//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);

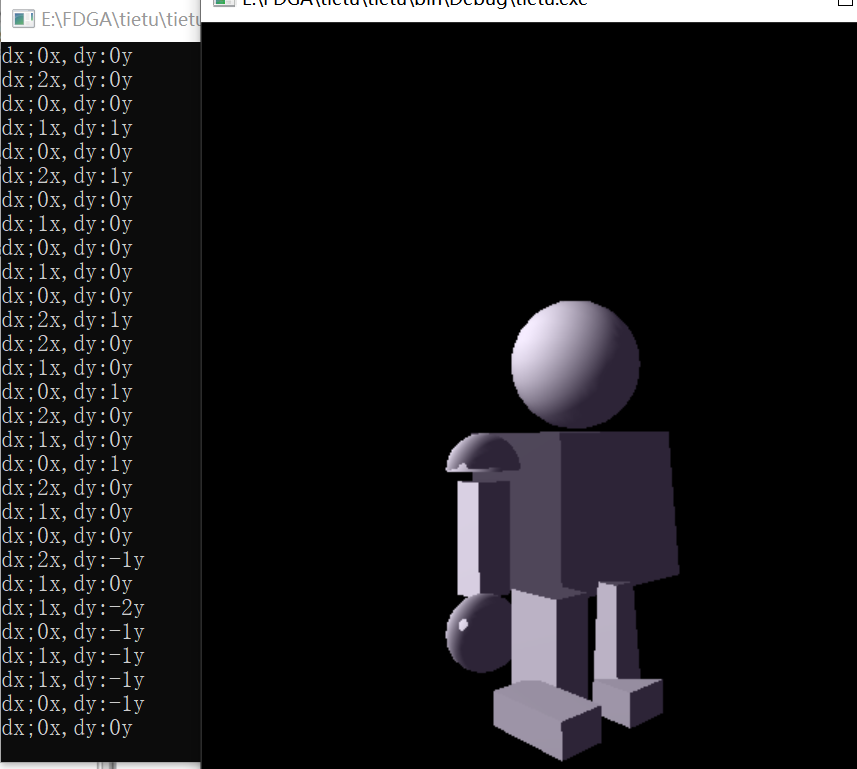
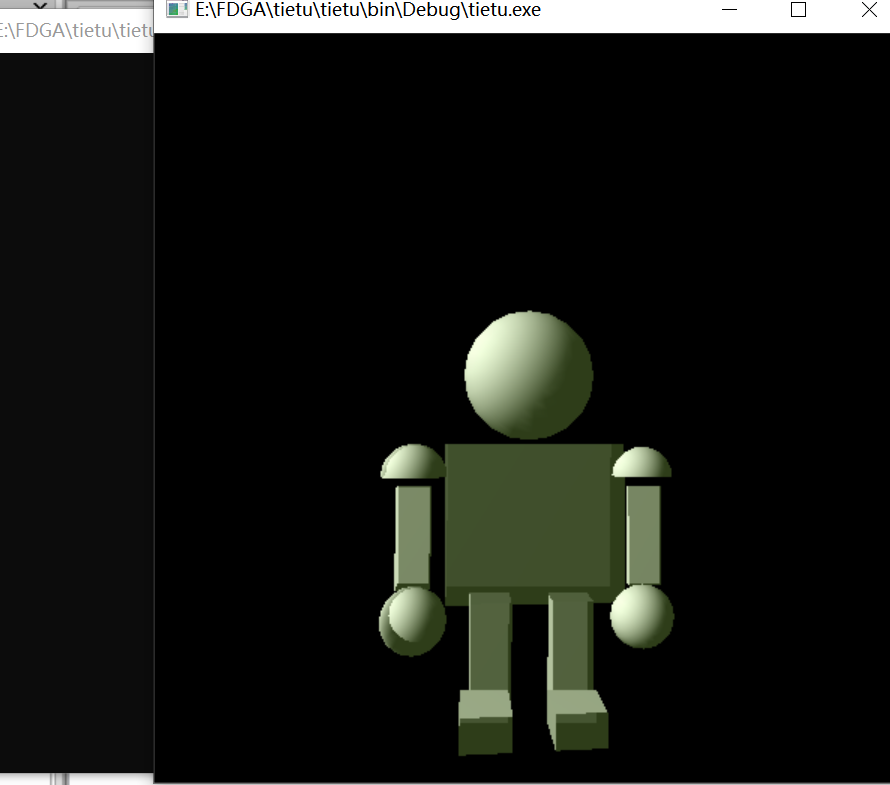
//循环

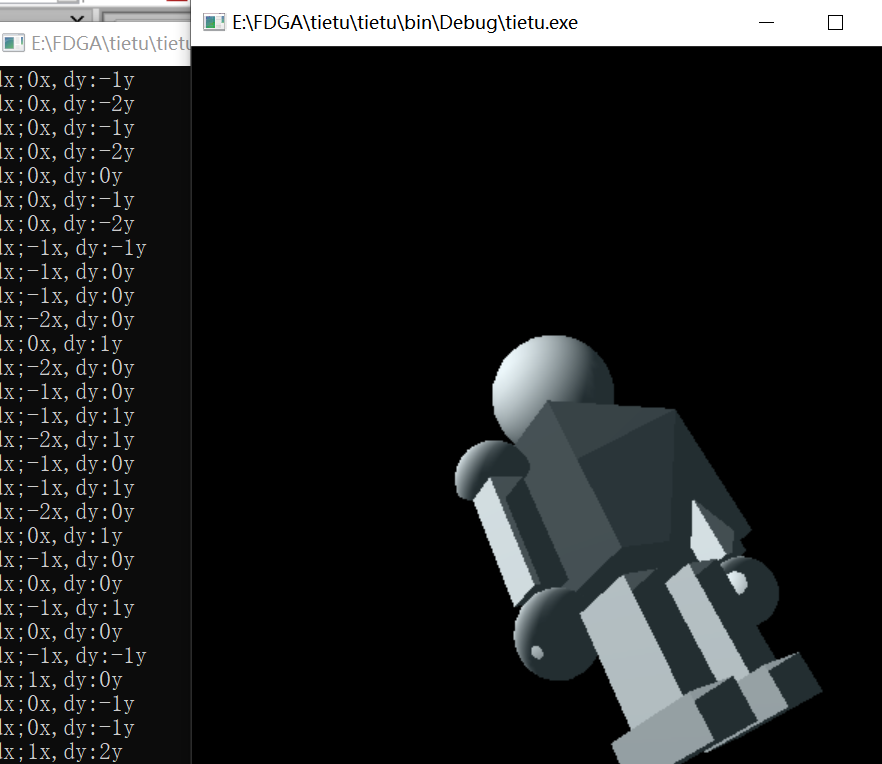
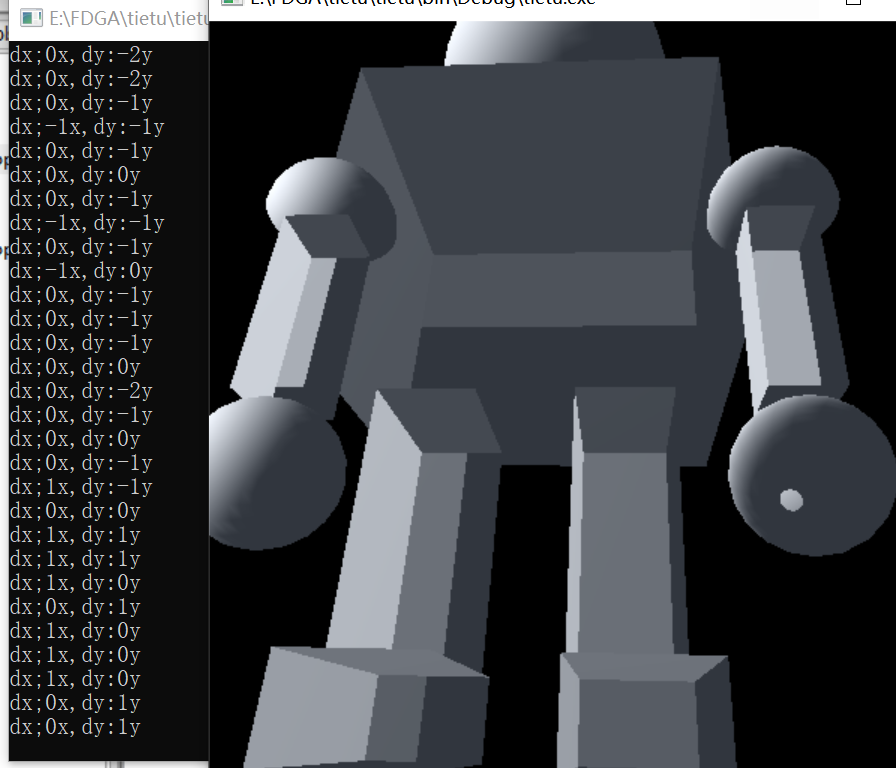
glutMainLoop();

return 0;

}

运行结果：





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

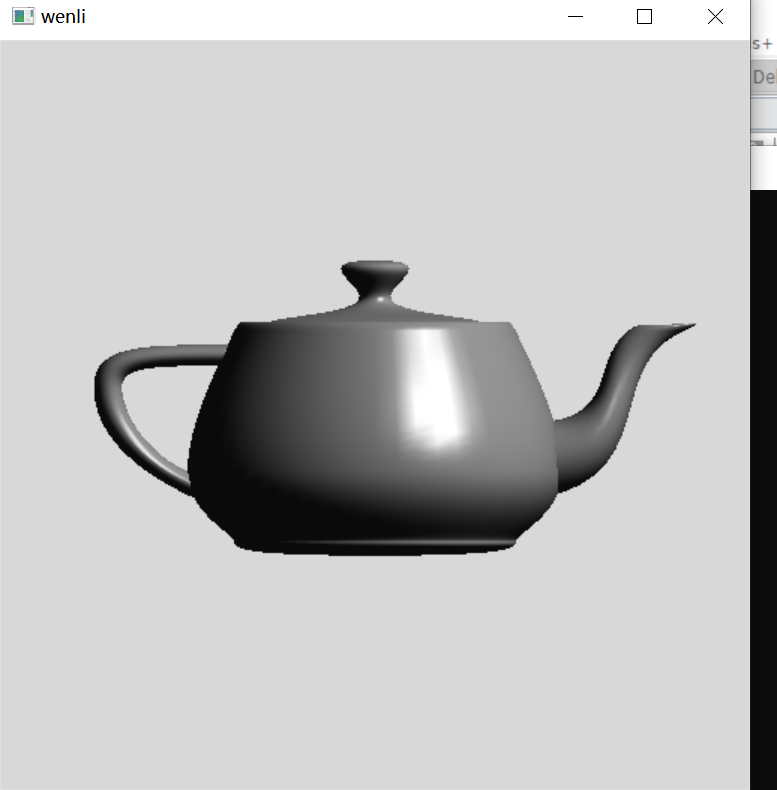
glutMainLoop(); //进入循环

return 0;

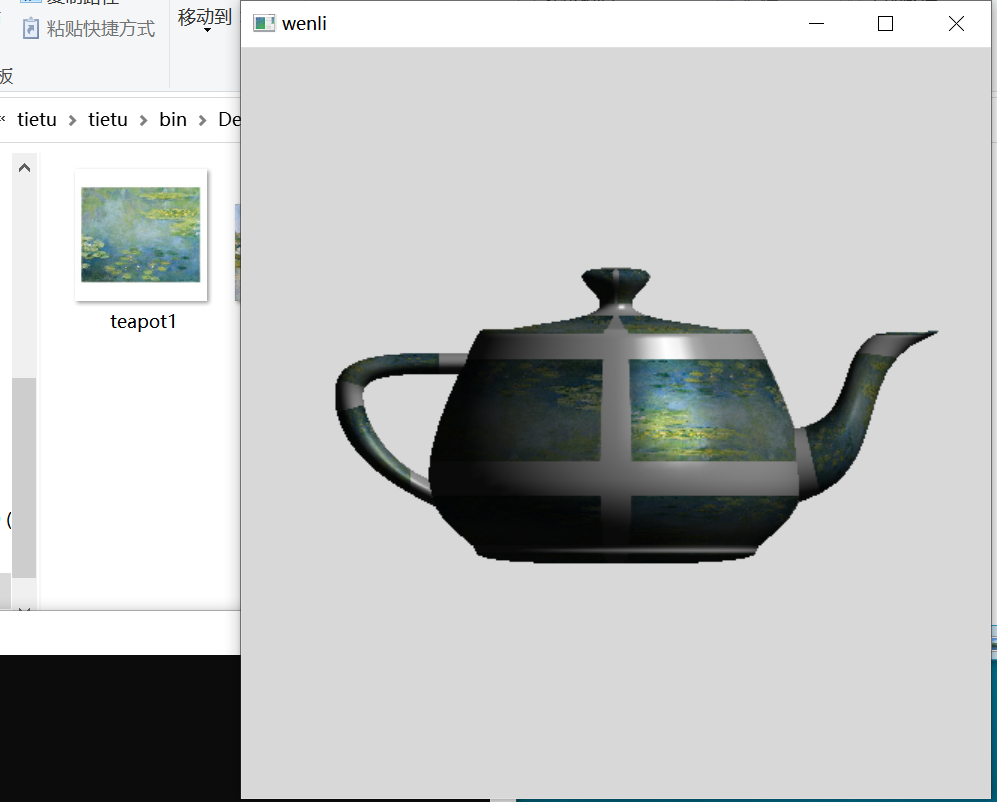
}

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。