/(1.25f\*WndHeight),-zMovement);

glPushMatrix();

glMultMatrixf(Transform.M);

for(int i = 0;i<order\*order\*order;i++)

{

DrawCube(cubes[i]);

}

glPopMatrix();

glFlush ();

}

if(isRotating)

{

if(rot90)

{

isRotating = false;

rot90 = false;

rotAngle = 0;

cube1 = -1;

cube2 = -1;

face1 = -1;

face2 = -1;

isSelected = false;

}

}

return id;

}

得到拾取到的物体数目hits之后，对拾取到的物体列表进行处理，找出离视点最近的一个物体：

int processHits (GLint hits, GLuint selectBuff[])

{

modelselect = -1;

if (hits >0)

{

int n=0;

unsigned int minz=selectBuff[1];

unsigned int temp;

for(int i=1;i<hits;i++)

{

temp = selectBuff[1+i\*4];

if (temp<minz)

{

n=i;

minz=temp;

}

}

modelselect = selectBuff[3+n\*4];

}

return modelselect;

}

在Update函数中对鼠标左键控制魔方面旋转的动作作出响应：

void Update (DWORD milliseconds)

{

if (g\_keys->keyDown [VK\_ESCAPE] == TRUE) TerminateApplication (g\_window);

if (g\_keys->keyDown [VK\_F1] == TRUE)

ToggleFullscreen (g\_window);

if (!isDragging)

{

if (isClicked)

{

if(!isRotating)

{

isDragging = true;

isSelected = true;

face1 = Draw();

cube1 = face1/6;

}

}

else if(isRClicked)

{

isDragging = true;

LastRot = ThisRot; ArcBall.click(&MousePt);

}

}

else

{

if (isClicked)

{

if(!isRotating)

{

int tempSelect = Draw();

if(tempSelect!= face1 && face2 == -1)

{

face2 = tempSelect;

}

}

}

else if(isRClicked)

{

Quat4fT ThisQuat;

ArcBall.drag(&MousePt, &ThisQuat);

Matrix3fSetRotationFromQuat4f(&ThisRot, &ThisQuat);

Matrix3fMulMatrix3f(&ThisRot, &LastRot);

Matrix4fSetRotationFromMatrix3f(&Transform, &ThisRot);

}

else

{

isDragging = false;

isSelected = false;

if(face2 != -1)

{

isRotating = true;

cube2 = face2/6;

getFaceCenter(face1,center1);

getFaceCenter(face2,center2);

if((center1[0] == center2[0])&&(fabs(center1[0])!= fabs(distance)))

{

rotAxis = 0;

layer = cubes[cube1].points[0][0];

if((center1[1]\*center2[2]-center1[2]\*center2[1])>0) delta = SPEED;

else delta = -SPEED;

}

else if((center1[1] == center2[1])&&(fabs(center1[1])!= fabs(distance)))

{

rotAxis = 1;

layer = cubes[cube1].points[0][1];

if((center1[2]\*center2[0]-center1[0]\*center2[2])>0) delta = SPEED;

else delta = -SPEED;

}

else if((center1[2] == center2[2])&&(fabs(center1[2])!= fabs(distance)))

{

rotAxis = 2;

layer = cubes[cube1].points[0][2];

if((center1[0]\*center2[1]-center1[1]\*center2[0])>0) delta = SPEED;

else delta = -SPEED;

}

else

{

isRotating = false;

}

face2 = -1;

}

cube1 = -1;

cube2 = -1;

}

}

}

**实验效果**

截图如下。以下是几张效果图：





