# 实验3 二维图形裁剪与三维变换观察

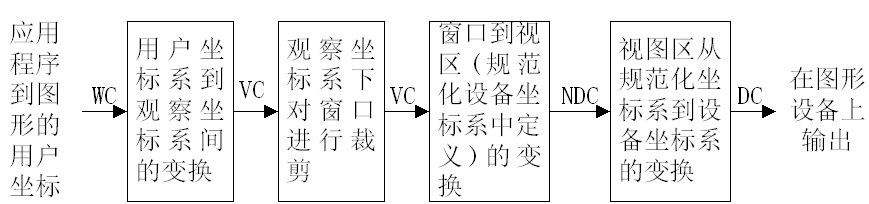
## 一、实验目的与要求：

1. 掌握二维观察基本流程；
2. 掌握二维点、线、多边形的裁剪算法；
3. 掌握三维图形基本几何变换、投影变换、观察变换；
4. 熟悉三维观察基本流程；
5. 掌握OpenGL三维图形变换。

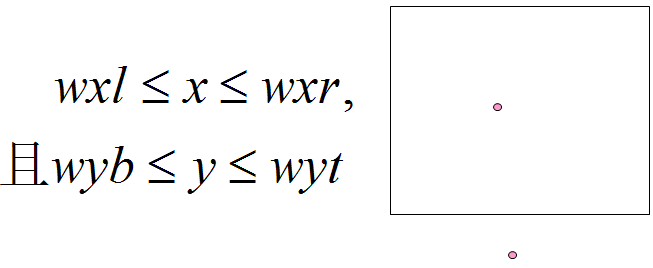
## 二、实验内容与步骤：

### 1 二维观察与窗口裁剪

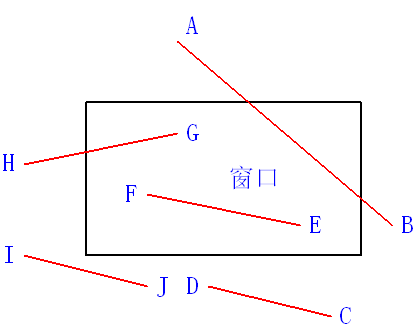
#### 1.1 二维观察基本流程



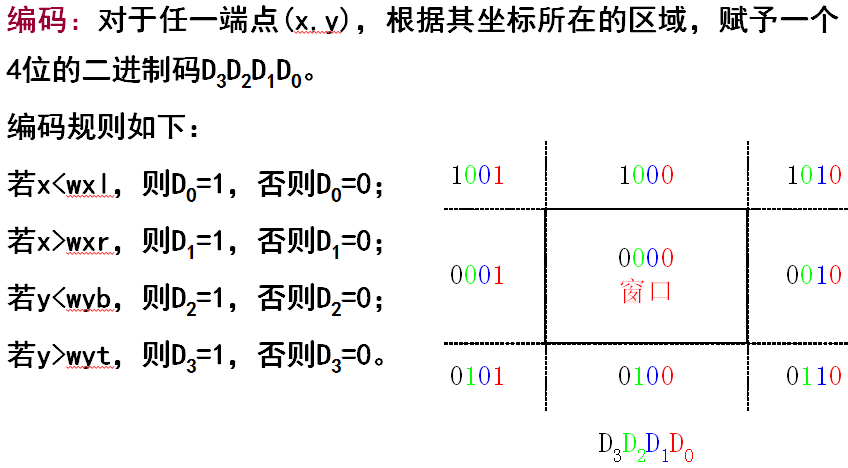
#### 1.2 点的裁剪



#### 1.3 直线段的裁剪



***1）Cohen-Sutherland算法 编码裁剪算法***



算法的步骤：

①输入直线段的两端点坐标：p1(x1,y1)、p2(x2,y2)，以及窗口的四条边界坐标：wyt、wyb、wxl和wxr。

②对p1、p2进行编码：点p1的编码为code1，点p2的编码为code2。

③若code1|code2=0，简取之，转⑥；否则，若code1&code2≠0，简弃之，转⑦；当上述两条均不满足时，进行步骤④ 。

④确保p1在窗口外部：若p1在窗口内，则交换p1和p2的坐标值和编码。

⑤按左、右、下、上的顺序求出直线段与窗口边界的交点，并用该交点的坐标值替换p1的坐标值。去掉p1s这一段。转②。

⑥画当前的直线段p1p2。

⑦算法结束。

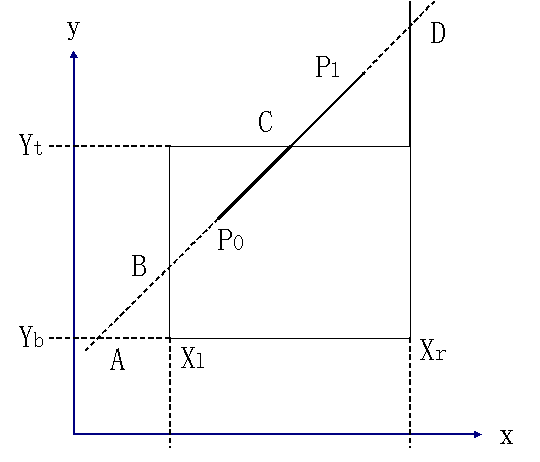
***2）中点分割算法***

①若code1|code2=0，对直线段应简取之，结束；否则，若code1&code2≠0，对直线段可简弃之，结束；当这两条均不满足时，进行步骤② 。

②找出该直线段离窗口边界最远的点和该直线段的中点。判中点是否在窗口内：若不在，则把中点和离窗口边界最远点构成的线段丢掉，以线段上的另一点和该中点再构成线段求其中点；如中点在窗口内，则又以中点和最远点构成线段，并求其中点，直到中点接近窗口边界，则该中点就是该线段落在窗口内的一个端点坐标。

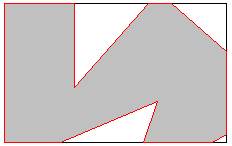
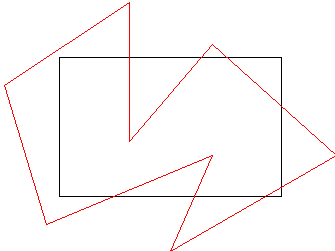
③如另一点在窗口内，则经②即确定了该线段在窗口内的部分。如另一点不在窗口内，则该点和所求出的在窗口上的那一点构成一条线段，重复步骤②，即可求出落在窗口内的另一点。

***3）梁友栋-Barsky算法***



设要裁剪的线段是P0P1。 P0P1和窗口边界交于A,B,C,D四点。算法的基本思想是:从A,B和P0三点中找出最靠近P1的点(P0)。从C,D和P1中找出最靠近P0的点(C)。那么P0C就是P0P1线段上的可见部分。

#### 1.4 多边形的裁剪



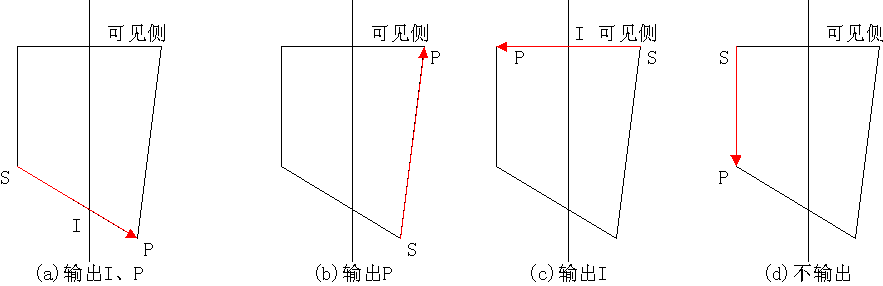
***1）Sutherland-Hodgeman多边形裁剪***

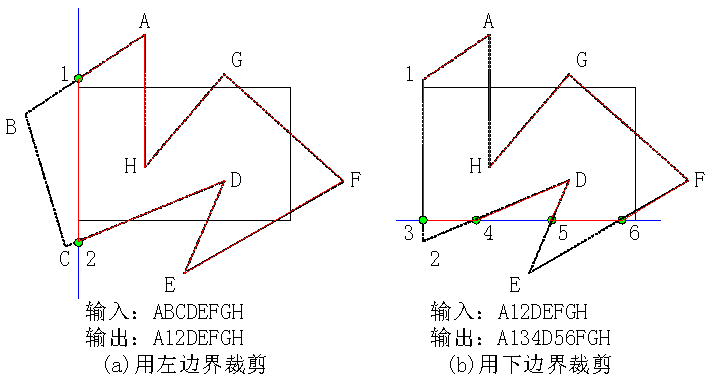
***逐边裁剪算法*基本思想**：每次用窗口的一条边对多边形进行裁剪。

**算法实施策略：**

**•为窗口各边界裁剪的多边形存储输入与输出顶点表。在窗口的一条裁剪边界处理完所有顶点后，其输出顶点表将用窗口的下一条边界继续裁剪。**

**•窗口的一条边以及延长线构成的裁剪线把平面分为两个区域，包含有窗口区域的一个域称为可见侧；不包含窗口区域的域为不可见侧。**





***2）Weiler-Atherton多边形裁剪***

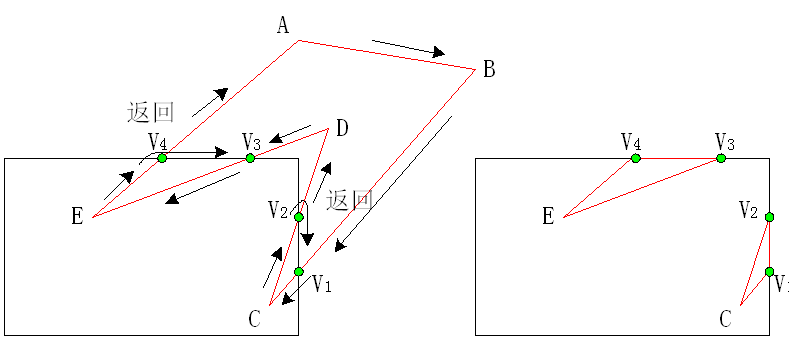
***双边裁剪算法***

***假定按顺时针方向处理顶点，且将用户多边形定义为Ps，窗口矩形为Pw。算法从Ps的任一点出发，跟踪检测Ps的每一条边，当Ps与Pw相交时（实交点），按如下规则处理：***

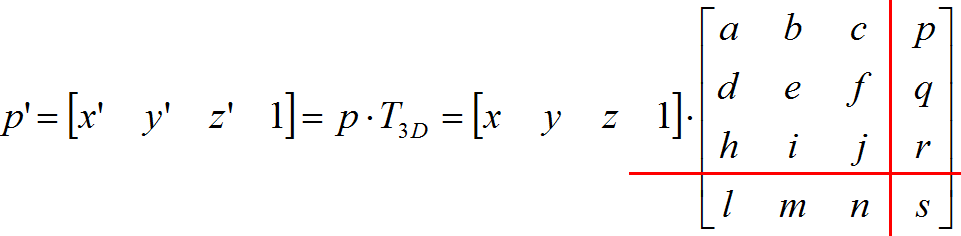
**(1)若是由不可见侧进入可见侧，则输出可见直线段，转(3)；**

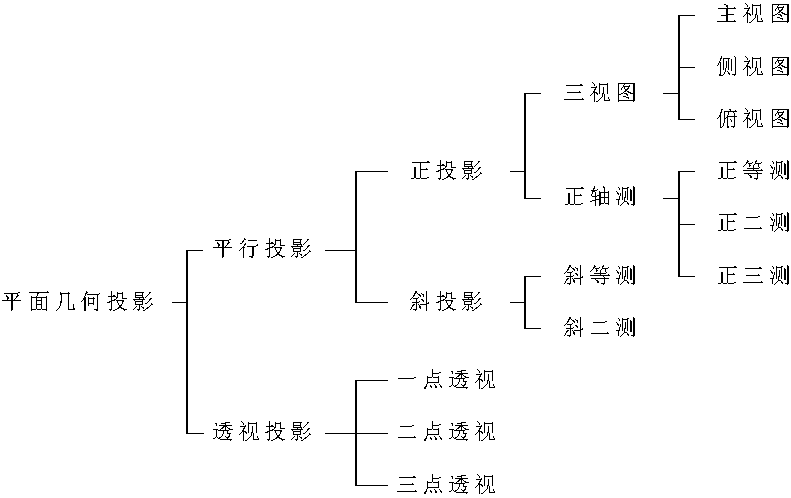
**(2)若是由可见侧进入不可见侧，则从当前交点开始，沿窗口边界顺时针检测Pw的边，即用窗口的有效边界去裁剪Ps的边，找到Ps与Pw最靠近当前交点的另一交点，输出可见直线段和由当前交点到另一交点之间窗口边界上的线段，然后返回处理的当前交点；**

**(3)沿着Ps处理各条边，直到处理完Ps的每一条边，回到起点为止。**

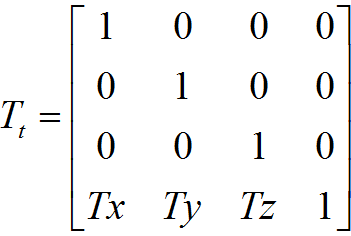


### 2 三维图形的几何变换、投影变换





#### 2.1 平移变换

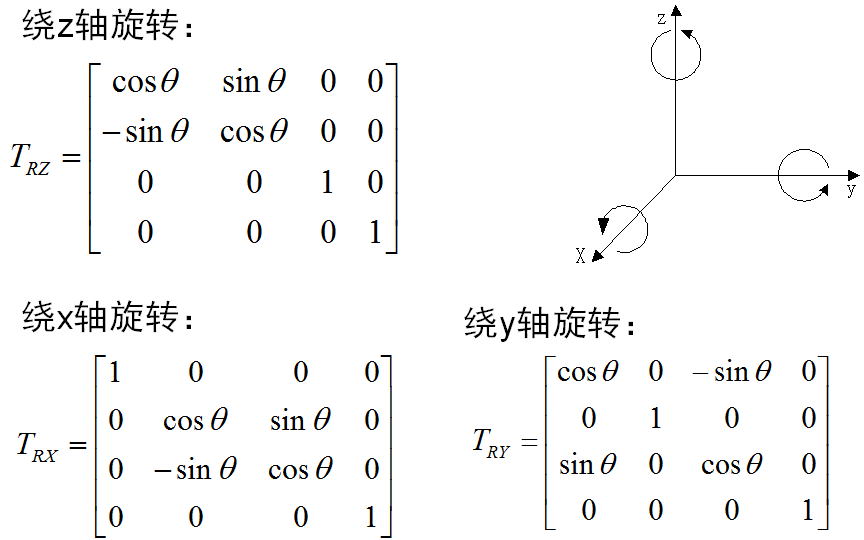


#### 2.2 比例变换

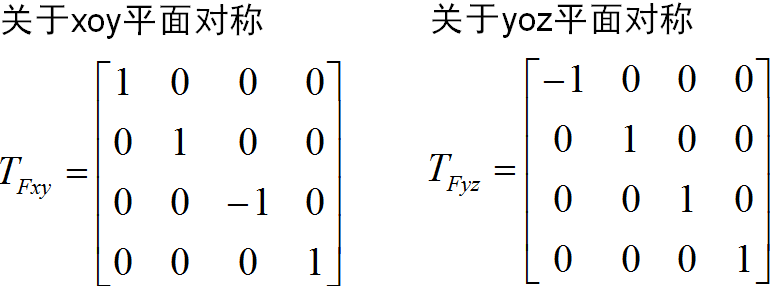
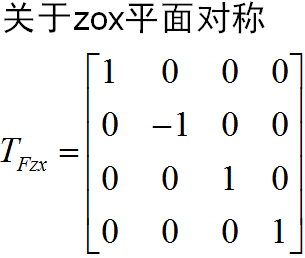
局部比例变换：

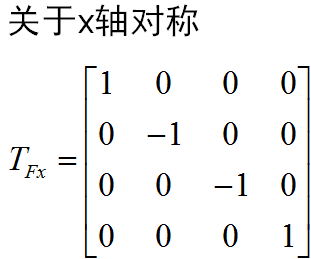
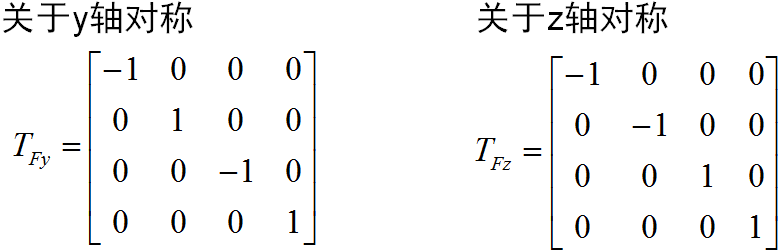
整体比例变换：

#### 2.3 旋转变换

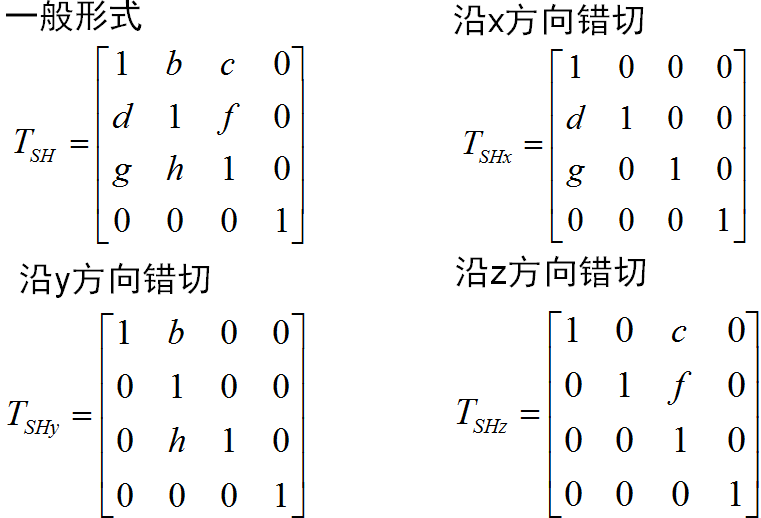


#### 2.4 对称变换

#### 2.5 错切变换



#### 2.6 相对任意参考点的复合变换

相对于参考点F(xf,yf,zf)作比例、旋转、错切等变换的过程分为以下三步：

(1)将参考点F移至坐标原点

(2)针对原点进行三维几何变换

(3)进行反平移

#### 2.7 相对任意方向的复合变换

针对任意方向轴的变换的五个步骤：



①使任意方向轴的起点与坐标原点重合，此时进行平移变换。

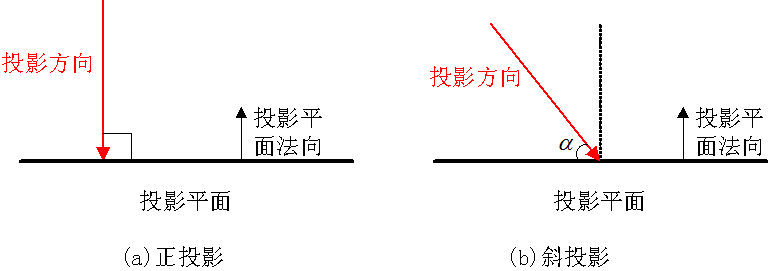
②使方向轴与某一坐标轴重合，此时需进行旋转变换，且旋转变换可能不止一次。

③针对该坐标轴完成变换。

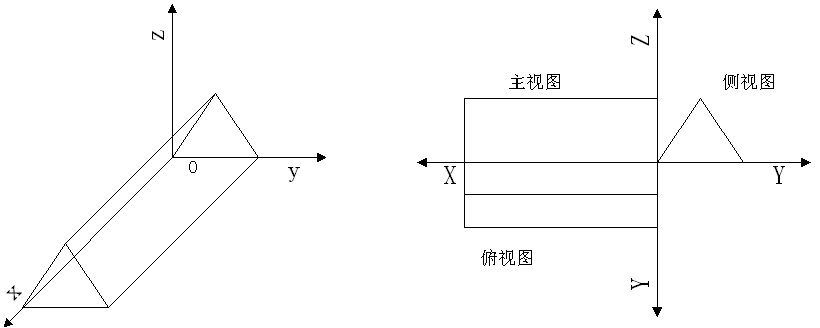
④用逆旋转变换使方向轴回到其原始方向。

⑤用逆平移变换使方向轴回到其原始位置。

#### 2.8 平行投影



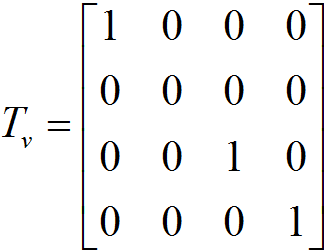
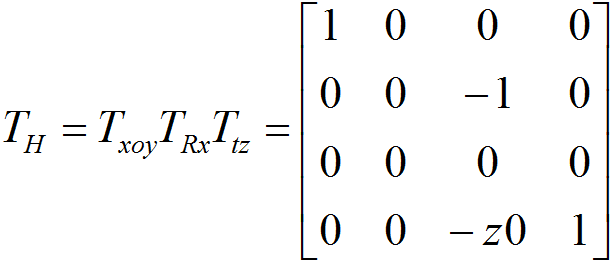
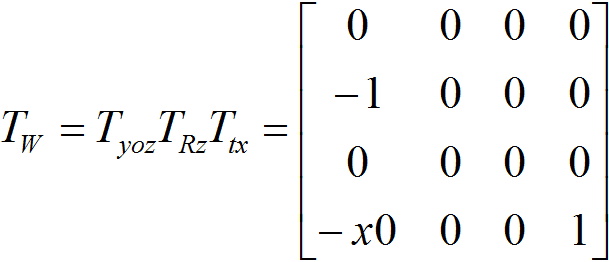
**1）正投影——三视图：**



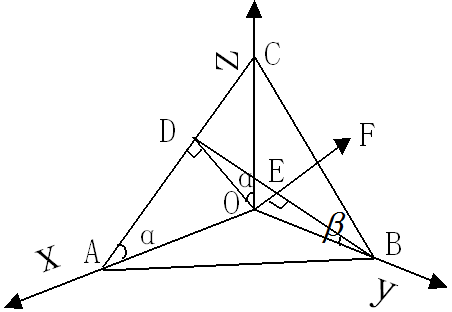
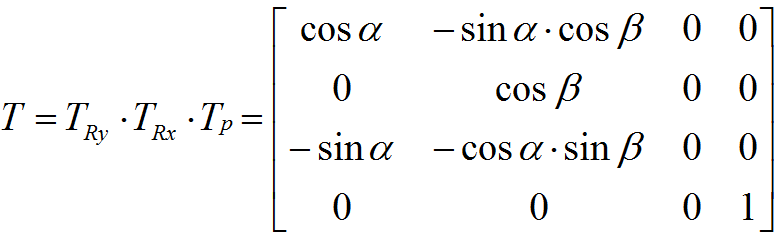
主视图：

俯视图：

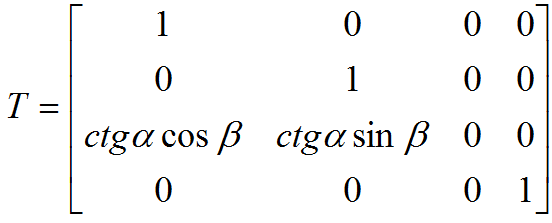
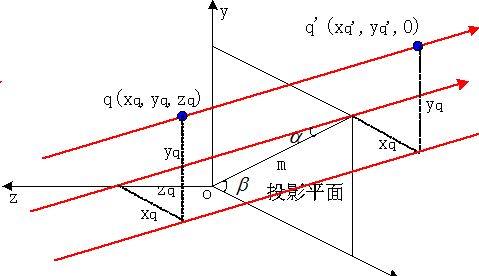
侧视图：

**2）正投影——正轴测（正等测、正二测、正三测）：**

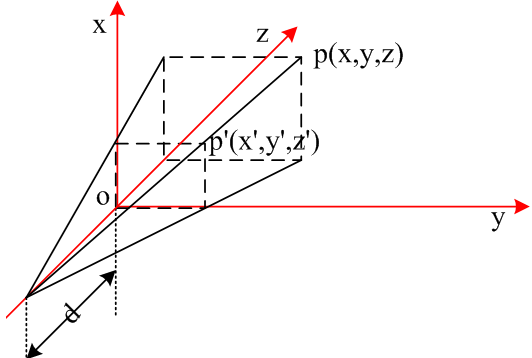
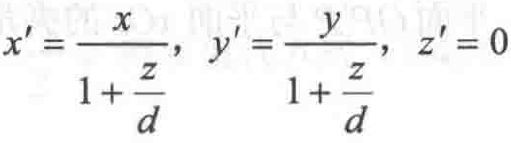
 

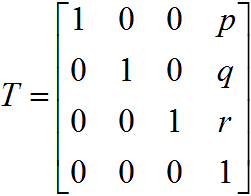
**3）斜投影——斜等测、斜二测：**



#### 2.9 透视投影

假定视点（投影中心）在Z轴上(z=-d处)，画面（投影面）与xOy面上：

 🡺 

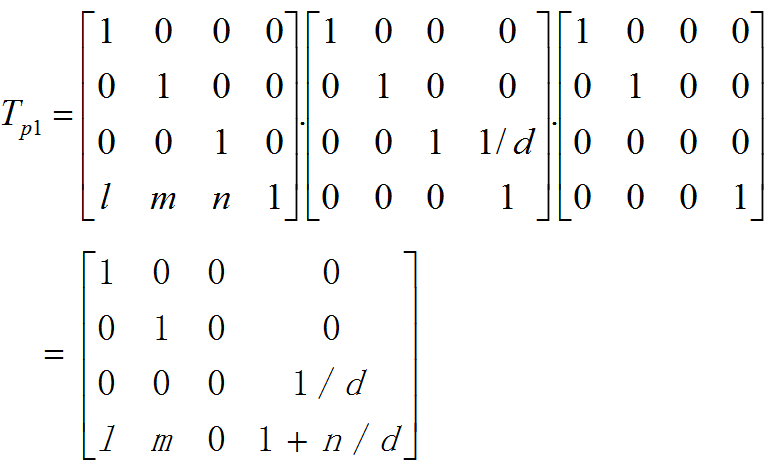


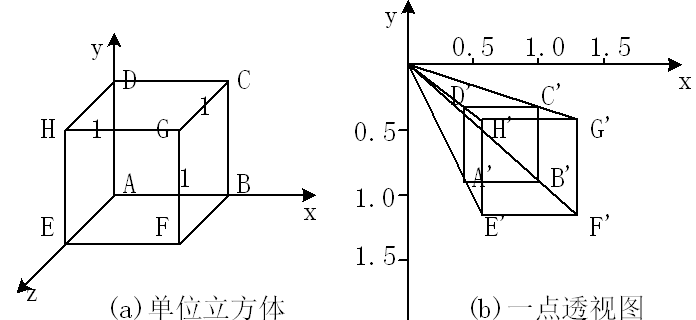
*一点透视的步骤：*

(1)将三维形体平移到适当位置l、m、n；

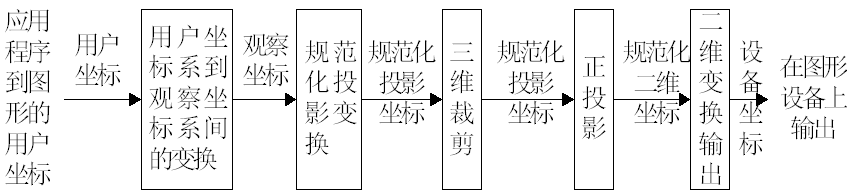
(2)令视点在z轴，进行透视变换；

(3)最后，为了绘制的方便，向xoy平面作正投影变换，将结果变换到xoy平面上。





### 3 三维观察基本流程



### 4 OpenGL基本变换

* 视图变换：指定观察者或摄影机的位置；
* 模型变换：在场景中移动对象；
* 模型视图变换：描述视图变换与模型变换的对偶性；
* 投影变换：对视见空间进行修剪和改变大小；
* 视见区变换：对窗口的最终输出进行缩放；

#### 4.1 OpenGL视图变换

* 视图变换函数（定义观察坐标系）

void gluLookAt（GLdouble eyex，GLdouble eyey，GLdouble eyez，GLdouble centerx，GLdouble centery，GLdouble centerz，GLdouble upx，GLdouble upy，GLdouble upz）；

#### 4.2 OpenGL模型变换

* 平移

void glTranslated(f)(GLdouble x，GLdouble y，GLdouble z)；

* 旋转

void glRotated(f)(GLdouble angle，GLdouble x，GLdouble y，GLdouble z )；

* 比例

void glScaled(f)(GLdouble x，GLdouble y，GLdouble z)；

#### 4.3 OpenGL投影变换

* 平行投影：

视景体是一个矩形的平行管道，也就是一个长方体，其特点是无论物体距离相机多远，投影后的物体大小尺寸不变。

void glOrtho (GLdouble left，GLdouble right，GLdouble bottom，GLdouble top，GLdouble near，GLdouble far)；

gluOrtho2D(GLdouble left，GLdouble right，GLdouble bottom，GLdouble top);

主要用于二维图像到二维屏幕上的投影，相当于glOrtho函数中的near和far默认设置为-1和1，所有二维物体的Z坐标都为0.0

* 透视投影

**void glFrustum (GLdouble left,GLdouble Right, GLdouble bottom,GLdouble top,GLdouble near, GLdouble far);**

此函数创建一个透视投影矩阵，并且用这个矩阵乘以当前矩阵。它的参数只定义近裁剪平面的左下角点和右上角点的三维空间坐标，即（left，bottom，-near）和（right，top，-near）；最后一个参数far是远裁剪平面的Z负值，其左下角点和右上角点空间坐标由函数根据透视投影原理自动生成。

**void gluPerspective (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);**

它也创建一个对称透视视景体，但它的参数定义于前面的不同，其操作是创建一个对称的透视投影矩阵，并且用这个矩阵乘以当前矩阵。参数fovy定义视野在X-Z平面（垂直方向上的可见区域）的角度，范围是[0.0, 180.0]；参数aspect是投影平面的纵横比（宽度与高度的比值）；参数zNear和Far分别是远近裁剪面沿Z负轴到视点的距离。



#### 4.4 视口变换

glViewport(GLint x, GLint y，GLsizei width, GLsizei height);

x和y指定了对应于屏幕上显示窗口中的矩形视区的左下角坐标，单位为像素。整型值width和height则指定了视区的宽度和高度。默认的视区大小和位置与显示窗口保持一致

#### 4.5 变换一般操作

* 切换变换矩阵模式：

glMatrixMode(GLenum mode)；

参数mode用于确定将哪个矩阵堆栈用于矩阵操作。

GL\_MODELVIEW： 模型视图矩阵堆栈

GL\_PROJECTION：投影矩阵堆栈

例如：glMatrixMode(GL\_PROJECTION)；

glLoadIdentity()；

* 变换矩阵（double mm[16]; 列优先，左乘矩阵）的入栈和出栈：

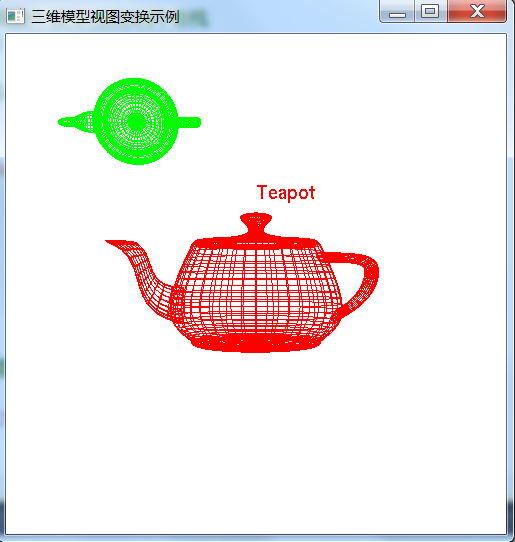
void glPushMatrix(void);

……

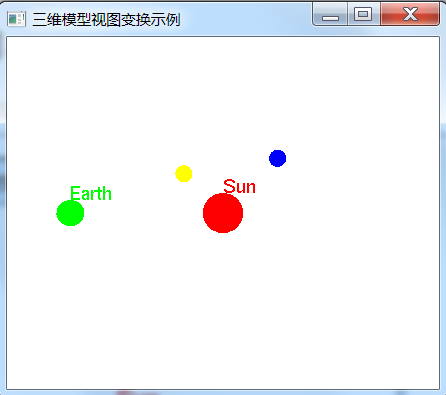
void glPopMatrix(void);

### 5 OpenGL实例程序学习

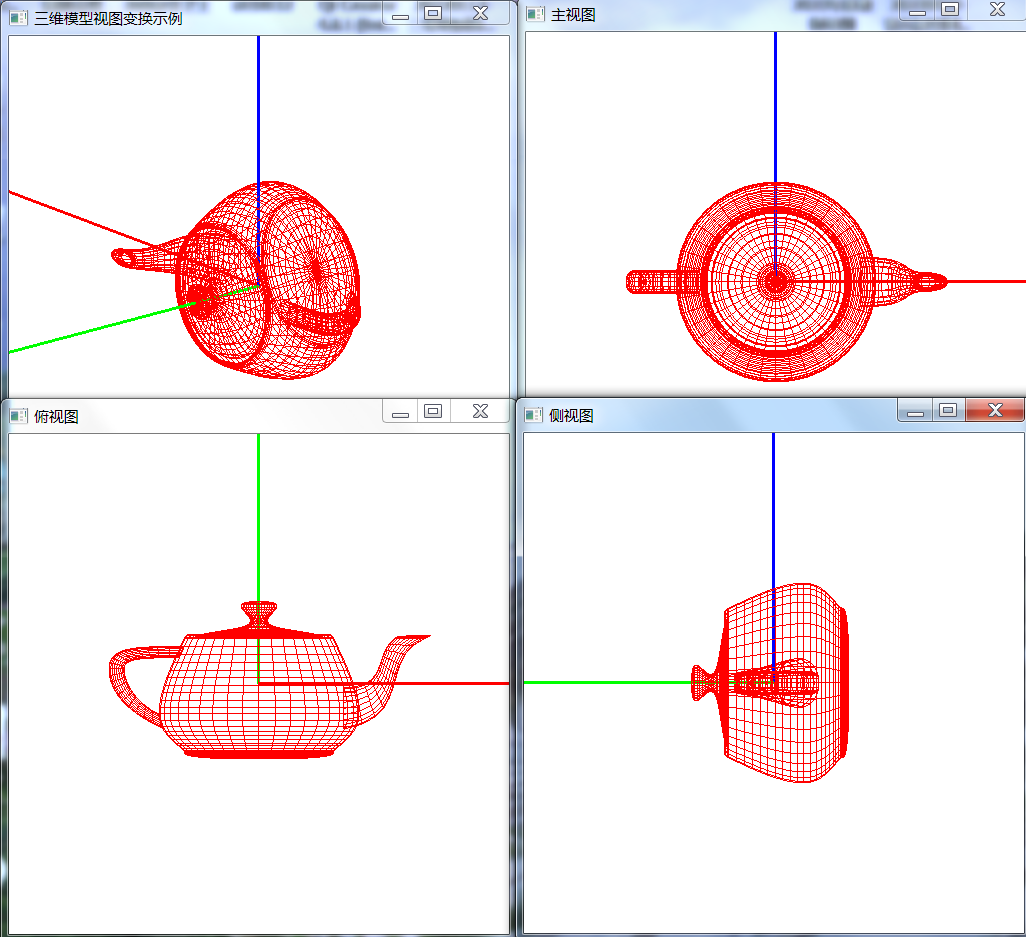
三维图形的视图变换【框架源码详见】，具体效果如下：



三维图形的模型变换【框架源码详见】，具体效果如下：



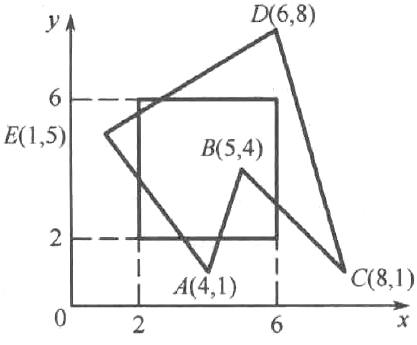
OpenGL三视图绘制【框架源码详见】，具体效果如下：



## 三、实验练习：

###### 1 编程实现直线段的Cohen-Sutherland编码裁剪算法、中点裁剪算法、梁友栋裁剪算法，并推广到任意折线的窗口裁剪；

###### 2 编程实现多边形的SH逐边裁剪算法、WA双边裁剪算法。验证p172-T6.15



###### 3 完善3D矩阵运算，实现模型的三维几何变换；

###### 4 掌握OpenGL三维图形的显示程序；

###### 5 基于OpenGL编程实现三视图的绘制；

###### 6 验证教材中的OpenGL例子。

## 四、提交报告：无