**《计算机图形学实验》综合实验报告**

**题目 行走的斑点茶壶**

**学 号 20201060378**

**姓 名 李静**

**指导教师 钱文华**

**日 期 2022年6月18日**

**摘要**

目前，真实感图形的研究是计算机图形学的一个热门话题，在本次实验中，使用opengl和codeblocks软件调用函数生成茶壶，并使用光照，纹理，线型对生成的茶壶进行渲染，然后设置360度旋转，上下左右平移等按键观察渲染后的茶壶，使得茶壶的形态尽量向真实感方向靠近。在最基础的茶壶基础上，添加了蓝绿黑色的纹理，再加上光照的照射，从人的视觉上看，茶壶表面有了纹理，也有了因为光照的照射引起的明暗变化。

**关键词：**茶壶 纹理 光照 旋转

目录

[实验背景及内容 3](#_Toc106478749)

[实验背景 3](#_Toc106478750)

[实验内容 3](#_Toc106478751)

[开发工具 3](#_Toc106478752)

[程序设计 3](#_Toc106478753)

[实现目的 4](#_Toc106478754)

[基本模块介绍 4](#_Toc106478755)

[关键算法的理论介绍 4](#_Toc106478756)

[纹理贴图算法 4](#_Toc106478757)

[光照算法 5](#_Toc106478758)

[程序实现步骤 6](#_Toc106478759)

[运行结果截图 7](#_Toc106478760)

[实验体会及总结 9](#_Toc106478761)

[参考文献 10](#_Toc106478762)

[附录 10](#_Toc106478763)

# 实验背景及内容

## 实验背景

为了让同学们了解三维渲染的效果，掌握光照跟踪原理算法，感受光照照射在三维物体上产生的明暗效果，掌握纹理贴图的原理算法，能够向实现真实感图形的方向靠拢，让同学们寓学于用，特布置了本实验让同学们自行尝试三维图形的渲染实现。

## 实验内容

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。

# 开发工具

笔记本电脑

OpenGL

CodeBlocks

# 程序设计

本实验首先生成一个茶壶，然后使用纹理贴图算法，将相应的图片贴到茶壶上。或者自己生成一个蓝绿黑的斑点图，贴在茶壶上。得到添加纹理后的茶壶。然后使用开启光照，为光照设定一个角度和光的颜色，光源。使得光照射在茶壶上，因为角度问题在不同部位产生明暗效果。最后将键盘的交互加入程序中，按相应的按键实现茶壶由最初的加光照的样子到加纹理，线型，旋转，上下左右平移等操作。

# 实现目的

使用Opengl和codeBlocks编程实现的茶壶，通过添加纹理，光照等等进行渲染，最终可以获得一个不是颜色单一的茶壶，而是贴上了纹理且在光照的照射下，茶壶的不同位置呈现出不同的明暗效果。向符合现实生活中人看物体的效果靠拢。

# 基本模块介绍

本实验代码分为三个模块。

第一个模块是纹理贴图，从本地读取bmp文件并将它的BGR格式转换为RGB格式。然后设定读入纹理的色彩，带宽，高度，边框等特点，然后选择纹理，设置纹理受光照影响和不受光照影响两种形式，根据不同的选择将纹理贴到茶壶表面，展现有无光照的不同显示效果。或者是不从本地获取纹理图片，而是自定义生成纹理。

第二个模块是光照的设置和茶壶绘制模式的设置，开启光照模式，设置光源位置和光的颜色，使用0号光源。然后设置茶壶绘制的模型为线型或者完全填充。

第三个模块实现和键盘的交互，设置键盘按键实现不同的功能，按空格键可使茶壶旋转，a右移,d右移，s下移,w上移，q退出，r加或不加纹理，o线型或非线型显示，p倾斜或不倾斜。这些功能还可以叠加实现。

# 关键算法的理论介绍

## 纹理贴图算法

纹理可以是一张图片，一组图片数据或者一段函数公式。纹理也可以用来存储大量的数据，这些数据可以发送到着色器上。它可以让模型的外表有更丰富的颜色图案。纹理映射的主要思想是将给定的纹理函数映射到物体表面上，再对物体表面进行光亮度计算。贴图时，要确定把图片的哪部分贴到哪个像素点就需要指定纹理坐标。当为顶点提供一系列的纹理坐标，在光栅化阶段会对纹理坐标进行插值计算，计算出每个对应像素点需要的纹理。纹理中包含一个用于跟屏幕上像素点对应的颜色值，许许多多的纹理里的颜色值显示在屏幕上的不同像素上就显示出来我们要的纹理。

使用纹理映射的3个步骤：

1. 创建一个纹理对象并加载纹理数据

（1）在OpenGl中使用纹理的第一步时创建纹理对象，然后将对象绑定到环境的纹理单元即：

glGenTextures(1,&m\_textureObj);//创建纹理对象并将函数返回值赋值到m\_textureObj

glBindTexture(m\_textureTarget,m\_textureObj);//绑定当前纹理对象

（2）加载贴图到纹理数据中，先把硬盘中的贴图读取到内存内，再从内存中加载纹理数据

2.为顶点数据添加纹理坐标

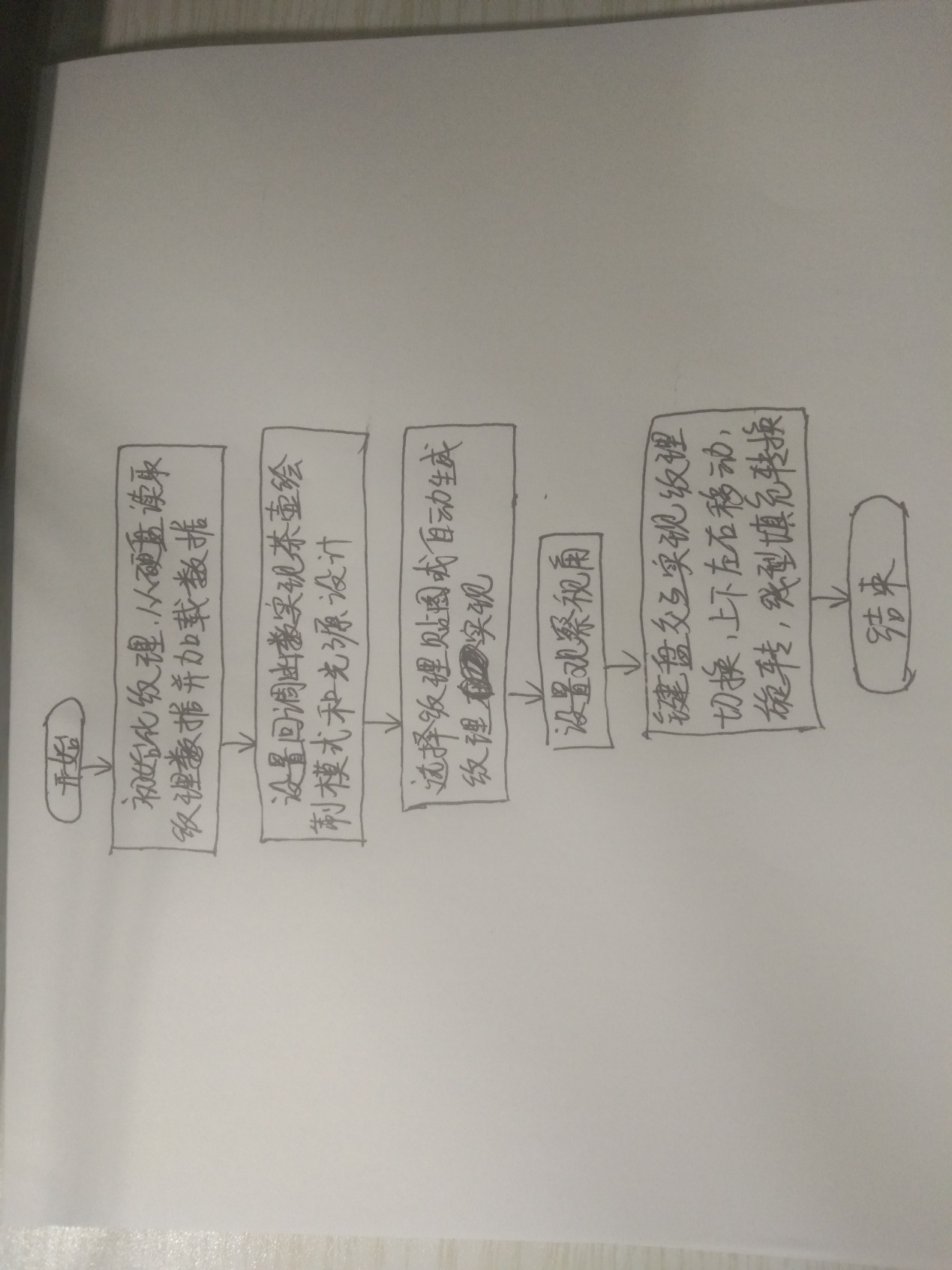
3.在着色器中使用纹理坐标从纹理中取样得到像素颜色

## 光照算法

光源是所有光照效果的基础，没有光源一切都无从谈起。OpenGL中提供了8个光源分别为GL\_LIGHT0、GL\_LIGHT1、……、GL\_LIGHT7,这几个光源的的位置、方向及发出的光线可以完全不同。因此使用OpenGL中的光源时只需要设置颜色，位置和方向等性质。在OpenGL中使用函数glLightfv（光源编号，光源特性，参数数据）来设置光源。光源特性主要可取GL\_AMBIENT（设置光源的环境光属性，默认值(0,0,0,1)）、GL\_DIFFUSE（设置光源的散射光属性，默认值(1,1,1,1)）、GL\_SPECULAR（设置光源的镜面反射光属性，默认值(1,1,1,1)）、GL\_POSITION（设置光源的位置，默认值(0,0,1,0)）。参数数据格式要求为数组形式，即数学上的向量形式。光源的性质设置完毕，还要调用glEnable(GL\_LIGHT0)来打开光源。

# 程序实现步骤

1. 调用函数画茶壶
2. 创建纹理对象并从硬盘中加载纹理数据
3. 选择纹理并贴图
4. 设置观察视角
5. 设置光源
6. 实现和键盘的交互



# 运行结果截图

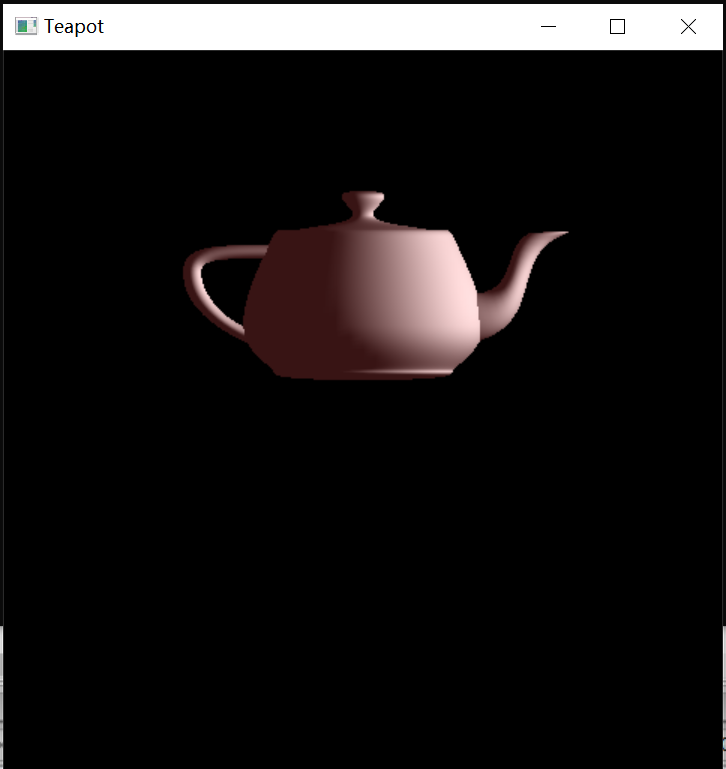


图1 初始的添加色彩和光照的茶壶

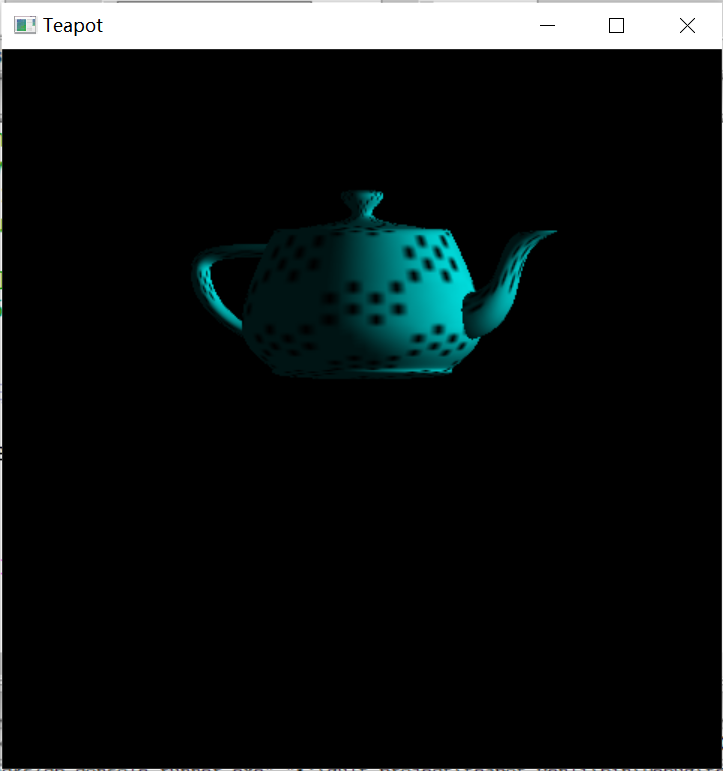


图2 生成斑点纹理加光照的茶壶

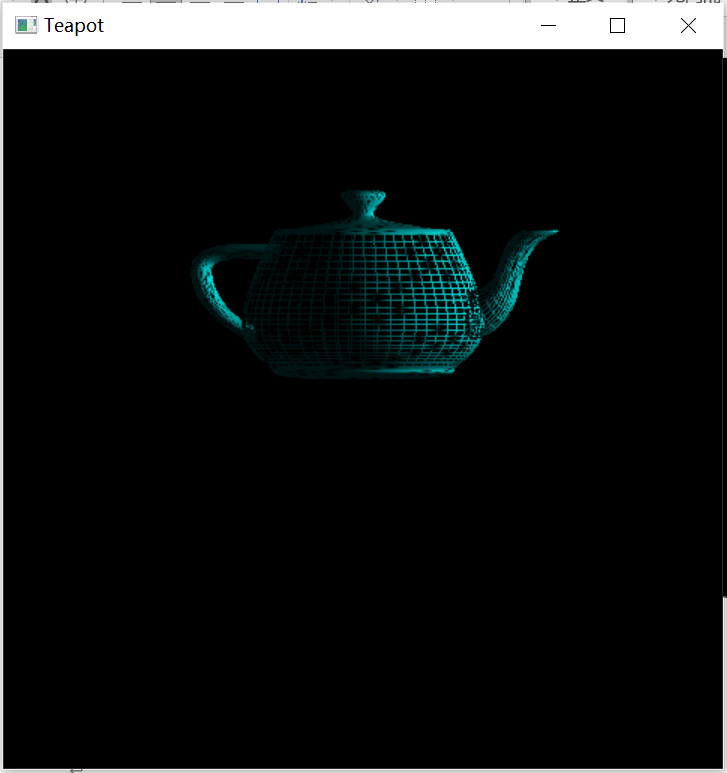


图3 加纹理和光照的线型绘制的茶壶

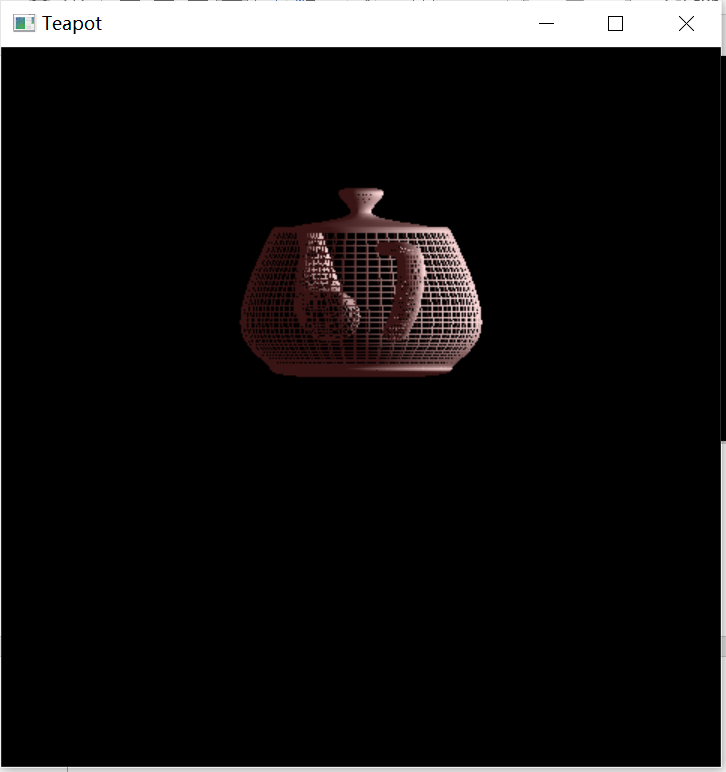


图4 旋转之后的线型茶壶

对实验结果的分析：本实验中的光照是从{5，2，2，1}这个角度照过来的，所以对于整个茶壶而言，各个部分就有比较明显的明暗变化，茶壶右边的位置就明显要比左边亮一些，光照引起的茶壶的明暗效果在以上四个图中都有体现。然后贴上纹理后，整个茶壶收光照的影响不变，只是表面的装饰发生了变化。然后图3和图4是使用线型绘制的茶壶，所以，看起来就像是透明的一样，能够看见茶壶的内部和被遮挡的部分，然后本实验还有旋转功能，按空格键茶壶就会360度旋转，看个人喜好哪个角度再按一次空格茶壶就停止旋转。图4就是茶壶旋转到任意角度停止的截图。

# 实验体会及总结

1.在本次实验过程中，使用了纹理贴图算法，纹理的自定义生成，OpenGL中的光照模型，三维旋转，三维平移，图形的绘制模式等计算机图形学的知识。

2.实验过程中也遇到了一些问题，纹理贴图时，图片的文件打不开，更改文件路径为绝对路径之后，整个程序运行后直接闪退，连茶壶都没了。研究了半天没有找出来错误，目前，这个问题还是没能解决。为了实现茶壶的纹理变化，于是改用另一种方法，直接自定义生成纹理图片，不需要从硬盘中读取文件，然后这样可以成功实现茶壶纹理的转换。

3.从运行的初始的茶壶图片来看，边缘还存在一些瑕疵比如在茶壶的盖子，壶把等地方有轻微的锯齿状，好像茶壶有一丢丢走样，所以本次实验还有很大的改进空间，不管从纹理贴图还是茶壶的走样上看。

4.感觉这次实验比较综合，能够把学过的很多东西都用到茶壶上去，整个实验做下来我觉得还是有点复杂，特别是那个纹理贴图的部分。

# 参考文献

[1]张璐. 真实感虚拟场景中光线跟踪算法的应用与改进[D].电子科技大学,2008.

[2]冯渊. OpenGL的光照处理及外部模型数据转化[D].电子科技大学,2004.

[3]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

源代码及注释：

#include <stdlib.h>

#include <windows.h>

#include <stdio.h>

#include <GL/glut.h>

#define BITMAP\_ID 0x4D42

#define Height 16

#define Width 16

//本程序包含了光照和纹理,上下左右平移、旋转以及线型图的茶壶

GLubyte image[Height][Width][3]; // 图像数据

float fTranslate;

float fRotate;

float fScale = 1.0f; // set inital scale value to 1.0f

int status = 0;

int status2 = 1;

bool bPersp = false;

bool bAnim = false;

bool bWire = false;

int wHeight = 0;

int wWidth = 0;

GLuint texture[3];

// 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）

//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/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

}

void generateTex()

{

//生成蓝绿黑的纹理图像

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

for (int j = 0; j < Width; j++) {

int x = ((i | 5 ) ^(j | 4 )) \* 255;

image[i][j][0] = 0;

image[i][j][1] = (GLubyte)x;

image[i][j][2] = (GLubyte)x;

}

}

}

//定义纹理的函数

void init()

{

glGenTextures(3, 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, "Monet.bmp");

texload(1, "Crack.bmp");

//下面生成自定义纹理

generateTex();

glBindTexture(GL\_TEXTURE\_2D, texture[2]);

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1); //设置像素存储模式控制所读取的图像数据的行对齐方式.

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, Width, Height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, image);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);//放大过滤，线性过滤

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);//缩小过滤，线性过滤

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);//S方向重复

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);//T方向重复

}

void Draw\_Triangle() // This function draws a triangle with RGB colors

{

glEnable(GL\_TEXTURE\_2D);

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

glPushMatrix();

glTranslatef(0, 0, 4+1);

glRotatef(90, 1, 0, 0);

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

glutSolidTeapot(5);

glPopMatrix();

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

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texture[status2]); //选择纹理texture[status2]

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);//设置纹理不受光照影响

glDisable(GL\_TEXTURE\_2D); //关闭纹理texture[status2]

}

void updateView(int width, int height)

{

glViewport(0, 0, width, height);//设置视窗大小

glMatrixMode(GL\_PROJECTION);//设置矩阵模式为投影

glLoadIdentity(); //初始化矩阵为单位矩阵

float whRatio = (GLfloat)width / (GLfloat)height; //设置显示比例

if (bPersp) {

gluPerspective(45.0f, whRatio, 0.1f, 100.0f); //透视投影

}

else {

glOrtho(-3, 3, -3, 3, -100, 100); //正投影

}

glMatrixMode(GL\_MODELVIEW); //设置矩阵模式为模型

}

void reshape(int width, int height)

{

if (height == 0) //如果高度为0

{

height = 1; //让高度为1（避免出现分母为0的现象）

}

wHeight = height;

wWidth = width;

updateView(wHeight, wWidth); //更新视角

}

void idle()

{

glutPostRedisplay();

}

float eye[] = {0, 0, 8};

float center[] = {0, 0, 0};

void key(unsigned char k, int x, int y)//按任意键可使茶壶旋转，a右移,d右移，s下移,w上移，q退出，r加或不加纹理，o线性或非线性显示，p倾斜或不倾斜，空格键旋转

{

switch (k)

{

case 27:

case 'q': {exit(0); break; }

case 'p': {bPersp = !bPersp; break; }

case ' ': {bAnim = !bAnim; break; }

case 'o': {bWire = !bWire; break; }

case 'a': {

eye[0] -= 0.2f;

center[0] -= 0.2f;

break;

}

case 'd': {

eye[0] += 0.2f;

center[0] += 0.2f;

break;

}

case 'w': {

eye[1] -= 0.2f;

center[1] -= 0.2f;

break;

}

case 's': {

eye[1] += 0.2f;

center[1] += 0.2f;

break;

}

case 'r': { //切换茶壶纹理

if (status == 0)status = 2;

else if (status == 2)status = 0;

break;

}

}

updateView(wHeight, wWidth);//更新视角

}

void redraw()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);//清除颜色缓存和深度缓存

glLoadIdentity(); //初始化矩阵为单位矩阵

gluLookAt(eye[0], eye[1], eye[2],

center[0], center[1], center[2],

0, 1, 0); // 场景（0，0，0）的视点中心 (0,5,50)，Y轴向上

if (bWire) {

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

//设置多边形绘制模式：正反面，线型

}

else {

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

//设置多边形绘制模式：正反面，填充

}

glEnable(GL\_DEPTH\_TEST);//开启深度测试

glEnable(GL\_LIGHTING); //开启光照模式

GLfloat white[] = { 0.9, 0.2, 0.2, 0.0 };

GLfloat light\_pos[] = { 5,2,2,1 };

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_pos);//光源位置

glLightfv(GL\_LIGHT0, GL\_AMBIENT,white);//定义白色

glEnable(GL\_LIGHT0);//开启第0号光源

glRotatef(fRotate, 0, 1.0f, 0); //旋转

glRotatef(-90, 1, 0, 0);

glScalef(0.2, 0.2, 0.2);//缩放

Draw\_Triangle();//绘制场景

if (bAnim) fRotate += 0.5f;//旋转因子改变

glutSwapBuffers(); //交换缓冲区

}

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

{

glutInit(&argc, argv);//对glut的初始化

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

//初始化显示模式:RGB颜色模型，深度测试，双缓冲

glutInitWindowSize(480, 480);//设置窗口大小

int windowHandle = glutCreateWindow("Teapot");//设置窗口标题

glutDisplayFunc(redraw); //注册绘制回调函数

glutReshapeFunc(reshape); //注册重绘回调函数

glutKeyboardFunc(key); //注册按键回调函数

glutIdleFunc(idle);//注册全局回调函数：空闲时调用

init(); //初始化纹理

glutMainLoop(); // glut事件处理循环

return 0;

}