第 3 章 变换与裁剪

CHAPTER 3

变换和裁剪是计算机图形学基本的操作。变换包括造型变换和取景变换，前者通过平移、旋转、缩放等手段将物体置于场景空间中的给定位置，并获得所需的形状；后者将场景中的景物变换到当前视点为原点、视线方向为 z 轴的摄像机坐标系中。裁剪又分为二维裁剪和三维裁剪，通常指去除位于显示窗口之外不可见的场景部分，以提高场景绘制的效率。

3.1 二维变换

二维图形变换主要有 3 类，即平移、旋转和缩放。将一个二维点 $P(x,y)$ 平移 (t_x,t_y) 后得到新点 $P'(x',y')$，其计算公式为：

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.1}$$

类似地，将 $P(x,y)$ 绕坐标原点按逆时针方向旋转 θ 角，记旋转后的新位置为 $P'(x',y')$，则：

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.2}$$

同样，经过缩放变换后，$P(x,y)$ 的新位置为：

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.3}$$

其中，s_x 和 s_y 分别为 x 和 y 分量的缩放量。为叙述方便，通常记上述平移、旋转和缩放矩阵为 $\boldsymbol{T}$、$\boldsymbol{R}$ 和 $\boldsymbol{S}$。

对物体进行上述 3 种变换时，变换次序是非常重要的。不同的变换次序会得到不同的变换结果。虽然上述 3 种变换不具有交换性，但是它们具有结合性。记 $\boldsymbol{A}$、$\boldsymbol{B}$ 和 $\boldsymbol{C}$ 为 3 个二维变换，结合性可以表示为：

$$\boldsymbol{ABC} = (\boldsymbol{AB})\boldsymbol{C} = \boldsymbol{A}(\boldsymbol{BC}) \tag{3.4}$$

在上述 3 个变换矩阵的基础上，通过变换的组合可以得到很多特殊的二维复合变换，如对称、剪切、刚体变换、仿射变换等。

3.2　三维变换

典型的三维变换流程如图 3.1 所示。场景造型在场景坐标系中实现，这里场景坐标系包括局部坐标系和世界坐标系。通常复杂的物体可以分解为若干形状规则的构形单元的组合，这些构形单元在它们各自的局部坐标系中具有简单的几何表示形式。这里局部坐标系可以是多层次的。然后通过造型变换将定义在不同局部坐标系中的景物变换到世界坐标系中组成整个场景。上述过程在场景造型阶段完成。场景造型完成后，用户指定照相机的位置、朝向，建立一个视点坐标系，然后通过取景变换将定义在世界坐标系中的场景变换到视点坐标系中。在视点坐标系中，需要根据当前视域和视线方向对场景进行裁剪和背面剔除。完成上述处理后，位于视点坐标系的场景被投影到视窗内部，这里视窗通常是一个二维矩形区域。再通过设备变换将视窗中的物体坐标变换到标准设备坐标，最后通过一个简单的二维视窗变换将设备坐标变换到以像素为单位的屏幕坐标系中。上述各种坐标系的示意图如图 3.2 所示。

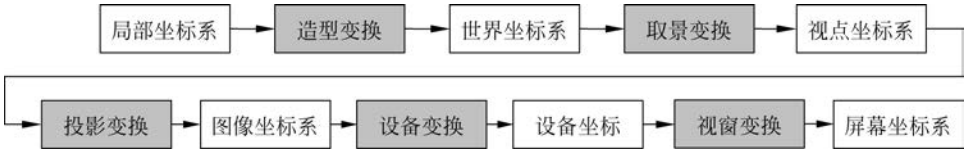


图 3.1　三维变换的流程图

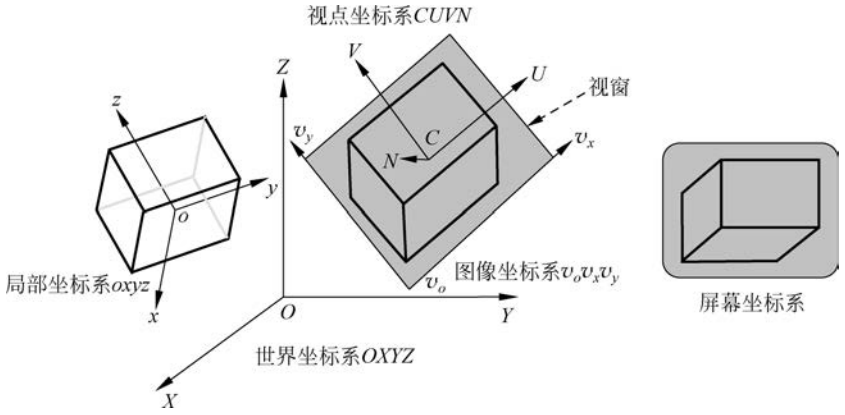


图 3.2　三维变换中的各种坐标系

3.2.1　场景坐标系和造型变换

要构造一个场景，首先需要建立一个世界坐标系，场景中的物体的形状和相对方位由它们在世界坐标系中的坐标所确定。为造型方便，物体的形状通常在各自的局部坐标系中定义。例如一个圆柱体，如果取它的轴线方向和某个坐标轴重合，那么它的定义就会变得十分简单。在具体应用中，可以首先在局部坐标系中定义一个标准圆柱体，然后通过三维几何变换得到所需要的位置和形状。显然，场景坐标系中的造型变换均为三维几何变换。

与二维情形相似，基本的三维几何变换为平移、旋转和缩放。为了统一起见，采用空间齐次坐标表示变换。记 $\boldsymbol{T}$ 为平移变换，平移量为(t_x,t_y,t_z)，假设 $P(x,y,z)$为一空间点，$P(x',y',z')$为经过三维变换后的对应点，那么对应的平移变换可以表示为：

$$\begin{pmatrix}x'\\y'\\z'\\1\end{pmatrix}=\begin{pmatrix}1&0&0&t_x\\0&1&0&t_y\\0&0&1&t_z\\0&0&0&1\end{pmatrix}\begin{pmatrix}x\\y\\z\\1\end{pmatrix} \tag{3.5}$$

记 $\boldsymbol{S}$ 为缩放变换，缩放因子为(s_x,s_y,s_z)，那么对应的变换可表示为：

$$\begin{pmatrix}x'\\y'\\z'\\1\end{pmatrix}=\begin{pmatrix}s_x&0&0&0\\0&s_y&0&0\\0&0&s_z&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}x\\y\\z\\1\end{pmatrix} \tag{3.6}$$

三维旋转变换则比二维情形稍微复杂一些，这是因为三维的旋转可以有多种情形，典型的为绕 3 个坐标轴的旋转和绕任意轴的旋转。若记 $\boldsymbol{R}_x$ 为绕 x 轴按逆时针方向旋转 θ 角的变换，其对应的变换矩阵为：

$$\begin{pmatrix}x'\\y'\\z'\\1\end{pmatrix}=\begin{pmatrix}1&0&0&0\\0&\cos\theta&-\sin\theta&0\\0&\sin\theta&\cos\theta&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}x\\y\\z\\1\end{pmatrix} \tag{3.7}$$

同样地，绕 y 轴和 z 轴旋转 θ 角的变换分别为：

$$\begin{pmatrix}x'\\y'\\z'\\1\end{pmatrix}=\begin{pmatrix}\cos\theta&0&\sin\theta&0\\0&1&0&0\\-\sin\theta&0&\cos\theta&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}x\\y\\z\\1\end{pmatrix} \tag{3.8}$$

$$\begin{pmatrix}x'\\y'\\z'\\1\end{pmatrix}=\begin{pmatrix}\cos\theta&-\sin\theta&0&0\\\sin\theta&\cos\theta&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}x\\y\\z\\1\end{pmatrix} \tag{3.9}$$

要实现绕任意轴的旋转，可以首先对坐标系进行变换，使得旋转轴和变换后坐标系的某个坐标轴重合，然后调用上述基本旋转变换。上述三维几何变换都是可逆的，同时它们可以通过组合的方式实现更为复杂的变换。

3.2.2 视点坐标系和取景变换

视点坐标系是以观察者的当前位置为原点、视线方向为 z 轴的坐标系，它类似于照相时所采用的坐标系，如图 3.2 所示。用户首先需要在场景坐标系中指定当前视点位置 C，作为坐标系的原点，C 可理解为照相机的位置。然后，用户指定视线方向单位矢量 $\boldsymbol{N}$，$\boldsymbol{N}$ 即照相机镜头所面向的方向。有了照相机的位置和朝向，照相机所拍摄的画面仍不能最后确定，用户还需指定一个向上的向量 **UP**，它的作用是确定相机拍照时画面的上下方向。可以通过

它和 $\boldsymbol{N}$ 确定坐标系的另一个方向 $\boldsymbol{V}$：

$$\boldsymbol{V}=\frac{\boldsymbol{N}\times\mathbf{UP}}{\|\boldsymbol{N}\times\mathbf{UP}\|} \tag{3.10}$$

最后，由 $\boldsymbol{N}$ 和 $\boldsymbol{V}$ 两个垂直向量确定一个坐标轴方向 $\boldsymbol{U}$：

$$\boldsymbol{U}=\boldsymbol{V}\times\boldsymbol{N} \tag{3.11}$$

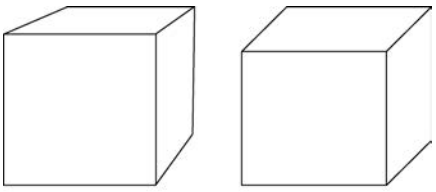
由上述坐标原点 C 以及 3 个相互垂直的单位矢量($\boldsymbol{U},\boldsymbol{V},\boldsymbol{N}$)共同组成了视点坐标系，$C=(C_x,C_y,C_z)$、$\boldsymbol{U}=(U_x,U_y,U_z)$、$\boldsymbol{V}=(V_x,V_y,V_z)$和 $\boldsymbol{N}=(N_x,N_y,N_z)$，那么由场景坐标系到视点坐标系的变换可以表示为：

$$\begin{pmatrix}u\\v\\n\end{pmatrix}=\begin{pmatrix}U_x & U_y & U_z\\V_x & V_y & V_z\\N_x & N_y & N_z\end{pmatrix}\begin{pmatrix}x-C_x\\y-C_y\\z-C_z\end{pmatrix} \tag{3.12}$$

其中，(x,y,z)为世界坐标系中的点，(u,v,n)为(x,y,z)在视点坐标系中的对应点。式(3.12)表示的变换称为取景变换。

3.2.3　投影坐标系和投影变换

场景中的物体是三维的，而显示屏幕是二维的。要在二维的显示屏幕上显示三维的场景，需要对场景进行投影变换。常见的投影变换有两类：一类是透视投影，这类投影符合人类的视觉，产生的投影效果很真实；另一类是平行投影，平行投影中，物体的相对度量保持不变，例如两个同方向等长线段的投影结果仍然是等长的，这种投影在建筑和机械设计中十分重要。两类投影的例子如图 3.3 所示。



(a) 透视投影　　(b) 平行投影

图 3.3　单位立方体的两种投影结果

投影变换是在视点坐标系 $CUVN$ 中进行的。对于透视投影，通常取视点坐标系中的(0,0,0)为投影点，投影平面取与视线方向垂直的平面 $n=d$，如图 3.4 所示。假设在视点坐标系中某点为(u,v,n)，该点在投影面上的对应点坐标(u_p,v_p)为：

$$u_p=\frac{u}{n/d}\quad v_p=\frac{v}{n/d} \tag{3.13}$$

如果记投影后的齐次坐标为(U,V,N,W)，那么透视投影(3.13)的变换矩阵可以表示为：

$$\begin{pmatrix}U\\V\\N\\W\end{pmatrix}=\begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 1/d & 0\end{pmatrix}\begin{pmatrix}u\\v\\n\\1\end{pmatrix} \tag{3.14}$$

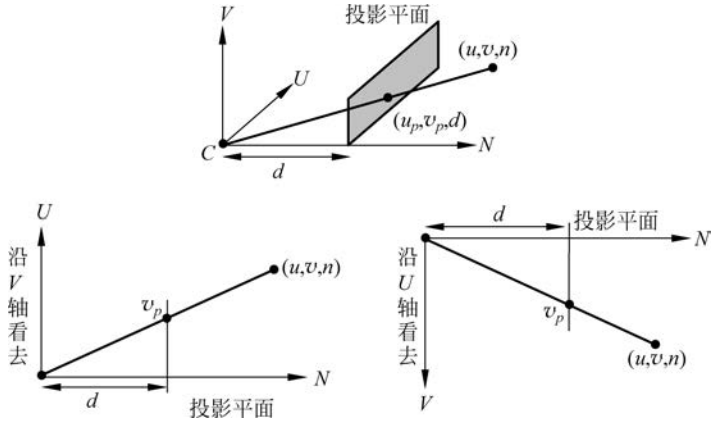


图 3.4　透视投影示意图

通过简单的运算可以得到：

$$\left(\frac{U}{W},\frac{V}{W},\frac{N}{W}\right)=\left(\frac{u}{n/d},\frac{v}{n/d},d\right)=(u_p,v_p,d) \tag{3.15}$$

与透视投影相比，平行投影比较简单。例如沿 N 轴、投影平面在 $n=0$ 的平行投影可以简单地表示为：

$$u_p=u \quad v_p=v \quad n_p=0 \tag{3.16}$$

由于显示窗口尺寸的限制，对于给定视点，场景中只有一部分景物能投影到显示窗口内构成画面。以当前视点为顶点，连接视点与显示窗口的 4 个角点的直线为棱边的棱锥体形成了场景的可见视域，称为视域锥。在视域锥中，去掉位于成像面之前的部分和视力所不及的后面部分，即为场景的有效视域，如图 3.5 所示。对于透视投影，这一有效视域是一平截头四棱锥体；对于平行投影，场景有效视域为一长方体。由于计算机图形学中常采用透视投影，我们通常将场景的有效视域称为视域四棱锥。在进行投影变换时，位于视域四棱锥外部的景物将会被剔除。如果物体的一部分位于视域四棱锥的外部、一部分位于视域四棱锥的内部，就需要将位于视域四棱锥外的物体部分剪切掉，这一操作称为裁剪。在本章的后面将介绍裁剪算法。

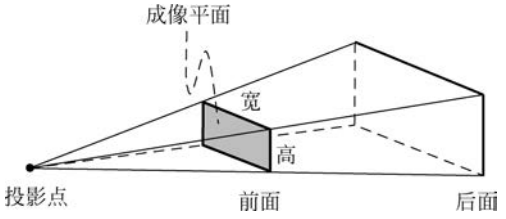


图 3.5　透视投影中的视域四棱锥

3.2.4　规格化设备坐标系和设备变换

经过前面的视域裁剪和投影变换，三维几何物体被投影到二维投影平面上称为视窗的矩形区域内，见图 3.2 中的 $v_o v_x v_y$ 坐标系中的矩形和图 3.5 中的矩形。此时，得到了物体投影后的二维齐次坐标表示。将二维齐次坐标除以最后一个坐标分量 ω，便得到了规格化设备坐标。

3.2.5　屏幕坐标系和视窗变换

为了能够在二维屏幕上显示投影结果，我们还需将定义在视窗中的投影结果转换到以像素为单位的屏幕坐标系，如图 3.2 所示。这个转换除了普通的二维变换之外，还包括投影结果的光栅化，即将连续的二维投影转换为离散的光栅表示。

3.3　裁剪

裁剪可以在投影变换之前进行(三维裁剪)，也可以在投影变换之后进行(二维裁剪)。

在二维裁剪中，裁剪窗口通常为一个矩形。根据裁剪对象的不同，可以将二维裁剪进一步分为点裁剪、线裁剪、多边形裁剪、字符裁剪等，如图 3.6～图 3.8 所示。点裁剪非常简单，仅仅是点的归类问题。线裁剪是二维裁剪中的基本方法，多边形裁剪和文本裁剪都可以由线裁剪算法导出。线裁剪的核心是高效地判断和舍弃位于显示窗口之外的线段或其一部分。最经典的二维线裁剪算法是 Cohn-Sutherland 算法，它通过对被裁剪线段的端点进行简单的编码分区和编码逻辑运算，快速地剔除一部分完全位于窗口之外的直线段。为了使二维线裁剪便于硬件实现，Sproull 和 Sutherland 又提出了中点法，线段与窗口的交点是通过对直线段递归地进行中点剖分和中点位置判定得到的。中点法的优点在于裁剪运算只有加法和除以 2 的运算，并且可以并行实现。Cyrus 和 Beck 提出了对于任意凸区域的参数化线裁剪算法，直线段首先在[0,1]区间上进行参数化，然后通过直线段和裁剪窗口的交点的参数值来进行取舍。梁友栋和 Barsky 改进了 Cyrus-Beck 参数化算法，使改进后的算法对于矩形窗口具有更高的裁剪效率。Nicholl-Lee-Nicholl 则采用更为仔细的判断方法，简单地抛弃明显位于窗口外部的直线段，从而减少了不必要的直线求交运算。当然，求交计算量的减少是以算法的复杂性为代价的。

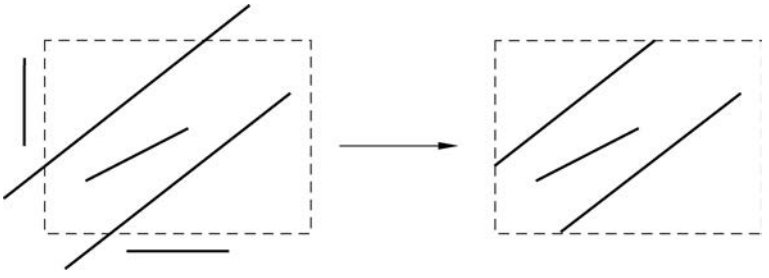


图 3.6　矩形窗口的线裁剪

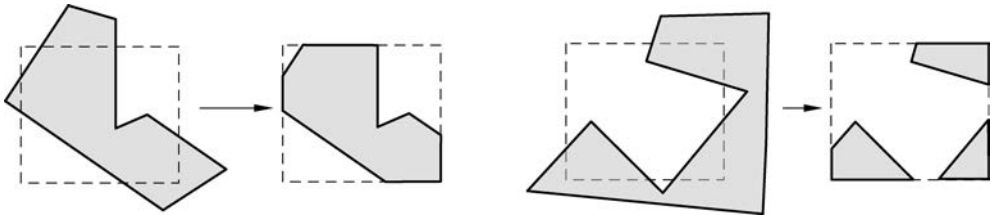


图 3.7　多边形裁剪

(a) 原始字符串　(b) 字符精度裁剪　(c) 像素精度裁剪

图 3.8　字符裁剪

与线裁剪相比，多边形裁剪则复杂一些。一个简单的做法是将多边形看作线段的集合，对于每条线段采用线裁剪算法，这样得到的结果在只显示线画图形时是可以接受的。但是当多边形需要显示成实区域时，上述方法就失效了。图 3.7 给出了两个多边形裁剪的例子，可以看出裁剪后的多边形可能取窗口的一部分作为多边形边界，裁剪后也有可能生成多个不相连的多边形。Sutherland-Hodgman 算法和 Liang-Barsky 算法是具有代表性的两个矩形裁剪窗口算法。Weiler-Atherton 算法则讨论了任意窗口的多边形裁剪。

在图形显示中的字符表示方法有两类：一类是矢量表示，另一类是点阵表示。对于矢量表示的字符，可以采用前面的多边形裁剪算法实现字符裁剪。对于点阵表示的字符，如果点阵是由软件生成的，点阵式字符的裁剪可以归结为点的裁剪问题；如果点阵式字符是由硬件生成的，裁剪就会变得比较复杂，一个简单的处理方法是：如果字符完全位于裁剪窗口内才会显示。图 3.8 为两种表示形式下字符的裁剪结果。

三维裁剪可以根据三维裁剪体的形状进行分类：对场景取平行投影时，其裁剪体是长方体；取透视投影时，其裁剪体为一个视域四棱锥。而当上述裁剪体为某些特殊的形状时，三维裁剪算法可以变得简单。对于一般的平行投影，可将裁剪体变换为如下标准裁剪体：

$$u=-1,\quad u=1,\quad v=-1,\quad v=1,\quad n=0,\quad n=1 \tag{3.17}$$

对于透视投影，其标准裁剪体通常取为：

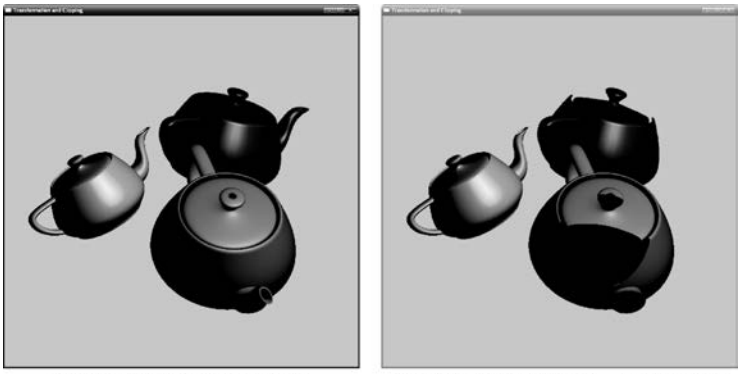
$$u=n,\quad u=-n,\quad v=n,\quad v=-n,\quad n=-n_{\min},\quad n=-1 \tag{3.18}$$

在上述两种标准裁剪体的情况下，二维线裁剪的算法，如 Cohen-Sutherland 裁剪、中点裁剪、Cyrus-Beck 裁剪、Liang-Barsky 裁剪都可以直接推广到三维情形。

3.4　变换与裁剪的实例

在本书的网上参考材料中，Ch03_Transformation 给出了基于 OpenGL 实现的三维变换与裁剪的实例。在实例中，我们通过对单位化的茶壶模型进行几何变换，得到了大小和空间方位不同的 3 个茶壶，对应的视域四棱锥的前裁剪平面和后裁剪平面的 z 值分别为 -30 和 30，如图 3.9(a)所示。通过设置视域四棱锥的不同大小，可以实现对场景中物体的三维裁剪，如图 3.9(b)所示，其中视域四棱锥的前裁剪平面和后裁剪平面的 z 值分别为 -20 和 20。由于茶壶的尺寸超出了视域四棱锥的范围，因此超出部分被视域四棱锥裁剪掉了。

在函数中，reshape()、glViewport(0,0,(GLsizei)w,(GLsizei)h)在投影平面上定义了一个视窗，位于视窗外的场景将被裁剪掉。为了使物体显示时不产生变形，通常取视窗与窗口的大小成比例。glMatrixMode(GL_PROJECTION)表示建立投影矩阵堆栈，也就是在设定视点坐标系的同时，将场景的物体投影到 $z=0$ 的平面上。glOrtho()函数的作用有两个：一方面，通过该函数建立起一个平行投影的视点坐标系；另一方面，它的内参数指定了如



(a) 3个经几何变换后的茶壶　　(b) 被视域四棱锥裁掉一部分的茶壶

图 3.9　变换与裁剪

图 3.5 所示的视域四棱锥。本例通过取不同的 ZVALUE 值，实现对场景中物体的完全显示或者三维裁剪。

renderTeapot()函数实现了对单位化茶壶的三维几何变换。在 OpenGL 中，几何变换和视点变换统一以一个矩阵堆栈表示，即 glMatrixMode(GL_MODELVIEW)。由于几何变换序列采用堆栈表示，因此前面的变换语句是最后执行的。例如在下述源程序中：

```
glTranslatef(x, y, z);
glRotatef(rotx, 1.0, 0.0, 0.0);
glRotatef(roty, 0.0, 1.0, 0.0);
glRotatef(rotz, 0.0, 0.0, 1.0);
glScalef(scalex, scaley, scalez);
```

首先对模型进行三维缩放变换，然后是关于 z 轴、y 轴和 x 轴的旋转，最后是平移变换。

习题

1. 将空间一个点(x,y,z)进行平移、旋转和缩放等一系列三维变换时，如果变换的顺序不同，则变换后的位置也不相同。请分别计算点(x,y,z)在如下三维变换的最终结果。

(1) 首先平移(2,0,4)，然后绕 x 轴旋转 90°，最后进行缩放变换(2,1,4)。

(2) 首先绕 x 轴旋转 90°，然后平移(2,0,4)，最后进行缩放变换(2,1,4)。

(3) 首先进行缩放变换(2,1,4)，然后绕 x 轴旋转 90°，最后平移(2,0,4)。

2. 请写出将空间一点(x,y,z)绕空间任意轴线旋转 θ 的变换矩阵。

3. 当一个三角形被矩形窗口裁剪后，可能的结果会有哪些？如果凸的 n 边形被矩形窗口裁剪，可能的结果又会有哪些？