**《计算机图形学实验》综合实验报告**

**题目 OpenGL实现三维图形渲染**

**学 号 20201060360**

**姓 名 唐庆勇**

**指导教师 钱文华**

**日 期 2022-6-19**

# 摘要：

使用照相机或者手机的拍照功能，我们能把一些三维的图形留在我们的二维平面上。而借助OpenGL的一些相关函数，我们同样可以在二维的平面中观察到三维的真实感图形，这一技术便叫做三维观察。简而言之，先建立观察用到的坐标系（相当于我们摄像头的位置，拍摄方向与相机的正向上方向）并对观察空间进行裁剪，接着利用投影函数将我们所构建的对象投影到观察平面上，最后辅之以光照模型和透明处理、纹理贴图等，就可以实现对一个真实感三维图形的渲染。而本文便运用三维观察与三维旋转，并加以光照与纹理贴图，实现了对一个三维茶壶的绘制与旋转。

**关键词：**三维观察、光照模型、纹理贴图、三维旋转

目录

[摘要： 2](#_Toc106751990)

[实验背景和内容 4](#_Toc106751991)

[实验背景： 4](#_Toc106751992)

[实验内容： 4](#_Toc106751993)

[开发工具： 4](#_Toc106751994)

[程序设计、实现目的及基本模块介绍 4](#_Toc106751995)

[关键算法介绍和程序实现步骤 5](#_Toc106751996)

[关键算法的理论介绍： 5](#_Toc106751997)

[三维观察： 5](#_Toc106751998)

[纹理贴图： 6](#_Toc106751999)

[光照模型： 6](#_Toc106752000)

[程序实现步骤： 7](#_Toc106752001)

[实验结果及存在问题： 7](#_Toc106752002)

[实验截图： 7](#_Toc106752003)

[实验结果分析： 9](#_Toc106752004)

[当前存在的问题： 9](#_Toc106752005)

[实验体会及小结： 9](#_Toc106752006)

[参考文献： 10](#_Toc106752007)

[附录： 10](#_Toc106752008)

# 实验背景和内容

## 实验背景：

经过一学期的计算机图形学和OpenGl的理论知识学习以及实验课的操作后，在掌握画线算法、二维以及三维变换、二维观察及三维观察等技术的基础上实现对三维图形的渲染，并加入纹理、色彩或光照等效果，意在加深所学知识，并为以后绘制更为复杂的三维图形（如动物、建筑物）等打下基础。

## 实验内容：

利用Visual C++, OpenGL, Java等工具，实现三维图形渲染，并且加入纹理、色彩、光照、阴影、透明等效果，可采用光线跟踪、光照明模型、纹理贴图、纹理映射等算法。

# 开发工具：

Codeblocks20.03以及OpenGL。

# 程序设计、实现目的及基本模块介绍

**该部分先从基本模块介绍入手，再描述每个模块中的程序设计及程序的目的。**

**1、光照实现模块：给予茶壶光照：**

void init(void)//定义光源及茶壶的性质，进而实现光照

**2、纹理贴图模块：实现对茶壶的贴图：**

unsigned char \*LoadBitmapFile(char \*filename, BITMAPINFOHEADER )\*bitmapInfoHeade

r)//读取纹理图片

void texload(int i, char \*filename) //加载纹理的函数

void initWenli()//初始化纹理的函数

**3、显示模块：实现茶壶的显示以用户的交互功能**

void display(void)//生成茶壶，开启光照渲染

void reshape(int w, int h)//在窗口大小改变时维持图形的比例

void myKeyboard(int key, int x, int y)//键盘回调函数，实现与用户的交互

# 关键算法介绍和程序实现步骤

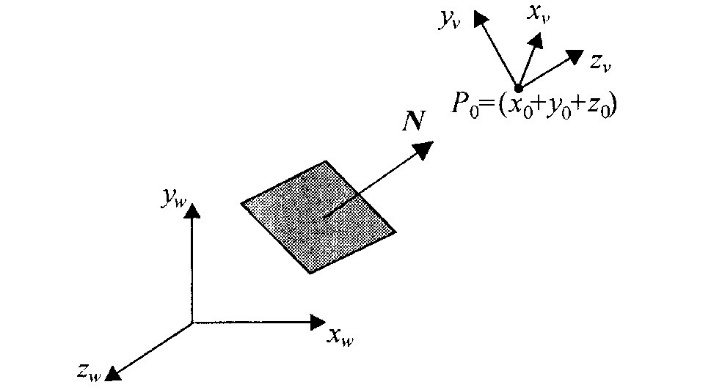
## 关键算法的理论介绍：

### 三维观察：

**1、三维观察坐标系的建立：**【1】

①挑选一个世界坐标点使其成为观察坐标系的原点。

②定义一个垂直于观察方向的平面，成为观察平面，该平面与xy平面平行。对象到观察平面的投影与场景在输出设备上的显示相对应。定义观察平面的法向量N为正z轴方向，即正z轴方向为观察方向的反方向。建立z的方向仅需要N的方向，与其模无关，因为可计算N得到规范化单位向量，如下图所示。



③指定某一向量 V 来选择观察向上向量它是用来标志物体朝向（朝上的方向）的矢量。定义正 y 轴方向为V 在观察平面上的投影。观察向上向量 V 的选择是任意的，只要 V 不平行 N即可。实际上在应用 中为了简化问题可以简单地选择 V 为平行于世界坐标系 y 轴的方向，即 V ＝（0，1，0）。

该点由GluLookAt()函数完成。

**2、三维世界坐标到三维观察坐标的变换**

①平移观察参考点到世界坐标系原点，记该步变换所使用的矩阵为T。

②进行旋转，让三维观察坐标轴对应到世界坐标轴，记该步变换所使用的矩阵为R。

该步可以经过三维复合变换得到，即R·T。

**3、实现三维观察**

运用下述函数即可实现三维观察：

glMatrixMode（GL\_MODEVIEW）；//设定当前操作矩阵为模型视图矩阵堆栈

glLoadIdentity（ ）；//指定一个4阶单位矩阵为当前的操作矩阵

glMultMatrixfv（T）；//将4阶矩阵 T 与当前矩阵相乘并将结果返回到当前矩阵，实现观察参考点与世界坐标系原点的重合

glMultMatrixfv（R）；//将4阶矩阵 R 与当前矩阵相乘并将结果返回到当前矩阵，实现观察坐标轴与世界坐标轴的重合

### 纹理贴图：

**1、定义纹理：**纹理在指定的数组中，此数组可以表达网格点上的纹理值。而它的函数为void glTexImage2D ()。

**2、控制纹理：**把纹理映射到模型上的第一步，会综合考虑“滤波”方面的问题（纹理图像的选取）与“重复与缩限”方面（纹理坐标的截取与纹理坐标的重复）的问题。

voidglTexParameterf(GLenum target,Lenumpname, GLfloat param); target参数为目标纹理,pname参数的取值有GL\_TEXTURE\_MIN\_FILTER、GL\_TEXTURE \_MAG\_FILTER、GL\_TEXTURE\_WRAP\_S、GL\_TEXT URE\_WRAP\_T。

**3、纹理贴图模式：**选择将纹理“贴”在图形上的方式。void glTexEnv {if} (GLenum target, GLenum pname, GLfloat param)。

**4、纹理坐标【2】：**将纹理图像映射到建好的几何模型上。当在处理 纹理映射的图景的时候,需要给模型每一个端点界定 其坐标,同时还要给纹理的各个顶点界定其坐标。 几 何坐标和纹理坐标， 前者可以让模型端点绘制在屏 幕上的具体方位, 后者可以判定其图象里是何元素 给予这个端点。 纹理坐标正常情况下是一到四维的, 假如用齐次坐标去体现,设为(s,t,r,q)。

它的函数为：gltexCoord2{sifd}(Glint s, Glint t);而 几何坐标的函数为glVertex\*()。

**5、纹理对象：**储存纹理的数据。

①生成纹理对象：利用函数glGenTexture（）来提供使用的纹理对象名称。

②构建和使用纹理对象：利用函数glBindTexture（）构建和使用纹理对象。

③删除纹理对象：使用glDeleteTextures （ ）去消除纹理对象所占有的资源。

**6、纹理数据的获取：**以手动生成或读取外部资源的方式获取纹理数据。

经过上述6步、便可成功实现纹理贴图。

### 光照模型：

1、创建并启动光源：任意发出辐射的对象称为一个光源。定义光源的位置后便可让其产生指定方向的光照。该步由函数gLight()完成。

此外，还需设立光照模型的性质：

①环境光：即使在黑暗的情况下，世界上通常也仍然有一些光亮（月亮、远处的光），所以物体几乎永远不会是完全黑暗的。因此需要一个背景光，它永远会给物体一些颜色，这便是环境光。

②漫反射：模拟光源对物体的方向性影响。它是冯氏光照模型中视觉上最显著的分量。物体的某一部分越是正对着光源，它就会越亮。

③镜面反射：模拟有光泽物体上面出现的亮点。镜面光照的颜色相比于物体的颜色会更倾向于光的颜色。

最后，启动光源，相当于打开手电筒。用函数glEnable()实现。

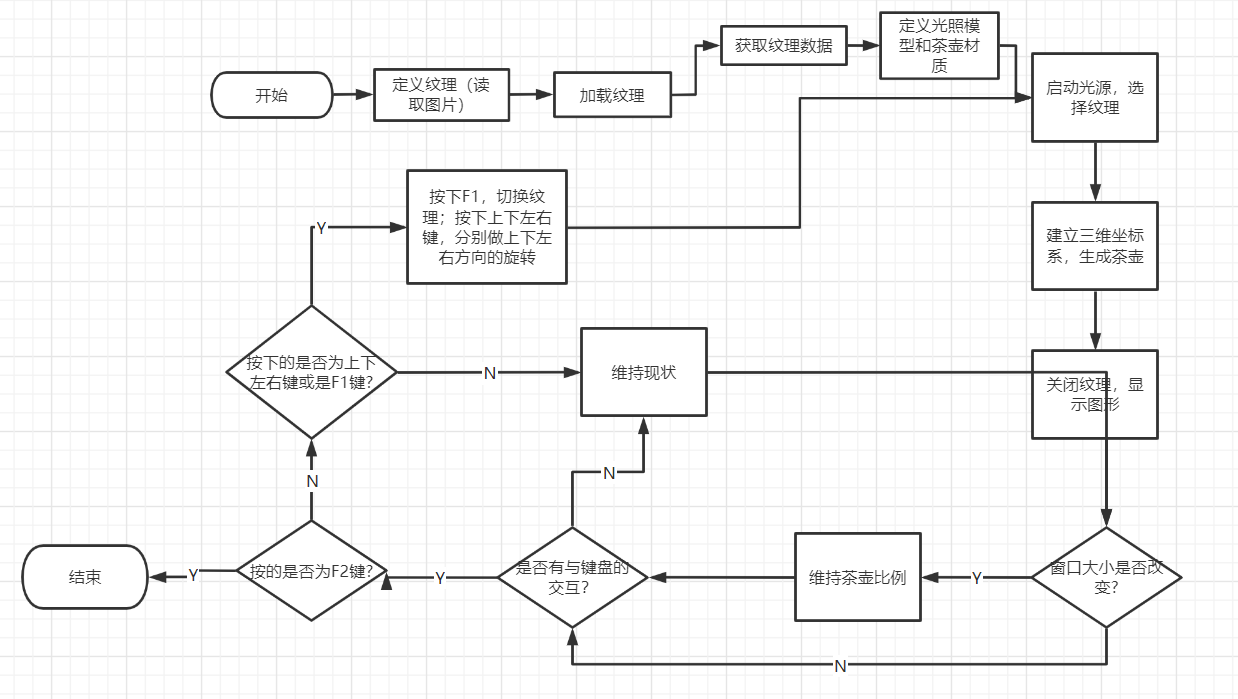
2、选择光照模式：创建光源后还要为场景选择确定光照模式, OpenGL的光照模式由三个部分组成，分别为全局环境光RGB值的设定, 观察点位置和物体是前面受光, 还是双面受光的确定。函数为:

gl Light Model\* ( )；

glShade Modle (GL-SMOOTH/FLAT ) 函数用来设定Gouraud或平面明暗处理方式。

3、设置材质的性质，该步与2点类似。该步由函数glMaterial()完成。

## 程序实现步骤：



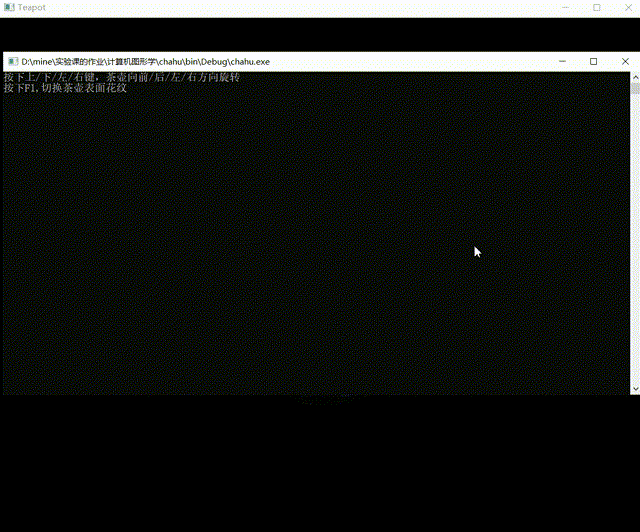
注：向上旋转是指按照用户观察到的二维平面的数值向上方向旋转，向下则与向上相反；

向左旋转是指按照用户观察到的二维平面的水平向左方向旋转，向右则与向左相反；

# 实验结果及存在问题：

## 实验截图：

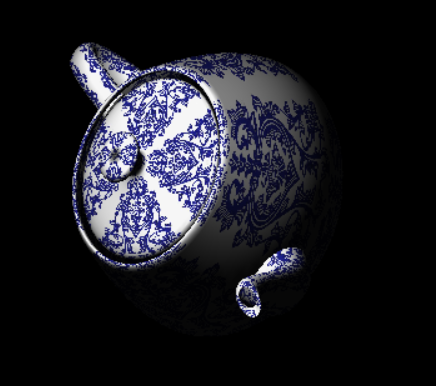
（1）动态演示（GIF）:



（2）静态截图：



原始茶壶 向上旋转（↑键）



向下旋转（↓键） 向左旋转（←键）



向右旋转（→键） 切换茶壶纹理（F1键）

## 实验结果分析：

程序首先生成了一个在用户看来面向东南方向的茶壶，并成功加载了第一个纹理（青花瓷花纹）与光照（光照方向在用户看来从左到右）。其后成功调用键盘回调函数，实现了对茶壶的三维旋转与纹理切换。

## 当前存在的问题：

①程序读入的图案被分成四部分贴在了茶壶的表面，从而导致以其他图案作为纹理图案时（如水墨画）不美观。

②对光照颜色的把握不够精准。

③茶壶使用函数直接生成，表面的刻画不够细致。

# 实验体会及小结：

通过本次实验，我对三维变换与三维观察等知识的理解更加深刻，同时接触并初步学习了纹理贴图及光照渲染等相关方面的知识。

在本次实验中，我也遇到了不少的问题。比如面对陌生知识的茫然：光照渲染以及纹理贴图等知识没有接触过；比如面对程序bug时的“憔悴”：读取纹理的失败，键盘回调无响应或是三维显示没有达到预想的效果，等等。但“世上无难事，只怕有心人”，通过在网络上查询相关书籍、在bilibili等平台观看演示视频以及与同学讨论等方式，这些问题便一一化解。

事实上，问题启发着我们去思考，启发着我们去动手实践。有句古话叫做“学而不思则罔，思而不学则殆”，这是说在学习上，不能只一昧地学，也不能只一昧地空想，只有“学而思”，并且“思而行”，我们才能真正学好一门课程，掌握一项技术。同时在面对问题时，不要惧怕困难，因为任何人的人生都不是一帆风顺的，直视挫折，勇敢应对，我们才可以穿过重重障碍，站在新的高度，接触到新的世界。

# 参考文献：

[1]蒋亚军.世界场景中三维观察变换的实现[J].吉首大学学报(自然科学版),2006(05):31-33.

[2]兰一麟涛,钱伟,田明银.基于OpenGL的三维纹理贴图绘制技术研究与实现[J].甘肃科技,2015,31(22):32-34.

# 附录：

#include <windows.h>

#include <GL/glut.h>

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

#define BITMAP\_ID 0x4D42

#define Height 16

#define Width 16

static GLfloat xRotate = 0.0;//x方向旋转角度

static GLfloat yRotate = 0.0;//y方向旋转角度

static GLfloat controlValue = 10.0;

GLuint texture[2];

int status;

//定义光源及茶壶的性质，进而实现光照

void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_SMOOTH);//设定平面明暗处理方式

// 定义光源

GLfloat light\_position[] = { -100.0,100.0,100.0,1.0 }; // （点）光源的位置

GLfloat light\_ambient[] = { 0.1f,0.1f,0.1f,1.0f }; // 环境光

GLfloat light\_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // 漫反射光，白色

GLfloat light\_specular[] = { 0.8f,0.8f,0.8f,1.0f }; // 镜面反射光，白色

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);//位置属性

// 开启光源

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_DEPTH\_TEST);

// 定义壶的材质

GLfloat mat\_ambient[] = { 0.1f,0.1f,0.1f,1.0f };

GLfloat mat\_diffuse[] = { 0.8f,0.8f,0.8f,1.0f }; // 材质的漫反射光，白色

GLfloat mat\_specular[] = { 0.8f,0.8f,0.8f,1.0f }; // 材质的镜面反射光，白色

GLfloat mat\_emission[] = { 0.0f,0.0f,0.0f,1.0f }; // 材质的幅射光，黑色

GLfloat mat\_shininess[] = { 5.0 };

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);//用光照计算当前材质的属性

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_EMISSION, mat\_emission);

glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);

}

//读纹理图片

unsigned char \*LoadBitmapFile(char \*filename, BITMAPINFOHEADER \*bitmapInfoHeader)

{

FILE \*filePtr; // 文件指针

BITMAPFILEHEADER bitmapFileHeader; // bitmap文件头

unsigned char\*bitmapImage; // bitmap图像数据

int imageIdx = 0; // 图像位置索引

unsigned char tempRGB; // 交换变量

// 以“二进制+读”模式打开文件filename

filePtr = fopen(filename, "rb");

if (filePtr == NULL) {

printf("file not open\n");

return NULL;

}

// 读入bitmap文件图

fread(&bitmapFileHeader, sizeof(BITMAPFILEHEADER), 1, filePtr);

// 验证是否为bitmap文件

if (bitmapFileHeader.bfType != BITMAP\_ID) {

fprintf(stderr, "Error in LoadBitmapFile: the file is not a bitmap file\n");

return NULL;

}

// 读入bitmap信息头

fread(bitmapInfoHeader, sizeof(BITMAPINFOHEADER), 1, filePtr);

// 将文件指针移至bitmap数据

fseek(filePtr, bitmapFileHeader.bfOffBits, SEEK\_SET);

// 为装载图像数据创建足够的内存

bitmapImage = new unsigned char[bitmapInfoHeader->biSizeImage];

// 验证内存是否创建成功

if (!bitmapImage) {

fprintf(stderr, "Error in LoadBitmapFile: memory error\n");

return NULL;

}

// 读入bitmap图像数据

fread(bitmapImage, 1, bitmapInfoHeader->biSizeImage, filePtr);

// 确认读入成功

if (bitmapImage == NULL) {

fprintf(stderr, "Error in LoadBitmapFile: memory error\n");

return NULL;

}

//由于bitmap中保存的格式是BGR，下面交换R和B的值，得到RGB格式

for (imageIdx = 0;imageIdx < bitmapInfoHeader->biSizeImage; imageIdx += 3) {

tempRGB = bitmapImage[imageIdx];

bitmapImage[imageIdx] = bitmapImage[imageIdx + 2];

bitmapImage[imageIdx + 2] = tempRGB;

}

// 关闭bitmap图像文件

fclose(filePtr);

return bitmapImage;

}

//加载纹理的函数

void texload(int i, char \*filename)

{

BITMAPINFOHEADER bitmapInfoHeader; // bitmap信息头

unsigned char\* bitmapData; // 纹理数据

bitmapData = LoadBitmapFile(filename, &bitmapInfoHeader);

glBindTexture(GL\_TEXTURE\_2D, texture[i]);

// 指定当前纹理的放大或缩小过滤方式

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

glTexImage2D(GL\_TEXTURE\_2D,

0, //mipmap层次(通常为，表示最上层)

GL\_RGB, //纹理有红、绿、蓝数据

bitmapInfoHeader.biWidth, //纹理宽带

bitmapInfoHeader.biHeight, //纹理高度

0, //边框(0=无边框, 1=有边框)

GL\_RGB, //bitmap数据的格式

GL\_UNSIGNED\_BYTE, //每个颜色数据的类型

bitmapData); //bitmap数据指针

}

//定义纹理的函数

void initWenli()

{

glGenTextures(2, texture); // 第一参数是需要生成标示符的个数, 第二参数是返回标示符的数组

texload(0, "C://Users//HUAWEI//Desktop//huawen.bmp");//茶壶花纹

texload(1, "C://Users//HUAWEI//Desktop//huawen1.bmp");//另外一个花纹

}

//显示函数

void display(void)

{

// 清除之前的深度缓存

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

glLoadIdentity();

// 与显示相关的函数

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texture[status]); //选择纹理texture[status]

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);//设置纹理受光照影响

gluLookAt(20.0,20.0,10.0,0.0,0.0,0.0,0.0,1.0,0.0);

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

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

//生成茶壶

glutSolidTeapot(50.0);

glDisable(GL\_TEXTURE\_2D); //关闭纹理texture[status]

glFlush();

glutSwapBuffers();

}

//维持图形比例

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-200, 200, -200 \* (GLfloat)h / (GLfloat)w, 200 \* (GLfloat)h / (GLfloat)w, -100.0, 100.0);

else

glOrtho(-200 \* (GLfloat)w / (GLfloat)h, 200 \* (GLfloat)w / (GLfloat)h, -200, 200, -100.0, 100.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

//键盘回调函数

void myKeyboard(int key, int x, int y)

{

switch (key)

{

case GLUT\_KEY\_DOWN:

{

xRotate = xRotate + 2.0;

if (xRotate > 360)

xRotate = xRotate - 360.0;

glutPostRedisplay();

break;

}

case GLUT\_KEY\_UP:

{

xRotate = xRotate - 2.0;

if (xRotate > 360)

xRotate = xRotate - 360.0;

glutPostRedisplay();

break;

}

case GLUT\_KEY\_LEFT:

{

yRotate = yRotate - 2.0;

if (yRotate > 360)

yRotate = yRotate - 360.0;

glutPostRedisplay();

break;

}

case GLUT\_KEY\_RIGHT:

{

yRotate = yRotate + 2.0;

if (yRotate > 360)

yRotate = yRotate - 360.0;

glutPostRedisplay();

break;

}

case GLUT\_KEY\_F1: { //切换茶壶纹理

if (status == 0)status = 1;

else if (status == 1)status = 0;

glutPostRedisplay();

break;

}

case GLUT\_KEY\_F2:

printf("感谢您的使用\n");

exit(0);

default:

break;

}

}

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

{

status = 0;//默认选用第一种花纹

printf("按下上/下/左/右键，茶壶向前/后/左/右方向旋转\n");

printf("按下F1,切换茶壶表面花纹\n");

printf("按下F2，程序退出\n");

glutInit(&argc, argv);

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

glutInitWindowSize(960, 960);

glutInitWindowPosition(100, 100);

glutCreateWindow("Teapot");

init();

initWenli();

glutDisplayFunc(display);//注册绘图函数

glutReshapeFunc(reshape);//注册调整窗口大小变化时的响应函数

glutSpecialFunc(&myKeyboard);//注册键盘回调函数

glutMainLoop();

return 0;

}