

# 浙江大学实验报告

专业：数字媒体技术

姓名：杨锐

学号：3180101941

日期：2020/04/04

地点：

课程名称：计算机图形学 指导老师：唐敏 成绩：\_\_\_\_\_

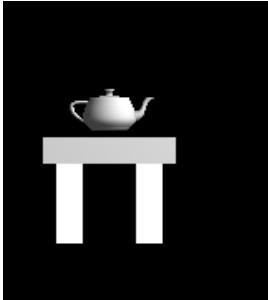
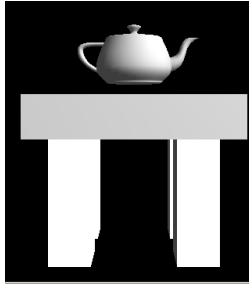
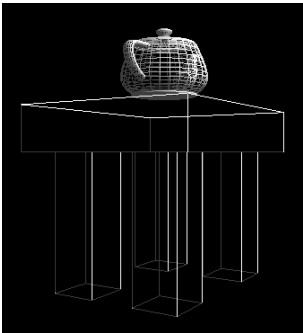
实验名称：OpenGL 三维观察 实验类型：基础实验 同组学生姓名：\_\_\_\_\_

## 一、实验目的和要求

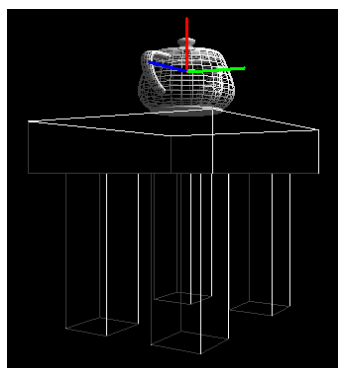
在模型变换实验的基础上，通过实现下述实验内容，掌握 OpenGL 中三维观察、透视投影、正交投影的参数设置，并能使用键盘移动观察相机，在透视投影和正交投影间切换，验证课程中三维观察的内容；进一步加深对 OpenGL 三维坐标和矩阵变换的理解和应用。

## 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

这次实验要求分别使用正投影和透视投影观察物体，并实现用键盘控制物体的移动和旋转，所以难点在于对opengl坐标系统的理解，以及动画效果的实现原理，因此我们从opengl函数的执行流程开始分析。

### 1. 动画效果实现原理

opengl的核心特点是：事件驱动和状态机。

事件驱动决定了opengl的代码运行逻辑。我们的程序从 `glutMainLoop()` 进入循环，在这个循环中处理之前放入queue的event（绘制、输入输出等），并在特定情况下trigger callback函数。这也就意味着，任何操作都必须在`glutMainLoop()`之前调用，一旦进入循环，程序只能按照已经设定好的顺序执行函数。

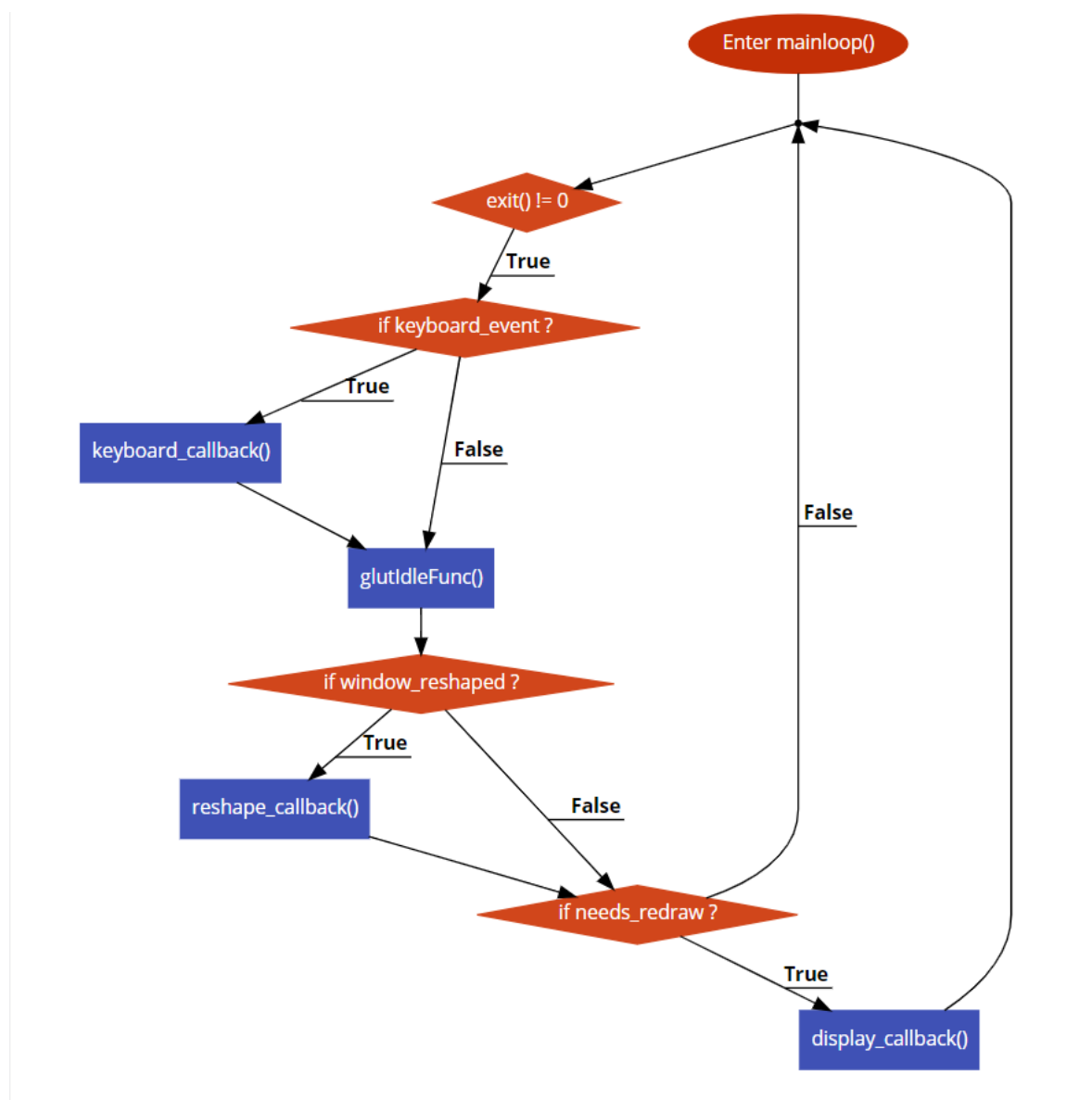
在这个实验中，`glutDisplayFunc` 传入`redraw`函数的句柄（指针），意味着每当窗口需要重新绘制时（当然包括第一次初始化），该函数就会调用`redraw`函数更新图像，这也是为什么我们需要在`redraw`的开头使用`glClear()`。例如当窗口的大小发生变化、图像显示发生了变化等情况时，我们`glutMainLoop()`会自动将`glutDisplayFunc()`加入下一次loop的event queue执行。

但是当我们希望使用键盘、鼠标移动物体时，仅仅使用`glutDisplayFunc()`是不够的。因为我们希望达到的效果是：按下某个键，窗口图像立刻发生改变，或者说，图像不停刷新以实现连续动画的效果。而在`glutKeyboardFunc(key)`中，我们仅仅是改变了某个参数的值，这时候`glutDisplayFunc()`并不会调用：换句话说，它在等待被调用。

因此我们需要使用`glutIdleFunc()`和`glutPostRedisplay()`来实现。

`glutIdleFunc()`将在空闲时调用：即当前loop中，没有event需要处理，那么调用它。因此，我们通常将图像绘制放入`glutDisplayFunc`中，将动画效果所需要的某些参数改变放入`glutIdleFunc()`中（所以我觉得已有的代码把`fRotate`放入`redraw`中是不太好的，因为`redraw`在idle之后执行，所以包括茶壶的旋转都应该放入Idle中，`redraw`应该专注于绘制每一帧的静态图像）。而`glutPostRedisplay()`并不是直接调用`display`函数，而是告诉程序，下一次loop时调用`display`函数重新绘制，本质上是将`redraw`函数queued诸如键盘操作这样的event后面。

这个实验中loop的流程图如下：



从流程图中我们可以清楚的看到，如果没有在idle中调用`glutPostRedisplay()`告诉程序需要display，那么我们只是不停地通过键盘输入改变参数，而没有将图像redraw到屏幕上。

## 2. opengl中的坐标系

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)（我调试了一下，结果是影响物体显示的初始位置，不常用），width和height设定了截取的图形以怎样的比例显示在视窗上，默认用原窗体的比例。

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，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。OpenGL本身没有相机机的概念，但我们可以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：

LookAt传入9个参数

```
gluLookAt(eye[0], eye[1], eye[2],
          center[0], center[1], center[2],
          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。

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

```
eye[1] -= 0.08f;
center[1] -= 0.08f;
```

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

```
eye[2] -= 0.08f;
//center[2] -= 0.08f;
```

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

```
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。

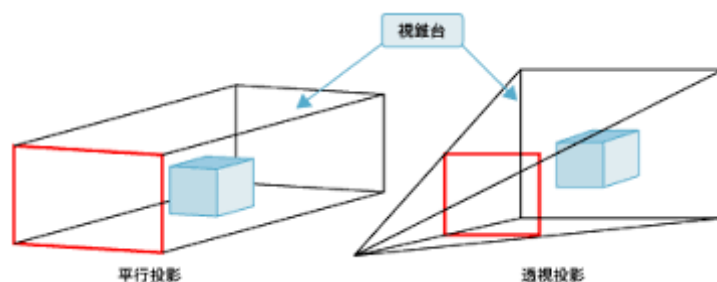
透视投影我们使用

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

fovy代表视角（以 $45^\circ$ 最常用），aspect代表宽高比，通常为width/height，near和far分别代表近处和远处

透视投影的矩阵我还没有完全推导出来，这里就不写了^^。

具体效果如图：



### 3.光照

为了使显示效果更真实，我们使用

```
glEnable (GL_DEPTH_TEST);
```

该函数会将图像投影到2D平面来判断物体的遮挡关系，也就是对于xy坐标相同的，只显示z较大的。

而光照属于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：

```
glLightfv(GL_LIGHT0, GL_POSITION, light_pos);/*void glLightfv(
    GLenum light,
    GLenum pname,
    const GLfloat *params
);*/
glLightfv(GL_LIGHT0, GL_AMBIENT, white);
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

### 4.其他

茶壶的移动和上次实验一样，我们设置一个数组储存茶壶每一次的位置向量，IJKL改变对应坐标，在绘制时移动

旋转通过一个参数来控制角度，一个bool型变量控制是否旋转，在空闲注册函数中进行旋转

在reshape时根据bool变量选择投影方式，redraw时选择渲染方式。

## 5.鼠标

在原来的基础上加了个简易的鼠标控制旋转，比较难操作🐼

```
glutMouseFunc(mouse);
glutMotionFunc(motion) //函数调用

void mouse(int button, int state, int x1, int y1)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        x = x1; //当左键按下时记录鼠标坐标
        y = y1;
    }
}

void motion(int x1, int y1)
{
    GLint dx = x - x1;
    GLint dy = y - y1;
    x_angle -= 360 * (GLfloat)dy / (GLfloat)wHeight; //根据屏幕上鼠标滑动的距离来设置旋转的角度
    y_angle -= 360 * (GLfloat)dx / (GLfloat)wWidth;
    x = x1; //记录此时的鼠标坐标，更新鼠标坐标
    y = y1;
    //glutPostRedisplay();
}

//在redraw绘制之前加入
glRotatef(x_angle, 1.0, 0.0, 0.0);
glRotatef(y_angle, 0.0, 1.0, 0.0);
```

## 五、心得

因为之前的实验没有太关注整个程序的运行和函数之间的关系，所以这次花了比较多的时间学习，经过这次实验，对opengl的运行逻辑有了更深的认识了。