《计算机图形学实验》综合实验报告

题目 基于 OpenGL 的三维图形的渲染

学 号 20201120571

姓 名 文尉鹏

指导教师 钱文华

日 期 2020.6.15

摘要

使用 OpenGL 实现三维图形渲染，渲染过程中加入了纹理、色彩、光照等效果，生成了一个渲染后的茶壶。

关键词：OpenGL, 三维图形渲染

目录

一、实验背景，实验内容

1.1 实验背景

随着计算机软、硬件突飞猛进的发展，计算机图形学在各个行业的应用也得到迅速普及和深入。随着几十年图形学的发展，计算机已进入三维时代，三维图形在人们周围无所不在。科学计算可视化、计算机动画和虚拟现实已经成为近年来计算机图形学的三大热门话题，计算机图形学领域的研究层次逐步深入，并以越来越快的速度向前发展。

在计算机中，由于用图形表达各种信息，其容量大、直观方便，更符合人们观察了解事物运动规律的习惯，所以三维图形技术在建筑虚拟、城市规划、场景漫游、效果场景制作、城市规划、房地产开发、虚拟教育、展馆展示、古迹复原、交通线路设计、3D 游戏等各方面都有广泛的实际应用。研究三维图形渲染能为后续发展奠定基础。

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。

二、开发工具，程序设计及实现目的及基本模块介绍

2.1 开发工具

Visual C++, OpenGL, Java 等工具。

2.2 程序设计

先生成茶壶，再生成纹理图像，建立渲染，生成光照，将这些效果附加在茶

壶上。

2.3 实现目的

实现茶壶的三维图形渲染，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。

2.4 基本模块介绍

模块	功能
makeStripeImage 函数	生成纹理图像
display 函数	画茶壶
rendering 函数	建立渲染，生成光照
reshape 函数	窗口改变时，调用的函数
mouse 函数	鼠标函数
motion 函数	进行旋转

三、关键算法的理论介绍和程序实现步骤

3.1 理论介绍

1、加载纹理：

使用 makeStripeImage() 来绘制纹理。然后从内存中读取纹理图并放到屏幕上，glPixelStorei(GL_UNPACK_ALIGNMENT, 1)，并定义纹理环境参数：调整当前亮度和颜色信息，使之适应纹理图像：glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)。

2、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); 其中前两个参数我们使用了 GL_TEXTURE_MAG_FILTER 和

GL_TEXTURE_MIN_FILTER，前者代表了待映射纹理像素少于一个纹理单元的像素时，以怎样的方式映射；后者代表了待映射纹理像素多于一个纹理单元的像素时，以怎样的方式映射。第三个参数我们使用了 GL_NEAREST 和 GL_LINEAR 两种，前者取 4 个坐标上最接近待映射纹理像素的颜色，后者取坐标上带映射纹理接近像素的平均值。

3、纹理贴图：

使用该函数：glTexImage1D(GL_TEXTURE_1D, 0, 3, imageWidth, 0, GL_RGB, GL_UNSIGNED_BYTE, stripeImage)来定义纹理，并要开启纹理，采用下列函数：glEnable(GL_TEXTURE_GEN_S);和 glEnable(GL_TEXTURE_1D)。

4、光照效果：

使用该函数开启光照效果 glEnable(GL_LIGHTING)；然后设置光照材质与位置，具体如下：

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);

通过 glLightfv（光源编号，光源特性，参数数据）来设置光源。其中，光源编号可取 GL_LIGHT0、GL_LIGHT1、……、GL_LIGHT7 共 8 个值。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。

其中 GL_LIGHT0 是来打开光源的，对于 GL_POSITION，其位置数组(x，y，z，w)定义了光源在空间中的位置。三维空间需要 4 个量，是因为这里采用的是齐次坐标，当 w≠0 时，它表示光源处于空间中(x，y，z)处，这时的光源称为定点光源；当 w＝0 时，根据齐次坐标的性质，它表示光源位于无穷远处，此时光源称为定向光源，其所有光线几乎是相互平等的，如太阳。

参数数据定义为：

GLfloat light_position[] = { -50.0, 100.0, 100.0, 0.0 };（光源位置）

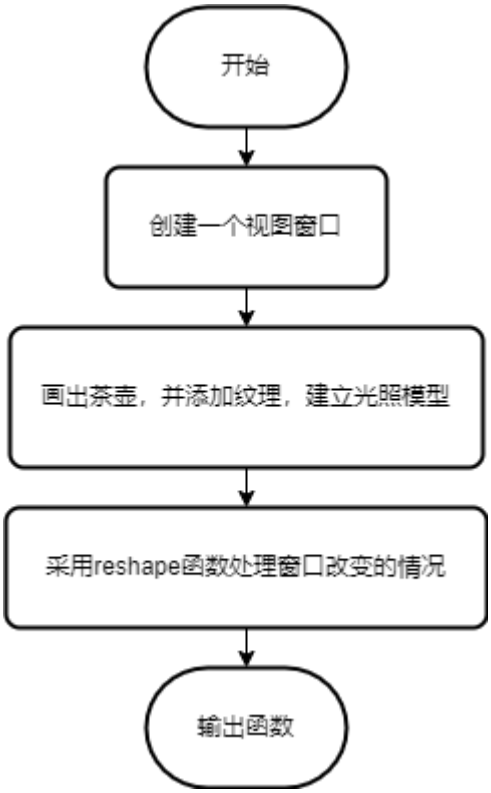
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };（环境光）

GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };（漫反射）

GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };（镜面光）

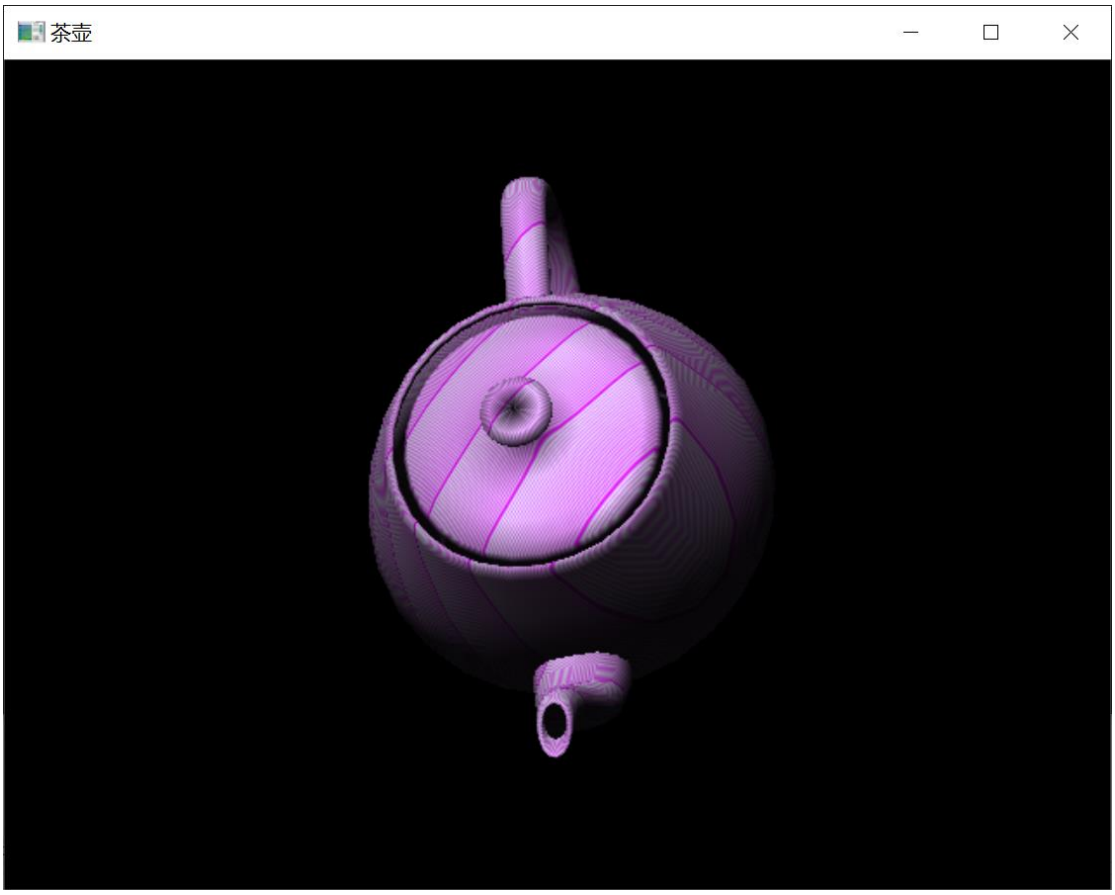
3.2 程序实现步骤

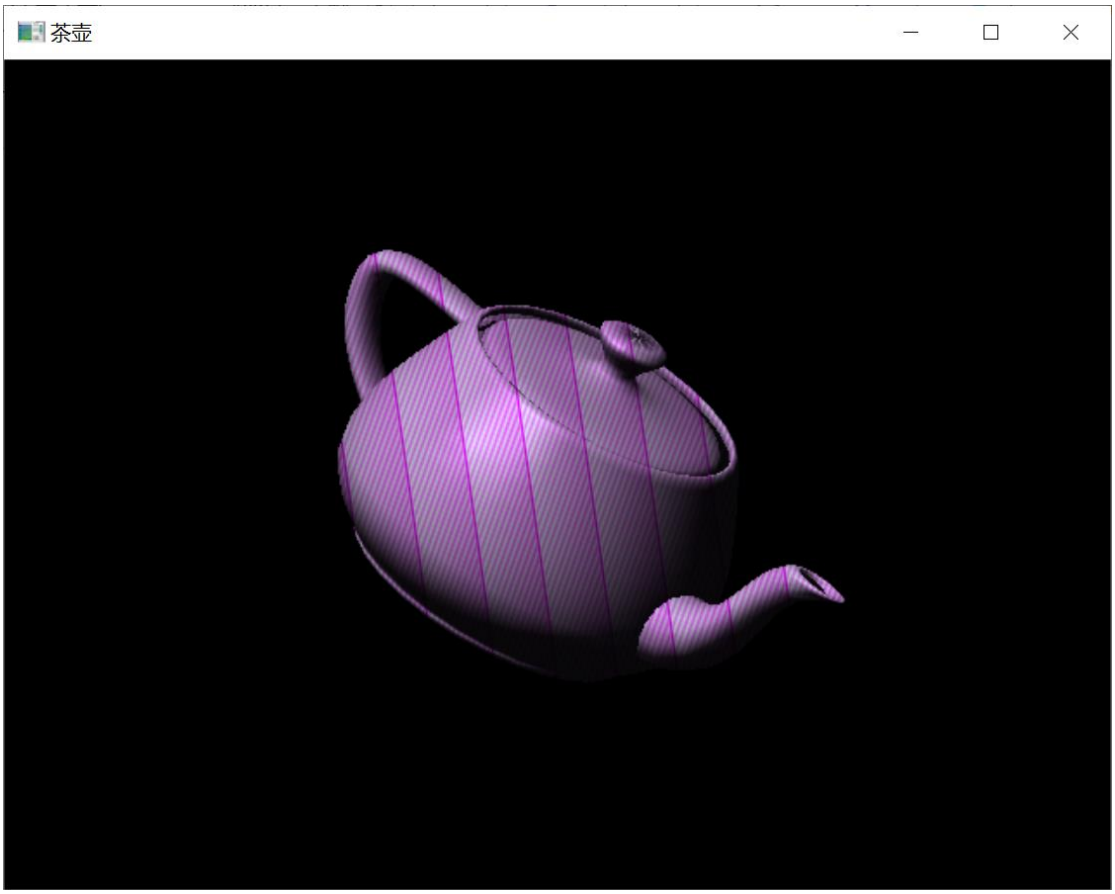
流程图如下：

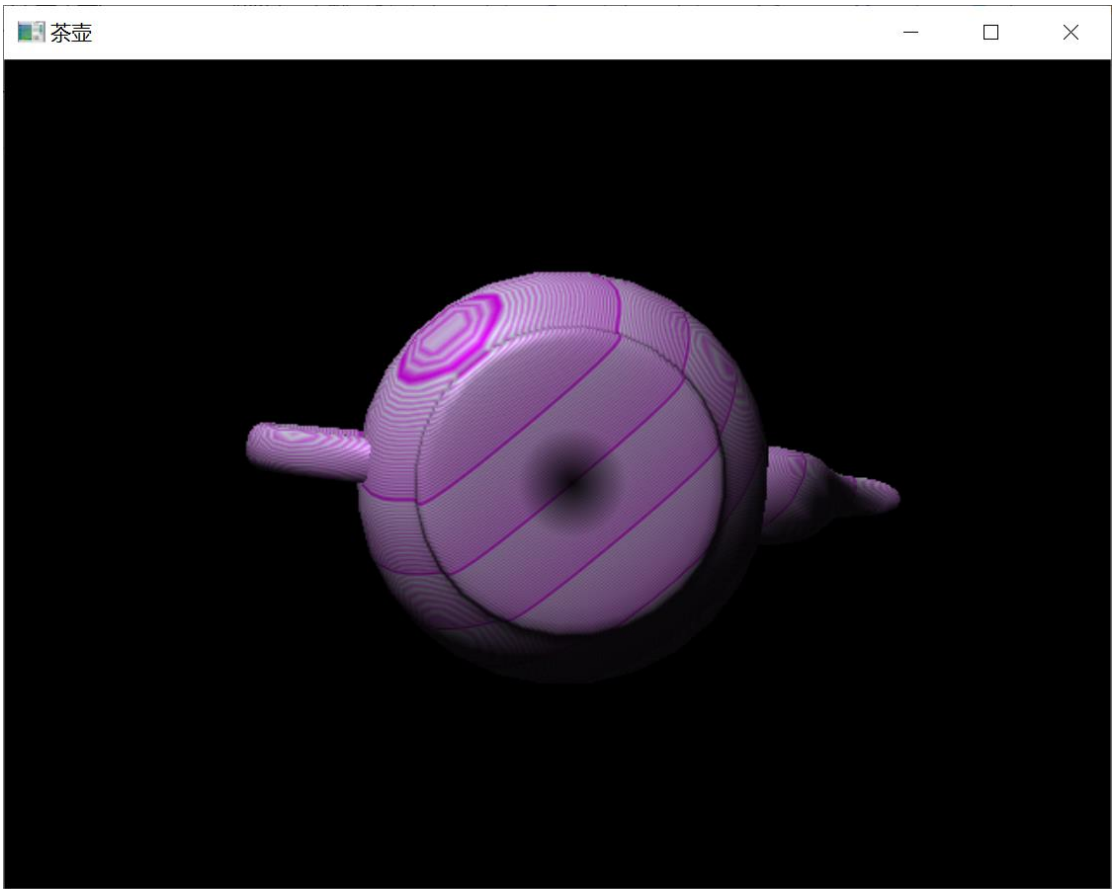


四、实验结果

4.1 实验运行截图




茶壶


茶壶

4.2 结果分析

使用 makeStripeImage() 函数来渲染茶壶，并可以通过鼠标来控制茶壶旋转，按鼠标右键可以实现自动旋转。

缺陷是不能实现图片贴图，要想改变纹理，只能修改函数来生成。

五、实验体会及小结

通过这次实验，我学到了很多，但也存在许多不足。

一开始，在进行图形渲染时，并不能很好的实现，与预料的不一致。然后我就不断尝试修改代码，以及在各种网站的程序实现，在这个过程中，我逐步学会并掌握了光照明模型、纹理贴图、纹理映射等算法，并最终将学到的知识融合到程序的实现。

但也由于自己能力不够，并不能照片的贴图，以后我也会加强这方面的学习。

六、参考文献

[1] OpenGL glLightfv 函数的应用以及光源的相关知识

https://blog.csdn.net/chy19911123/article/details/46413121

[2] [OpenGL]茶壶与纹理

https://blog.csdn.net/ZJU_fish1996/article/details/51419541

[3] openGL 实现旋转的茶壶

https://blog.csdn.net/qq_15267341/article/details/83273109

七、附录

源代码：

```
#include <windows.h>
#include<GL/glut.h>
#define imageWidth 50
```

```
GLfloat roate = 0.0;//  设置旋转速率
GLfloat rote = 0.0;//旋转角度
GLfloat anglex = 0.0;//X  轴旋转
GLfloat angley = 0.0;//Y  轴旋转
GLfloat anglez = 0.0;//Z  轴旋转
GLint WinW = 400;
GLint WinH = 400;
GLfloat oldx;//当左键按下时记录鼠标坐标
GLfloat oldy;

GLubyte stripeImage[3 * imageWidth];

void init(void) {

}

//定义纹理图像
void makeStripeImage(void)
{
	int j;
	for (j = 0; j < imageWidth; j++)
	{
		stripeImage[3 * j] = 1000;
		stripeImage[3 * j + 1] = 255 /2 * j;
		stripeImage[3 * j + 2] = 255;
	}
}
/*  参数设置  */
GLfloat sgenparams[] = { 1.0, 1.0, 1.0, 0.0 };
```

```
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };//材质的镜面反射系数
GLfloat mat_shininess[] = { 100.0 };//材质的镜面光指数
// 光源 0
GLfloat light_position[] = { -50.0, 100.0, 100.0, 0.0 };//光源位置
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };//环境光
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };//漫反射
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };//镜面光
//生成茶壶
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);//开启深度测试
    glDepthFunc(GL_LESS);//深度测试函数
    glColor3f(1.0, 0.0, 0.0);
    glLoadIdentity();//加载矩阵
    glPushMatrix();//矩阵入栈
    glRotatef(rote, 0.0f, 1.0f, 0.0f);
    glRotatef(anglex, 1.0, 0.0, 0.0);
    glRotatef(angley, 0.0, 1.0, 0.0);
    glRotatef(anglez, 0.0, 0.0, 1.0);
    rote += roate;
    glutSolidTeapot(50);//绘制茶壶
    glPopMatrix();//矩阵出栈
    glutPostRedisplay();
    glutSwapBuffers();
}
//建立渲染，生成光照
void rendering(void)
{
```

```
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);//光暗处理
    makeStripeImage();//绘制纹理
    //函数设定从内存中读取纹理图并放到屏幕上的方式
    //指定内存中每个象素行起始的排列要求为字节排列（1）
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    //定义纹理环境参数：调整当前亮度和颜色信息，使之适应纹理图像
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);
    //纹理绕转使用重复方式
    glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S,
GL_REPEAT);
    //定义纹理放大和缩小函数均为 GL_LINEAR
    glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    //定义纹理
    glTexImage1D(GL_TEXTURE_1D, 0, 3, imageWidth, 0, GL_RGB,
GL_UNSIGNED_BYTE, stripeImage);
    //控制纹理坐标的生成
    //指定单值纹理生成参数
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
    //指定纹理坐标生成函数,系数由 sgenparams 指定
    glTexGenfv(GL_S, GL_OBJECT_PLANE, sgenparams);

    glEnable(GL_TEXTURE_GEN_S);//开启纹理坐标映射
    glEnable(GL_TEXTURE_1D);//开启纹理
```

```
    glEnable(GL_LIGHT0);//开启 0 光源
    //设置材质
    glMaterialf(GL_FRONT, GL_SHININESS, 64.0);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    glEnable(GL_LIGHTING);//开启光照效果
    //设置光照材质与位置
    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);

}
//窗口改变时，调用的函数
void   reshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-100.0, 100, -100.0 * (GLfloat)h / (GLfloat)w, 100.0 * (GLfloat)h
/ (GLfloat)w, -1000.0, 1000.0);
    else
        glOrtho(-100.0 * (GLfloat)w / (GLfloat)h, 100.0 * (GLfloat)w /
(GLfloat)h, -100.0, 100.0, -1000.0, 1000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

```
void mouse(int button, int state, int x, int y) //  鼠标函数
{
	if (button == GLUT_LEFT_BUTTON)
	{
		if (state == GLUT_DOWN)
		{
			roate = 0;
			rote = 0;
			oldx = x;//当左键按下时记录鼠标坐标
			oldy = y;
		}
	}
	if (button == GLUT_RIGHT_BUTTON)
	{
		if (state == GLUT_DOWN)
		{
			roate += 1.0f;
		}
	}
}

void motion(int x, int y)
{
	GLint deltax = oldx - x;
	GLint deltay = oldy - y;
	anglex += 360 * (GLfloat)deltax / (GLfloat)WinW;//根据屏幕上鼠标滑动的距离来设置旋转的角度
	angley += 360 * (GLfloat)deltay / (GLfloat)WinH;
```

```
        anglez += 360 * (GLfloat)deltay / (GLfloat)WinH;
        oldx = x;//记录此时的鼠标坐标，更新鼠标坐标
        oldy = y;//若是没有这两句语句，滑动是旋转会变得不可控
        glutPostRedisplay();
}

int main(int argc, char** argv)
{
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(640, 480);
        glutInitWindowPosition(100, 150);
        glutInit(&argc, argv);
        glutCreateWindow("茶壶");
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutMouseFunc(mouse);
        glutMotionFunc(motion);
        rendering();
        glutMainLoop();
        return 0;
}
```