实验 8　OpenGL 模型视图变换 参考资料

1．实验目的：

理解掌握 OpenGL 程序的模型视图变换。

2．实验内容：

（1）阅读实验原理，运行示范实验代码，理解掌握 OpenGL 程序的模型视图变换；

（2）根据示范代码，尝试完成实验作业；

3．实验原理：

我们生活在一个三维的世界——如果要观察一个物体，我们可以：

1、从不同的位置去观察它（人运动，选定某个位置去看）。（视图变换）

2、移动或者旋转它，当然了，如果它只是计算机里面的物体，我们还可以放大或缩小它（物体运动，让人看它的不同部分）。（模型变换）

3、如果把物体画下来，我们可以选择：是否需要一种“近大远小”的透视效果。另外，我们可能只希望看到物体的一部分，而不是全部（指定看的范围）。（投影变换）

4、我们可能希望把整个看到的图形画下来，但它只占据纸张的一部分，而不是全部（指定在显示器窗口的那个位置显示）。（视口变换）这些，都可以在 OpenGL 中实现。

从“相对移动”的观点来看，改变观察点的位置与方向和改变物体本身的位置与方向具有等效性。在 OpenGL 中，实现这两种功能甚至使用的是同样的函数。

由于模型和视图的变换都通过矩阵运算来实现，在进行变换前，应先设置当前操作的矩阵为“模型视图矩阵”。设置的方法是以 GL_MODELVIEW 为参数调用 glMatrixMode 函数，像这样：glMatrixMode(GL_MODELVIEW);该语句指定一个 4×4 的建模矩阵作为当前矩阵。

通常，我们需要在进行变换前把当前矩阵设置为单位矩阵。把当前矩阵设置为单位矩阵的函数为：glLoadIdentity();我们在进行矩阵操作时，有可能需要先保存某个矩阵，过一段时间再恢复它。当我们需要保存时，调用 glPushMatrix（）函数，它相当于把当前矩阵压入堆栈。当需要恢复最近一次的保存时，调用 glPopMatrix（）函数，它相当于从堆栈栈顶弹出一个矩阵为当前矩阵。

OpenGL 规定堆栈的容量至少可以容纳 32 个矩阵，某些 OpenGL 实现中，堆栈的容量实际上超过了 32 个。因此不必过于担心矩阵的容量问题。通常，用这种先保存后恢复的措施，比先变换再逆变换要更方便，更快速。注意：模型视图矩阵和投影矩阵都有相应的堆栈。使用 glMatrixMode 来指定当前操作的究竟是模型视图矩阵还是投影矩阵。在代码中，视图变换必须出现在模型变换之前，但可以在绘图之前的任何时候执行投影变换和视口变换。

1.display()程序中绘图函数潜在的重复性强调了：在指定的视图变换之前，应该使用 glLoadIdentity()函数把当前矩阵设置为单位矩阵。

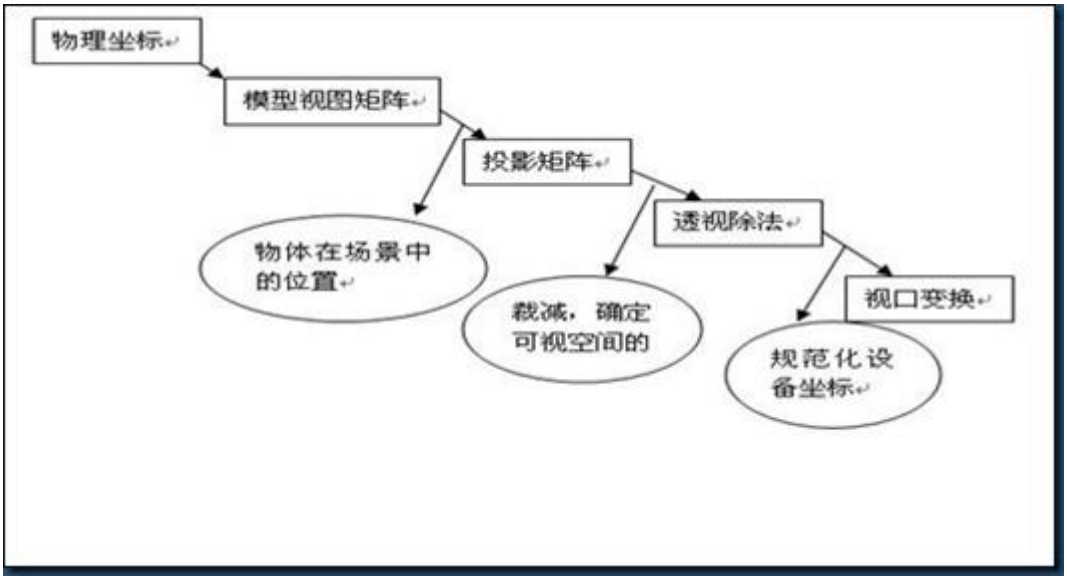
2.在载入单位矩阵之后，使用 gluLookAt()函数指定视图变换。如果程序没有调用 gluLookAt()，那么照相机会设定为一个默认的位置和方向。在默认的情况下，照相机位于原点，指向 Z 轴负方向，朝上向量为(0,1,0)。

3.一般而言，display()函数包括：视图变换 + 模型变换 + 绘制图形的函数(如 glutWireCube())。display()会在窗口被移动或者原来先遮住这个窗口的东西被一开时，被重复调用，并经过适当变换，保证绘制的图形是按照希望的方式进行绘制。

4.在调用 glFrustum()设置投影变换之前，在 reshape()函数中有一些准备工作：视口变换 + 投影变换 + 模型视图变换。由于投影变换，视口变换共同决定了场景是如何映射到计算机的屏幕上的，而且它们都与屏幕的宽度，高度密切相关，因此应该放在 reshape()中。reshape()会在窗口初次创建，移动或改变时被调用。

OpenGL 中矩阵坐标之间的关系

物理坐标*模型视图矩阵*投影矩阵*透视除法*规范化设备坐标——〉窗口坐标



（1）视图变换函数 gluLookAt(0.0，0.0，5.0，0.0，0.0，0.0，0.0，1.0，0.0，)设置照相机的位置

把照相机放在（0，0，5），镜头瞄准（0，0，0），朝上向量定为（0，1，0）朝上向量为照相机指定了一个唯一的方向。如果没有调用 gluLookAt，照相机就设定一个默认的位置和方向，在默认情况下，照相机位于原点，指向 Z 轴的负方向，朝上向量为（0，1，0）

glLoadIdentity()函数把当前矩阵设置为单位矩阵。

（2）使用模型变换的目的是设置模型的位置和方向

（3）投影变换，指定投影变换类似于为照相机选择镜头，可以认为这种变换的目的是确定视野，并因此确定哪些物体位于视野之内以及他们能够被看到的程度。

除了考虑视野之外，投影变换确定物体如何投影到屏幕上，OpenGL 提供了两种基本类型的投影，1、透视投影：远大近小；2、正投影：不影响相对大小，一般用于建筑和 CAD 应用程序中

（4）视口变换

视口变换指定一个图象在屏幕上所占的区域

（5）绘制场景

4．示范代码： 太阳系

```
#include <GL/glut.h>
 #include <stdlib.h>
static int year = 0, day = 0;
void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_FLAT);
}
void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glPushMatrix();
   glutWireSphere(1.0, 20, 16); /* draw sun */
   glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);
   glTranslatef (2.0, 0.0, 0.0);
   glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
   glutWireSphere(0.2, 10, 8); /* draw smaller planet */
   glPopMatrix();
   glutSwapBuffers();
}
void reshape (int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
```

```
    glLoadIdentity ();
    gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}
void keyboard (unsigned char key, int x, int y)
{
    switch (key) {
        case 'd':
            day = (day + 10) % 360;
            glutPostRedisplay();
            break;
        case 'D':
            day = (day - 10) % 360;
            glutPostRedisplay();
            break;
        case 'y':
            year = (year + 5) % 360;
            glutPostRedisplay();
            break;
        case 'Y':
            year = (year - 5) % 360;
            glutPostRedisplay();
            break;
        case 27:
            exit(0);
            break;
        default:
             break;
```

```
    }
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv[0]);
    init ();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

代码说明：

上面所描述的这个程序绘制一个简单的太阳系，其中有一颗行星和一颗太阳，它们是用同一个球体绘制函数绘制的。为了编写这个程序，需要使用 glRtate*()函数让这颗行星绕太阳旋转，并且绕自身的轴旋转。还需要使用 glTranslate*()函数让这颗行星远离太阳系原点，移动到它自己的轨道上。记住，可以在 glutWireSphere()函数中使用适当的参数，在绘制两个球体时指定球体的大小。

为了绘制这个太阳系，首先需要设置一个投影变换和一个视图变换。在这个例子中，可以使用 glutPerspective() 和 gluLookat().

绘制太阳比较简单，因为它应该位于全局固定坐标系统的原点，也就是球体函数进行绘图的位置。因此，绘制太阳时并不需要移动，可以使用 glRotate*()函数绕一个任意的轴旋转。绘制一颗绕太阳旋转的行星要求进行几次模型变换。这颗行星需要每天绕自己的轴旋转一周，每年沿着自己的轨道绕太阳旋转一周。

为了确定模型变换的顺序，可以从局部坐标系统的角度考虑。首先，调用初始的 glRotate*()函数对局部坐标系统进行旋转，这个局部坐标系统最初与全局固定坐标系统是一致的。接着，可以调用 glTranslate*()把局部坐标系统移动到行星轨道上的一个位置。移动的距离应该等于轨道的半径。因此，第一个 glRotate*()函数实际上确定了这颗行星从什么地方开始绕太阳旋转（或者说，从一年的什么时候开始）。

第二次调用 glRotate*()使局部坐标轴进行旋转，因此确定了这颗行星在一天中的时间。当调用了这些函数变换之后，就可以绘制这颗行星了。

5. 实验作业：

（1）验证并实现上述代码。

实验扩展：

（2）尝试在太阳系中增加一颗卫星，一颗行星。提示：使用 glPushMatrix()和 glPopMatrix()在适当的时候保存和恢复坐标系统的位置。如果打算绘制几颗卫星绕同一颗行星旋转，需要在移动每颗卫星的位置之前保存坐标系统，并在绘制每颗卫星之后恢复坐标系统。尝试把行星的轴倾斜。