

# 浙江大学实验报告

专业： 计算机科学与技术

姓名： 余启航

学号： 3190103324

日期： 2021.11.19

地点： 曹光彪西 501

课程名称： 计算机图形学 指导老师： 童若锋 成绩： \_\_\_\_\_

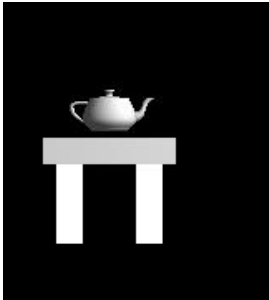
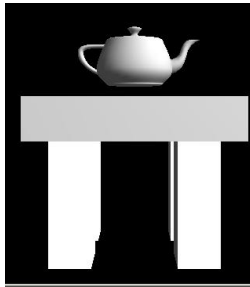
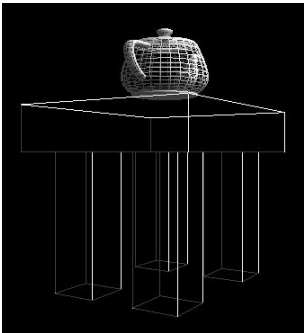
实验名称： OpenGL 三维观察 实验类型： 基础实验 同组学生姓名： \_\_\_\_\_

## 一、实验目的和要求

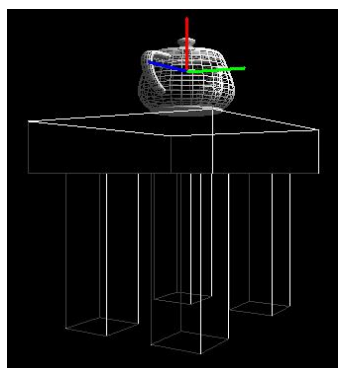
在模型变换实验的基础上，通过实现下述实验内容，掌握 OpenGL 中三维观察、透视投影、正交投影的参数设置，并能使用键盘移动观察相机，在透视投影和正交投影间切换，验证课程中三维观察的内容；进一步加深对 OpenGL 三维坐标和矩阵变换的理解和应用。

## 二、实验内容和原理

使用 Visual Studio C++编译已有项目工程，并修改代码生成以下图形：

 <p>正投影</p>	 <p>透视投影</p>
	使用键盘改变 camera 位置与观察方向 (按键为 W、S、A、D、Z、C，也可以自行设定)

添加键盘对茶壶的控制，主要是茶壶沿着桌面的平移操作（如下图中绿色和蓝色标示）和茶壶绕自身轴（如下图中红色标示）的旋转操作；按键为：l,j,i,k,e。具体对应关系可查看参考答案中的操作指南。



### 三、主要仪器设备

Visual Studio C++

glut.zip

Ex3-vs2010 工程

### 四、操作方法和实验步骤

#### 1. 明确目标

实验模板已经实现了现况显示，整体旋转。因此，我们需要实现其他按键的功能，包括茶壶相对桌子的移动，投影方式切换以及 camera 的变化。

#### 2. 设置茶壶移动

查找到绘制茶壶的代码如下：

```
1.  glPushMatrix();
2.  glTranslatef(0, 0, 5);
3.  glRotatef(90, 1, 0, 0);
4.  glutSolidTeapot(1);
5.  glPopMatrix();
```

我们需要实现它的桌面移动和自身旋转，因此需要增加参数用于设置旋转和平移：

```
1.  GLfloat teaxyz[3] = { 0,0,5 };
2.  float tearotate = 0.0f;
```

因此绘制函数进行修改，最终变成：

```
1.  glPushMatrix();
2.  glTranslatef(teaxyz[0], teaxyz[1], teaxyz[2]);
3.  glRotatef(90, 1, 0, 0);
4.  glRotatef(tearotate, 0, 1, 0);
5.  glutSolidTeapot(1);
6.  glPopMatrix();
```

根据键盘按键对参数进行修改，使得回调函数执行时能产生显示的变化：

```
1.  case 'l': {
2.      if(teaxyz[0]<2.0f)
3.          teaxyz[0] += 0.05f;
4.      break;
5.  }
6.  case 'j': {
7.      if (teaxyz[0] >-2.0f)
8.          teaxyz[0] -= 0.05f;
9.      break;
10. }
11. case 'i': {
12.     if (teaxyz[1] <2.0f)
13.         teaxyz[1] += 0.05f;
```

```

14.     break;
15.     }
16. case 'k': {
17.     if (teaxyz[1] > -2.0f)
18.         teaxyz[1] -= 0.05f;
19.     break;
20.     }
21. case 'e': {
22.     tearotate += 10;
23.     if(tearotate>360)
24.         tearotate -=360;
25.     break;
26.     }

```

将初始界面的屏幕向内方向视为前方，按键 I 会使其前移，K 使其后移，J 使其左移，L 使其右移，E 使其旋转。

### 3. 设置投影模式

目前已经实现了正投影以及投影切换的回调，只需要设置透视投影的具体实现：

```

1.  if (bPersp){
2.     gluPerspective(45.0f, 1, 5.0f, 40.0f);
3.  }

```

透视投影可以以该函数进行设置，其函数原型为：

```

1.  void gluPerspective( GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);

```

fovy 是视线上下张开的角度值，值越小，视野范围越狭小，值越大，视野范围越宽阔；zNear 表示近裁剪面到眼睛的距离，zFar 表示远裁剪面到眼睛的距离。aspect 表示裁剪面的宽 w 高 h 比，这个影响到视野的截面有多大。

### 4. 设置 camera 位置和观察方向

相机的设置如下：

```

1.  gluLookAt(eye[0], eye[1], eye[2],
2.     center[0], center[1], center[2],
3.     0, 1, 0);

```

因此调整 camera 的位置和方向需要对参数 eye, center 进行调整。Z,C 调整的是距离，只需要对 eye[2] 进行调整，而相机在 X,Y 方向移动时，为了使得投影方向不变，在调整相机位置时，需要同时修改观察中心的位置，使得观察方向不变：

```

1.  case 'a': {
2.     eye[0] += 0.05f;
3.     center[0] += 0.05f;
4.     break;
5.  }
6.  case 'd': {
7.     eye[0] -= 0.05f;
8.     center[0] -= 0.05f;
9.     break;

```

```

10.         }
11.  case 'w': {
12.      eye[1] -= 0.05f;
13.      center[1] -= 0.05f;
14.      break;
15.  }
16.  case 's': {
17.      eye[1] += 0.05f;
18.      center[1] += 0.05f;
19.      break;
20.  }
21.  case 'z': {
22.      eye[2] -= 0.1f;
23.      break;
24.  }
25.  case 'c': {
26.      eye[2] += 0.1f;
27.      break;
28.  }

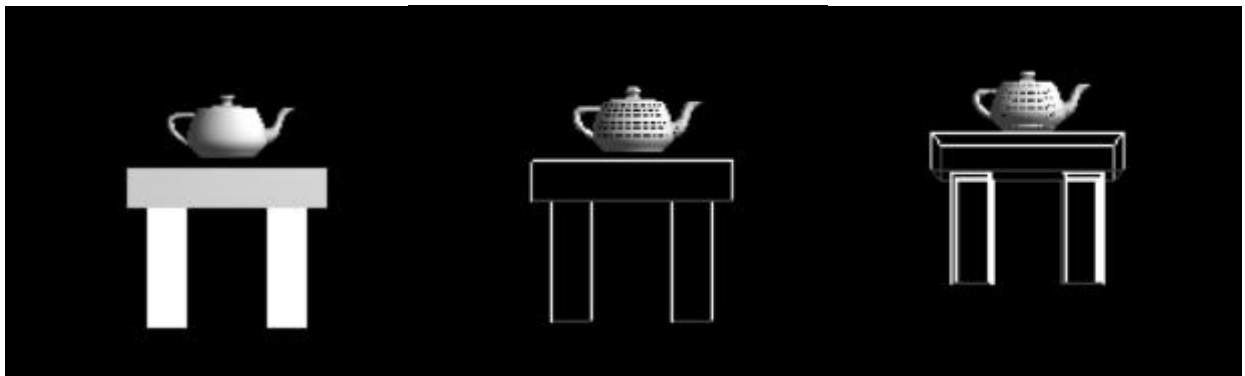
```

## 五、实验数据记录和处理

无数据需要记录处理。

## 六、实验结果与分析

### 1. 显示模式和投影模式切换



### 2. camera 位置变化

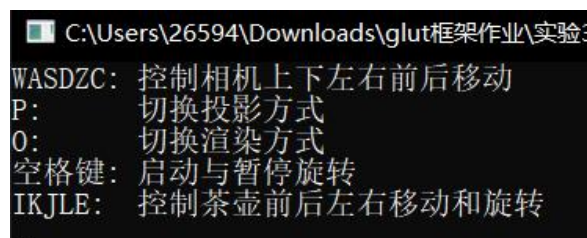


### 3. 茶壶移动和旋转



### 七、讨论、心得

1. 本次实验代码位于一个文件内，可读性和逻辑性比较低，改为多文件，根据不同功能进行文件分割，有利于代码结构观察，更具有可读性。
2. 操作按键只能通过记忆，每次需要查看文本文档的记录，很不方便。可以在终端窗口输出规则，方便每次调试以及最后的使用。



3. 按键识别只设置了小写字母，可以增加大写字母增加灵活性。可以在 `case` 增加判断，也可以直接对输入进行判断，如果发现是大写字母，则改为对应的小写字母。
4. 在上次实验中，我们只需要在每次绘制图像时改变物体本身的位置就可以实现平移、旋转和缩放，而这次实验涉及到使用正射投影和透视投影来观察并移动物体，就必须要对坐标系统进行更深入的理解了。在绘制图像之前，我们在 `reshape(int width, int height)` 函数中调用 `updateview(int width, int height)` 设置坐标系统的初始化。

`glViewport (GLint x, GLint y, GLsizei width, GLsizei height)` 其中 `x,y` 代表视窗左下角的坐标，默认为 `(0, 0)`，`width` 和 `height` 设定了截取的图形以怎样的比例显示在视窗上。

在分析投影模式前，我们先分析一下坐标系统的变换。在绘制图像时，我们使用的是局部空间，也就是以每一个物体对应一个坐标系统，以自身中心为原点，在这次实验中，我们直接调用 `glutSolidTeapot(GLdouble size)` 和 `glutSolidCube(GLdouble size)` 绘制。接下来，为了将不同的物体放置在一起，我们需要将绘制完成的图像放在世界空间中，这个变换其实就是通过平移、缩放、旋转这三个变换矩阵实现。当然在这之前我们需要将矩阵模式设置为 `MODELVIEW`，而 `MODELVIEW` 其实包括了两个过程：`MODEL` 代表 `Model Matrix`，将局部空间转换到世界空间，`VIEW` 代表 `View Matrix` 将世界空间转换为观察空间，所以我们可以理解为，当一个物体被绘制好并摆放好之后，它就已经处于观察空间下了。

在观察空间下我们进行相机的移动的操作。通过把场景中的所有物体往相反方向移动的方式来模拟出摄像机，产生一种我们在移动，而不是场景在移动的感觉。观察空间的变换通过 `LookAt` 矩阵实现：

1. `gluLookAt(eye[0], eye[1], eye[2],`
2. `center[0], center[1], center[2],`
3. `0, 1, 0);`

,第一行代表相机的位置，第二行代表相机的指向的方向，第三行代表相机头顶的向量，具体矩阵如下：

$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

P 是位置向量，D 是方向向量，RU 分别是右、上向量。

5. 实际上我们是创建了一个以相机为坐标原点的坐标系统，方向向量是 z 轴，R、U 分别是 X 轴和 Y 轴，R 通过 U 和 D 点乘得到，并通过这个系统去观察世界空间中的物体。因此要实现 WASDZC 移动相机，我们只需要同时改变位置向量和方向向量即可（否则不是平视，物体会有倾斜的感觉），以 W 上移为例，我们同时使相机下移，则物体相对上移

```
1. case 'w': {
2.     eye[1] -= 0.05f;
3.     center[1] -= 0.05f;
4.     break;
5. }
```

以 Z 靠近为例，我们将相机 z 位置坐标减小，则物体相对靠近，由于始终是平视，所以方向向量 z 可以不变

```
1. case 'z': {
2.     eye[2] -= 0.1f;
3.     break;
4. }
```

最后我们通过设置矩阵模式为 PROJECTION，使用投影矩阵转换剪裁空间，即对观察坐标系中的图像进行剪裁，并决定哪些点以怎样的形式出现在屏幕上。正射投影我们使用

```
1. glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble
    zNear, GLdouble zFar);
```

我们可以理解定义了一个立方体容器，6 个参数决定了立方体的长宽高和位置，实际显示效果类似于 2D 映射，在移动过程中物体大小不会发生变化，所以前后移动观察不出来。这和我们实际体验不同，因此我们使用透视投影来解决这个问题。透视投影我们使用

```
1. gluPerspective (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);
```

fovy 代表视角，aspect 代表宽高比，通常为 width/height，near 和 far 分别代表近处和远处透视投影的矩阵。

6. 光照属于 opengl 中非常重要的领域，对光照得体的模拟会使渲染的场景增加很多视觉吸引力。最基本的光照模型有三种“环境光/漫反射光/镜面反射光”这里选择的是环境光：

```
1. glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
2. glLightfv(GL_LIGHT0, GL_AMBIENT, white);
3. glEnable(GL_LIGHT0);
```

POSITION 和 light\_pos 定义光源位置，AMBIENT 和 white 定义白色环境光，GL\_LIGHT0 是光源种类的宏定义。