《计算机图形学实验》综合实验报告

题目 三维图形渲染

学 号 20191060064

姓 名 白文强

指导教师 钱文华

日 期 2020 年 12 月 25 日

摘要

本实验要实验对三维图形的渲染，在实验中，我对茶壶进行了纹理、色彩、光照、透明等效果的渲染。其中，在渲染纹理时，采用纹理贴图方法，读取*.bmp文件对茶壶进行映射；为了更明显地体现对茶壶色彩的渲染，设置了按键，可以在纹理渲染和色彩渲染之间进行切换；为了体现透明效果，在茶壶前绘制了一块薄板做遮挡，通过按键调整其透明度，由此可以调整茶壶的可见性。

为了更好的体现三维图形的渲染效果，程序设计了利用键盘和鼠标的交互系统，可以控制茶壶围绕三个轴进行旋转、控制薄板的透明度，可以利用鼠标滚轮对茶壶进行缩放，全面体现程序的功能与渲染效果。

关键字：三维渲染；光照；纹理；透明

目录

一、实验背景，实验内容

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。

二、开发工具，程序设计及实现目的及基本模块介绍

开发工具：OpenGL

实现目的：绘制三维茶壶，并对其渲染以达到看起来尽可能真实的目的，同时加入三维变化与交互功能，增强物体显示的多样性

程序基本模块：

readImg()函数，读取纹理图片；

LoadTexure()函数，加载并绑定纹理

init()函数，设置填充颜色、开启深度测试、设定反射系数、创建并绑定纹理等

drawBoard()函数，绘制一个薄板

display()函数，清空并设置颜色、设定光源位置和光源类型、绘制茶壶和薄板等

reshape()函数，当窗口尺寸改变时，图形比例不发生变化

keyboard()函数，键盘交互函数：

当按下键盘“Q”或“q”时，茶壶绕 x 轴逆时针旋转

当按下键盘“W”或“w”时，茶壶绕 x 轴顺时针旋转

当按下键盘“A”或“a”时，茶壶绕 z 轴逆时针旋转

当按下键盘“S”或“s”时，茶壶绕 z 轴顺时针旋转

当按下键盘“Z”或“z”时，茶壶绕 z 轴顺时针旋转

当按下键盘“X”或“x”时，茶壶绕 z 轴逆时针旋转

当按下键盘“+”时，薄板不透明度增加

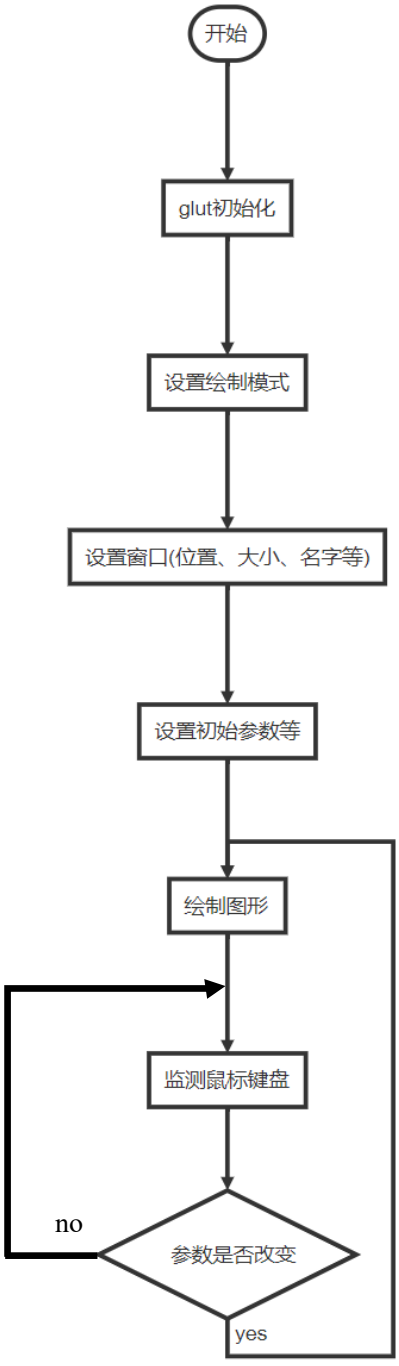
当按下键盘“-”时，薄板不透明度降低

myMouse()函数，鼠标交互函数：

当鼠标左键单击时，茶壶实现由纹理填充与颜色填充之间的转换

鼠标滚轮可以控制茶壶的大小，实现比例缩放

三、关键算法的理论介绍和程序实现步骤（可画流程图）



四、程序源代码及注释

```
#include <windows.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

float s=1;
bool is_wl=true;
static GLuint texture;
float theta1,theta2,theta3; //转动角度
float alpha=0.8;//透明值
static GLfloat *currentCoeff;
static GLenum currentPlane;
static GLint currentGenMode;
static GLfloat xequalzero[] = {1.0, 0.0, 0.0, 0.0};

GLubyte*  readImg(char* filename, int * imagewidth, int * imageheight)
{
    //打开文件
    int pixellength;
    GLubyte *  pixeldata ;
    FILE* pfile=fopen(filename,"rb");
    if(pfile == 0) exit(0);

    //读取图像大小
    fseek(pfile,0x0012,SEEK_SET);
    fread(imagewidth,sizeof(*imagewidth),1,pfile);
    fread(imageheight,sizeof(*imageheight),1,pfile);

    //计算像素数据长度
    pixellength=(*imagewidth)*3;
    while(pixellength%4 != 0)pixellength++;
    pixellength *= (*imageheight);

    //读取像素数据
    pixeldata = (GLubyte*)malloc(pixellength);
    if(pixeldata == 0) exit(0);

    fseek(pfile,54,SEEK_SET);
    fread(pixeldata,pixellength,1,pfile);

    fclose(pfile);
    return pixeldata;
```

```
}

void LoadTexture(char * filename, GLuint &texture)
{
    GLubyte * data;
    GLint width, height;

    //读文件
    data=readImg(filename, &width, &height );
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);//绑定纹理
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

    //线性滤图
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    //自动生成纹理坐标，使用球面映射
    currentCoeff = xequalzero;
    currentGenMode = GL_OBJECT_LINEAR;
    currentPlane = GL_OBJECT_PLANE;

    //生参数 1:GL_UNPACK_ALIGNMENT ，指定 OpenGL 如何从数据缓存区中解包图像数据
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, width, height, 0, GL_BGR_EXT, GL_UNSIG
NED_BYTE, data);

    glEnable(GL_TEXTURE_2D);//打开纹理映射
    free(data); //释放纹理图像数据，纹理数据已由上一句生成并保存到纹理缓存中，使用
完毕后应用 glDeleteTextures 释放纹理缓存
}
void init(void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);

    glEnable(GL_DEPTH_TEST);        //开启深度测试
    glDepthFunc(GL_LESS);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);


    GLfloat mat_diffuse[] = { 1, 1, 1, 1.0 };  //漫反射系数，对应 RGBA 四个分量，
A 分量在混合开启后有效

```

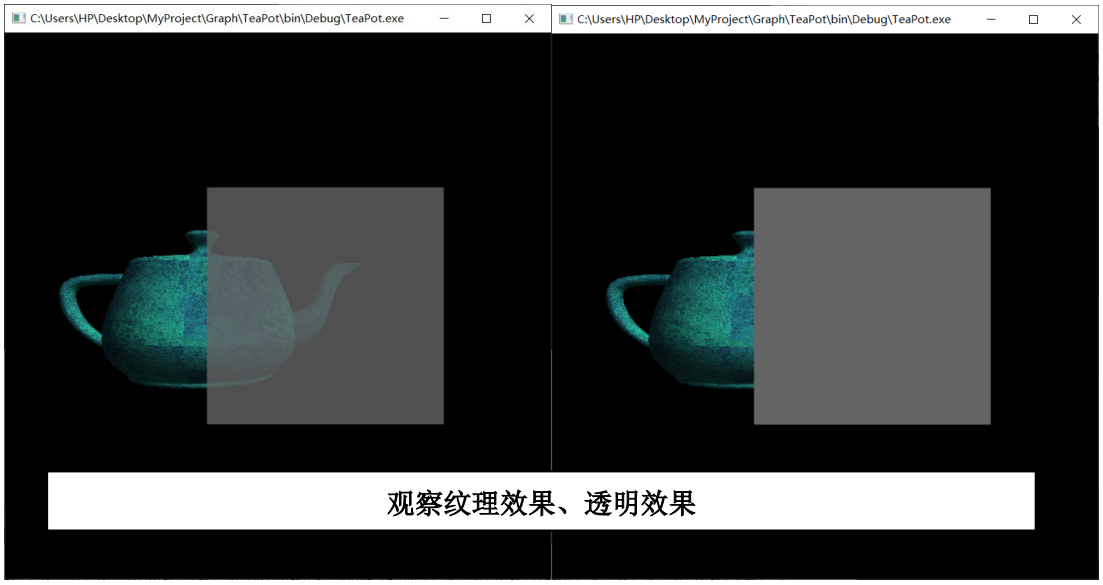
```
83.     GLfloat mat_specular[] = { 0.8, 0.7, 0.8, 1.0 }; //镜面反射系数,对应 RGBA 四个分量，A 分量在混合开启后有效
84.     GLfloat mat_shininess[] = { 50.0 };
85.
86.
87.     glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_diffuse);//设置环境光系数和漫反射光系数
88.     glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);  //set material 镜面光反射
89.     glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);//设置材料反射指数
90.
91.     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA); //指定混合函数
92.
93.     glEnable(GL_BLEND);         //启用混合状态
94.     glEnable(GL_COLOR_MATERIAL);//材质跟踪当前绘图色
95.     glEnable(GL_LIGHTING);
96.     glEnable(GL_LIGHT0);
97.
98.     //创建并绑定纹理
99.     LoadTexture("C:\\Users\\HP\\Desktop\\MyProject\\Graph\\TeaPot\\铜锈.bmp",texture);
100. }
101. void drawBoard()
102. {
103.     glColor4f(0.5,0.5,0.5,alpha);
104.
105.     glBegin(GL_QUADS);
106.     glVertex3f(-0.5,-0.5,1);
107.     glVertex3f(-0.5,0.5,1);
108.     glVertex3f(0.5,0.5,1);
109.     glVertex3f(0.5,-0.5,1);
110.     glEnd();
111. }
112. void display(void)
113. {
114.     glClear (GL_COLOR_BUFFER_BIT |GL_DEPTH_BUFFER_BIT );
115.
116.     GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };//最后一个参数为 0，表示光源在无穷远处
117.
118.     glLightfv(GL_LIGHT0, GL_POSITION, light_position);//光源位置
119.
120.     glEnable(GL_LIGHTING);
121.     glMatrixMode(GL_MODELVIEW);  //使用模型视图矩阵栈保存后面的几何变换
```

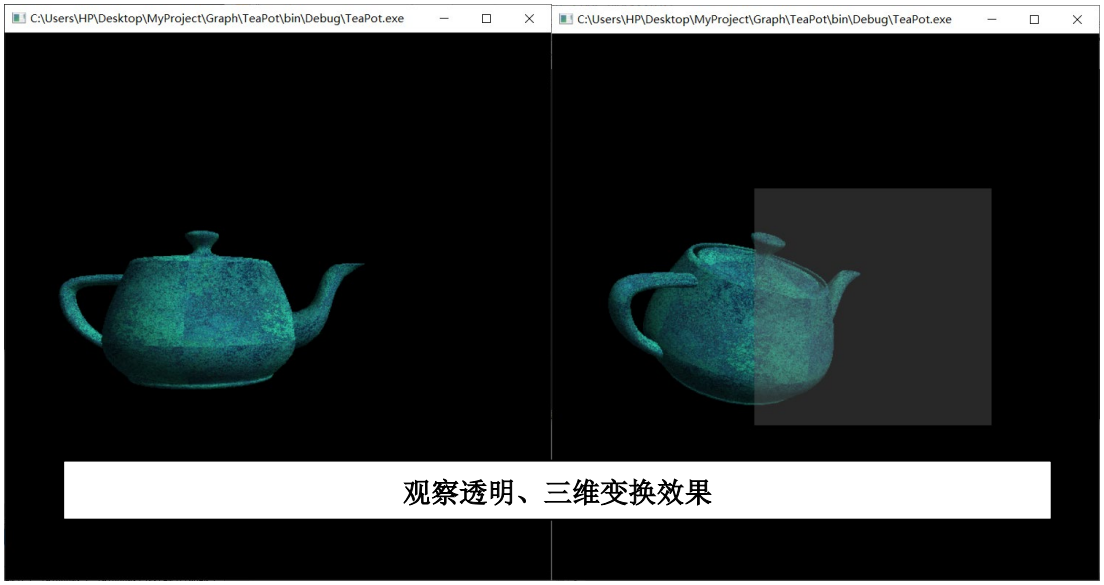
```
122.     glPushMatrix();                //保存世界坐标系到观察坐标系的变换矩阵
123.
124.     glTranslatef(-0.8,0,0);
125.     glPushMatrix();
126.
127.     glColor4f (1,1,1,1); //启用光源后，不在使用颜色函数对物体着色
128.     glBindTexture(GL_TEXTURE_2D, texture);
129.     glRotatef(theta1,1.0f,0.0f,0.0f);
130.     glRotatef(theta2,0.0f,1.0f,0.0f);
131.     glRotatef(theta3,0.0f,0.0f,1.0f);
132.     if(is_wl)
133.         glEnable(GL_TEXTURE_2D);//打开纹理
134.     else
135.     {
136.         glDisable(GL_TEXTURE_2D);//关闭纹理
137.         glColor3f(1,0,0);//设置红色
138.     }
139.     glutSolidTeapot(1*s);
140.     glPopMatrix();
141.
142.
143.     glPushMatrix();
144.     glDisable(GL_DEPTH_TEST); // 关闭深度测试
145.     glDisable(GL_TEXTURE_2D);//关闭纹理
146.     glTranslatef(1,0,2);
147.
148.     drawBoard();//绘制透明薄板
149.
150.     glEnable(GL_TEXTURE_2D);//打开纹理
151.     glEnable(GL_DEPTH_TEST);//打开深度测试
152.     glPopMatrix();
153.
154.     glPopMatrix();                 //恢复世界坐标系到观察坐标系的变换矩阵
155.     glutSwapBuffers();
156. }
157.
158. void reshape (int w, int h)
159. {
160.     glViewport (0, 0, (GLsizei) w, (GLsizei) h);  //使视区大小保持与窗口的显
    示区域大小一致
161.     glMatrixMode(GL_MODELVIEW);                    //使用模型视图矩阵栈
162.     glLoadIdentity();
163.
164.
```

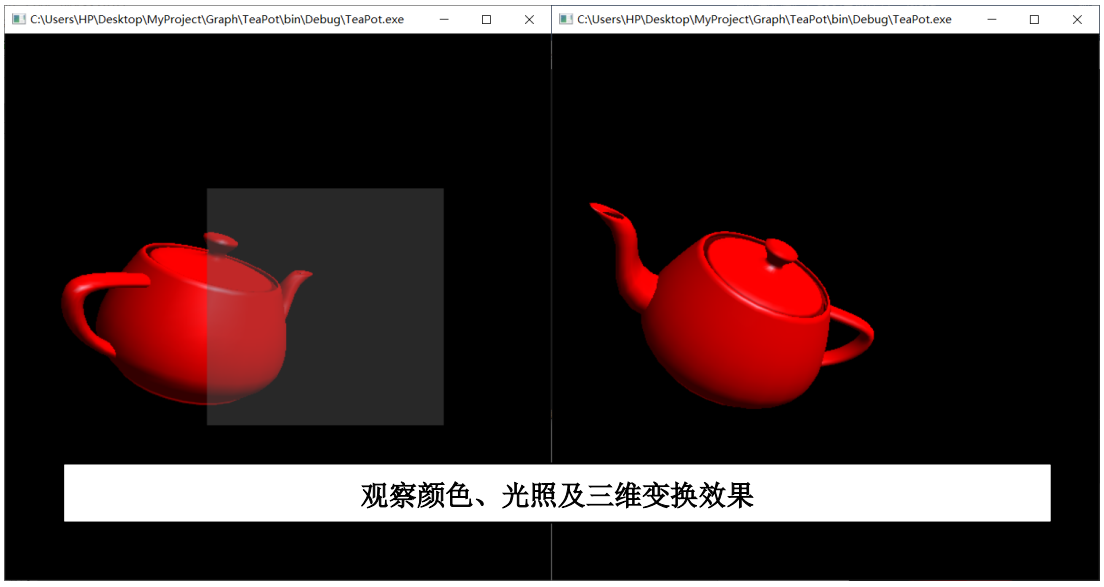
```
165.     gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
166.
167.     glMatrixMode (GL_PROJECTION);                    //使用投影矩阵栈，准备设定投影矩阵
168.     glLoadIdentity ();                               //初始化投影矩阵为单位矩阵
169.
170.     gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 0.5, 30.0);
171. }
172.
173. void keyboard (unsigned char key, int x, int y)
174. {
175.     switch (key)
176.     {
177.     case '+':
178.         if(alpha<1)
179.             alpha+=0.05;
180.         break;
181.     case '-':
182.         if(alpha>0)
183.             alpha-=0.05;
184.         break;
185.     case 'q':
186.     case 'Q':
187.         theta1+=5.0;
188.         break;
189.     case 'w':
190.     case 'W':
191.         theta1-=5.0;
192.         break;
193.     case 'a':
194.     case 'A':
195.         theta2+=5.0;
196.         break;
197.     case 's':
198.     case 'S':
199.         theta2-=5.0;
200.         break;
201.     case 'z':
202.     case 'Z':
203.         theta3+=5.0;
204.         break;
205.     case 'x':
206.     case 'X':
```

```
207.         theta3-=5.0;
208.         break;
209.     default:
210.         break;
211.     }
212.     glutPostRedisplay();
213. }
214. void myMouse(int button, int state, int x, int y) //滑轮控制缩放
215. {
216.     if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
217.     {
218.         is_wl = !is_wl;//左键显示\取消纹理
219.     }
220.     if(button== 3 ) //滑轮上滑放大
221.     {
222.         s+= 0.005;
223.     }
224.     if(button== 4) //滑轮下滑缩小
225.     {
226.         s-=0.005;
227.     }
228.     glutPostRedisplay(); //重新调用绘制函数
229. }
230. int main(int argc, char** argv)
231. {
232.     glutInit(&argc, argv);
233.     glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
234.     glutInitWindowSize (600, 600);
235.     glutInitWindowPosition (100, 100);
236.     glutCreateWindow (argv[0]);
237.     init ();
238.     glutDisplayFunc(display);
239.     glutReshapeFunc(reshape);
240.     glutKeyboardFunc(keyboard);
241.     glutMouseFunc(myMouse);
242.     glutMainLoop();
243.     return 0;
244. }
```

五、实验运行结果，结果分析及不足







六、实验体会及小结

通过本次实验，对 openGL 有了更深的理解，本次实验中，涉及到许多从未使用过的概念和函数，通过查询 openGL 帮助文档，学到了纹理贴图的方法即加载位图图像将图像中的纹理绑定到要绘制的图形上去，学到了设置透明度、设置光照及开启与关闭，学到了设置材料的反射系数，以及设置交互功能时调用 glutPostRedisplay()函数重新调用绘制函数以把利用键盘或鼠标修改后的参数重新代入绘制函数再把新图形绘制出来的方法

在完成实验的过程中，遇到的问题及解决方法主要有：

（1）本次实验用到了大量 openGL 库函数，对库函数的不熟悉是遇到的第一个问题，解决方法为查看 openGL 帮助文档，通过查看里面的例程，熟悉了很多库函数及实现功能如设置纹理贴图的方法。

（2）在绘制时，出现了下面的情况

即理应被遮挡的地方没有被遮挡，可以透过前面看到后面，这种情况出现的原因是没有打开深度测试，后查询帮助文档时发现要打开深度测试即 glEnable(GL_DEPTH_TEST)即可。