**计算机图形学实验**

**姓 名： 黄珀芝**

**学 号： 20201050331**

**专 业： 计算机科学与技术**

**教 师： 钱文华**

**摘要**

本文在分析和研究OpenGL下实现三维真实感效果处理技术的基础上,在场景中绘制三维物体模型,通过两个实例展示了不同功能，其中具有纹理贴图和自定义光照方向等功能，定义了光照模型、材质属性、视口大小、投影方式和纹理映射,从而实现了三维图形的真实感显示,并具有改变光照条件,设置渲染模式等功能。两个实例分别是机器人的3D绘制和茶壶的贴图纹理。

关键词:OpenGL 三维图形 真实感 纹理映射 光照

**目 录**

[1 实验背景及内容 1](#_Toc280776659)

[2 程序设计及实现目的 2](#_Toc280776660)

[3 关键算法的理论介绍和程序实现 4](#_Toc280776661)

[4 实验运行及测试分析 7](#_Toc280776662)

[5 实验体会及小结 2](#_Toc280776663)1

# 1 实验背景及内容

OpenGL(Open Graphic Library)是由SGI公司的IRIS GL 图形库发展而来的三维真实感图形生成工具,鉴于它的跨平台、高质量、高效率、功能完善等特点,已经成为各种平台下的三维图形制作及交互式场景处理的工业标准,被广泛地运用于科学计算可视化、计算机动画和虚拟现实等计算机图形学热点问题的解决之中。

在Windows平台下,OpenGL和DirectX是两个开发三维图形应用程序的标准,OpenGL 提供了二维和三维建模、变换、光线处理、色彩处理、纹理映射、运动模糊、动画和实时交互等功能,是绘制真实感三维图形、建立三维交互场景、实现虚拟现实的高性能图形开发工具软件包。与DirectX 相比,用OpenGL 来绘制三维地形具有图形质量高、程序可移植性好等优点[2]。

本文利用三维图形库OpenGL ,在codeblock GLUT环境下开发了两种种基于OpenGL 的三维真实感图形。

本次实验要求利用Visual C++, OpenGL, Java等工具，实现三维图形渲染，自定义三维图形，三维图形不能仅仅是简单的茶壶、球体、圆柱体、圆锥体等图形，渲染过程须加入纹理、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。

本文构造了两个程序实例来完成任务——鼠标控制机器人的真实感图形以及自由选择纹理贴图的茶壶三维。

# 2 程序设计及实现目的

想要实现任务目标，首先明确程序设计的要点以及实现目标是什么。

2.1 定义光照模型及材质属性

要使STL 格式实体实现真实感图形显示,建立光照模型是必须的。为增强真实感图形显示的效果,也可以加入材质属性的定义。在opengl中, 光照模型由环境反射光(Ambient Light)、漫反射光(Diffuse Light)、镜面反射光(Specular Light)等组成,它们可模拟真实的光照效果。在光照中还应加入消隐处理,以确定可见面,隐藏不可见面:

glDepthFunc(GL-LESS); glEnable(GL-DEPTH-TEST)[4];

这两个函数都可以在后面的程序中找到。

材质颜色是通过材质对R、G、B的近似反光率来定义的,如同光源模型,也分成环境反射、漫反射、镜面反射成分, 函数glMaterial实现了材质的设置。材质属性与光源特性的结合就是观察的最终效果。

2.2 视口及投影

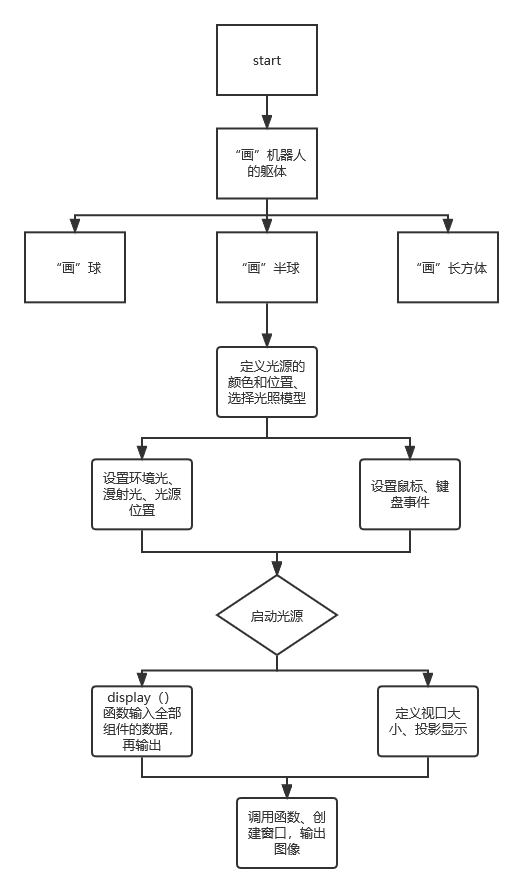
三维实体要在二维屏幕上显示出来,须经以下坐标转换:世界坐标系→用户坐标系→投影坐标系→窗口坐标系,各步转换通过矩阵操作实现。OpenGL 使用相应的函数来事先设置矩阵参数,为实体坐标转换作好初始化工作,其中最重要的是设置视口和投影。视口可设置成应用程序窗口大小,通过函数glViewport实现。OpenGL 中观察物体的空间称为视景体,视景体投影变换来定义。显示在屏幕上的内容就是视景体中的内容,只有把三维物体绘制在视景体中,才能显示在屏幕上。本文采用的正交投影定义的是一个长方体的视景体,物体在屏幕上的显示尺寸不受所处距离远近的影响。OpenGL 使用函数glOrtho()来定义正交投影[3]。

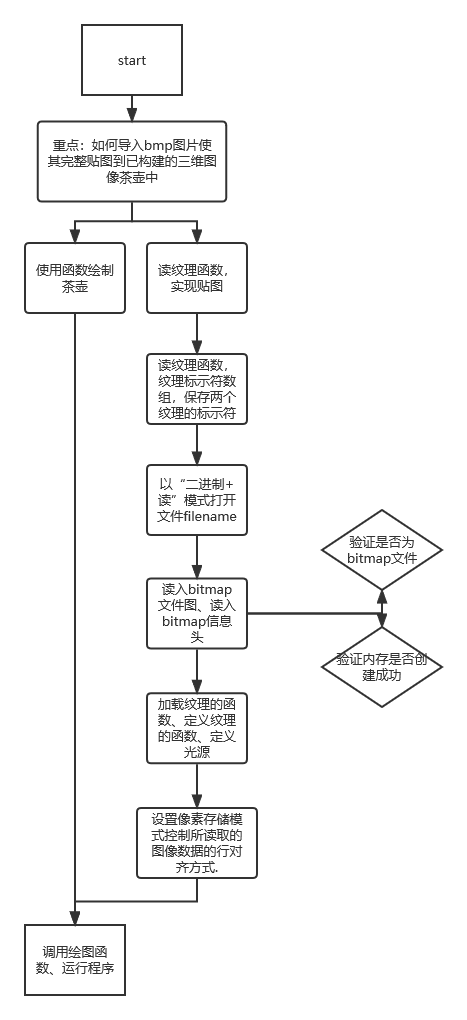
2.3 纹理映射

纹理映射通常是将2D 图形(通常是位图)映射到3D 物体上的一种技术,是三维图形真实感效果处理的关键,它通过将图像粘贴于几何模型表面来增强图形的真实感,在很大程度上能够简化复杂模型的几何描述[1]。

纹理的映射实际上是纹理坐标和物体空间坐标一一对应的过程。本设计中采用的是多分辨率纹理映射,在动态场景中,当带纹理的物体远离视点时,纹理图像必须随物体变小而变小,为此,必须对纹理图像进行滤波,使其大小适合映射在物体上,同时避免在视觉上产生人工雕琢的效果,如晃动、闪光和闪烁。为了避免这种现象,可以指定一系列预先通过滤波生成的、分辨率递减的纹理。

# 3 算法的理论介绍和程序实现

3.1具有光照阴影的可鼠标键盘操控机器人

3.2具有纹理贴图效果的茶壶

3.3重要算法、函数的理论介绍

|  |  |
| --- | --- |
| LoadBitmap（） | 将.bmp添加到资源里， LoadBitmap(IDB\_BITMAP1) |
| glEanble(GL\_LIGHTING)  glDisable(GL\_LIGHTING)  glLightModel  glLight  glMaterial  glColorMaterial  glNormal | 开启光照计算（默认情况下OpenGL关闭光照计算）  关闭光照计算  设置光照模型  设置光的参数  设置物体材质  设置物体材质 |
| 纹理贴图四要素 | 在OpenGL中实现贴图，需要知道以下四个名词：纹理对象、纹理单元、采样对象和采样Uniform变量。 |

# 4 实验运行及测试分析

4.1机器人代码

#include <windows.h>

#include <stdlib.h>

#include <stdarg.h>

#include <stdio.h>

#include <GL/glut.h>

#include <GL/gl.h>

#include <GL/glaux.h>

#include <mmsystem.h>

#include <math.h>

#define SOLID 1

#define WIRE 2

int moveX,moveY;

int spinX = 0;

int spinY = 0;

int des = 0;

void init() {

//定义光源的颜色和位置

GLfloat ambient[] = { 0.5, 0.8, 0.1, 0.1 };

GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat position[] = { -80.0, 50.0, 25.0, 1.0 };

//选择光照模型

GLfloat lmodel\_ambient[] = { 0.4, 0.4, 0.4, 1.0 };

GLfloat local\_view[] = { 0.0 };

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_SMOOTH);

//设置环境光

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

//设置漫射光

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

//设置光源位置

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

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, lmodel\_ambient);

glLightModelfv(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, local\_view);

//启动光照

glEnable(GL\_LIGHTING);

//启用光源

glEnable(GL\_LIGHT0);

}

//画球

void drawBall(double R, double x, double y,double z, int MODE) {

glPushMatrix();

glTranslated(x,y,z);

if (MODE == SOLID) {

glutSolidSphere(R,20,20);

} else if (MODE ==WIRE) {

glutWireSphere(R,20,20);

}

glPopMatrix();

}

//画半球

void drawHalfBall(double R, double x, double y,double z, int MODE) {

glPushMatrix();

glTranslated(x,y,z);

GLdouble eqn[4]={0.0, 1.0, 0.0, 0.0};

glClipPlane(GL\_CLIP\_PLANE0,eqn);

glEnable(GL\_CLIP\_PLANE0);

if (MODE == SOLID) {

glutSolidSphere(R,20,20);

} else if (MODE ==WIRE) {

glutWireSphere(R,20,20);

}

glDisable(GL\_CLIP\_PLANE0);

glPopMatrix();

}

//画长方体

void drawSkewed(double l, double w, double h, double x, double y, double z, int MODE) {

glPushMatrix();

glScaled(l, w, h);

glTranslated(x, y, z);

if (MODE == SOLID) {

glutSolidCube(1);

} else if (MODE ==WIRE) {

glutWireCube(1);

}

glPopMatrix();

}

void display(void) {

//清除缓冲区颜色

glClear(GL\_COLOR\_BUFFER\_BIT);

//定义白色

glColor3f(1.0, 1.0, 1.0);

//圆点放坐标中心

glLoadIdentity();

//从哪个地方看

gluLookAt(-2.0, -1.0, 20.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glPushMatrix();

glRotated(spinX, 0, 1, 0);

glRotated(spinY, 1, 0, 0);

glTranslated(0, 0, des);

//头

drawBall(2, 0, 1, 0, SOLID);

//身体

drawSkewed(5, 4.4, 4, 0, -0.75, 0, SOLID);

//肩膀

drawHalfBall(1, 3.5, -2.1, 0, SOLID);

drawHalfBall(1, -3.5, -2.1, 0, SOLID);

//胳膊

drawSkewed(1, 3, 1, 3.5, -1.3, 0, SOLID);

drawSkewed(1, 3, 1, -3.5, -1.3, 0, SOLID);

//手

drawBall(1, 3.5, -6.4, 0, SOLID);

drawBall(1, -3.5, -6.4, 0, SOLID);

//腿

drawSkewed(1.2, 3, 2, 1, -2.4, 0, SOLID);

drawSkewed(1.2, 3, 2, -1, -2.4, 0, SOLID);

//脚

drawSkewed(1.5, 1, 3, 0.9, -9.2, 0, SOLID);

drawSkewed(1.5, 1, 3, -0.9, -9.2, 0, SOLID);

glPopMatrix();

glutSwapBuffers();

}

//鼠标点击事件

void mouseClick(int btn, int state, int x, int y) {

moveX = x;

moveY = y;

GLfloat ambient[] = { (float)rand() / RAND\_MAX, (float)rand() / RAND\_MAX, (float)rand() / RAND\_MAX, 0.1 };

//设置环境光

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

//启用光源

glEnable(GL\_LIGHT0);

}

//键盘事件

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

switch (key) {

case 'a':

spinX -= 2;

break;

case 'd':

spinX += 2;

break;

case 'w':

des += 2;

break;

case 's':

des -= 2;

break;

}

glutPostRedisplay();

}

// 鼠标移动事件

void mouseMove(int x, int y) {

int dx = x - moveX;

int dy = y - moveY;

printf("dx;%dx,dy:%dy\n",dx,dy);

spinX += dx;

spinY += dy;

glutPostRedisplay();

moveX = x;

moveY = y;

}

void reshape(int w, int h) {

//定义视口大小

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

//投影显示

glMatrixMode(GL\_PROJECTION);

//坐标原点在屏幕中心

glLoadIdentity();

//操作模型视景

gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);

glMatrixMode(GL\_MODELVIEW);

}

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

//初始化

glutInit(&argc, argv);

//设置显示模式

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

//初始化窗口大小

glutInitWindowSize(500, 500);

//定义左上角窗口位置

glutInitWindowPosition(100, 100);

//创建窗口

glutCreateWindow(argv[0]);

//初始化

init();

//显示函数

glutDisplayFunc(display);

//窗口大小改变时的响应

glutReshapeFunc(reshape);

//鼠标点击事件，鼠标点击或者松开时调用

glutMouseFunc(mouseClick);

//鼠标移动事件，鼠标按下并移动时调用

glutMotionFunc(mouseMove);

//键盘事件

glutKeyboardFunc(keyPressed);

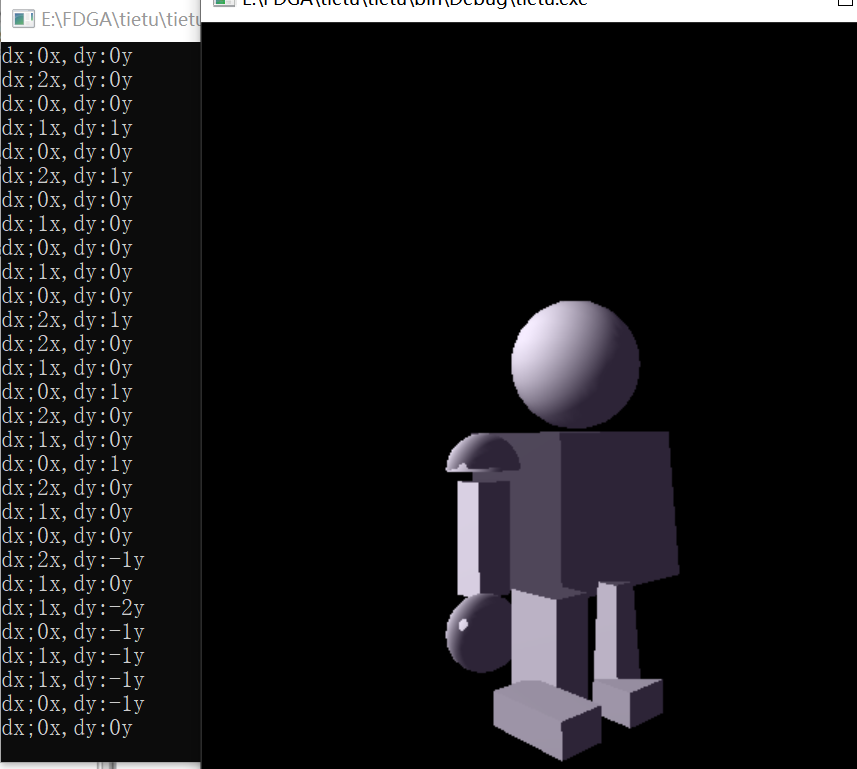
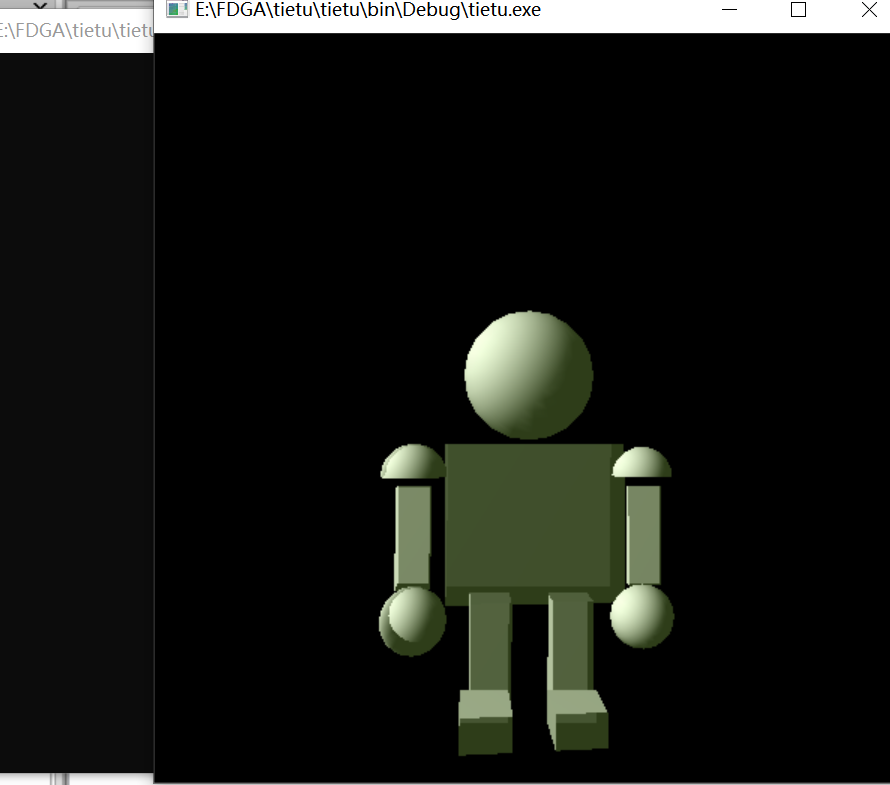
//循环

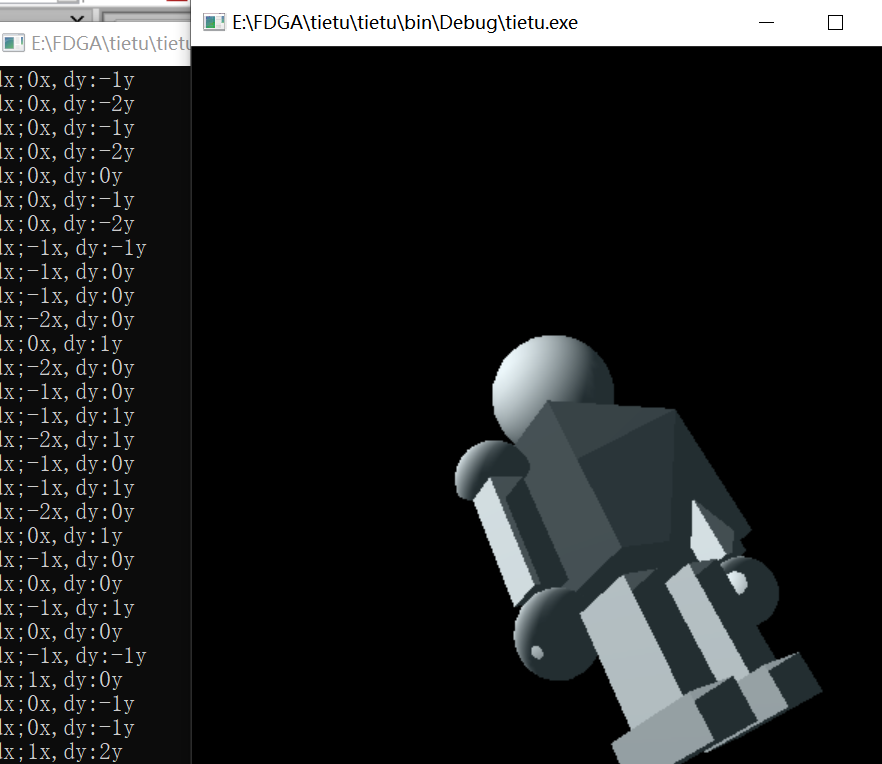
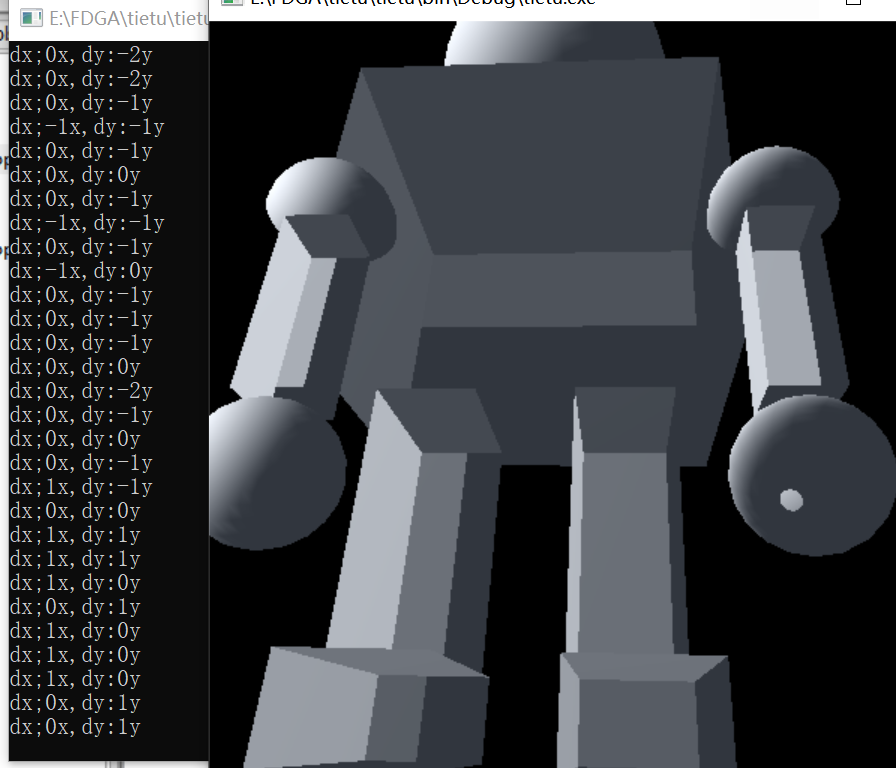
glutMainLoop();

return 0;

}

运行结果：





4.2贴图茶壶代码

#include <windows.h>

#include <stdlib.h>

#include <stdarg.h>

#include <stdio.h>

#include <GL/glut.h>

#include <GL/gl.h>

#include <GL/glaux.h>

#include <mmsystem.h>

#include <math.h>

//读纹理函数：

#define BITMAP\_ID 0x4D42

#define TEX\_NUM 4

GLuint filter; // 滤波类型

GLuint Texture[TEX\_NUM];

char \*TextureName[] = {

"teapot1.BMP",

"teapot2.BMP",

"teapot3.BMP",

"teapot4.BMP",

};

// 读纹理函数

// 纹理标示符数组，保存两个纹理的标示符

// 描述: 通过指针，返回filename 指定的bitmap文件中数据。

// 同时也返回bitmap信息头.（不支持-bit位图）

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) 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/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, //我们希望该纹理有红、绿、蓝数据

bitmapInfoHeader.biWidth, //纹理宽带，必须是n，若有边框+2

bitmapInfoHeader.biHeight, //纹理高度，必须是n，若有边框+2

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

}

//绘制茶壶

GLint GenTeapotList()

{

GLint lid = glGenLists(1);

glNewList(lid, GL\_COMPILE);

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

GLfloat mat\_diffuse[] = { 0.55, 0.55, 0.55, 1.0 };

GLfloat mat\_specular[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat mat\_shininess[] = { 90.0 };

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);

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

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

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

glutSolidTeapot(0.5);

glEndList();

return lid;

}

//定义纹理的函数：

void init(void) //

{

glEnable(GL\_DEPTH\_TEST);//打开深度测试

//定义光源

GLfloat position1[] = { 1.0, 1.0, 1.0, 0.0 };

glLightfv(GL\_LIGHT0, GL\_POSITION, position1);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

//定义纹理

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

glGenTextures(TEX\_NUM, Texture);

for (int i = 0; i < TEX\_NUM; i++) {

texload(i, TextureName[i]);

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

//设置像素存储模式控制所读取的图像数据的行对齐方式.

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

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

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

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

}

glDisable(GL\_TEXTURE\_2D);

}

void display(void)

{glClearColor(0.85f, 0.85f, 0.85f, 1.0f);

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

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, Texture[0]); //选择纹理图案

glCallList(GenTeapotList());

glFlush();

}

void reshape(GLsizei w, GLsizei h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

}

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

{

glutInit(&argc, argv); //初始化工具包

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH); //设置显示模式

glutInitWindowPosition(0, 0); //设置窗口在屏幕上的位置

glutInitWindowSize(500, 500); //设置窗口的大小

glutCreateWindow("wenli"); //打开屏幕窗口

glutReshapeFunc(reshape);

glutDisplayFunc(display); //调用绘图函数

init(); //必要的其他初始化函数

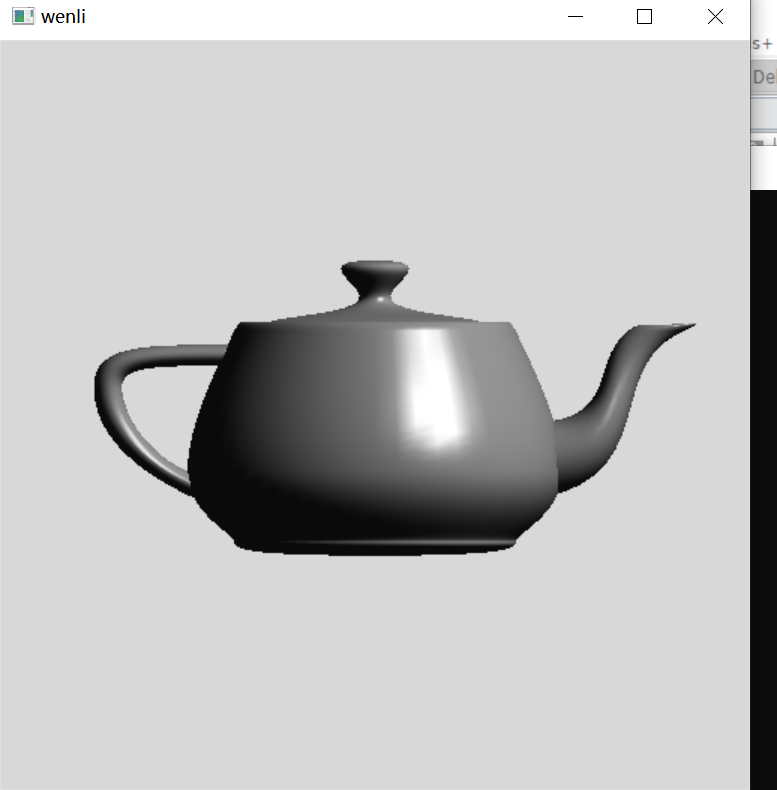
glutMainLoop(); //进入循环

return 0;

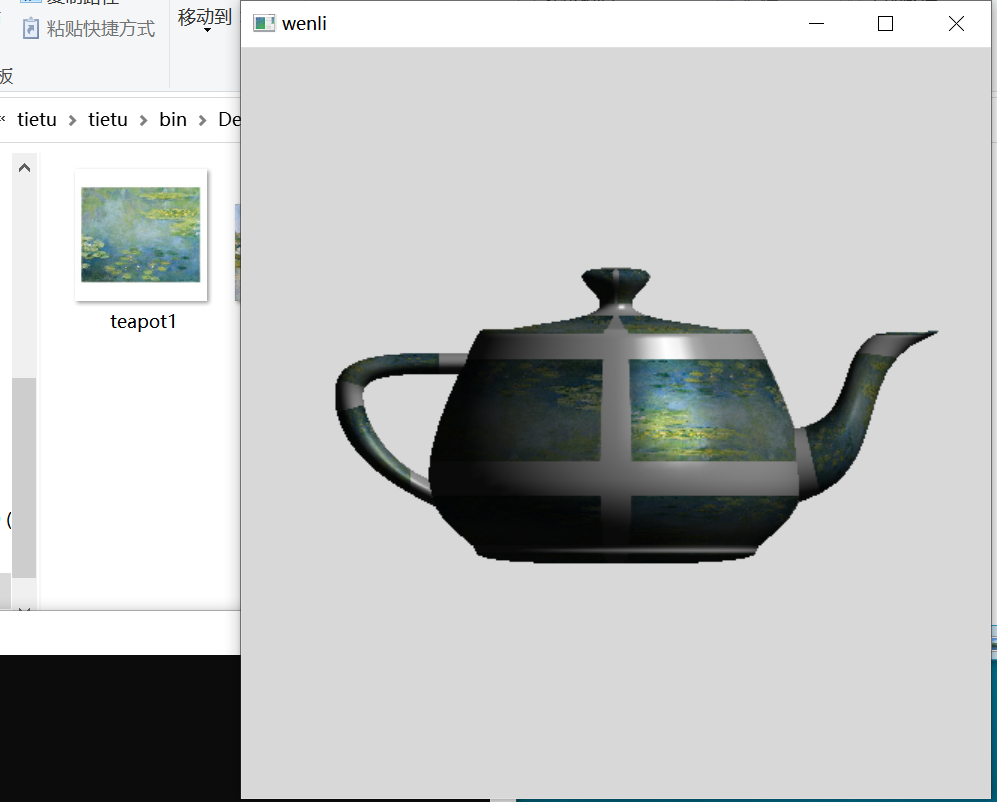
}

运行结果：

1、未贴图前：



1. 贴“teapot1.BMP”图片后



# 5 实验体会与小结

5.1实验存在的问题有：

1. 在实验贴图的过程中，导入BMP图的过程并不顺利，一共两次出现的问题有：

①Failed to open DIB file ；

②unknow DIB file format；

第一个是指找不到DIB图片在哪里无法导入。查阅了很多论坛资料后，发现出现第一个问题的原因是没有把BMP图片和exe文件放在同一个文件下导致计算机找不到（哪怕我在程序里面把图片地址粘贴上去也找不到，不知道是因为不能这样做还是我用的方法不对），在我把图片和exe文件放在一个文件夹下后，在codeblocks运行依旧显示这个错误，只能呈现未贴图的效果。但是在文件夹的exe文件直接点击运行时，却发现可以实现贴图效果。目前依然没有找到良好的解决方法，想要看贴图效果只能直接打开exe文件，并且要求相关的BMP图片与exe放在一起。

第二个是指这个图片的格式是未知的无法打开。有人说是因为采用位图位数不统一,有8位,也有24位,把8位改为24位后可以能贴，但是我排除了这个解决方法，因为我修改几遍仍旧不行。有人说是因为图片编码方式没改，但是我也排除了这个问题——我是用一个网站直接将jpg图片转为bmp图片格式，图片编码方式并没有错。就这样反复修改后，突然这个错误就消失了......然后出现了第一个找不到文件的错误，解决方法最终参考上面。

1. 画的三维图像不够复杂多变，且第一个实验中的机器人是采用坐标系将几个基础三维图形组件组合起来的，并没有多大难度，第二个实验的茶壶是可以直接参考opengl本身函数的，在构建三维图像这方面依然需要努力。
2. 实现的功能不够全面。之所以做了两个实验是因为我无法将两个实验融为一体，即同时在一个复杂的（由多个函数构成的）三维图像中做到多种渲染，函数调用过程与opengl本身函数的多样性让我应付不过来，修改了多次之后依然无法整体实现。目标是能够做出一个真正真实感强烈，能够鼠标键盘交互切换视角、能够旋转供使用者查看整体，自定义光源可以随鼠标移动，能够每一面都贴不同的图与不同纹理的复杂三维图像或者场景。

5.2实验小结

本文研究了利用OpenGL 的库函数进行三维图形绘制的方法,在codeblock中使用OpenGL 库函数对机器人和茶壶两种三维实体模型进行各种基本图形操作,有改变光照条件,设置材质属性、定义视口大小和投影方式,以及进行简单的纹理映射处理,从而实现了三维图形的真实感显示。