《计算机图形学实验》综合实验报告

题目　基于 OpenGL 的茶壶 3D 渲染效果

学　号　20201060335

姓　名　袁新宇

学　院　信息学院

专　业　计算机科学与技术

指导教师　钱文华

日　期　2022 年 6 月 17 日

目录

摘要：

在本次实验中，通过使用 OpenGL 工具，调用其中的光照名模型、纹理映射等算法，为该实验的渲染对象——茶壶的表面指定相应的纹理图像以及光照效果，从而实现真实感的效果。本次实验中，茶壶主体为红色和蓝色交替，并伴有着淡淡的绿色的条纹。

关键词：　OPenGL；三维图形渲染；光照模型；纹理映射

1. 实验背景与内容

1.1 实验背景

OpenGL(Open GraphicLibrary，开放式图形库)是一个三维的计算机和模型库，在三维真实感图形制作中具有优秀的性能，可以独立于窗口系统、操作系统和硬件系统环境的图形开发环境，是一个与硬件无关的 API，可以在不同的硬件平台上得到实现。

在本次实验中，所要求的是实现三维图形的渲染，也就是说使用 OpenGL 的所有要素都是三维的，但是电脑屏幕却是二维的，因此在渲染过程中，需要将 3D 坐标转换为适应屏幕的 2D 坐标。纹理模式可以由一个矩形数组进行定义，也可以由一个修改对象表面光强度值的过程来定义，这种方法称为纹理映射。而纹理可定义为一维、二维或三维图案。而为在场景中添加光照，需要设定一个或多个光源，并设置它们的属性，同时还要对物体的材质属性进行设定，最后选择一种光照模型。

1.2 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。

2. 开发工具与程序设计

2.1 开发环境

硬件：笔记本电脑

操作系统：Windows 10

所用软件：Visual Studio 2022

图形交互设备：键盘

2.2 实验目的

通过自定义茶壶，并且使用 OpenGL 程序的三维图形渲染效果，包括采用光线跟踪、光照明模型、纹理贴图、纹理映射等算法，能够实现为茶壶加上纹理、光照、色彩、阴影、透明等效果，从而绘制出更具有真实感的图形。同时通过本次实验，对相应的三维图形渲染过程的理论知识得到更深的了解，巩固自己的编程能力。

2.3 程序设计

本次实验，我将程序设计分成了多个模块。通过多模块的设计，各自实现相应的功能。同样，各模块之间也会有对应的联系，而不仅仅只是分开来工作，以此来提高程序的运行效率，达到预期的实验效果。

2.4 基本模块介绍

基本模块（函数）	功能
MyKey 函数	键盘函数，实现按下某些键盘对茶壶进行操作的功能
MySpecialKey 函数	特殊键盘函数，实现某些特殊键盘的操作
SetImage 函数	设置纹理，生成平面纹理坐标
Init 函数	启用纹理和光照模型，设置对应的参数
reshape 函数	当窗的形状发生改变时，修正窗内的图形显示
display 函数	绘图函数，将茶壶画出来
main 函数	程序的入口函数，初始化环境变量，设置窗口、标题，注册回调函数

3. 关键算法的理论介绍及程序实现步骤

3.1 关键算法的理论介绍

（1）启用光照模型

首先要设置光源和材料的属性，例如本次试验中语句 GLfloat light_ambient[] = { 0.6, 0.5, 0.4, 1.0 }设置了环境光的参数。

在设置完相应的属性参数之后，还需要调用 glEnable(GL_LIGHTING) 和 glEnable(GL_LIGHT0)用来将 OpenGL 光源和设置好的光源（LIGHT0）激活，接着对象表面就用包括已激活的每一光源贡献的光照计算来绘制（当设置了多个光源的时候）。

最后，为了将光源以及材料的各种特性如位置、类型、颜色和投射效果等与每一个光源结合在一起，因此还要调用设定特性的函数 glMaterialfv(materialName,lightProperty,propertyValue)\gllightfv(lightName,lightProperty,propertyValue)，前者是关于材质，后者是关于光源。其中的参数第一个是名称，第二个是属性，第三个是对应属性的参数。通过以上步骤，完成了光照模型的启动。

（2）启用纹理映射

在程序的开始，构建纹理构建函数 SetImage，用来形成对应的二维表面纹理映射，其中在程序中 stripeImageWidth 和 stripeImageHeight 分别是纹理的长和高。

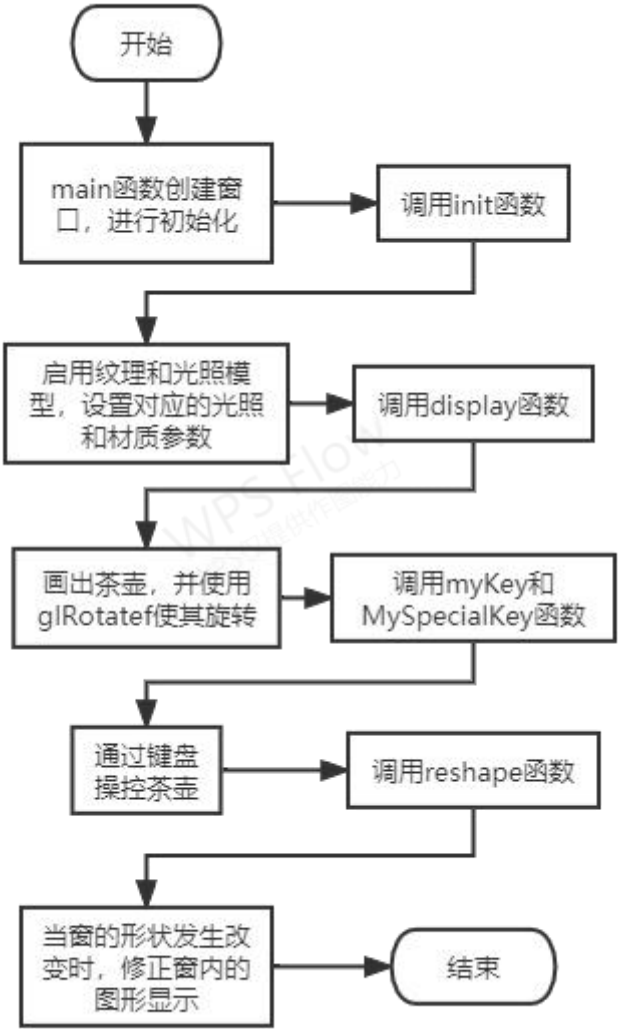
设置完纹理样式后，需要调用函数 glTexParameteri 选择最近纹理颜色或者插值纹理颜色，如程序中的语句 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR) 和 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)所示。

同时，为了建立二维 RGBA 纹理空间，我们需要调用函数 glTexImage2D 指定纹理数组的宽和高，且必须是 2 的次幂。对于二维纹理图案，按自下而上次序设定纹理数组元素的元素值。从颜色图案的左下角开始，将该数组第一行的元素设定为与纹理空间底行对应的 RGBA 值，将数组最后一行元素设定为与矩形纹理空间顶行对应的 RGBA 值。最后需要调用 glEnable(GL_TEXTURE_2D)函数启用纹理。

另外，可以调用函数 glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, applicationMethod)来选择纹理映射的方法。本次实验中，将 applicationMethod

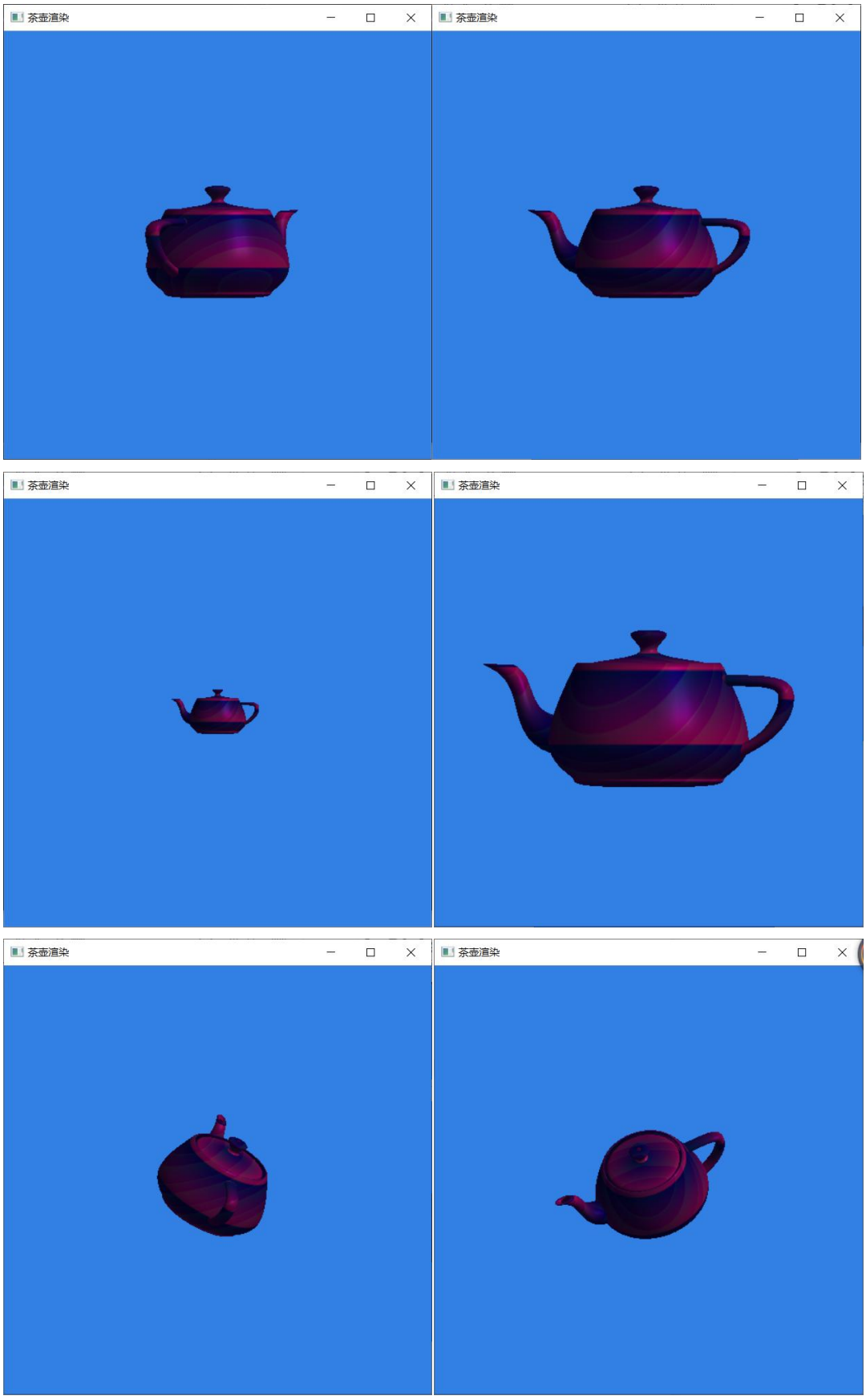
赋值为 GL_MODULATE，用纹理颜色调制当前颜色值，纹理单位颜色与几何图形颜色相乘。

3.2 程序实现流程图



4. 实验结果

4.1 运行截图


茶壶渲染
茶壶渲染
茶壶渲染
茶壶渲染
茶壶渲染
茶壶渲染

4.2 结果及问题分析

通过本次实验的结果可以看出，在绘制基本茶壶的基础之上，所添加的光照模型使得茶壶没被光照到的部分显示出了一小块阴影部分，说明光照明效果具有较好的成果。另外，通过纹理设置函数，为茶壶的表面设置了相应的颜色主体，为红色和蓝色的逐渐渐变，并还伴有着非常浅的绿色条纹，实现了茶壶的纹理映射。

另外，可以通过键盘的控制，使茶壶进行旋转、缩放，实现了人机交互的功能。

但是，在本次试验中，仍存在一定的问题，比如说茶壶的纹理设置过于暗淡，导致所生成的茶壶看不清，这可能与光照的相应参数、材质的属性以及生成的平面纹理坐标有着较大的关系。另外，键盘控制模块也存在一定的问题，使用特殊键盘时（上下左右键）时，之后按下的键还会包含之前按下键的操作。最后，本次实验也没有实现纹理贴图的功能，只是单纯地实现了简单的纹理映射。

5. 实验体会及小结

通过本次实验，我学到了很多知识，更加清晰地认识了三维图形的渲染过程，学到了关于纹理映射和光照明模型的许多理论知识，还有关于键盘控制的知识。通过学习网络上如 CSDN 的许多优秀代码，并在自己的程序中合理运用并做出相应的改进，能够更好地提高我的代码编写能力和逻辑思维能力，将所学的理论知识更好地进行运用，从而加深理解。

同时，我深深体会到了计算机图形学在绘制真实感图形中有着非常重要的作用，它能在现实生活中运用到方方面面，带来各种各样的令人惊艳的视觉效果。

在本次实验中，不可避免的遇到了一些问题，其中一些经过相应的分析以及查阅资料很好地解决了，比如纹理坐标的设置和设置特殊键盘的操作，但是另外一些问题如上文 4.2 中所述的一样，因此在以后的学习中需要更加加深相应部分的学习，解决自己在该学科中的软弱部分。

6. 参考文献

[1] 童立靖,陈静. OpenGL 纹理函数应用方法研究[J].软件导刊, 2015(07):33-35.

[2] 大卫 施耐德.OpenGL 编程指南[M].北京.机械工业出版社.2010:138-142.

[3] 郭昳,邱玉宝.OpenGL 中纹理贴图、滤波、光照和雾化效果编程初步[J].电脑编程技巧与维护,2005(03):73-76.

[4] 僧德文,李仲学,李春民.基于 OpenGL 的真实感图形绘制技术及应用[J].计算机应用研究,2005(03):173-175.

7. 附录

源代码：

```
#include<GL/glut.h>
#include <cstdlib>
#include <cstdio>
#include <cmath>
#define stripeImageWidth 28
#define stripeImageHeight 28

GLubyte stripeImage[stripeImageWidth][stripeImageHeight][4];
// 初始化材质属性、光源、照明模型和深度缓冲区

GLfloat rx, ry, rz = 0.0;            // 物体 旋转
GLdouble size=0.5;                   // 茶壶尺寸

void SetImage(void){       //设置纹理
      int i, j;
      for (i = 0; i < stripeImageWidth; i++){
            for (j = 0; j < stripeImageHeight; j++){
                  stripeImage[i][j][0] = (GLubyte)(i * 8-1);
                  stripeImage[i][j][1] = (GLubyte)(j * 13- 1);
                  stripeImage[i][j][2] = (GLubyte)222;
                  stripeImage[i][j][3] = (GLubyte)112;
            }
      }
}

static GLfloat xequalzero[] = { 0.3,0.8,0.9, 1.0 };
static GLfloat* currentCoeff;
static GLenum currentPlane;
static GLint currentGenMode;
static float roangles;

void init(void){
      GLfloat mat_ambient[] = { 0.6, 0.2, 0.7, 1.0 };       //环境光照分量
      GLfloat mat_specular[] = { 0.6, 0.2, 0.7, 1.0 };      //镜面反射参数
      GLfloat mat_diffuse[] = { 0.8, 0.1, 0.5, 1.0 };       //漫反射光照分量
```

```
GLfloat mat_shininess[] = { 85.0 };                    //高光指数

GLfloat light_position[] = { 1.0, 1.0, 1.0,0.0 };     //光源位置
GLfloat light_specular[] = { 0.9, 0.1, 0.5,1.0 };     //折射光参数
GLfloat light_diffuse[] = { 0.6, 0.5, 0.4, 1.0 };     //满射光参数
GLfloat light_ambient[] = { 0.6, 0.5, 0.4, 1.0 };     //环境光参数

glClearColor(0.2, 0.5, 0.9, 1.0);
glEnable(GL_DEPTH_TEST);       //消除隐藏曲面
glShadeModel(GL_SMOOTH);
SetImage();

glPixelStorei(GL_UNPACK_ALIGNMENT, 1);     //提取像素方式
glTexParameteri(GL_TEXTURE_2D,       GL_TEXTURE_WRAP_S,       GL_REPEAT);
//设置环绕模式
glTexParameteri(GL_TEXTURE_2D,    GL_TEXTURE_MAG_FILTER,    GL_LINEAR);
//缩小过滤器，线性过滤
glTexParameteri(GL_TEXTURE_2D,    GL_TEXTURE_MIN_FILTER,    GL_LINEAR);
//放大过滤器，线性过滤
glTexImage2D(GL_TEXTURE_2D, 0, 4, stripeImageWidth, stripeImageHeight, 0,
GL_RGBA, GL_UNSIGNED_BYTE, stripeImage);//载入纹理
glTexEnvf(GL_TEXTURE_ENV,      GL_TEXTURE_ENV_MODE,      GL_MODULATE);
//指定环境模式为：纹理单位颜色与几何图形颜色相乘。


currentCoeff = xequalzero;
currentGenMode = GL_OBJECT_LINEAR;
currentPlane = GL_OBJECT_PLANE;
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
glTexGenfv(GL_S, currentPlane, currentCoeff);   //自动生成纹理坐标
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_2D);     //启用纹理
glEnable(GL_LIGHTING);         //激活 OpenGL 光源
glEnable(GL_LIGHT0);           //启用此光照
glEnable(GL_AUTO_NORMAL);   //以分析方式计算图面法向量
glEnable(GL_NORMALIZE);       //使用 glNormal 指定的正常向量在转换后缩
放为单位长度。

roangles = 50.0f;

//光照
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);           //设置材料漫反射
指数
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);          //设置材料环境光
```

照指数
```
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);      //设置材料反射指数（纯镜面反射）
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);   //设置材料反射指数（材料反射指数）

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);    //建立光源（光源位置）
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);          //建立光源 （漫反射光分量强度）
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);        //建立光源 （折射光分量强度）
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_ambient);      //建立光照模式
}

void display(void){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    //glutSolidSphere (1.0, 20, 16);
     glPushMatrix();
     glRotatef(roangles, 0.0, 1.0, 0.0);
    glutSolidTeapot(size);
     glPopMatrix();
     glRotatef(rx, 1.0f, 0.0f, 0.0f);                                  // 绕 X 轴旋转
     glRotatef(ry, 0.0f, 1.0f, 0.0f);                                  // 绕 Y 轴旋转
     glRotatef(rz, 0.0f, 0.0f, 1.0f);                                  // 绕 Z 轴旋转
    glFlush();
}

void myKey(unsigned char key, int x, int y) { //键盘
     switch (key){
          case 'a':case 'A':
               roangles += 5.0;
               glutPostRedisplay();     //重新调用绘制函数
               break;
          case 's':case 'S':
               roangles -= 5.0;
               glutPostRedisplay();
               break;
               //控制大小
          case 'w':case'W':
               size += -0.1;
               glutPostRedisplay();
               break;
          case 'd':case'D':
               size +=    0.1;
```

```
                glutPostRedisplay();
                break;
            case 27:exit(0); break;
        }

        if (roangles> 360) roangles-= 360;
        if (roangles< 0) roangles+= 360;
}

void mySpecialKey(int key, int x, int y) {
        switch (key) {
        case GLUT_KEY_LEFT:
                rx += 1;
                glutPostRedisplay();      //重新调用绘制函数
                break;
        case GLUT_KEY_RIGHT:
                rx -= 1;
                glutPostRedisplay();
                break;
        case GLUT_KEY_UP:
                rz += 1;
                glutPostRedisplay();
                break;
        case GLUT_KEY_DOWN:
                rz -= 1;
                glutPostRedisplay();
                break;
        case 27:exit(0); break;
        }
}

void reshape(int w, int h){
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);
       glMatrixMode(GL_PROJECTION);     //  指定当前操作矩阵为投影矩阵。
       glLoadIdentity();                          //重置坐标系统
       if (w <= h)
            glOrtho(-1.5, 1.5, -1.5 * (GLfloat)h / (GLfloat)w,
                1.5 * (GLfloat)h / (GLfloat)w, -10.0, 10.0);
       else
            glOrtho(-1.5 * (GLfloat)w / (GLfloat)h,
                1.5 * (GLfloat)w / (GLfloat)h, -1.5, 1.5, -10.0, 10.0);

       glMatrixMode(GL_MODELVIEW);    //指定当前操作矩阵为模型视图矩阵。
       glLoadIdentity();
```

```
}

int main(int argc, char** argv){
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(500, 500);
	glutInitWindowPosition(100, 100);
	 glutCreateWindow("茶壶渲染");

	init();
	glutDisplayFunc(display);
	 glutKeyboardFunc(myKey);
	 glutSpecialFunc(mySpecialKey);

	glutReshapeFunc(reshape);
	glutMainLoop();
	return 0;
}
```