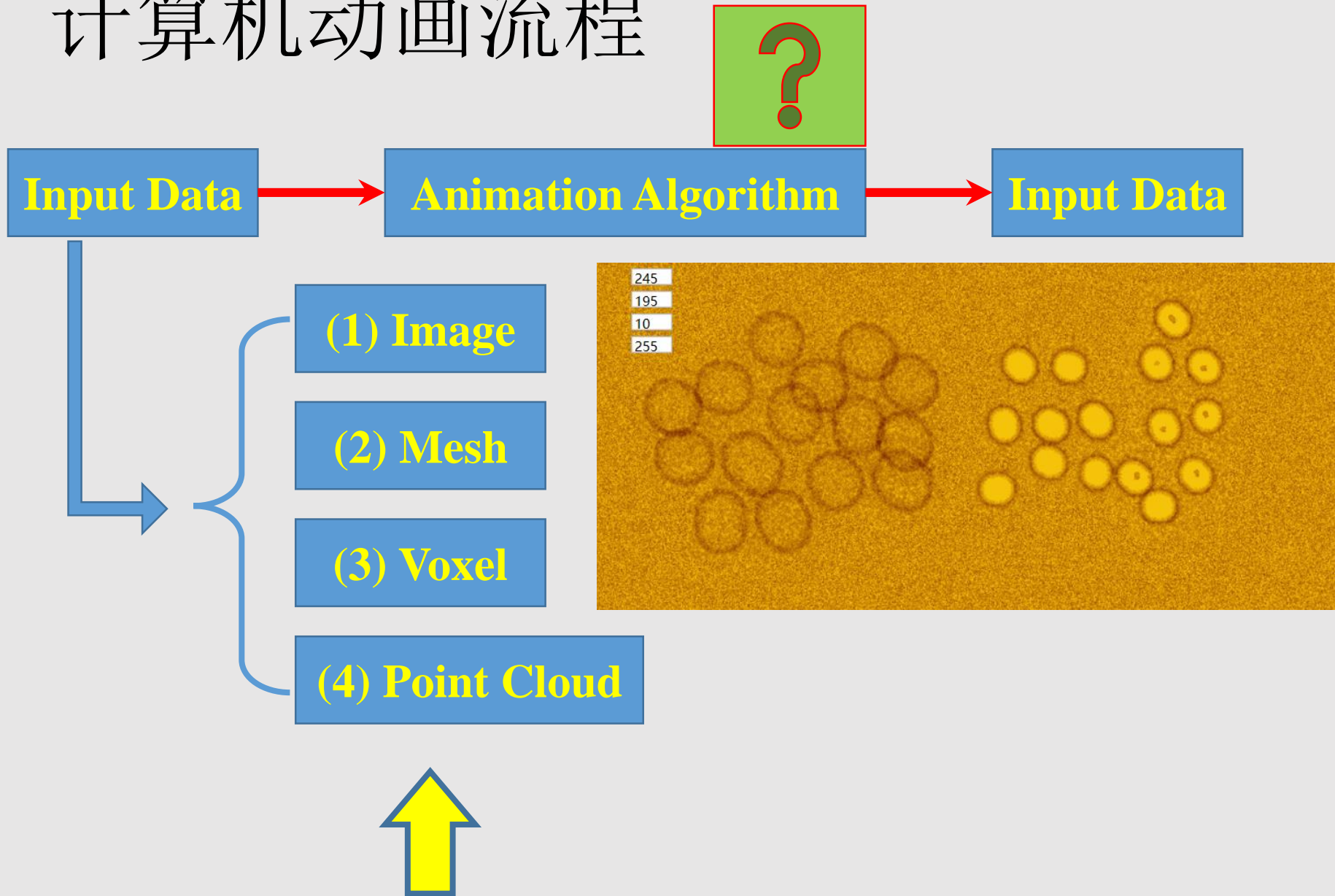
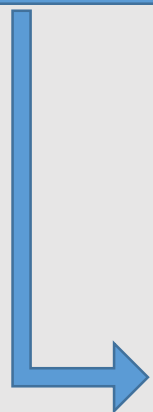


# 计算机动画流程



# 计算机动画流程

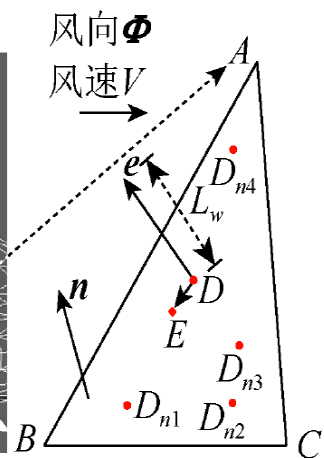
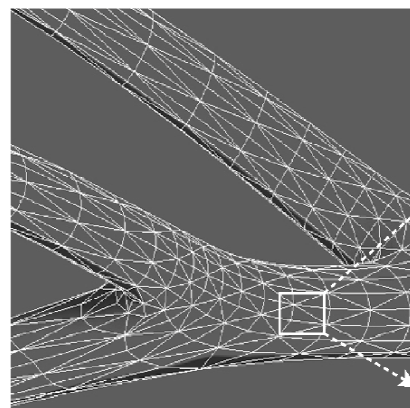
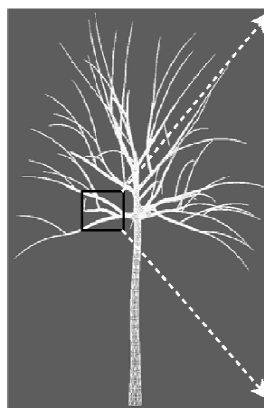


(1) Image

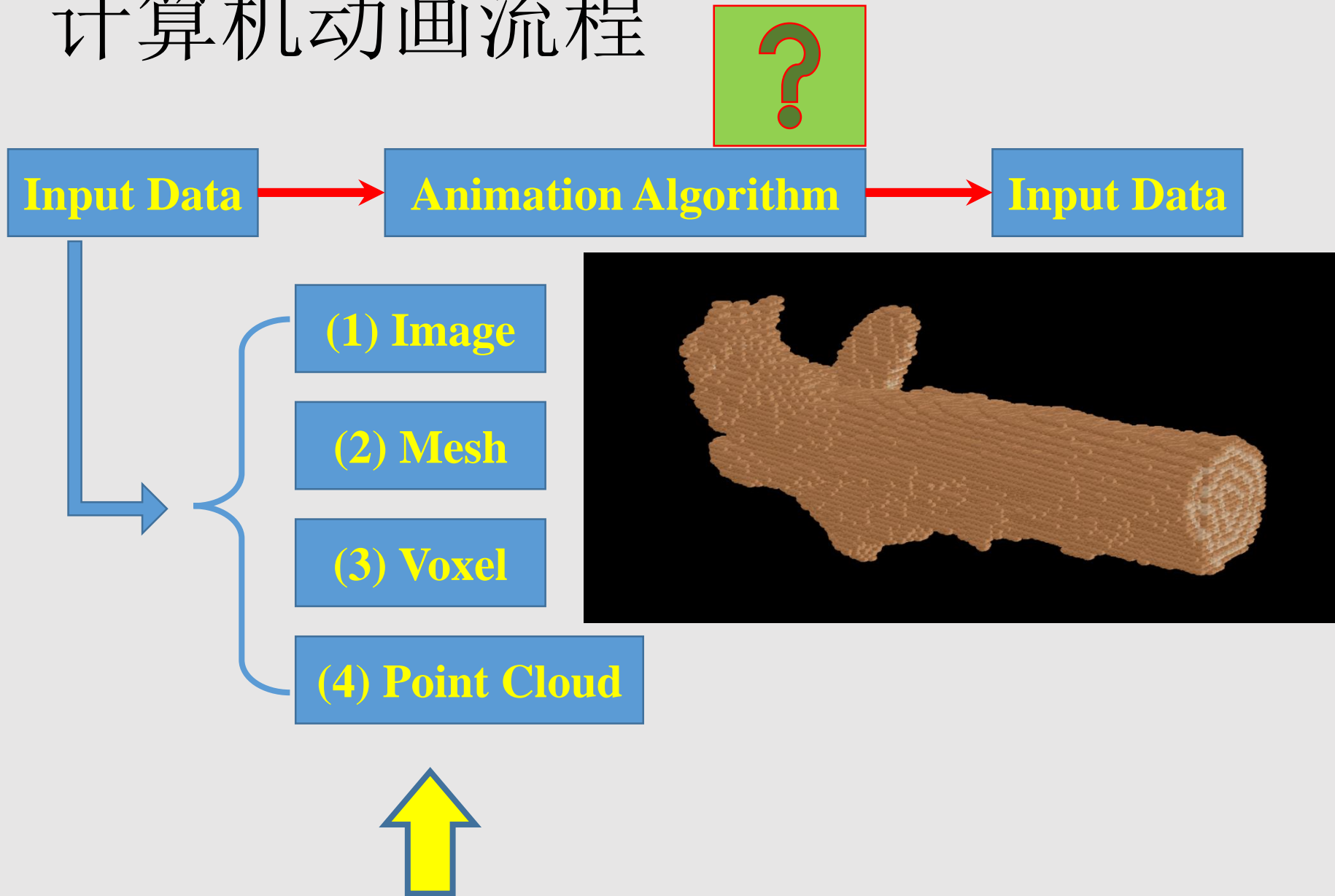
(2) Mesh

(3) Voxel

(4) Point Cloud



# 计算机动画流程



# 计算机动画流程

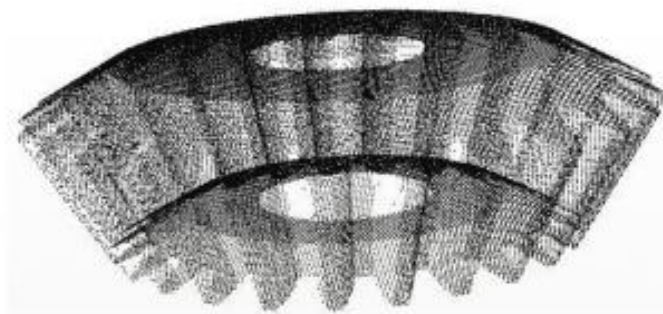
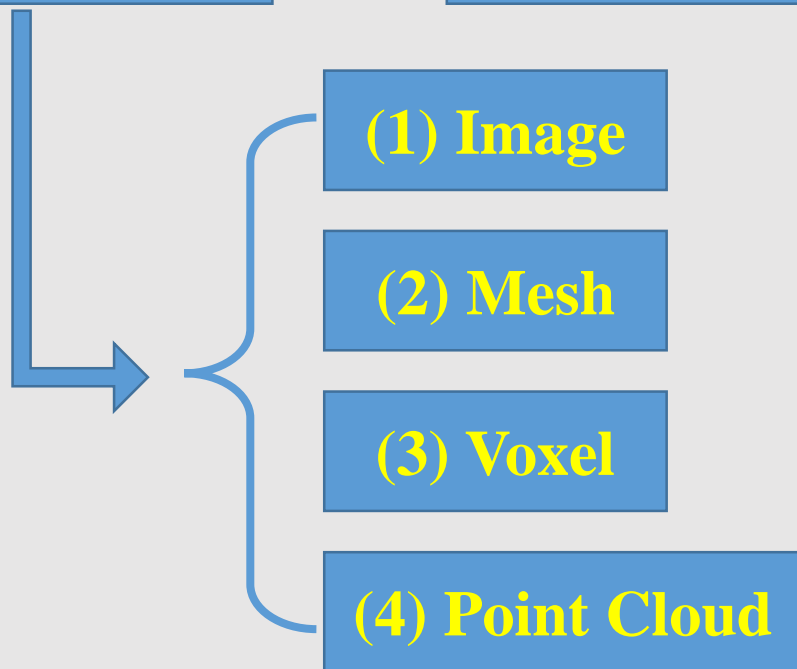


图2 锥齿轮的点云数据

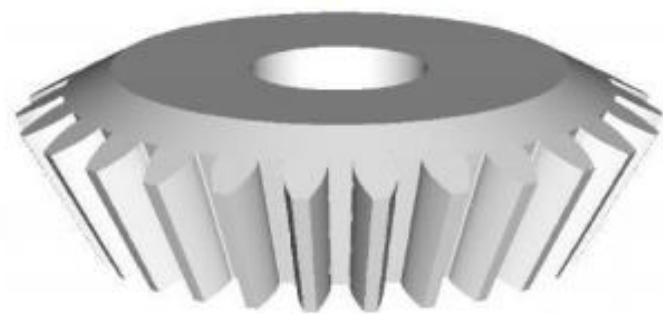


图3 锥齿轮实体模型

## 第二章 计算机动画技术背景

### 2.1 三维空间变换

2.1.1 变换的数学基础

2.1.2 几何变换

2.1.3 坐标变换

2.1.4 投影变换

2.1.5 图形的显示流程

2.1.6 OpenGL中的变换

### 2.2 曲线与曲面

2.2.1 基础知识

2.2.2 B é zier曲线/曲面

2.2.3 B样条曲线/曲面

2.2.4 NURBS曲线/曲面

2.2.5 Coons曲面

2.2.6 三维物体的描述方法

## 2.1 三维空间变换

2.1.1 变换的数学基础

2.1.2 几何变换

2.1.3 坐标变换

2.1.4 投影变换

2.1.5 图形的显示流程

2.1.6 OpenGL中的变换

## 2.1.1 图形变换的数学基础

### 2.1.1.1 矢量运算

设两个矢量： $V_1(x_1, y_1, z_1), V_2(x_2, y_2, z_2)$

(1) 两矢量之和： $V_1 + V_2 = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$

(2) 两矢量的点积： $V_1 \bullet V_2 = x_1 * x_2 + y_1 * y_2 + z_1 * z_2$

(3) 矢量长度： $|V_1| = (V_1 \bullet V_1)^{1/2} = (x_1 * x_1 + y_1 * y_1 + z_1 * z_1)^{1/2}$

(4) 两矢量的叉积：

$$V_1 \times V_2 = \begin{vmatrix} i & j & k \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix} = (y_1 z_2 - y_2 z_1, z_1 x_2 - z_2 x_1, x_1 y_2 - x_2 y_1)$$

## 2.1.1.2 矩阵运算

(1) 矩阵加法:

$$A + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots a_{2n} + b_{2n} \\ \cdots & \cdots & \cdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots a_{mn} + b_{mn} \end{bmatrix}$$

(2) 数乘矩阵:

$$kA = \begin{bmatrix} ka_{11} & ka_{12} & \cdots ka_{1n} \\ ka_{21} & ka_{22} & \cdots ka_{2n} \\ \cdots & \cdots & \cdots \\ ka_{m1} & ka_{m2} & \cdots ka_{mn} \end{bmatrix}$$



(3) 矩阵乘法运算:

$$\begin{aligned} C = A \bullet B &= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix} \end{aligned}$$

$$C = (c_{ij})_{m \times p} = A_{m \times n} \bullet B_{n \times p}$$

(4) 零矩阵的运算:

$$A_{m \times n} + 0_{m \times n} = A_{m \times n}$$

(5) 单位矩阵:

$$I = \begin{bmatrix} 1 & & & \\ & \bullet & & \\ & & \bullet & \\ & & & \bullet \\ & & & & 1 \end{bmatrix}$$

$$A_{m \times n} \bullet I = A_{m \times n}$$

$$I \bullet A_{m \times n} = A_{m \times n}$$

(6) 逆矩阵:

设 $A$ 是一个 $n$ 阶矩阵, 如果有 $n$ 阶矩阵 $B$ 存在, 使得

$$A \bullet B = B \bullet A = I$$

则说 $A$ 是一个非奇异矩阵, 并说 $B$ 是 $A$ 的逆。

任何非奇异矩阵只能有一个逆矩阵。

(7) 转置矩阵:

矩阵  $A = (a_{ij})_{m \times n}$  的行列互换得到的  $n \times m$  矩阵叫做  $A$  的转置矩阵, 记作  $A^T$ 。

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \cdots & \cdots & \cdots & \cdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

- 转置矩阵的基本性质

$$(1) \quad \left(A^T\right)^T = A$$

$$(2) \quad (\partial A)^T = \partial A^T$$

$$(3) \quad (A+B)^T = A^T + B^T$$

$$(4) \quad (A \bullet B)^T = B^T \bullet A^T$$

(8) 矩阵运算的基本性质：

- 矩阵加法适合交换律与结合率

$$A + B = B + A$$

$$A + (B + C) = (A + B) + C$$

- 数乘矩阵适合分配率与结合率：

$$\partial(A + B) = \partial A + \partial B$$

$$\partial(A \bullet B) = (\partial A) \bullet B = A \bullet (\partial B)$$

$$(\partial + \beta)A = \partial A + \beta A$$

$$\partial(\beta A) = (\partial \beta)A$$

- 矩阵的乘法适合结合率：

$$A(B \bullet C) = (A \bullet B)C$$

- 矩阵的乘法对加法适合分配率：

$$(A + B)C = AC + BC$$

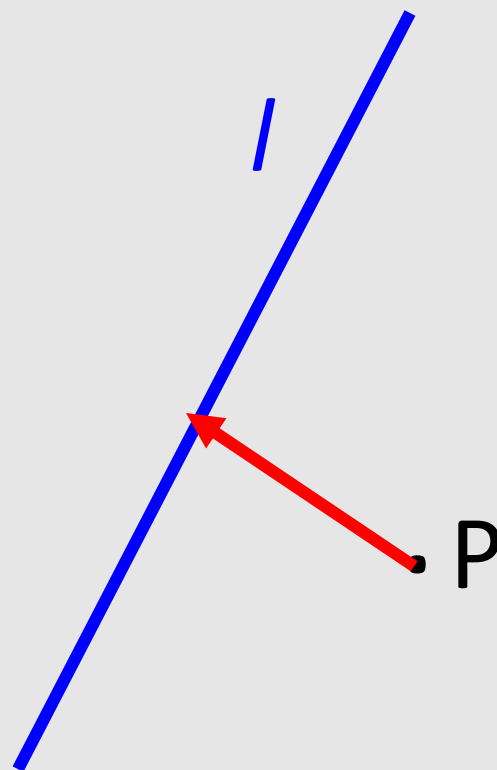
$$C(A + B) = CA + CB$$



- 矩阵的乘法不适合交换率：

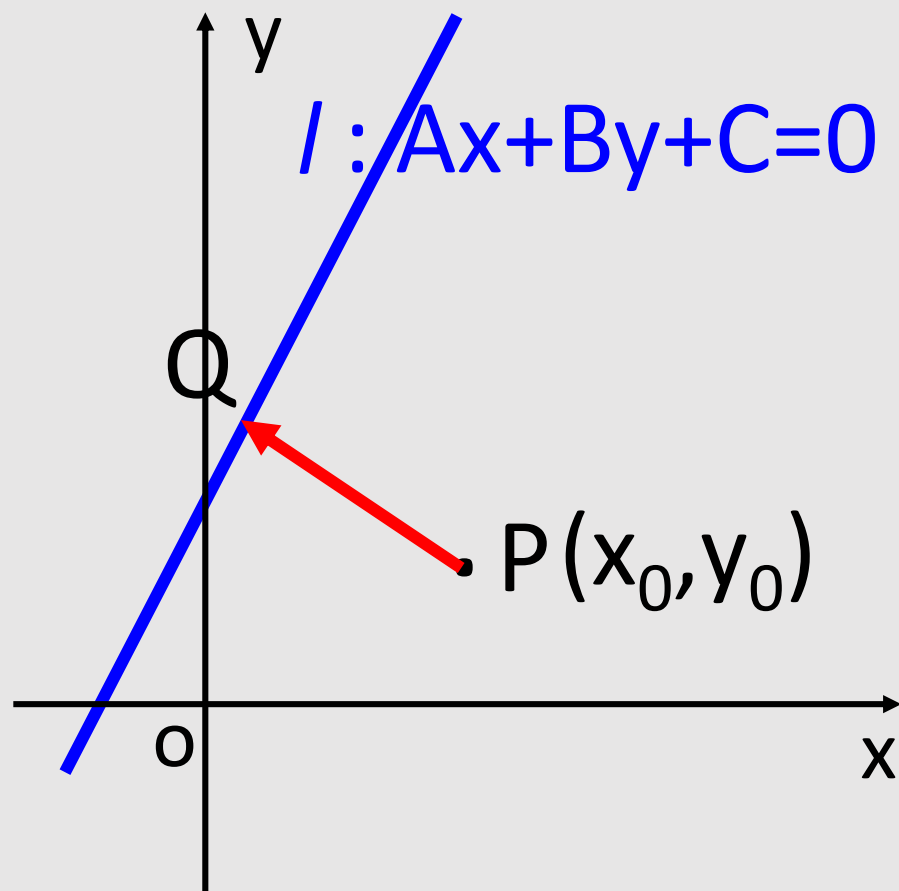
一般情况下，  $AB \neq BA$

# 点到直线的距离

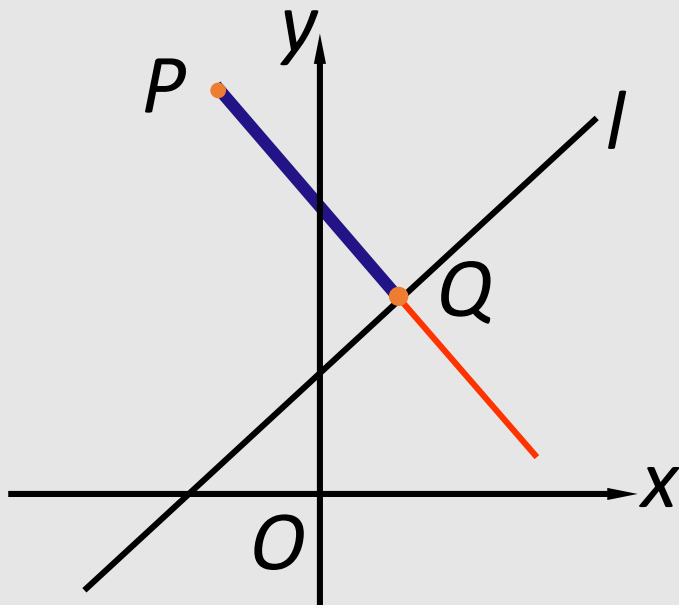


## 问题举例

# 点到直线的距离



**问题：** 求点 $P(x_0, y_0)$  到直线 $l: Ax+By+C=0$ 的距离。



$P(x_0, y_0)$

$l: Ax+By+C=0$

法一：写出直线PQ的方程，与 $l$ 联立求出点 $Q$ 的坐标，  
然后用两点间的距离公式求得  $|PQ|$ 。

法二：  $P(x_0, y_0)$ ,  $l: Ax + By + C = 0$ , 设  $AB \neq 0$ ,

$\because AB \neq 0, \therefore$  这时  $l$  与  $x$  轴,  $y$  轴都相交,

过  $p$  作  $x$  轴的平行线, 交  $l$  与点  $R(x_1, y_0)$ ;

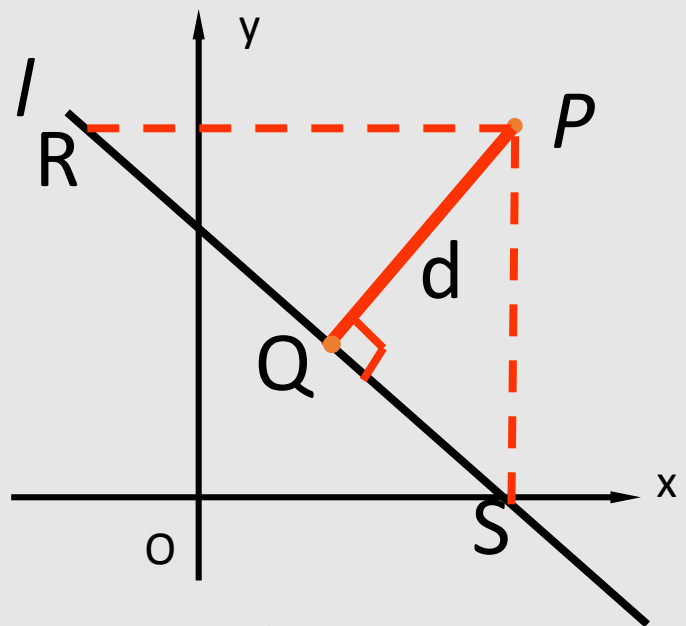
作  $y$  轴的平行线, 交  $l$  与点  $S(x_0, y_2)$

$\therefore Ax_1 + By_0 + C = 0, Ax_0 + By_2 + C = 0$

$$\therefore x_1 = \frac{-By_0 - C}{A}, y_2 = \frac{-Ax_0 - C}{B}$$

$$\therefore |PR| = |x_0 - x_1| = \left| \frac{Ax_0 + By_0 + C}{A} \right|, |PS| = |y_0 - y_2| = \left| \frac{Ax_0 + By_0 + C}{B} \right|$$

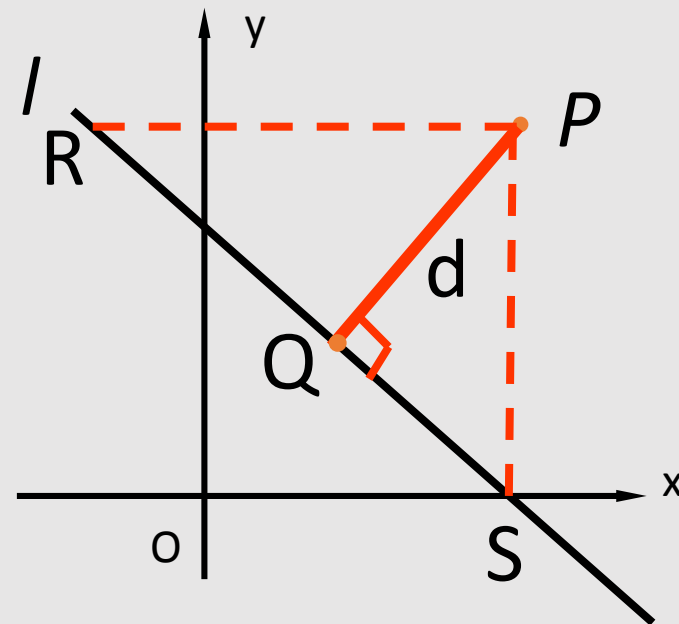
$$\therefore |RS| = \sqrt{PR^2 + PS^2} = \frac{\sqrt{A^2 + B^2}}{|AB|} |Ax_0 + By_0 + C|$$



由三角形面积公式可得：

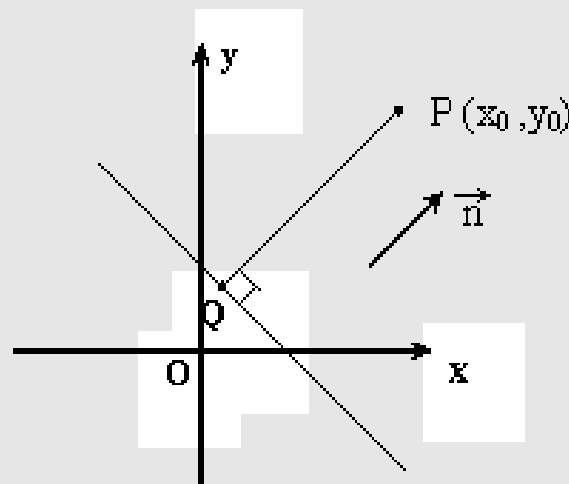
$$\begin{aligned}d \cdot |RS| &= |PR| \cdot |PS| \\ \therefore d \cdot \frac{\sqrt{A^2 + B^2}}{|AB|} |Ax_0 + By_0 + C| \\ &= \left| \frac{Ax_0 + By_0 + C}{A} \right| \cdot \left| \frac{Ax_0 + By_0 + C}{B} \right|\end{aligned}$$

$$\therefore d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$



注：❶ 在使用该公式前，须将  
直线方程化为一般式。

❷  $A=0$ 或 $B=0$ ，此公式也成立，  
但当 $A=0$ 或 $B=0$ 时一般不用此  
公式计算距离。



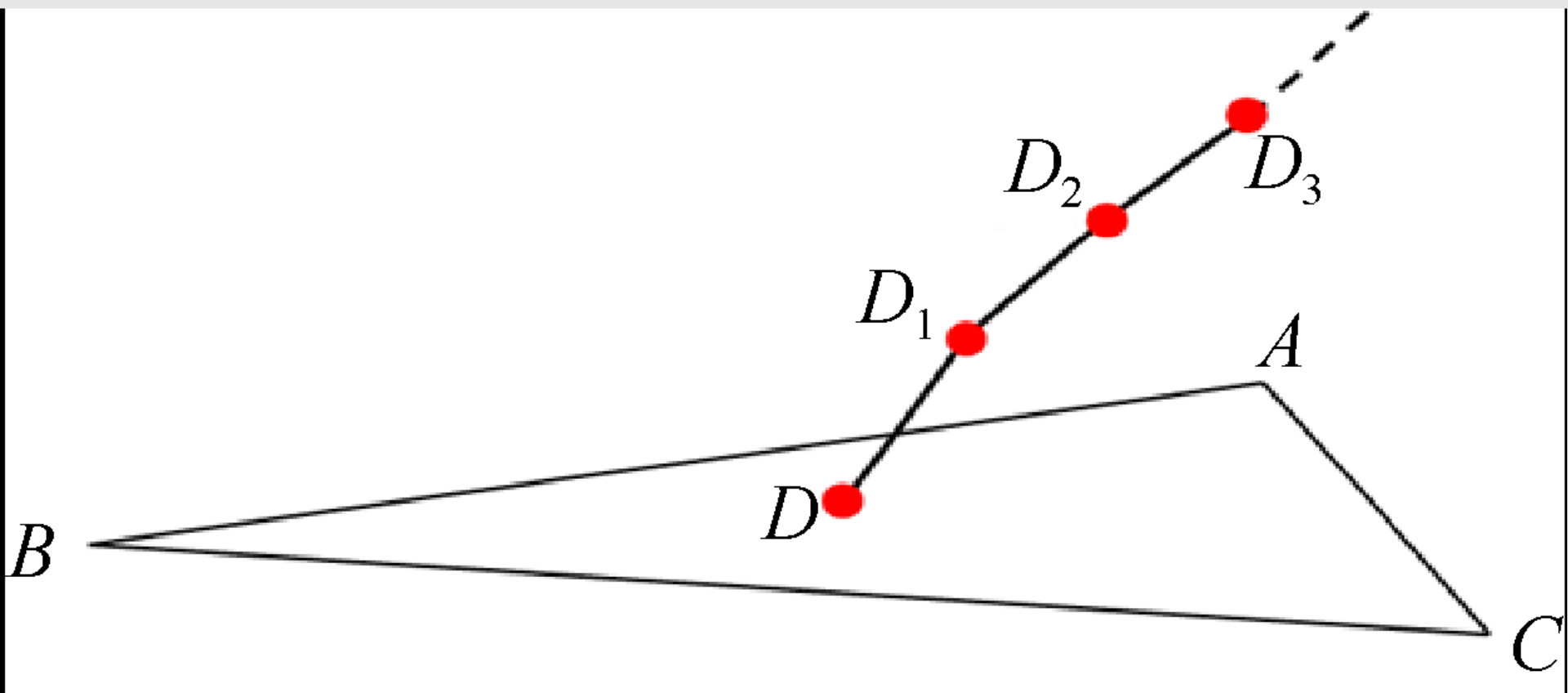
$$\overrightarrow{PQ} = (x_Q - x_0, y_Q - y_0), \quad \lambda \vec{n} = (\lambda A, \lambda B)$$

$$\begin{cases} x_Q - x_0 = \lambda A \\ y_Q - y_0 = \lambda B \end{cases} \Rightarrow \begin{cases} x_Q = x_0 + \lambda A \\ y_Q = y_0 + \lambda B \end{cases}, \text{ 由于点 } Q \text{ 在直线上, 所以满足直线方程}$$

$$A(x_0 + \lambda A) + B(y_0 + \lambda B) + C = 0, \text{ 解得 } \therefore \lambda = -\frac{Ax_0 + By_0 + C}{A^2 + B^2}$$

$$\therefore |PQ| = |\lambda| |\vec{n}| = \frac{|Ax_0 + By_0 + C|}{A^2 + B^2} \sqrt{A^2 + B^2} = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$

实例



实例 图 硬化的模拟形态示意图

[Jump to 推荐图书](#)



## 2.1 三维空间变换

2.1.1 变换的数学基础

2.1.2 几何变换

2.1.3 坐标变换

2.1.4 投影变换

2.1.5 图形的显示流程

2.1.6 OpenGL中的变换

## 2.1.2 三维几何变换

2.1.2.1 平移变换

2.1.2.2 缩放变换

2.1.2.3 旋转变换

2.1.2.4 变形变换

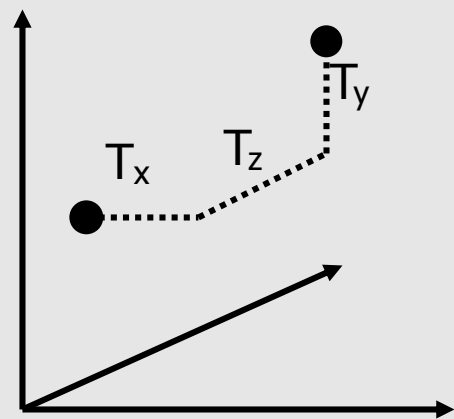
2.1.2.5 对称变换

## 2.1.2.1 平移变换

- 每个三维点 $(x,y,z)$ 对应于一个齐次坐标 $[x,y,z,1]$ 。所有的三维变换都可通过乘以一个 $4 \times 4$ 的变换矩阵来进行。
- 平移变换

点 $(x,y,z)$ 沿 $x$ 轴方向平移 $T_x$ 距离，沿 $y$ 轴方向平移 $T_y$ 距离，沿 $z$ 轴方向平移 $T_z$ 距离，变成点 $(x',y',z')$ ，这一变换过程的变换矩阵为：

$$T_3(T_x, T_y, T_z) = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

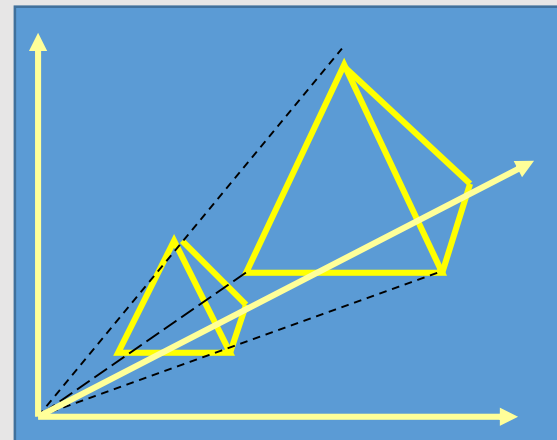


$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix}^T = T_3(T_x, T_y, T_z) \bullet \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$$

## 2.1.2.2 缩放变换

- 设一个点沿 $x$ ,  $y$ ,  $z$ 轴缩放的比例分别为 $S_x$ ,  $S_y$ ,  $S_z$ , 则缩放变换矩阵可表示为:

$$S_3(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



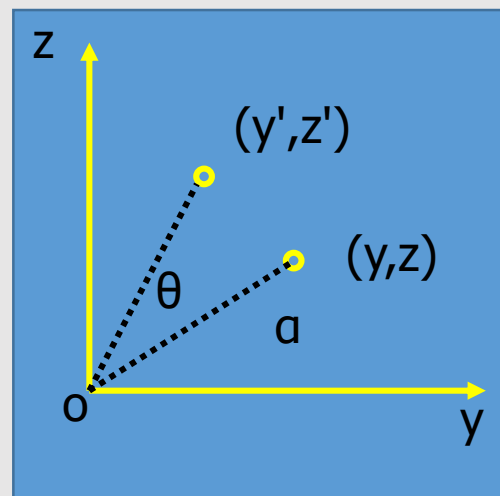
$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix}^T = S_3(S_x, S_y, S_z) \bullet \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$$

- 当 $|S_x|, |S_y|, |S_z|$ 分别大于1时, 为物体的放大; 小于1时, 为缩小变换;
- 当 $|S_x|, |S_y|, |S_z|$ 皆等于1时, 即为恒等变换;
- 当 $S_x, S_y, S_z$ 分别小于0时, 作相应坐标平面的镜面变换。

## 2.1.2.3 旋转变换 — 绕坐标轴旋转

### ■ 绕X轴变换

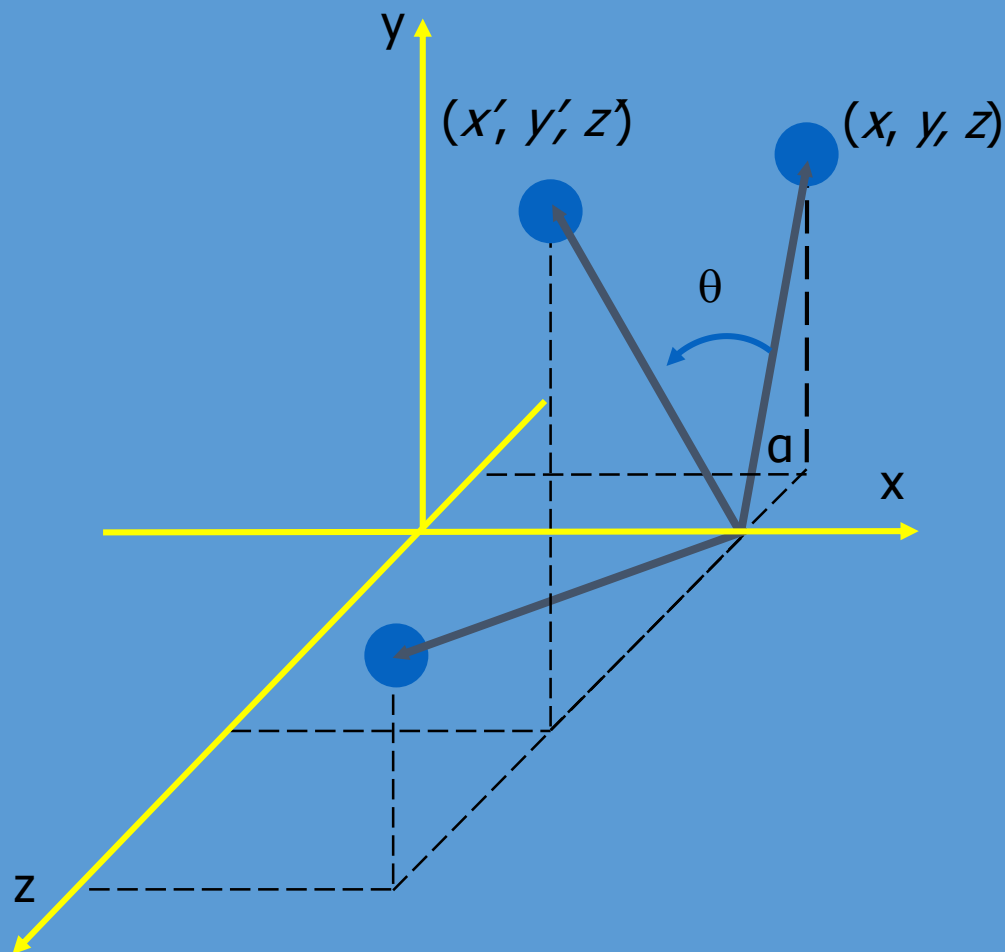
空间上的立体绕X轴旋转时，立体上各点的X坐标不变，只是Y、Z坐标发生相应的变化。



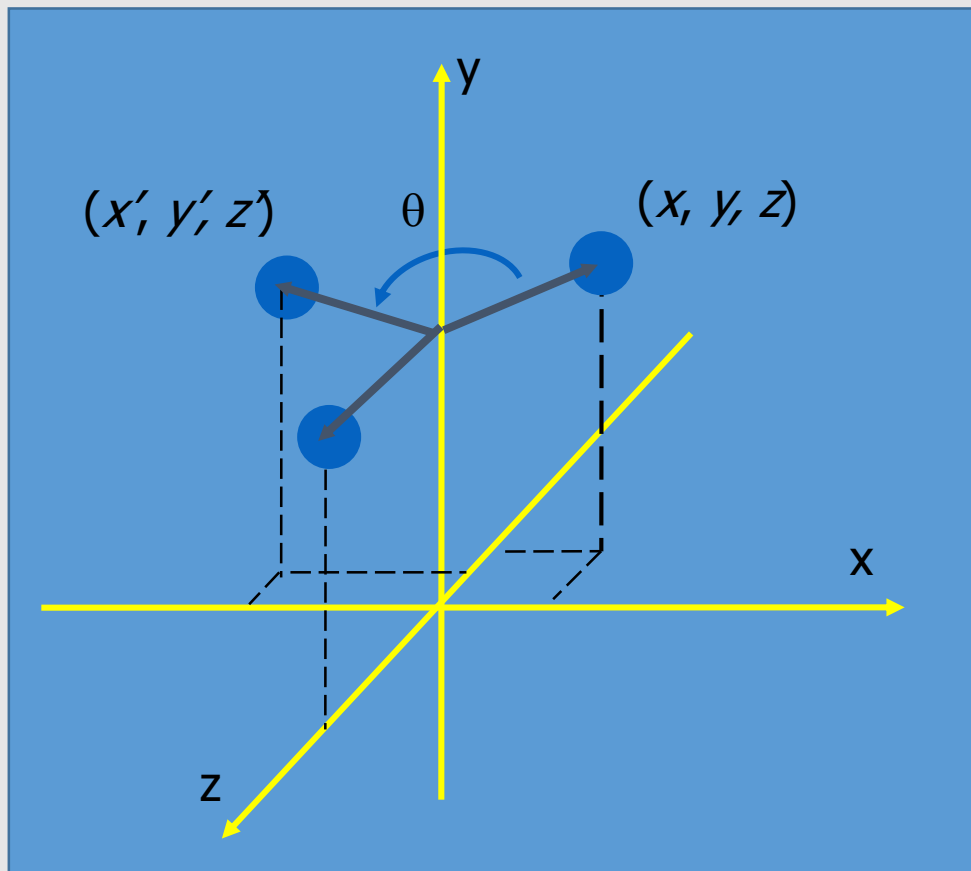
$$x' = x$$

$$y' = p \cos(\alpha + \theta) = y \cdot \cos \theta - z \cdot \sin \theta$$

$$z' = p \sin(\alpha + \theta) = y \cdot \sin \theta + z \cdot \cos \theta$$

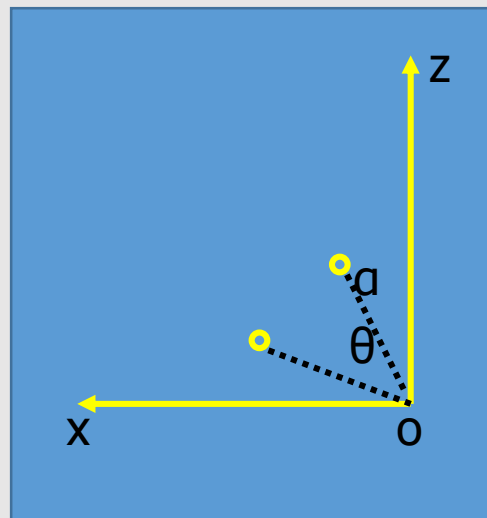


## 2.1.2.3 旋转变换 — 绕坐标轴旋转



### ■ 绕Y轴旋转

此时，Y坐标不变，X，Z坐标相应变化。

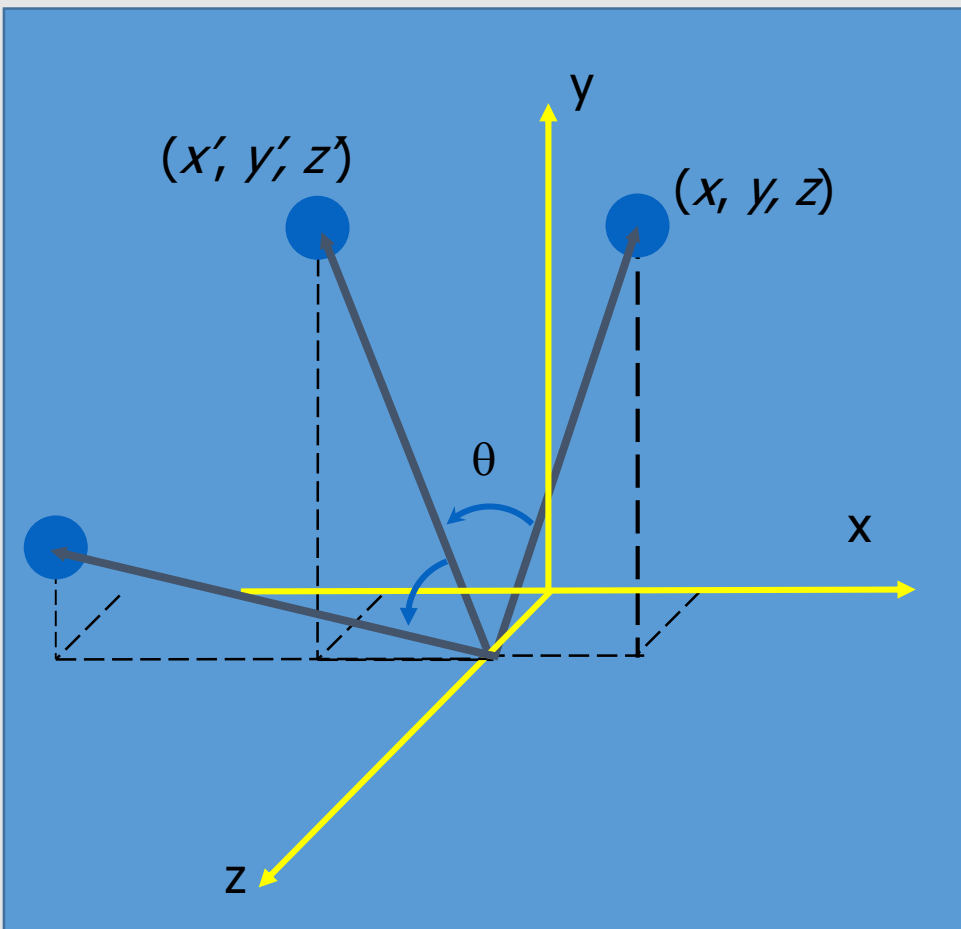


$$x' = \rho \sin(\alpha + \theta) = x * \cos\theta + z * \sin\theta$$

$$y' = y$$

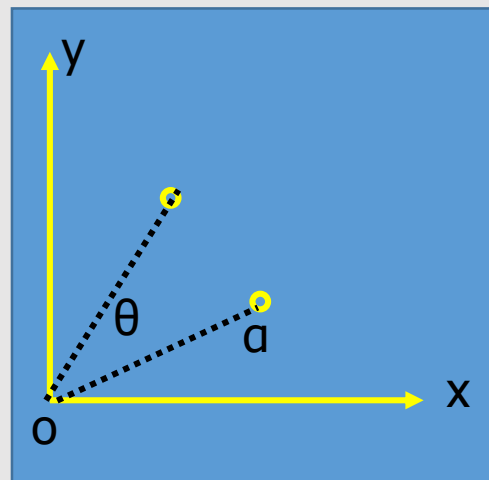
$$z' = \rho \cos(\alpha + \theta) = z * \cos\theta - x * \sin\theta$$

## 2.1.2.3 旋转变换 — 绕坐标轴旋转



### ■ 绕Z轴旋转

此时，Z坐标不变，X，Y坐标相应变化。



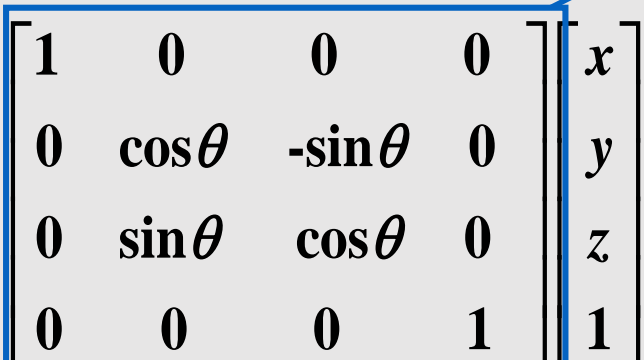
$$x' = \rho \cos(\alpha + \theta) = x \cdot \cos \theta - y \cdot \sin \theta$$

$$y' = \rho \sin(\alpha + \theta) = x \cdot \sin \theta + y \cdot \cos \theta$$

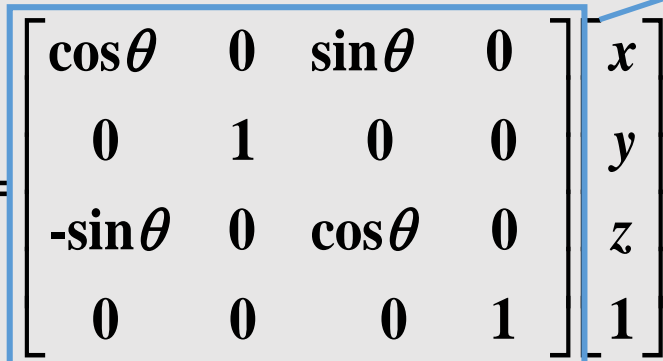
$$z' = z$$

## 2.1.2.3 旋转变换 — 绕坐标轴旋转 $R_x(\theta)$

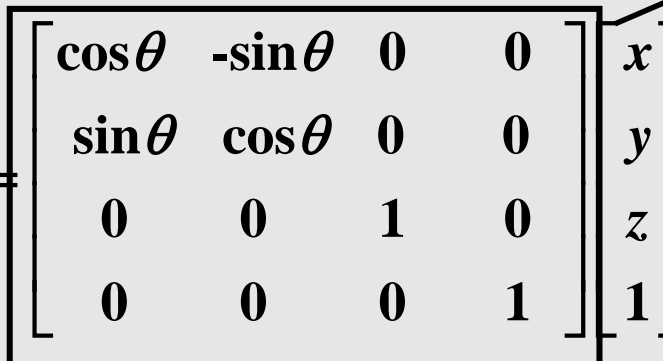
绕x轴旋转：

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$


绕y轴旋转：

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix}^T = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$


绕z轴旋转：

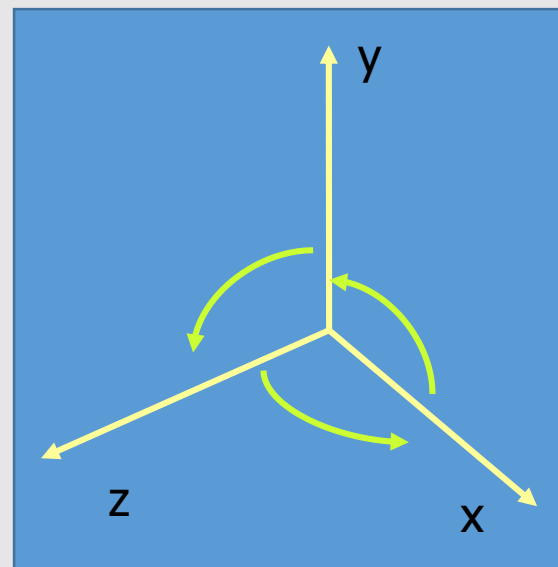
$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix}^T = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$




## 2.1.2.3 旋转变换 — 旋转的方向

- 旋转角度为 $\theta$ 时，点的旋转方向：

旋转轴	相应的旋转方向
x轴	从y轴到z轴
y轴	从z轴到x轴
z轴	从x轴到y轴

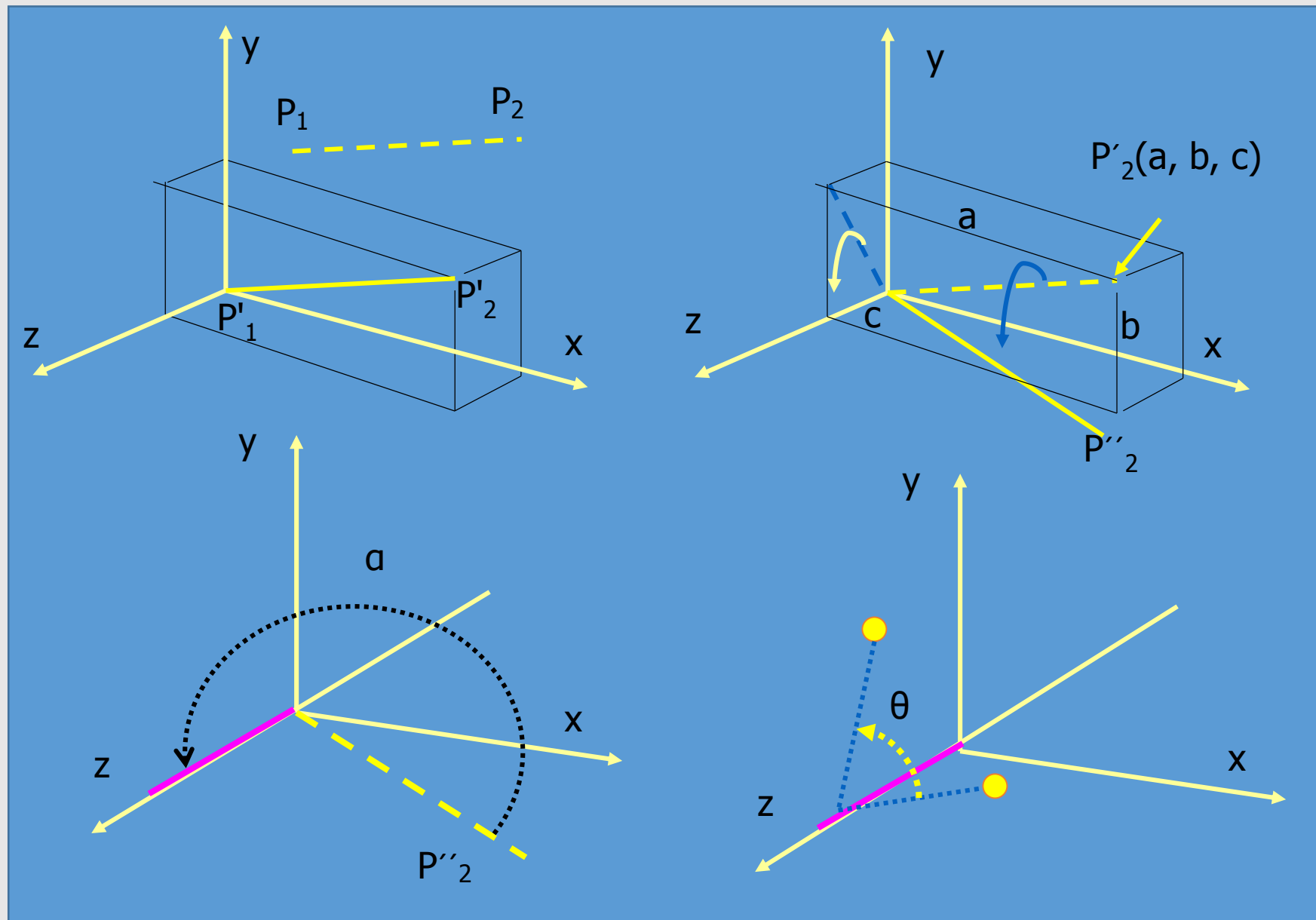


- 这样定义旋转方向的原因是为了保证所用的旋转矩阵是相同的。

## 2.1.2.3 旋转变换 — 绕任意轴旋转

- 求绕任意直线旋转的矩阵的原则：
  - 任意变换的问题  $\longrightarrow$  基本几何变换的组合
  - 绕任意直线旋转的问题  $\longrightarrow$  绕坐标轴旋转的组合

# 绕任意轴旋转 一点绕直线 $P_1P_2$ 旋转 $\theta$ 角

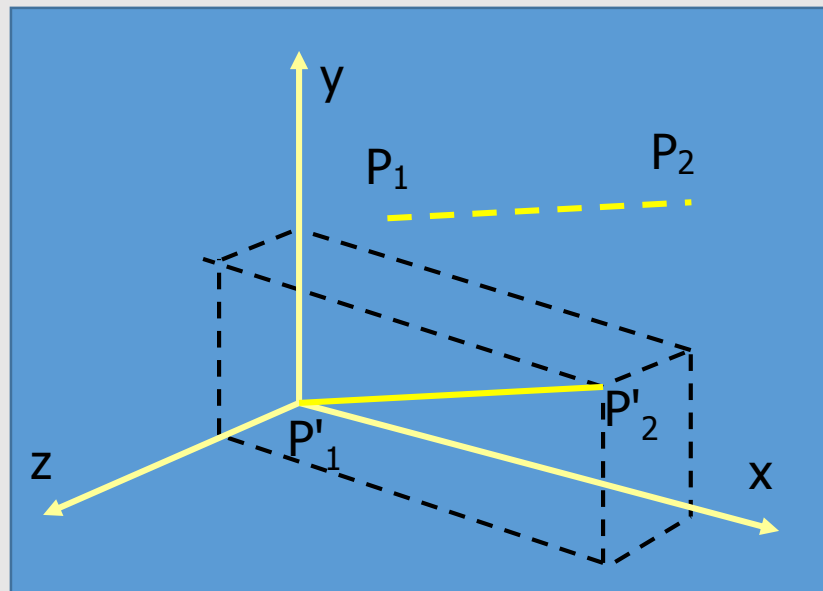


# 绕任意轴旋转 一点绕直线 $P_1P_2$ 旋转 $\theta$ 角

- 绕直线 $P_1P_2$ 旋转 $\theta$ 角的过程可分解为下列步骤：
  1. 把点 $P_1(x_1, y_1, z_1)$ 移至原点；
  2. 绕 $x$ 轴旋转，使直线与 $xoz$ 平面重合；
  3. 绕 $y$ 轴旋转，使直线与 $z$ 轴重合；
  4. 绕 $z$ 轴旋转 $\theta$ 角；
  5. 执行步骤(3)的逆变换；
  6. 执行步骤(2)的逆变换；
  7. 执行步骤(1)的逆变换；

# 绕任意轴旋转 一点绕直线 $P_1P_2$ 旋转 $\theta$ 角

- 步骤(1): 把点 $P_1(x_1, y_1, z_1)$ 移至原点, 变换矩阵为:



$$T_3(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# 绕任意轴旋转 一点绕直线 $P_1P_2$ 旋转 $\theta$ 角

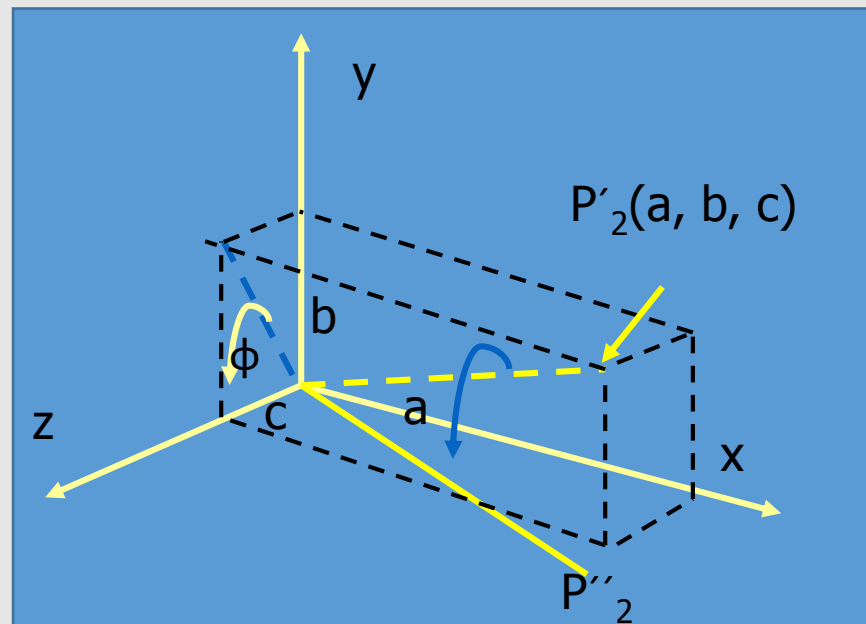
- 步骤(2): 绕x轴旋转, 使直线与xoz平面重合。可知:

$$\cos \phi = \frac{c}{\sqrt{b^2 + c^2}}$$

$$\sin \phi = \frac{b}{\sqrt{b^2 + c^2}}$$

- 设 $d_1 = (b^2 + c^2)^{1/2}$ , 则变换矩阵为:

$$R_X(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d_1 & -b/d_1 & 0 \\ 0 & b/d_1 & c/d_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



# 绕任意轴旋转 一点绕直线 $P_1P_2$ 旋转 $\theta$ 角

- 步骤(3): 绕y轴旋转, 使直线与z轴重合, 此刻 $P'_2$ 的坐标已变为 $P''_2(a,0,d_1)$ , 可知:

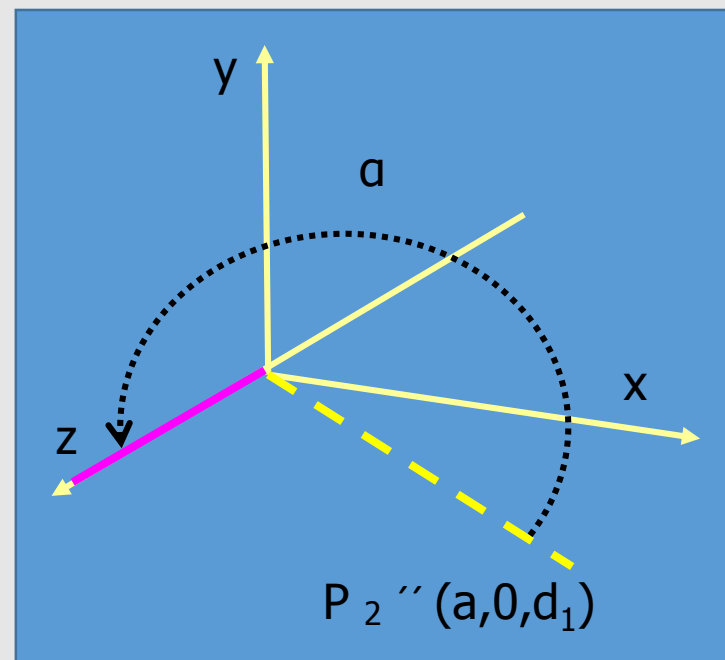
$$\cos \alpha = \frac{\overrightarrow{P_2''} \cdot \overrightarrow{u_z}}{|\overrightarrow{P_2''}| \times |\overrightarrow{u_z}|} = \frac{d_1}{\sqrt{a^2 + d_1^2}}$$

$$\overrightarrow{P_2''} \times \overrightarrow{u_z} = \overrightarrow{u_y} \cdot |\overrightarrow{P_2''}| \times |\overrightarrow{u_z}| \times \sin \alpha = \overrightarrow{u_y} \cdot (-a)$$

$$\Rightarrow \sin \alpha = \frac{-a}{\sqrt{a^2 + b^2 + c^2}}$$

令 $d_2 = (a^2 + b^2 + c^2)^{1/2}$ , 则变换矩阵为:

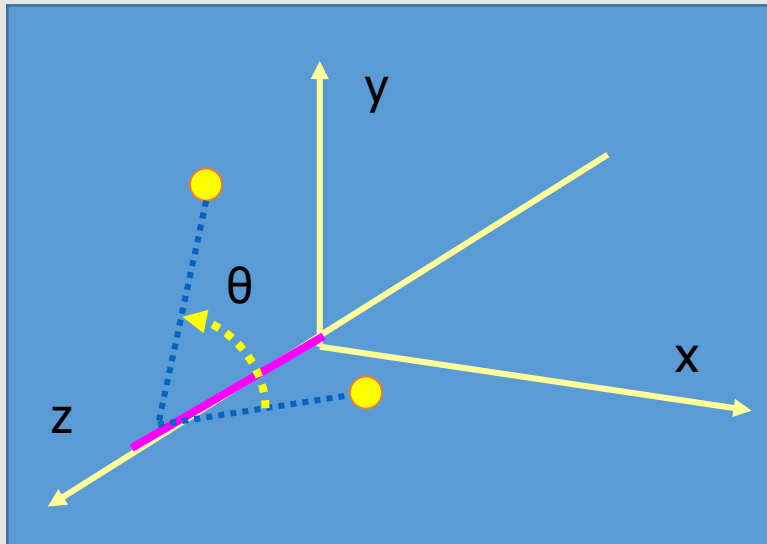
$$R_y(\alpha) = \begin{bmatrix} d_1 / d_2 & 0 & -a / d_2 & 0 \\ 0 & 1 & 0 & 0 \\ a / d_2 & 0 & d_1 / d_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



# 绕任意轴旋转 一点绕直线 $P_1P_2$ 旋转 $\theta$ 角

- 步骤(4): 绕 $z$ 轴旋转 $\theta$ 角, 变换矩阵为:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



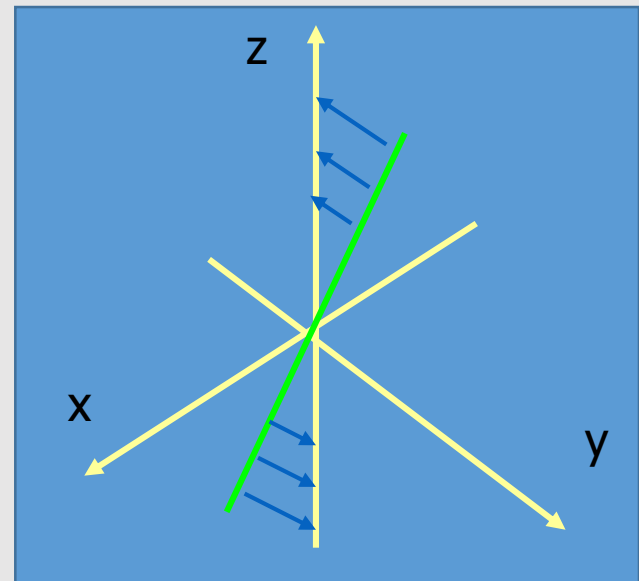


# 绕任意轴旋转 一点绕直线 $P_1P_2$ 旋转 $\theta$ 角

- 步骤(5)，执行步骤(3)的逆变换，变换矩阵为 $R_y(-\alpha)$ ；
  - 步骤(6)，执行步骤(2)的逆变换，变换矩阵为 $R_x(-\phi)$ ；
  - 步骤(7)，执行步骤(1)的逆变换，变换矩阵为 $T_3(x_1, y_1, z_1)$ 。
- 
- 综上，绕直线 $P_1P_2$ 旋转 $\theta$ 角的变换矩阵为：
$$R(\theta) = T_3(x_1, y_1, z_1) R_x(-\phi) R_y(-\alpha) R_z(\theta) R_y(\alpha) R_x(\phi) T_3(-x_1, -y_1, -z_1)$$
  - 注意：变换的过程有多种选择。如果中间的几个旋转次序变了，则各个矩阵的对应矩阵参数也会不同。

## 2.1.2.4 变形变换(错切变换)

- 对于过原点的一条直线，如果希望把它变换成另一条不同的过原点的直线，可以通过变形变换来实现。它可以产生变形的效果。例如：一个正方体可通过三维变形变换变成一个平行六面体。



- 这里只考虑比较简单的情况。把一条不在xoy平面上的过原点的直线变换成z轴，对应的z轴坐标都保持不变。
  - 这样的“z-变形”变换可以考虑在yoz平面和xoz平面上进行组合变形。

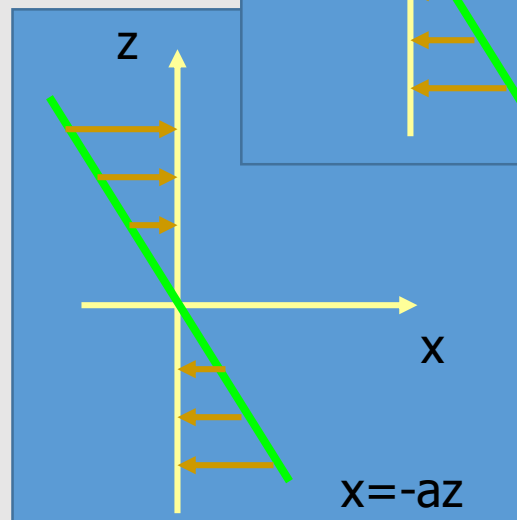
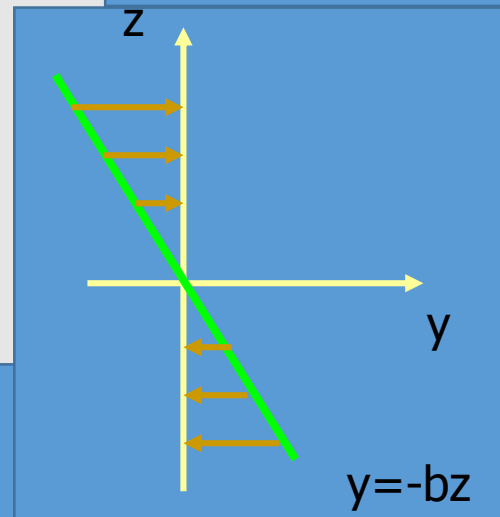
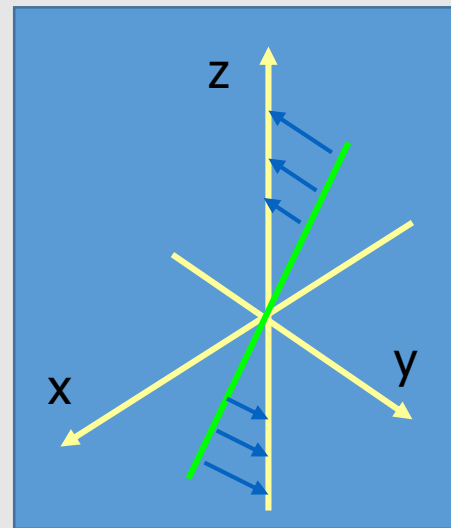
## 2.1.2.4 变形变换(错切变换)

- 对于 $yoz$ 平面上的变形情况，考虑直线 $y=-bz$ ，则变形后的直线方程为：

$$\begin{cases} y' = y - (-bz) = y + bz \\ z' = z \end{cases}$$

- 对于 $xoz$ 平面上的变形情况，考虑直线 $x=-az$ ，则变形后的直线方程为：

$$\begin{cases} x' = x - (-az) = x + az \\ z' = z \end{cases}$$

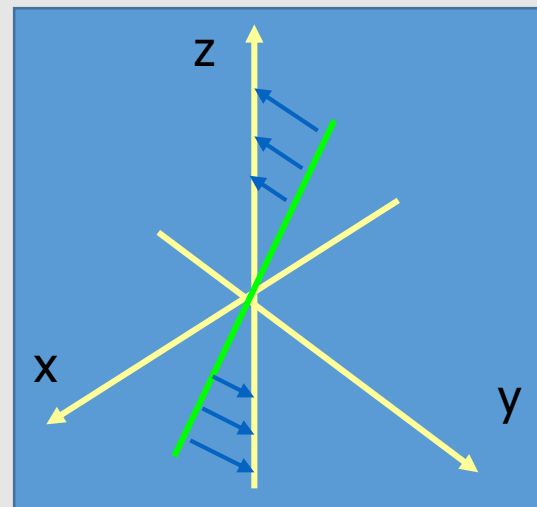


## 2.1.2.4 变形变换(错切变换)

- 则z-变形变换的矩阵表达式如下：

$$shz = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

其中a, b参数由直线方程所决定。



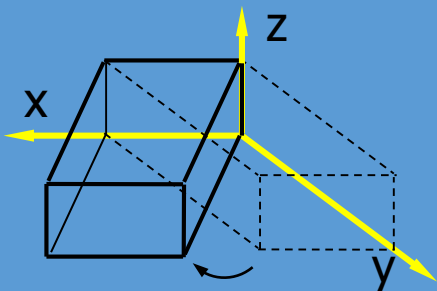
- 设直线上任一点 $P(x_1, y_1, z_1) \neq 0$ 经过z变形变换，将变成 $P(0, 0, z_1)$ ，即：

$$shz \bullet [x_1, y_1, z_1, 1]^T = [x_1 + az_1, y_1 + bz_1, z_1, 1]^T = [0, 0, z_1, 1]^T$$

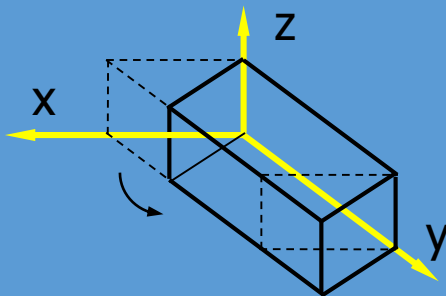
$$\text{即} \quad \begin{cases} x_1 + az_1 = 0 \\ y_1 + bz_1 = 0 \end{cases} \quad \text{则} \quad \begin{cases} a = -x_1 / z_1 \\ b = -y_1 / z_1 \end{cases}$$

- 如果该直线位于x, y平面，则 $z_1=0$ ，无法把它变换成z轴。

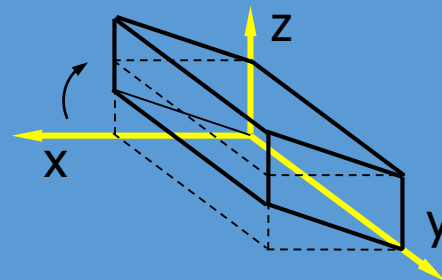
## 2.1.2.4 变形变换(错切变换)



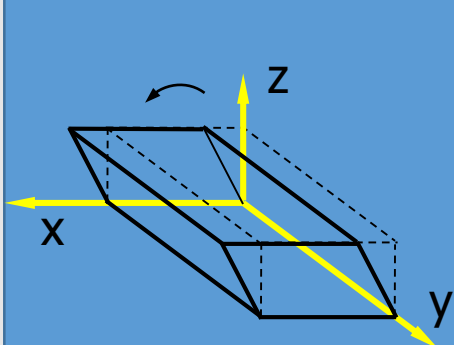
沿x含y错切



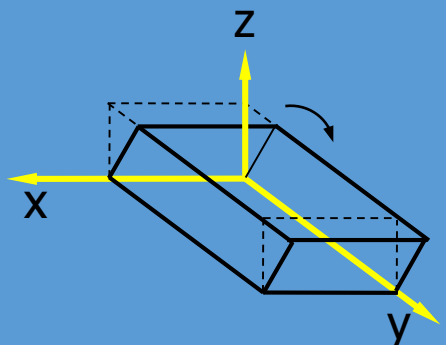
沿y含x错切



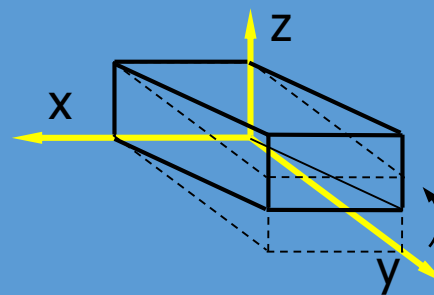
沿z含x错切



沿x含z错切



沿y含z错切



沿z含y错切

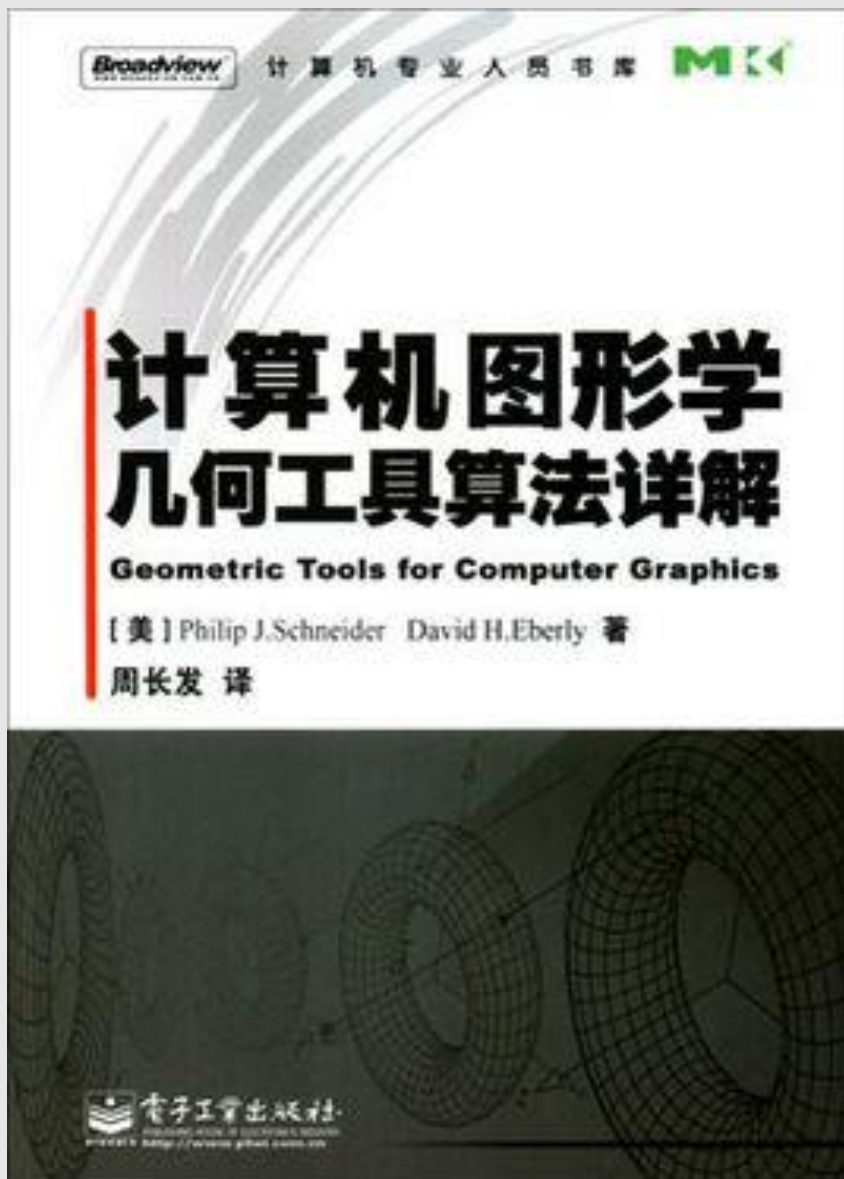
## 2.1.2.5 对称变换

- 关于坐标平面xoy的对称变换：

$$SY_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 关于其它坐标平面的变换类似。

## 2.1.2.5 工具书推荐



电子书下载:

<http://vdisk.weibo.com/s/Ad5c3dUIPAMJ>

<http://download.csdn.net/download/zg260/7615241>

作者: (美)施奈德 (Schneider Philip J.)  
(美)埃伯利 (Eberly David H.)

·出版日期: 2005年01月

·ISBN: 7121005158

·条形码: 9787121005152

·版本: 第1版

·装帧: 平装

·开本: 16

·正文语种: 中文

·丛书名: 计算机专业人员书库

·外文书名: Geometric Tools for  
Computer Graphics

[Jump back](#)

## 2.1 三维空间变换

2.1.1 变换的数学基础

2.1.2 几何变换

2.1.3 坐标变换

2.1.4 投影变换

2.1.5 图形的显示流程

2.1.6 OpenGL中的变换



## 2.1.3 坐标变换

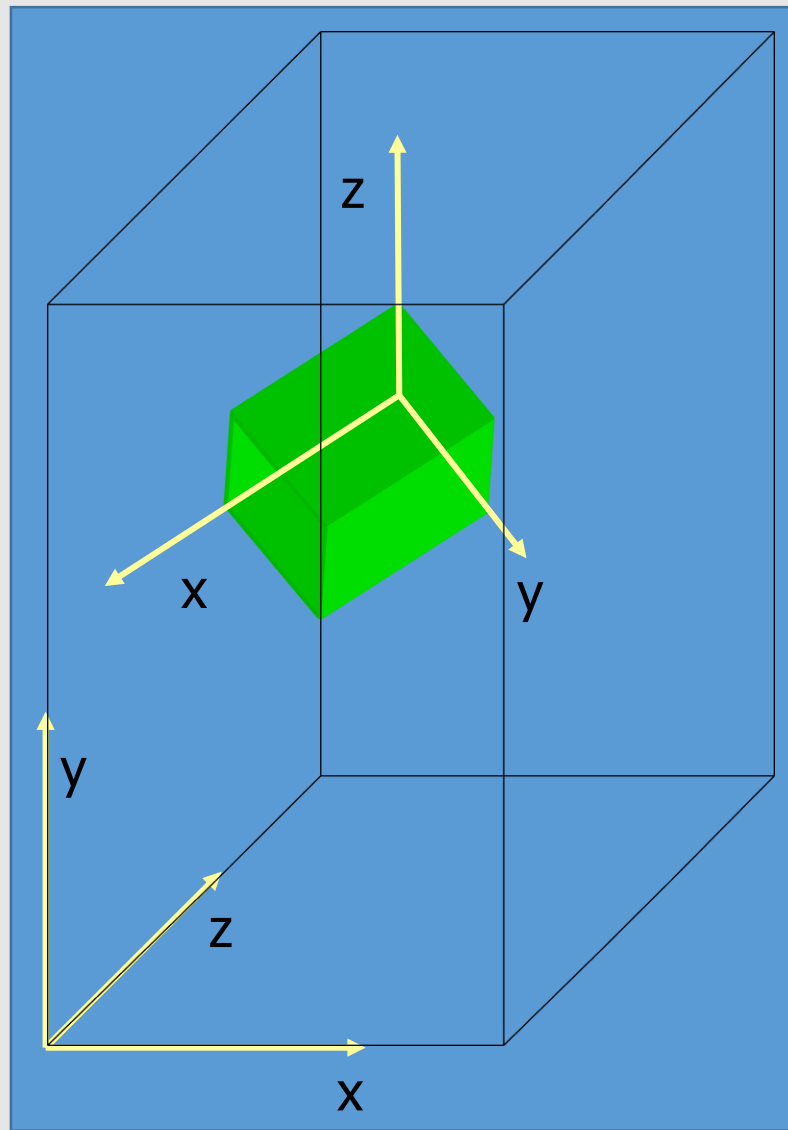
2.1.3.1 坐标系

2.1.3.2 坐标变换

2.1.3.3 几何变换和坐标变换的关系

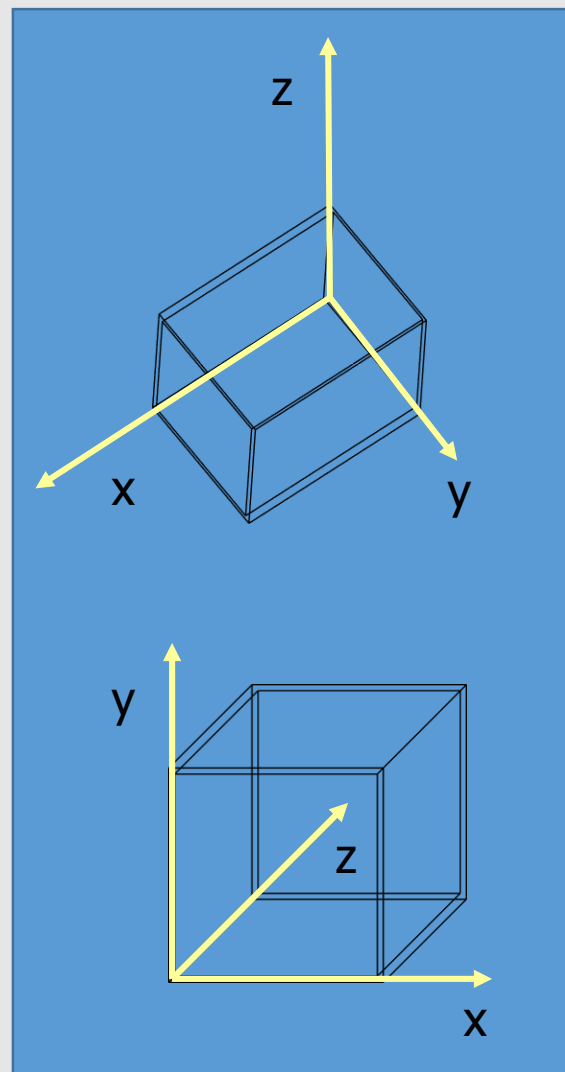
## 2.1.3.1 坐标系

- 对图形的描述、图形的输入输出都是在坐标系中进行的。
- 现实的物体具有高度、宽度、和深度。它们可以用三维坐标系的x轴、y轴和z轴来表示。
- 三维坐标系是一个直角坐标系；坐标系内任何一点可以由一个有序的三元组(x, y, z)来表示。每个坐标表示该点与坐标原点之间沿相应坐标轴的距离。



## 2.1.3.1 坐标系

- 在现实世界中，人们通常使用右手坐标系表示物体的位置，因此它又经常被称为世界坐标系(World Coordinate System)。
- 在计算机图形显示时，一般采用另一种三维坐标系：计算机的屏幕是一个平面。指定它的左下角为原点， $x$ 轴正向向右延伸， $y$ 轴正向向上延伸。另外，定义 $z$ 轴从原点开始，指向屏幕内部，表示深度。这个坐标系称作左手坐标系。



## 2.1.3.1 坐标系

- **世界坐标系** WC(World Coordinate System): 包括常用的直角坐标系、几何坐标系等各种坐标系, 用来直接描述对象。或称为**用户坐标系** UC (User Coordinate System), 取值范围为整个实数域。
- **设备坐标系** DC(Device Coordinate System): 图形的显示是在设备上进行的, 在设备上描述图形的坐标系称为设备坐标系 DC(Device Coordinate System), 取值范围受设备的输入输出的精度以及画面有效范围的限制。屏幕上显示的图形均以**其一个像素**点单位为量化单位。

## 2.1.3 坐标变换

2.1.3.1 坐标系

2.1.3.2 坐标变换

2.1.3.3 几何变换和坐标变换的关系

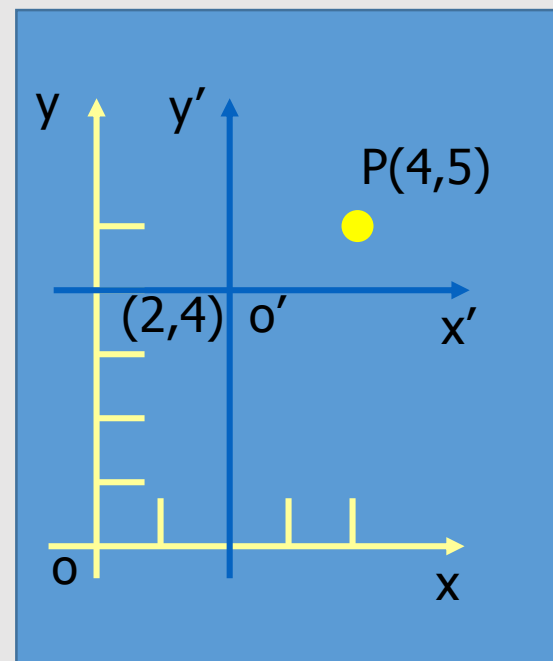
## 2.1.3.2 坐标变换

- 同一物体在**不同的坐标系**中描述时，会有**不同的坐标值**。在甲坐标系中定义一个物体，那么只要把乙坐标系的坐标轴及坐标原点变换成甲坐标系的坐标轴及坐标原点，就可以在乙坐标系中定义物体了。

### ■ 例1：坐标系平移

- 坐标系 $x'o'y'$ 的原点 $o'$ 在坐标系 $xoy$ 的坐标为 $(2, 4)$ ，则从 $x'o'y'$ 到 $xoy$ 的变换矩阵为 $T_2(-2, -4)$ ， $xoy$ 坐标系中一点 $P(4, 5)$ 在 $x'o'y'$ 坐标系中的坐标矢量为：

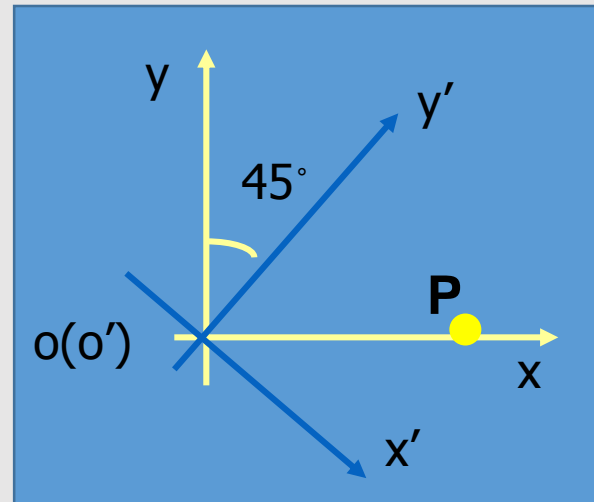
$$T_2(-2, -4) \cdot [4, 5, 1]^T = [2, 1, 1]^T$$



## 2.1.3.2 坐标变换

- 例2：坐标系旋转

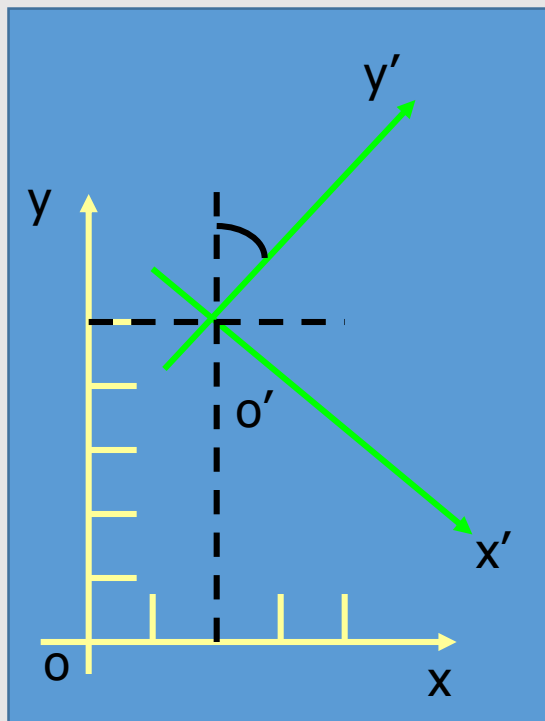
- $x'o'y'$ 坐标系与 $xoy$ 坐标系原点重合，将 $x'o'y'$ 坐标系逆时针旋转 $45^\circ$ 就变成了 $xoy$ 坐标系，则变换矩阵为：



$$R_2\left(\frac{\pi}{4}\right) = \begin{bmatrix} \cos \frac{\pi}{4} & -\sin \frac{\pi}{4} & 0 \\ \sin \frac{\pi}{4} & \cos \frac{\pi}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## 2.1.3.2 坐标变换

- 例3：坐标系复合变换
  - 求变换矩阵为平移矩阵和旋转矩阵的乘积矩阵： $R_2(\pi/4) \cdot T_2(-2, -4)$





## 2.1.3 坐标变换

2.1.3.1 坐标系

2.1.3.2 坐标变换

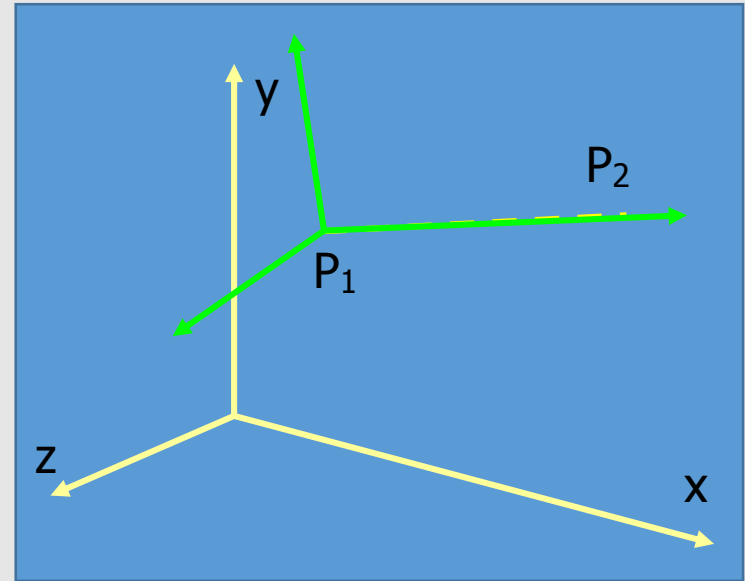
2.1.3.3 几何变换和坐标变换的关系

## 2.1.3.3 几何变换和坐标变换的关系

- 两者的原理及效果是相同的。都是利用变换矩阵来实现对图形的变换。
- 根据条件的不同，灵活选用几何变换或坐标变换可以简化问题。
  - 几何变换：直观；
  - 坐标变换：便于求解复杂问题。

## 2.1.3.3 几何变换和坐标变换的关系

- 前面利用几何变换求解的问题：
  - 绕空间中的直线 $P_1P_2$ 旋转的问题。
- 也可以利用坐标变换来实现：
  - 以 $P_1$ 点为坐标原点， $P_1P_2$ 为 $z$ 轴构建一个直角坐标系。则问题就简化为绕 $z$ 轴旋转的问题。
  - 如何选择 $x$ 轴、 $y$ 轴？



思考题？！

## 2.1 三维空间变换

2.1.1 变换的数学基础

2.1.2 几何变换

2.1.3 坐标变换

2.1.4 投影变换

2.1.5 图形的显示流程

2.1.6 OpenGL中的变换

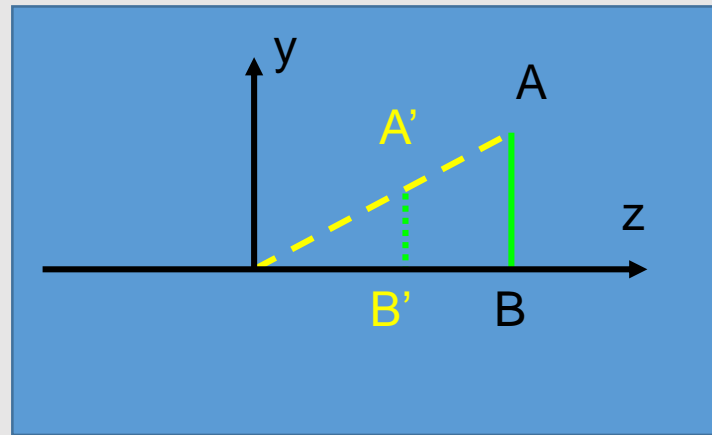
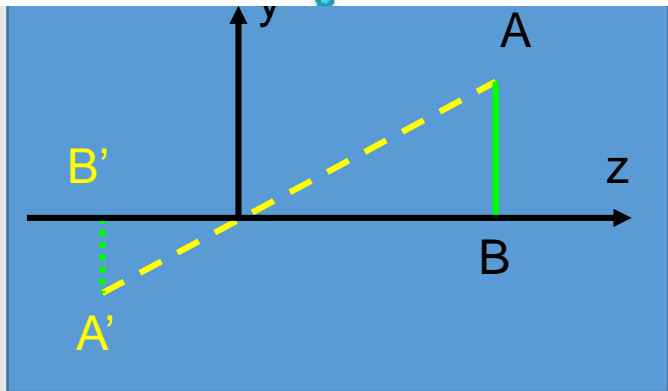
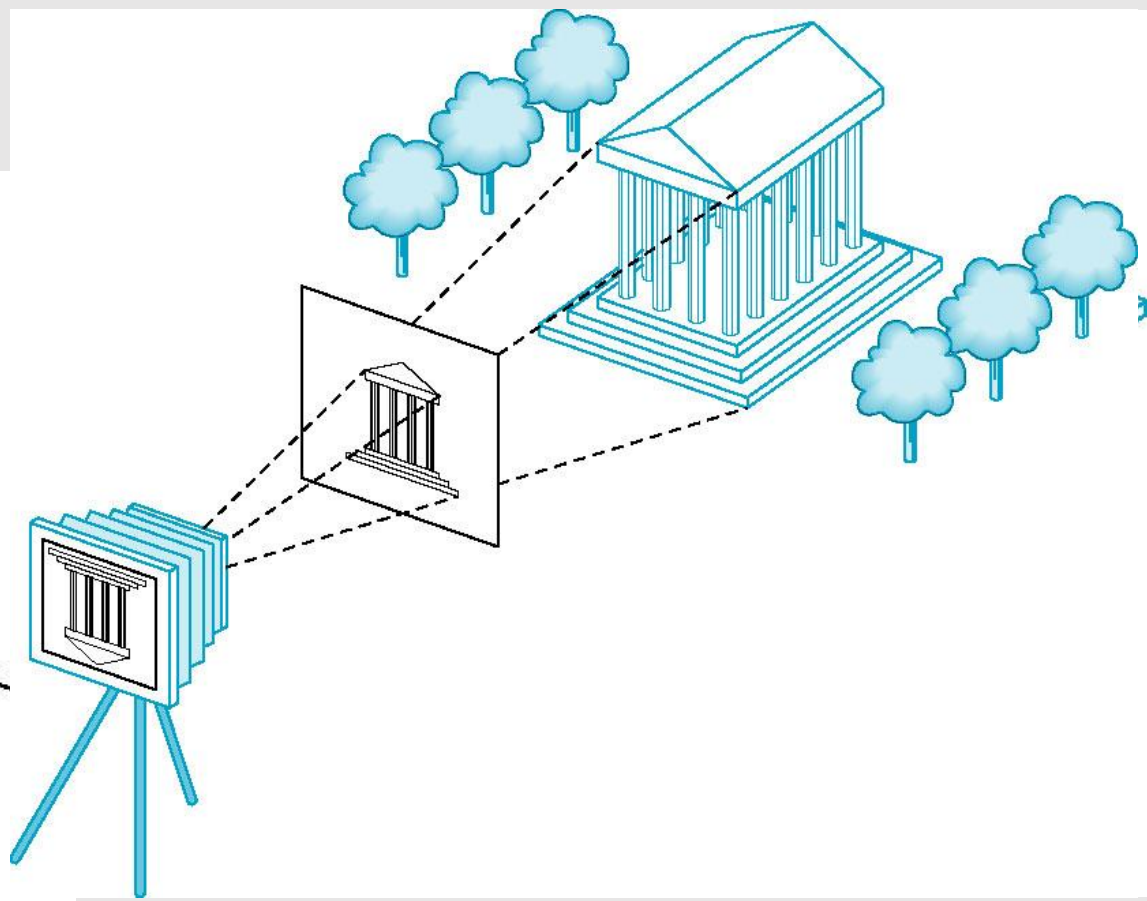
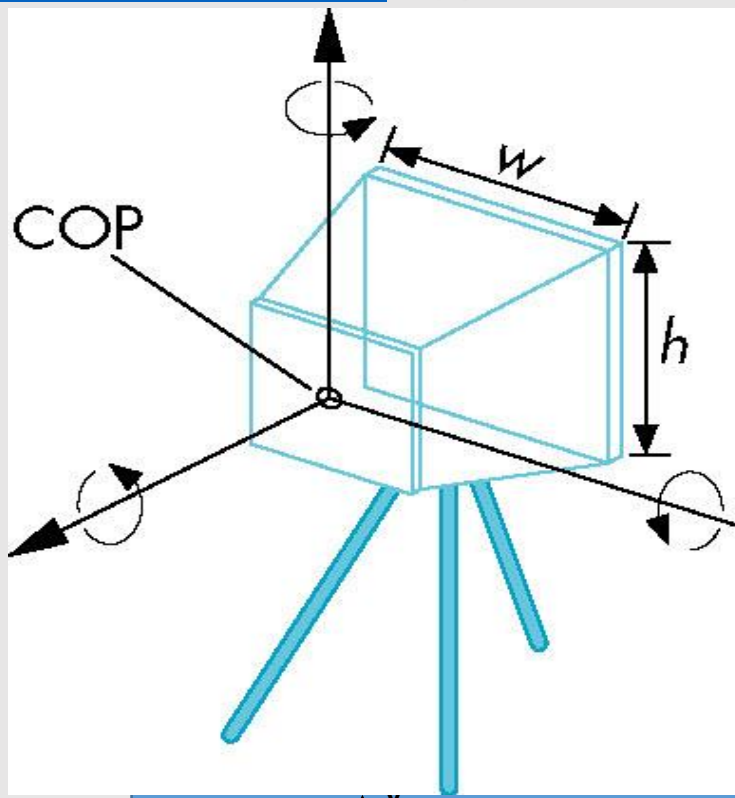
## 2.1.4 投影变换

2.1.4.1 概 述

2.1.4.2 透视投影

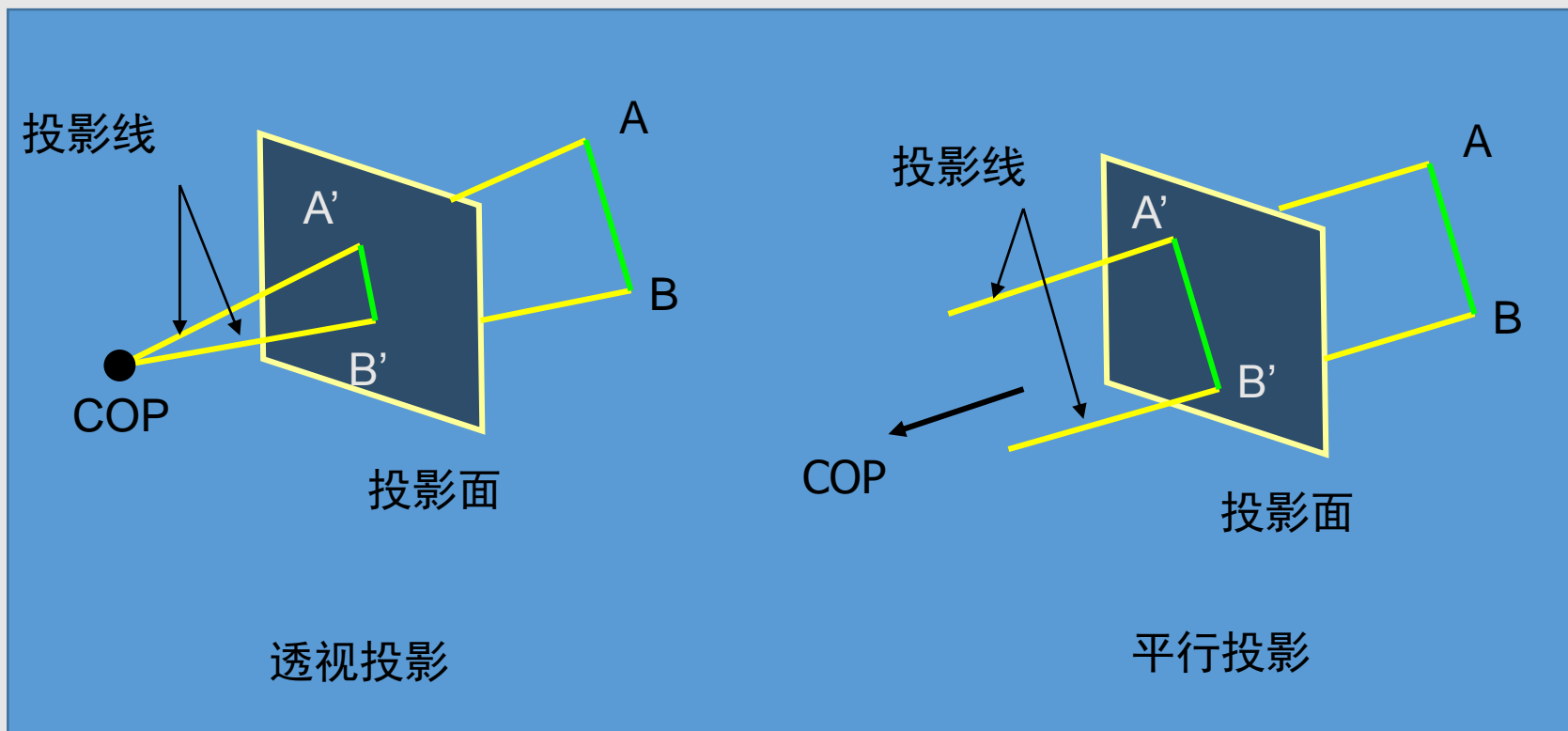
2.1.4.3 平行投影

## 2.1.4.1 概述

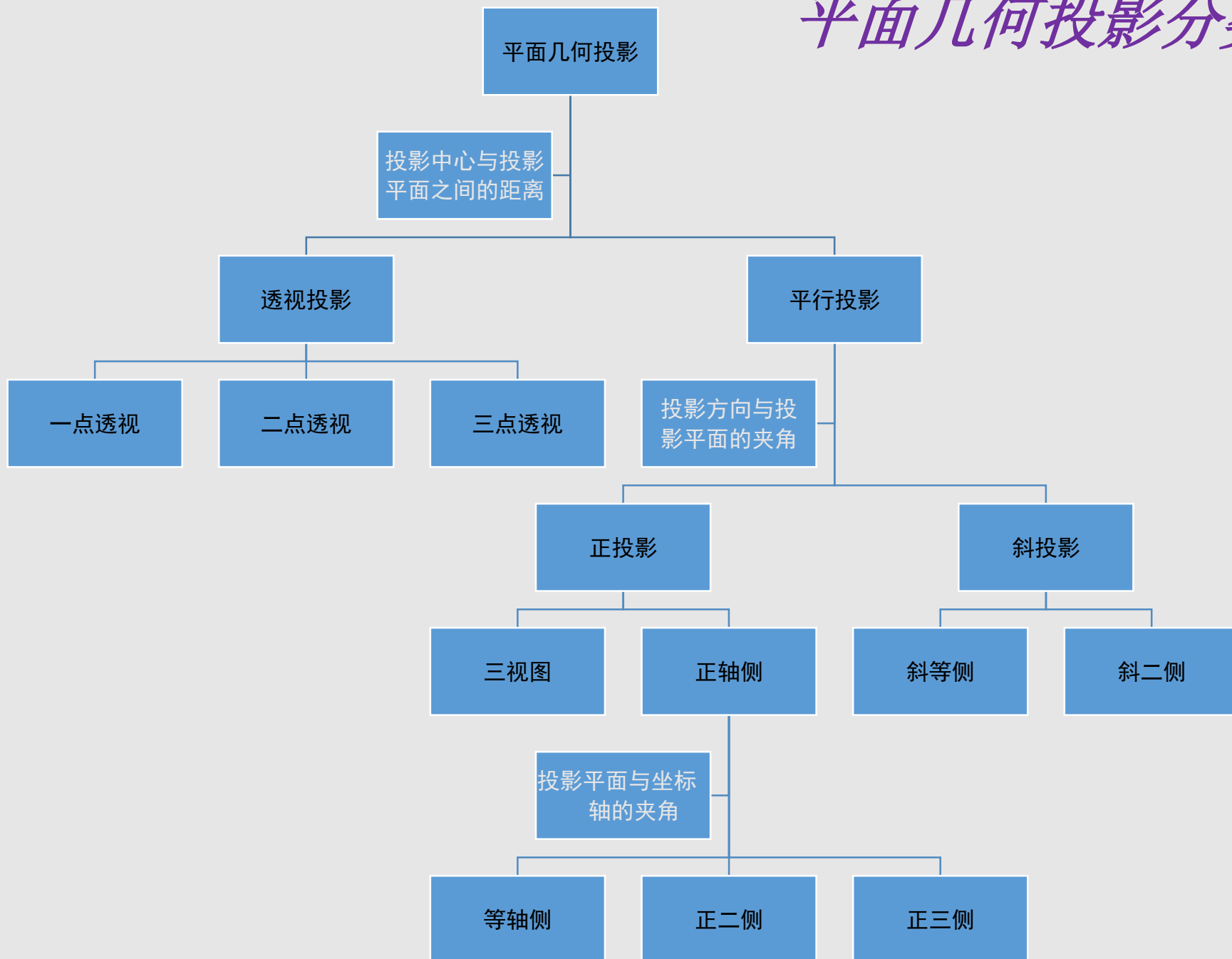


## 2.1.4.1 概述

- 根据投影中心(COP:Center of Projection)与投影平面之间的距离，投影变换可分为**平行投影**以及**透视投影**。



# 平面几何投影分类





## 2.1.4 投影变换

2.1.4.1 概 述

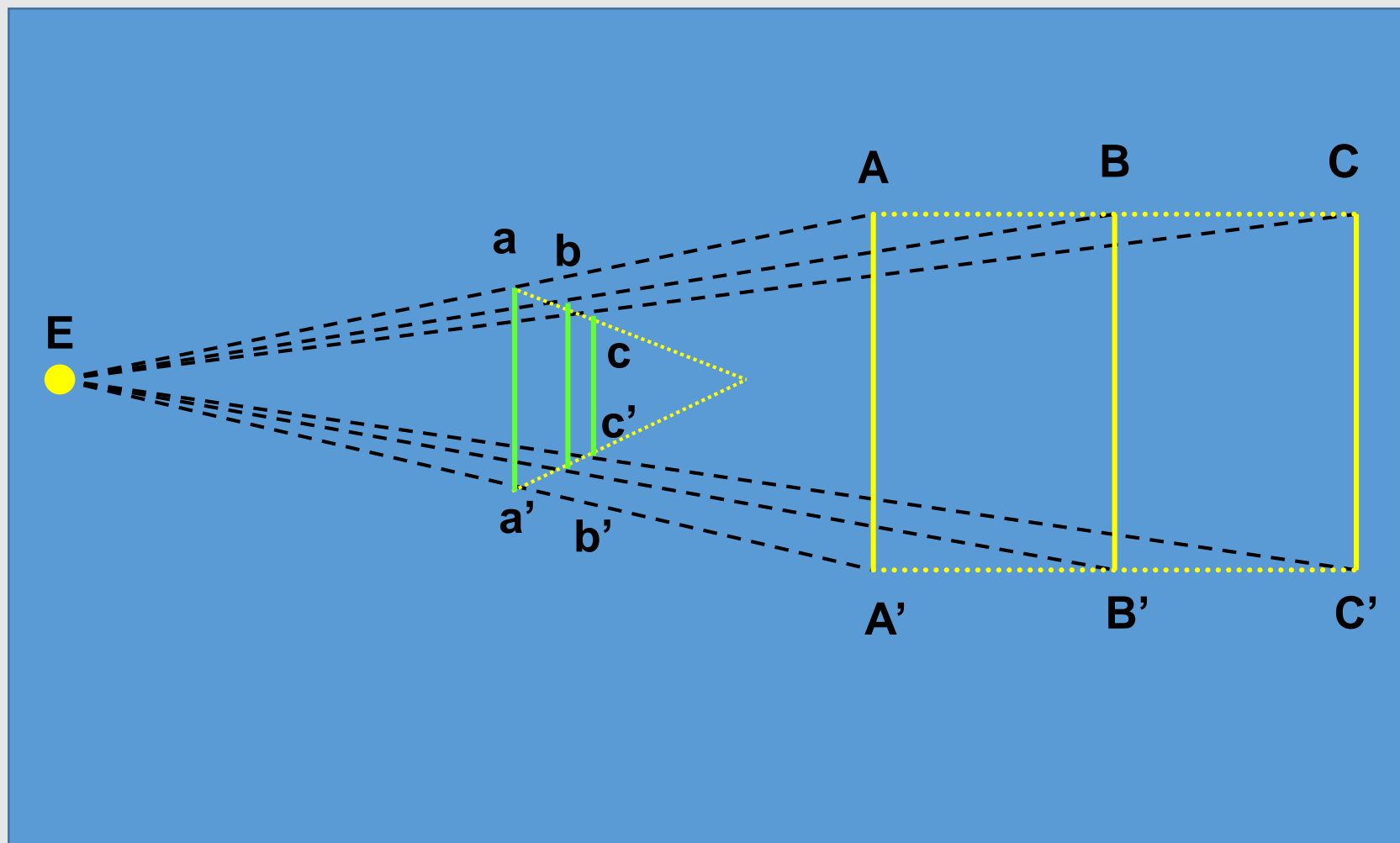
2.1.4.2 透视投影

2.1.4.3 平行投影



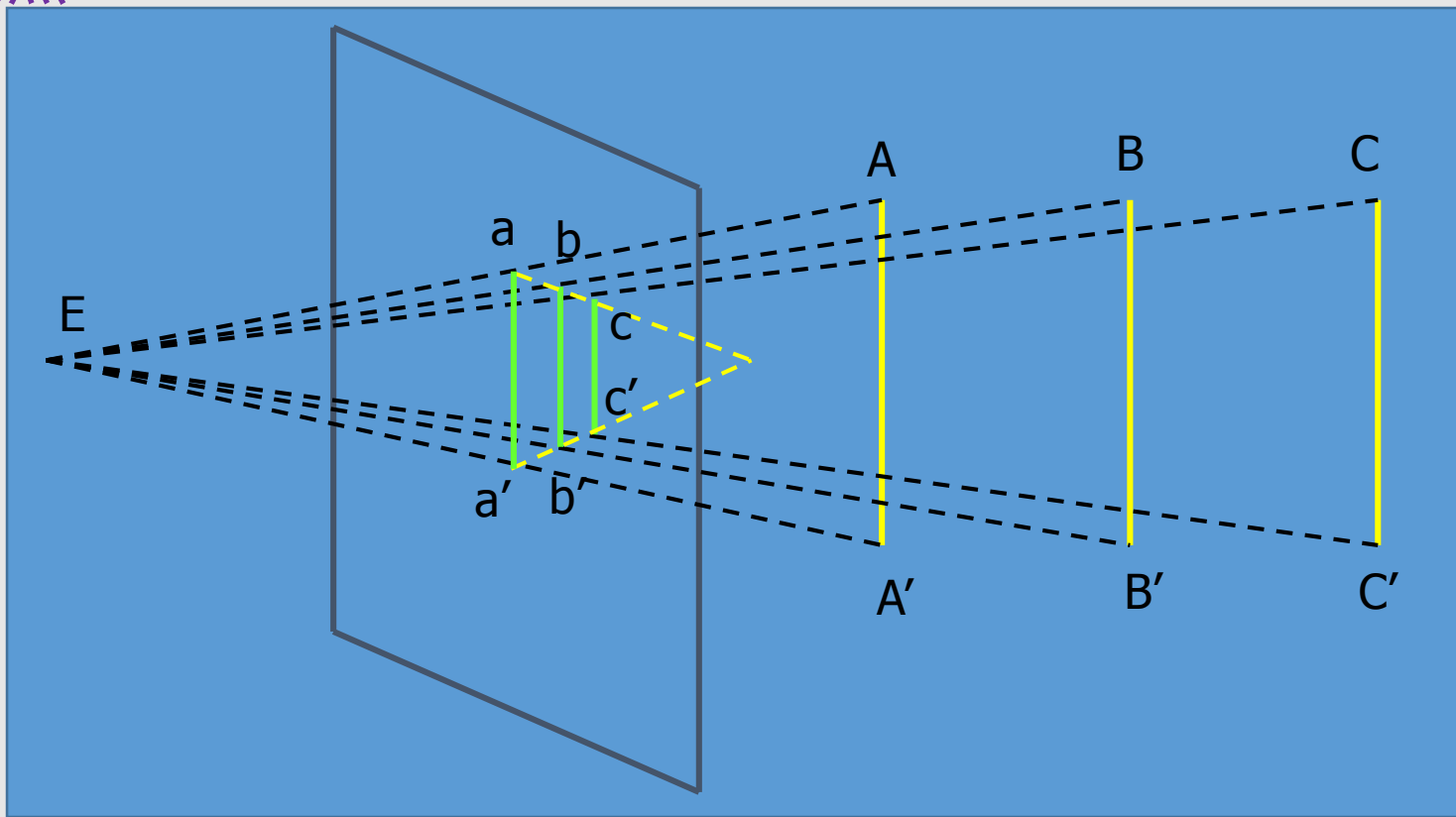
## 2.1.4.2 透视投影

- 产生透视的原因，可用下图来说明：



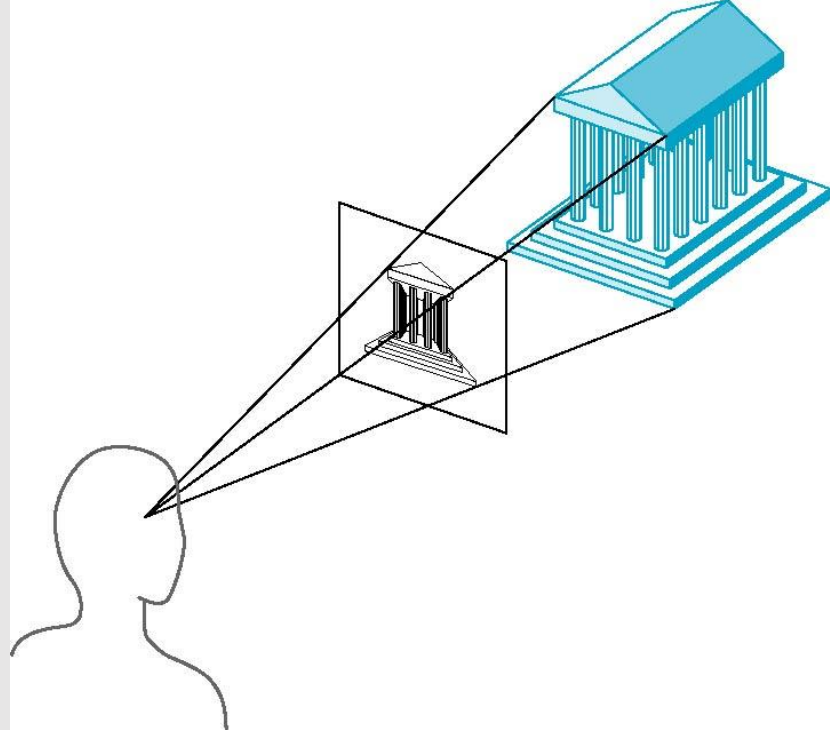
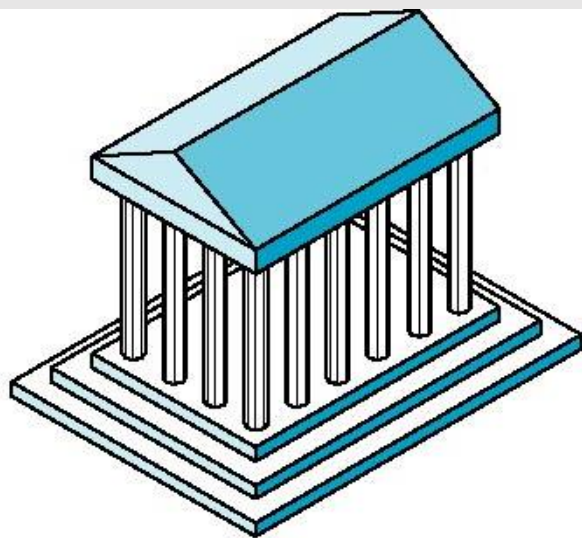
## 2.1.4.2 透视投影

- 若连接 $abc$ 及 $a'b'c'$ ，它们的连线汇聚于一点。
- 实际上， $ABC$ 与 $A'B'C'$ 的连线是两条互相平行的直线，这说明空间任何一束不平行于投影平面的平行线的投影将汇聚在一点，即 $abc$ 与 $a'b'c'$ 的连线，必交于一点，这点我们称之为灭点。



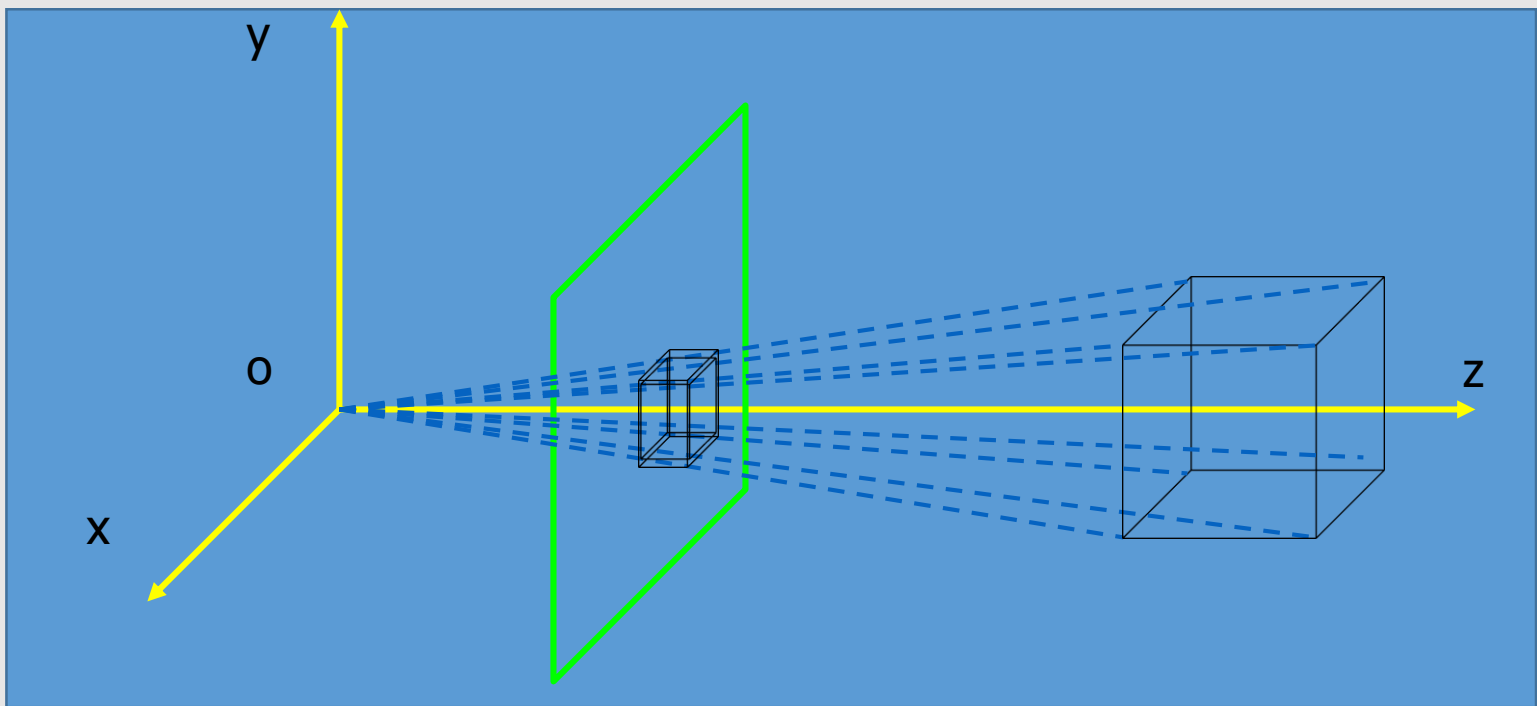


## 2.1.4.2 透视投影



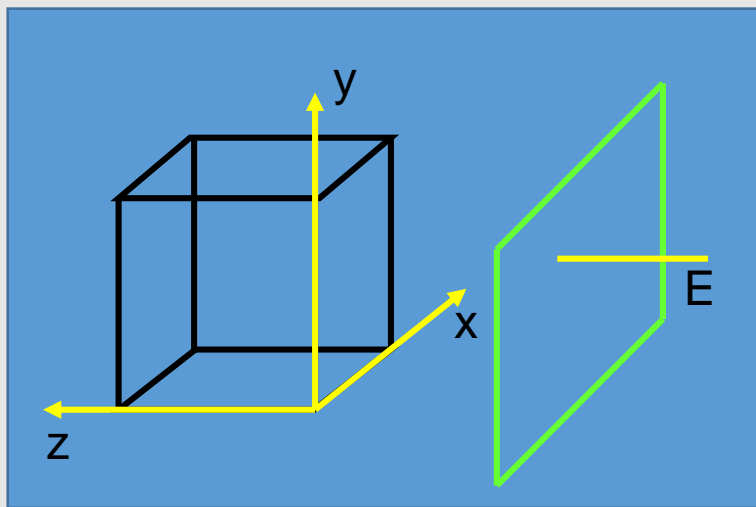
## 2.1.4.2 透视投影

- **主灭点**: 平行于坐标轴的平行线产生的灭点。
- **主灭点数**是和**投影平面切割坐标轴的数量**相对应的。
  - 若投影平面仅切割 $z$ 轴, 则 $z$ 轴是投影平面的法线。平行于 $xy$ 平面的直线也平行于投影平面, 因而没有灭点, 所以只在 $z$ 轴上有一个主灭点。

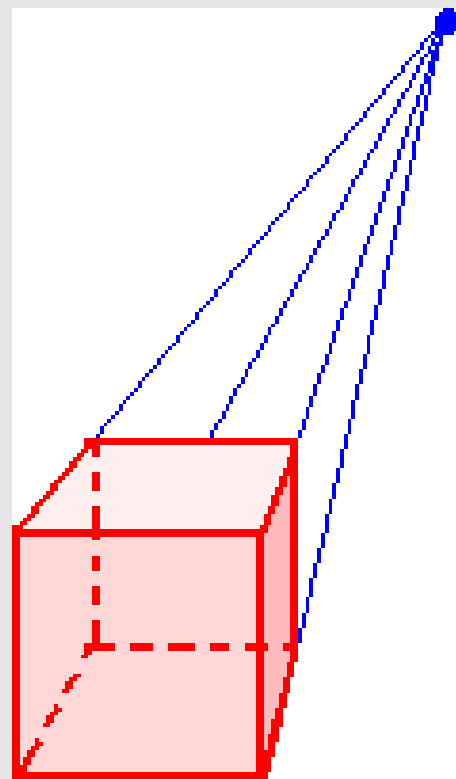


## 2.1.4.2 透视投影

- 透视投影的视线是从视点(观察点)出发，视线是不平行的，透视投影按照主灭点的个数分为一点透视，两点透视和三点透视。

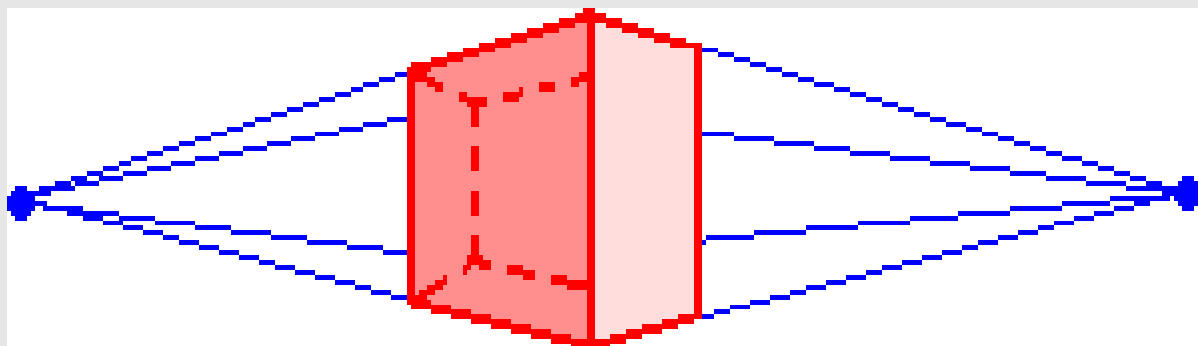
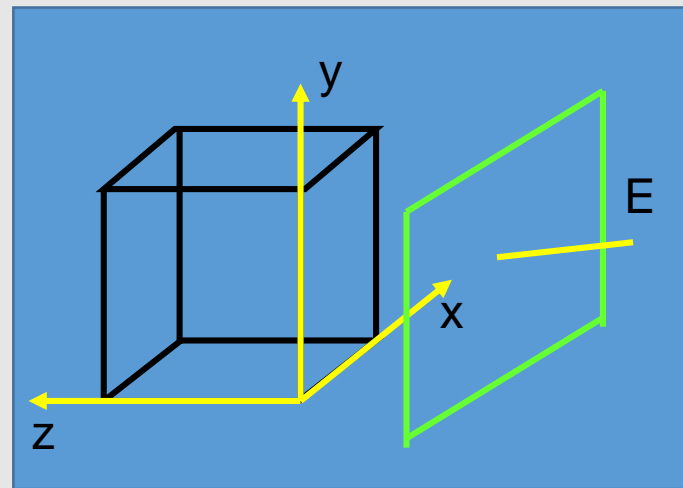


- 人眼从正面去观察一个立方体，当z轴与投影平面垂直时，另两根轴ox,oy轴平行于投影平面。这时的立方体透视图**只有一个灭点**，即与投影面垂直的那组平行线的透视投影交于一点。



## 2.1.4.2 透视投影—两点透视

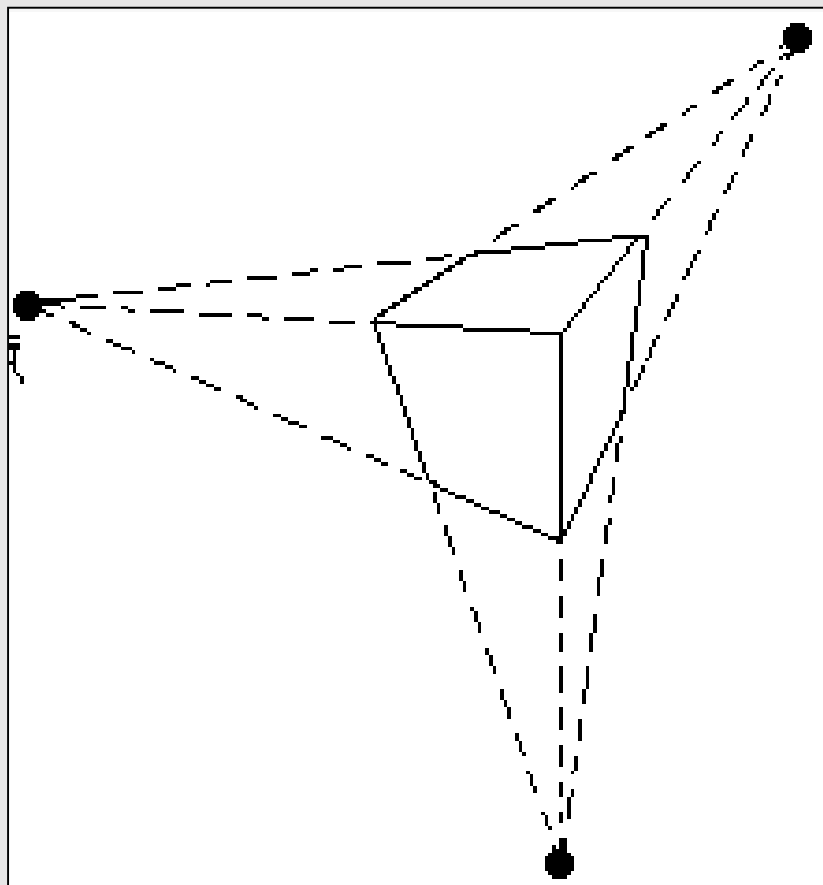
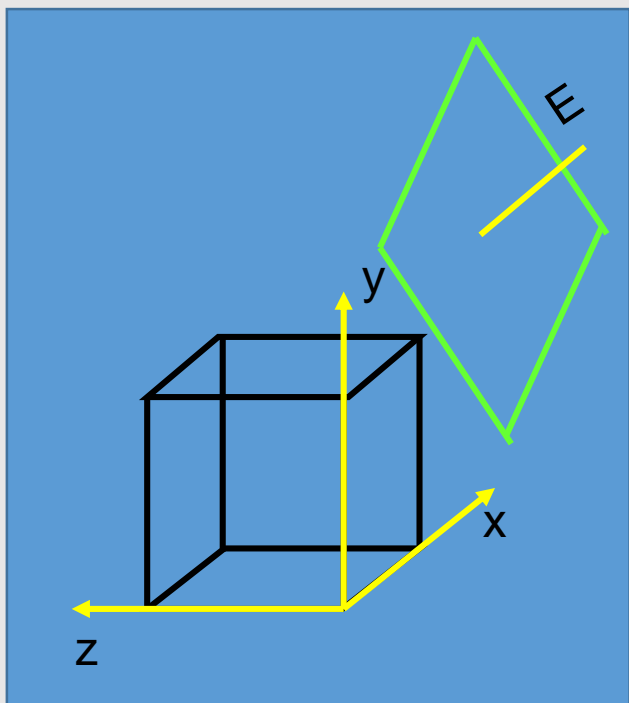
- 人眼观看的立方体是绕 $y$ 轴旋转一个角度之后，再进行透视投影。三坐标轴中 $oy$ 轴与投影平面平行，这时除平行于 $oy$ 轴的那组平行线外，其它两组平行线的透视投影分别在投影平面的左右两侧，作出的立方体透视图产生两个主灭点。





## 2.1.4.2 透视投影—三点透视

- 投影平面与三坐标轴均不平行。
- 这时的三组平行线均产生主灭点。



## 2.1.4.2 透视投影——一点透视

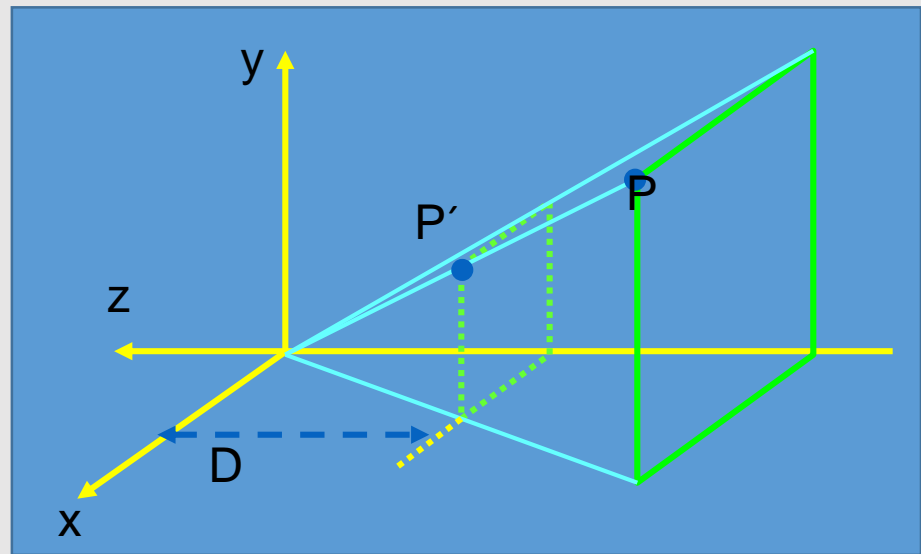
- 假设投影中心为原点，投影面平行于xy平面，方程为 $z=D$ 。
- 设物体上一点  $P(x_1, y_1, z_1)$ , 它的投影点为  $P'(x', y', z')$ , 则  $z'=D$ 。
- 投影线的方程为:

$$\begin{cases} x' = x_1 t \\ y' = y_1 t \\ z' = z_1 t \end{cases}$$

把 $z'=D$ 代入的，得  $t = D/z_1$ ,

则:

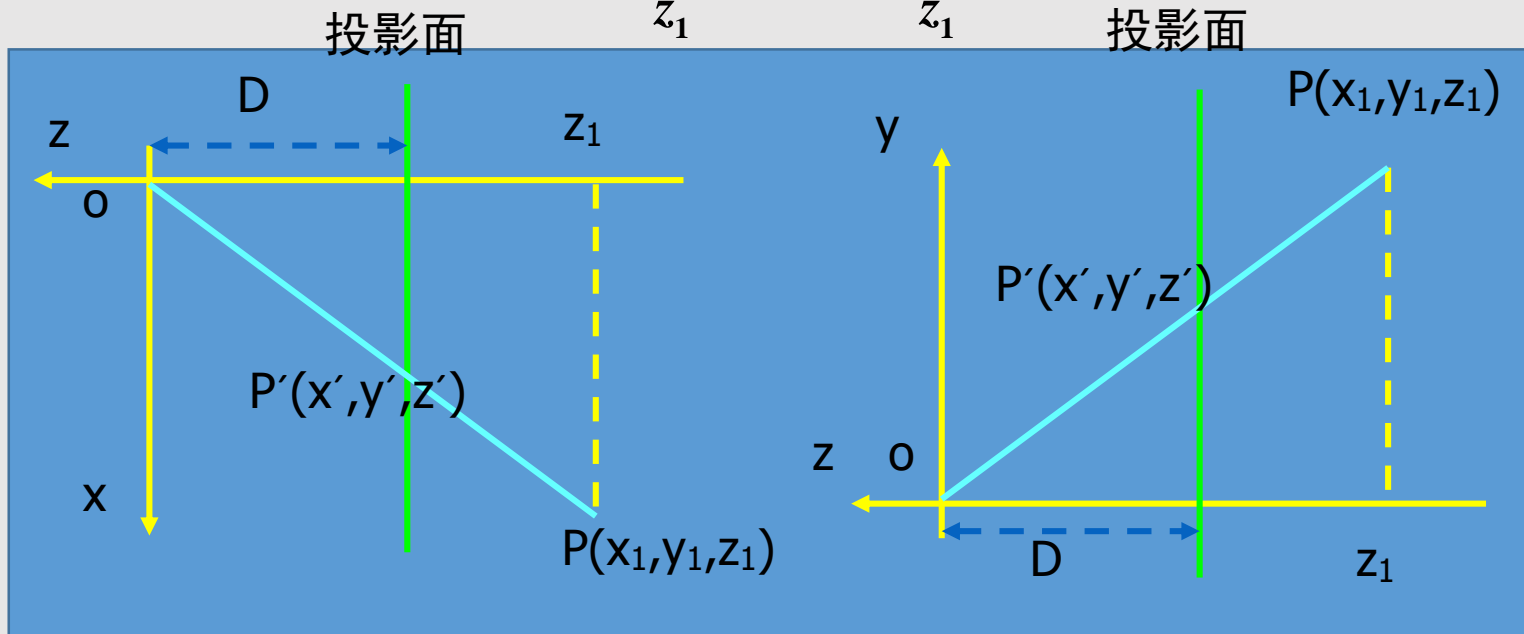
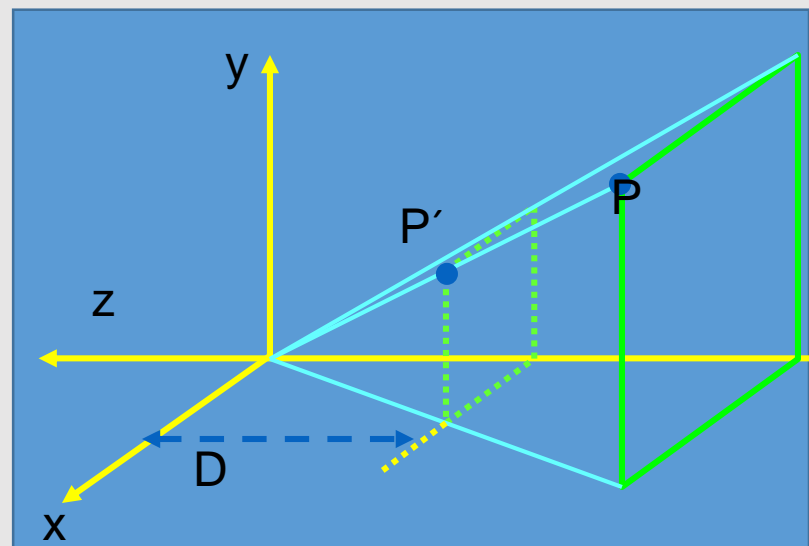
$$x' = \frac{x_1}{z_1} D ; y' = \frac{y_1}{z_1} D$$



## 2.1.4.2 透视投影——一点透视

- 若D固定,  $z_1 \uparrow$ , 则  $x' \downarrow$ ,  $y' \downarrow$ ;  
即对象越远, 成像越小。
- 若视点取无穷远( $D=\infty$ )时, 则:  
 $x' = x_1$ ,  $y' = y_1$ , 此时, 透视投影变为正平行投影。

$$x' = \frac{x_1}{z_1} D ; y' = \frac{y_1}{z_1} D$$



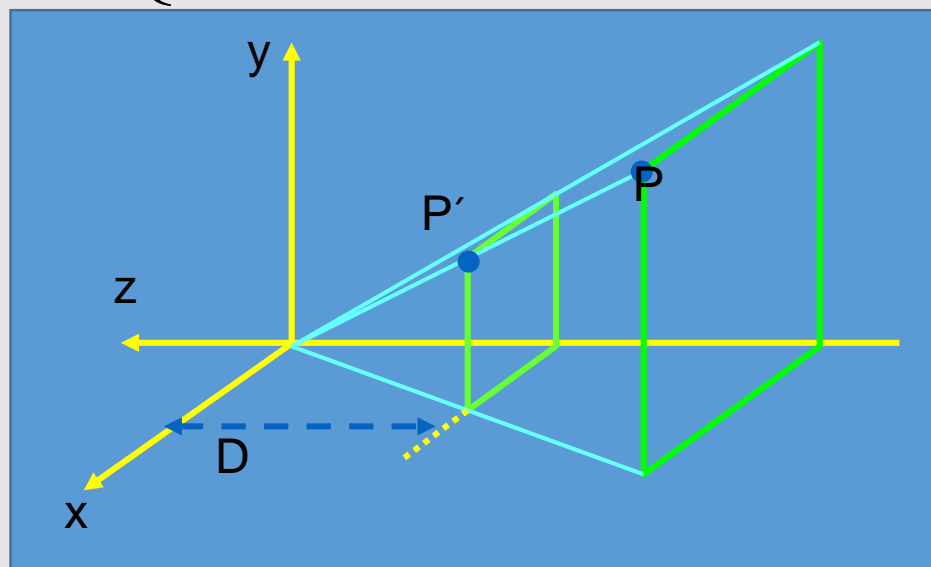
## 2.1.4.2 透视投影——一点透视

- 由上面分析，一点透视投影的变换矩阵为：

$$\begin{bmatrix} x' \\ y' \\ z' \\ z_1 / D \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/D & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

$$\begin{cases} x' = \frac{x_1}{z_1 / D} \\ y' = \frac{y_1}{z_1 / D} \\ z' = D \end{cases}$$

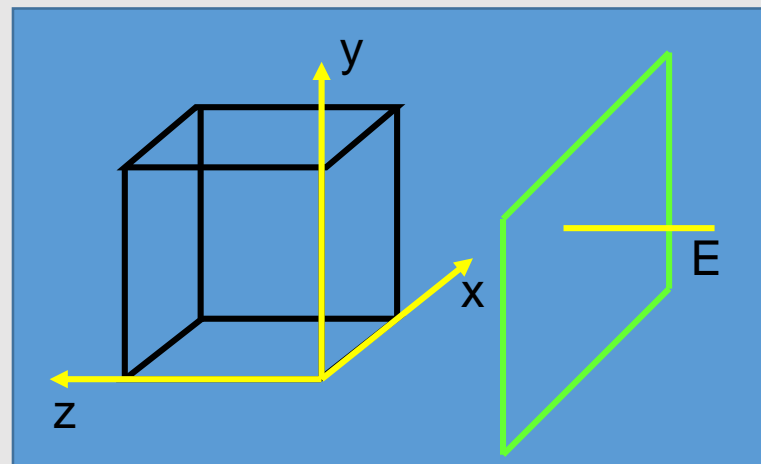
- 主灭点？



## 2.1.4.2 透视投影

- 在变换矩阵中，第四行的 $p, q, r$ 起透视变换作用；
  - 当 $p, q, r$ 中有一个不为0时即为一点透视投影。
  - 当 $p, q, r$ 中有两个不为0时的透视变换即为二点透视变换。
  - 若 $p, q, r$ 都不为0，则可得到有三个灭点的三点透视。

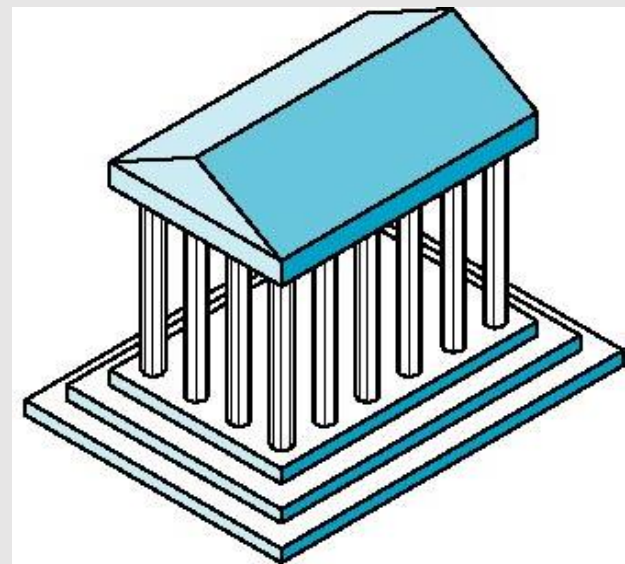
$$M_{Per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & r & 0 \end{bmatrix}$$



- 旋转物体，获得两点、三点透视图。

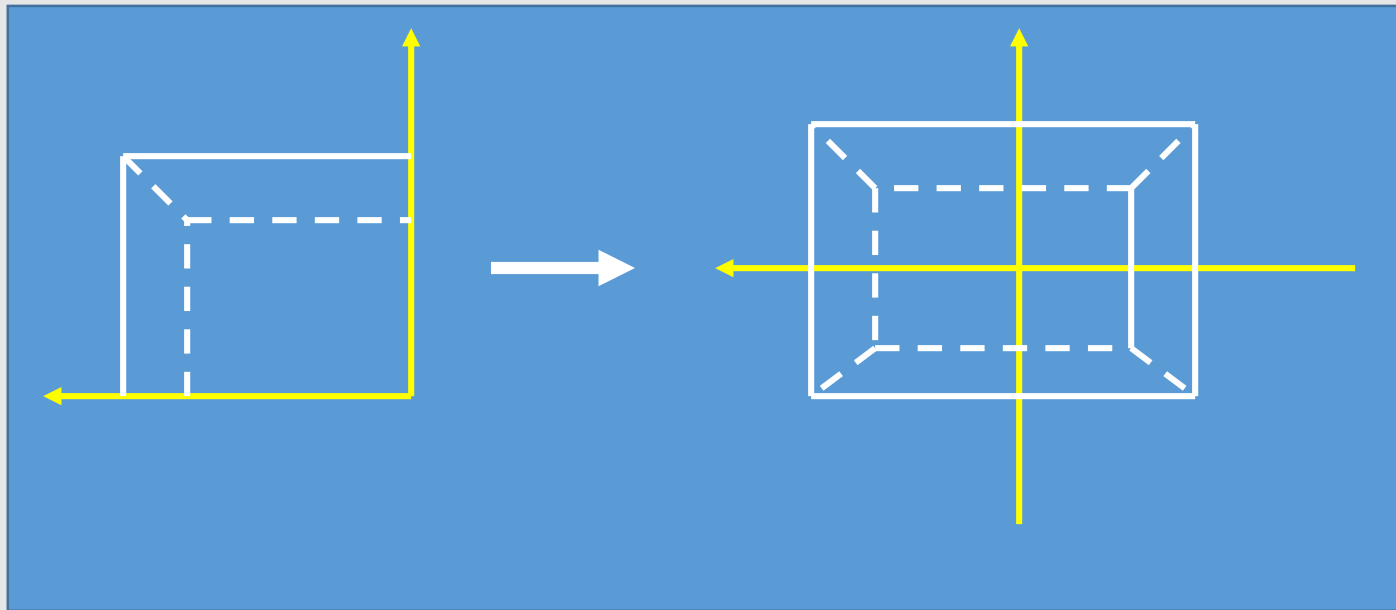
## 2.1.4.2 透视投影

- 具有**真实感**。
- 与投影面不平行的平面中，线条的**平行关系**、**长度**、及**角度**会发生改变。



## 2.1.4.2 透视投影—透视投影的技巧

- 在生成一点透视图时，为了**避免将物体安置在坐标系原点**，而产生左图所示的透视效果，通常在透视变换前，先将立体作一平移变换。
- 对于两点或三点透视，也可利用平移、旋转等操作来获得较好的投影效果。



## 2.1.4 投影变换

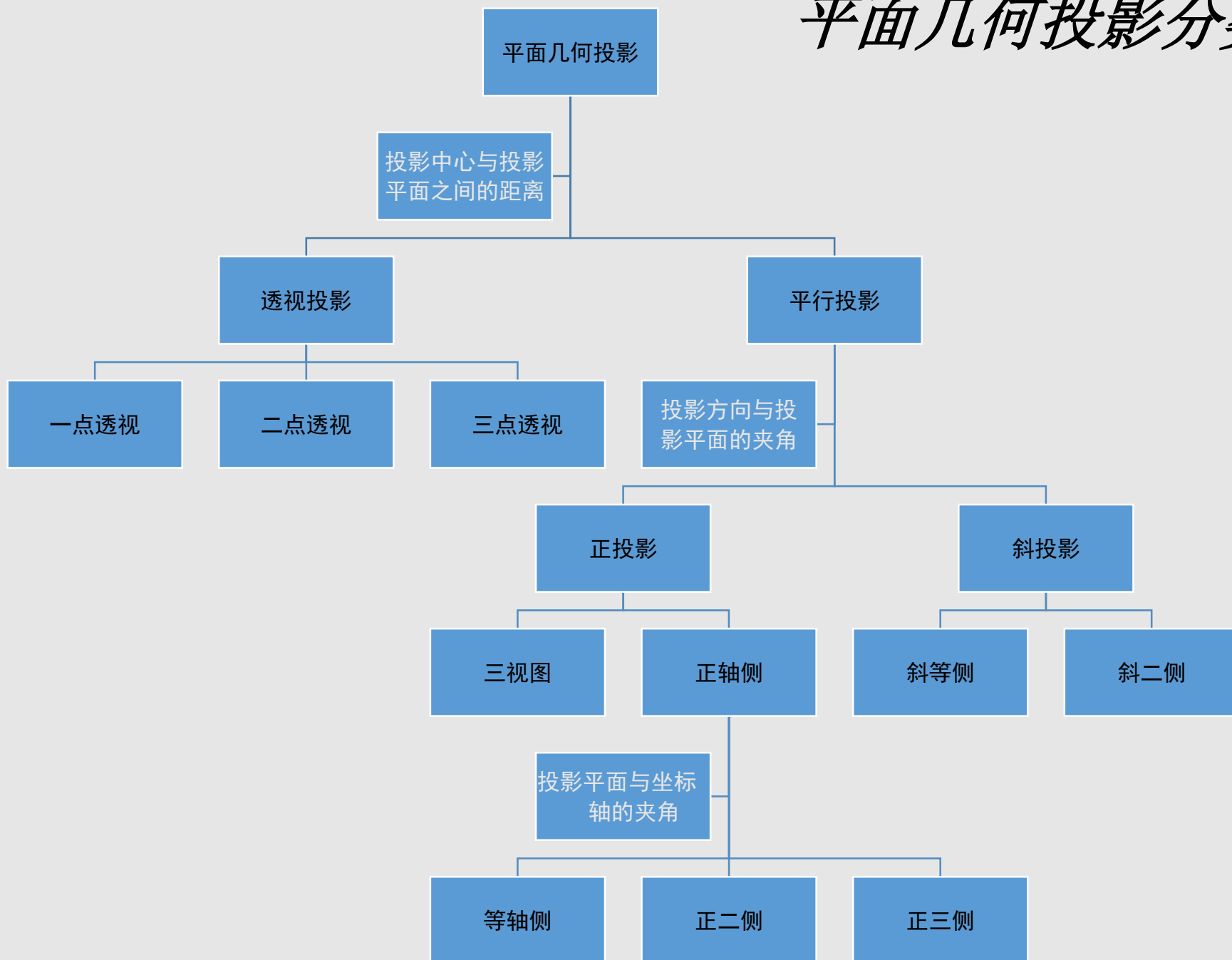
2.1.4.1 概 述

2.1.4.2 透视投影

2.1.4.3 平行投影



# 平面几何投影分类



## 2.1.4.3 平行投影

■ 根据投影线是否垂直于投影平面，平行投影可分为：

■ 正平行投影

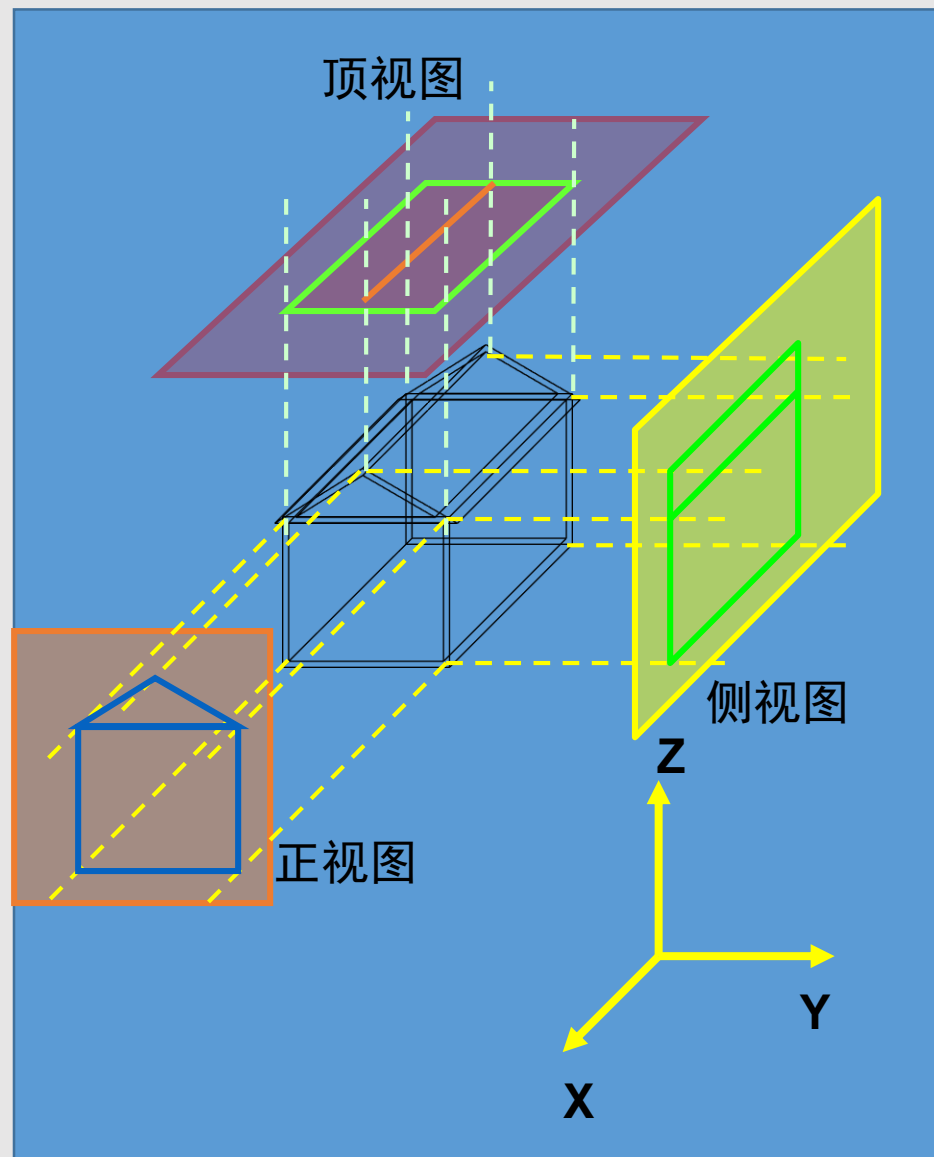
■ 三视图

■ 正轴侧投影

■ 斜平行投影

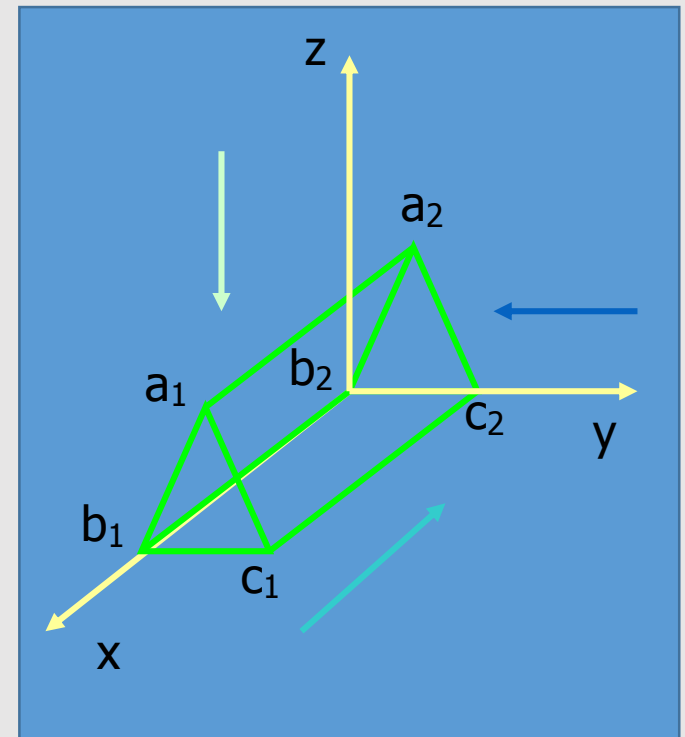
■ 斜等侧

■ 斜二侧



# 1. 正平行投影 — 三视图

- 正平行投影：投影方向垂直于投影平面时称为正平行投影。
  - 三视图
  - 正轴侧投影
- 三视图：三个投影面和坐标轴相互垂直。
- 三视图的生成就是把 $x$ 、 $y$ 、 $z$  坐标系下的形体投影到 $x=0$ ， $y=0$ ， $z=0$ 的平面，一般还需将三视图在一个平面上画出（再变换到 $u$ 、 $v$ 、 $w$  坐标系）。

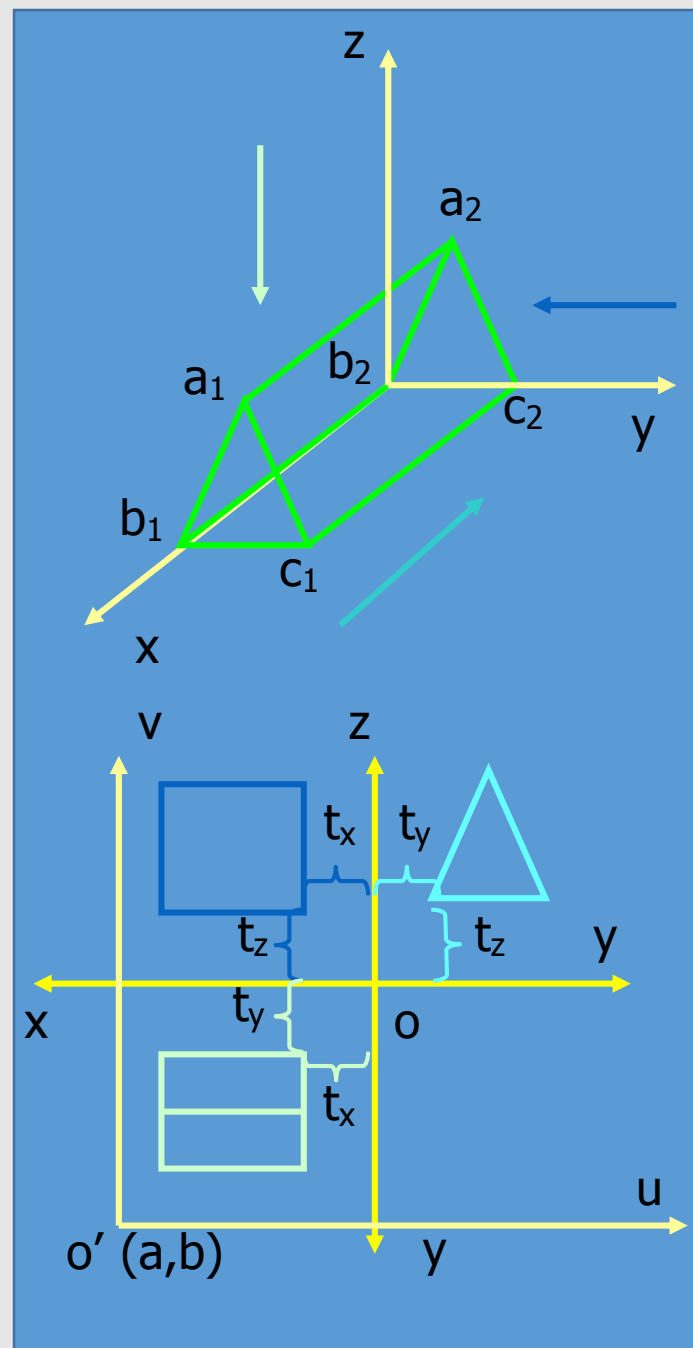
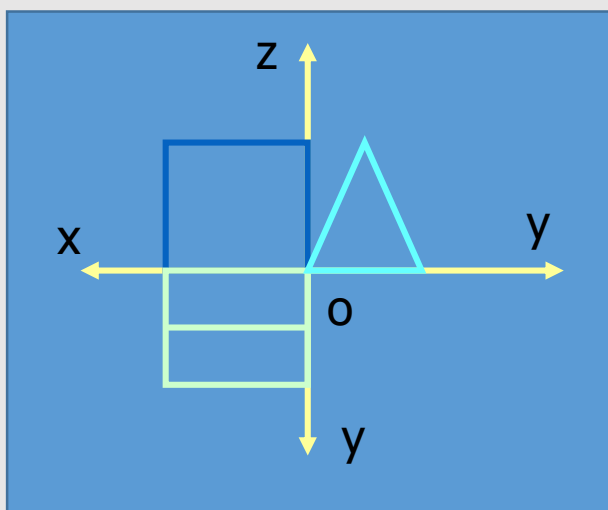


# 1. 正平行投影 — 三视图

- 变换矩阵:

正视图  
(y方向)

$$\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & a-t_x \\ 0 & 0 & 1 & b+t_z \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

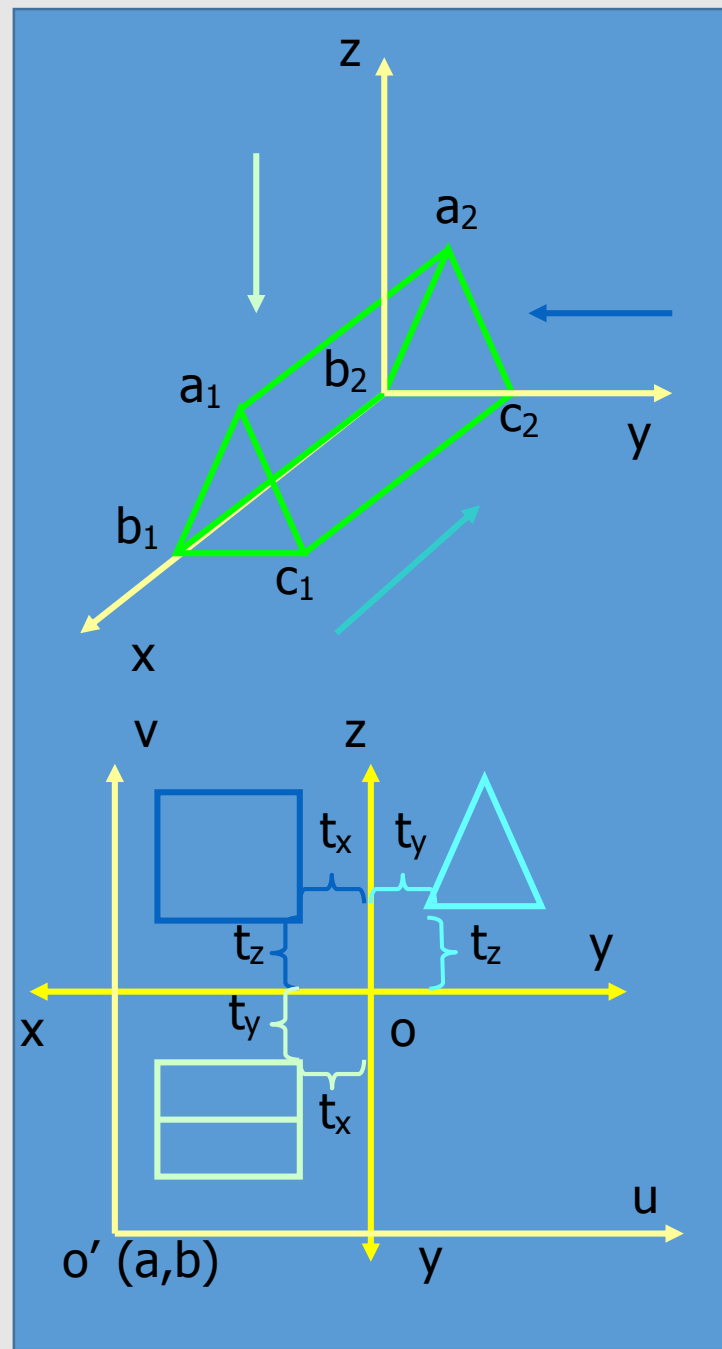
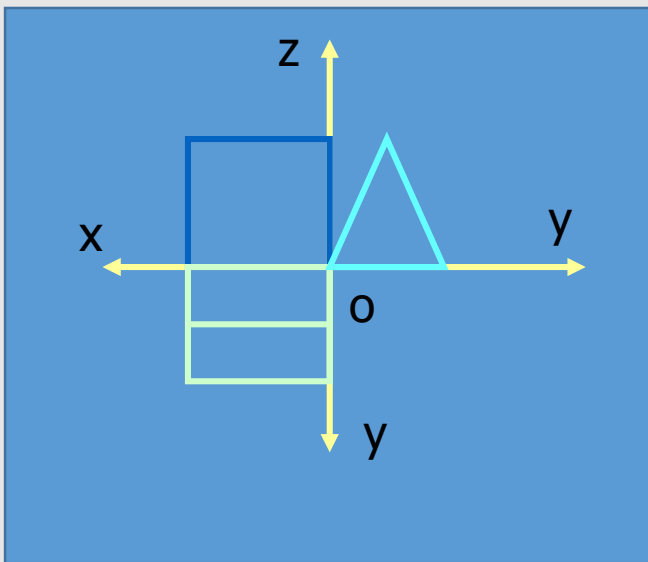


# 1. 正平行投影 — 三视图

- 变换矩阵:

顶视图  
(z方向)

$$\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & a-t_x \\ 0 & -1 & 0 & b-t_y \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

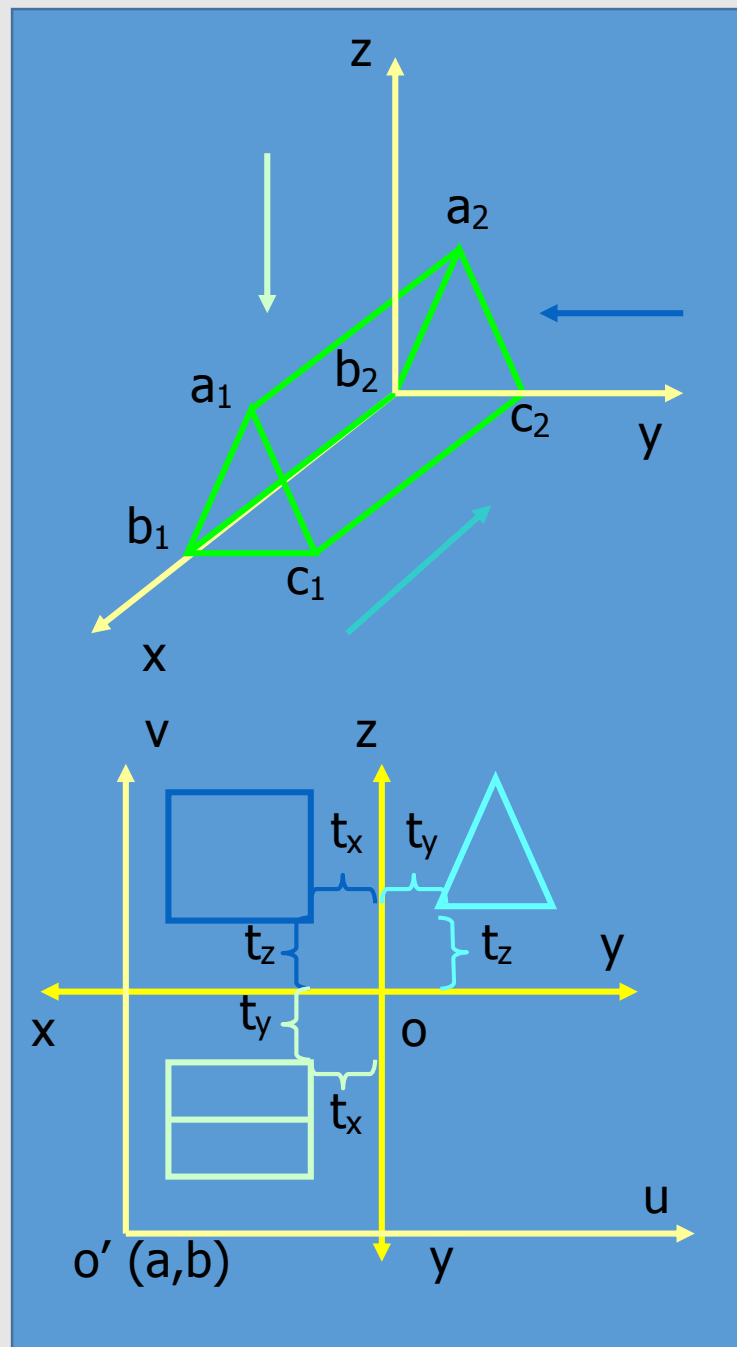
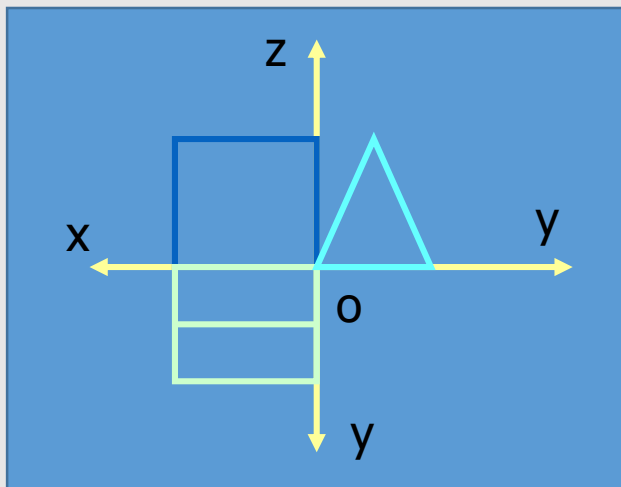


# 1. 正平行投影 — 三视图

- 变换矩阵:

侧视图  
(x方向)

$$\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & a+t_y \\ 0 & 0 & 1 & b+t_z \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



# 1. 正平行投影 — 三视图

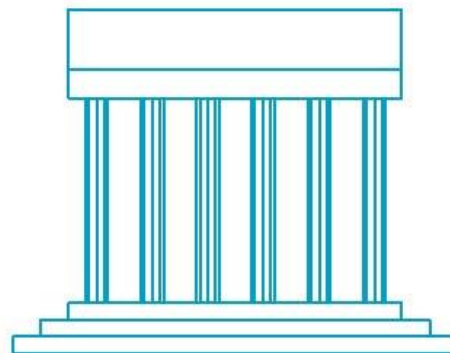
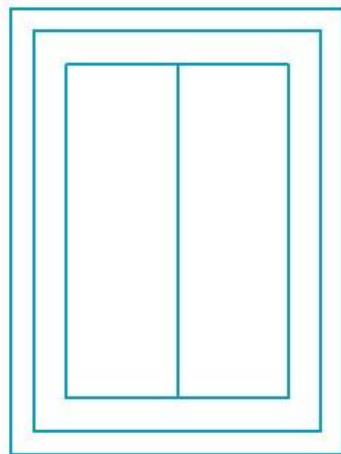
- 优点:

保留了长度、  
角度及物体的形状。



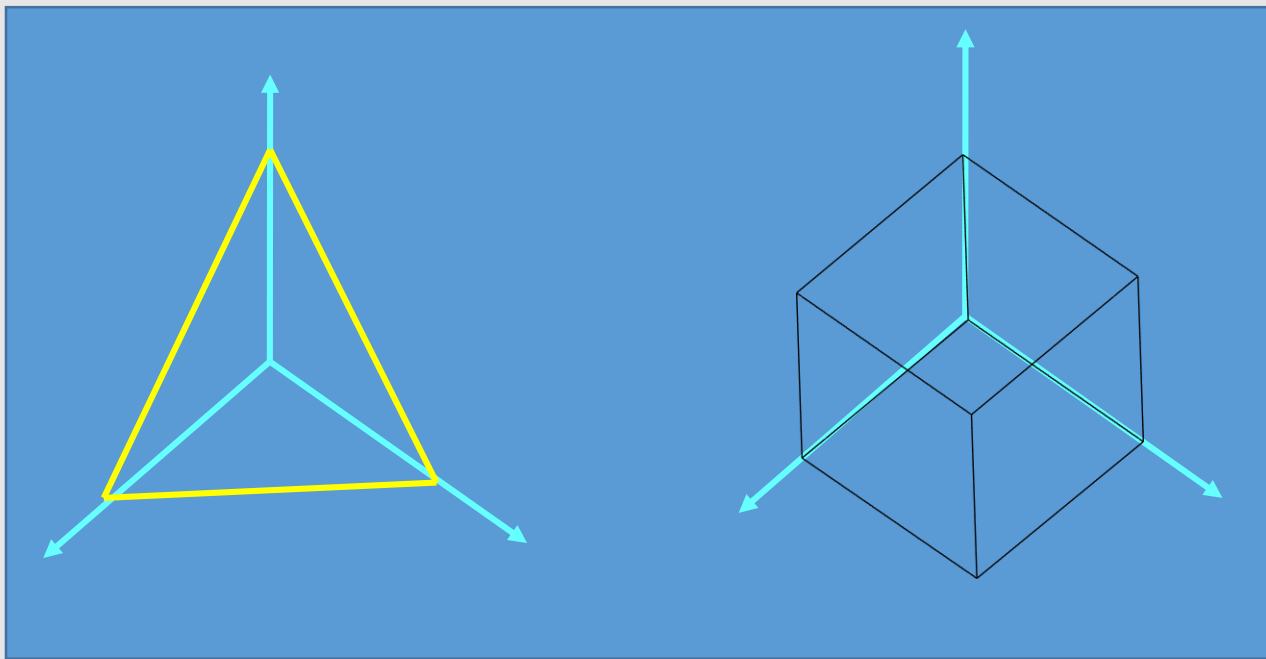
- 缺点:

没有立体感。  
(丢失了许多面的  
信息)



# 1. 正平行投影 — 正轴侧投影

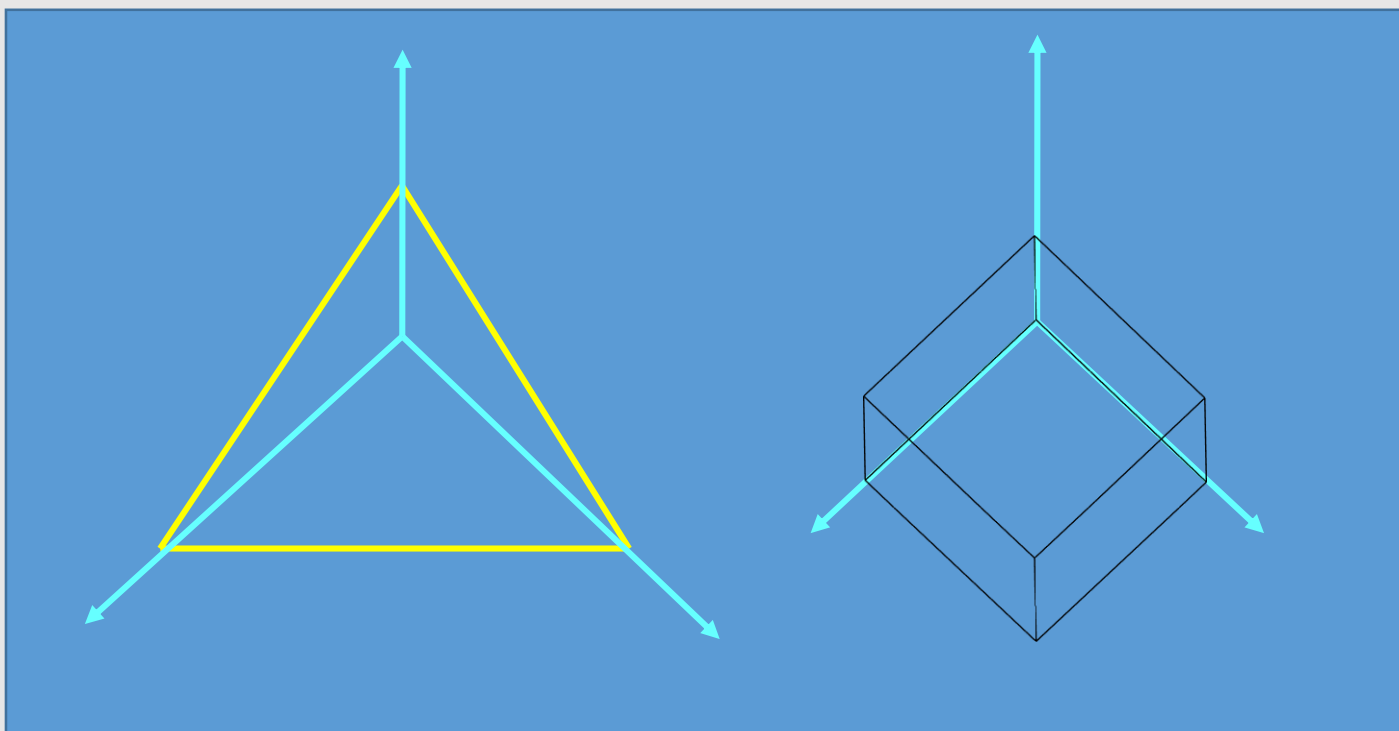
- 当投影方向不取坐标轴方向，投影平面不垂直于坐标轴时，产生的正投影称为正轴测投影。
- 正轴测投影分类：正等侧、正二侧、正三侧。
  - 正等侧：投影平面与三个坐标轴的交点到坐标原点的距离都相等。沿三个轴线具有相同的变形系数。





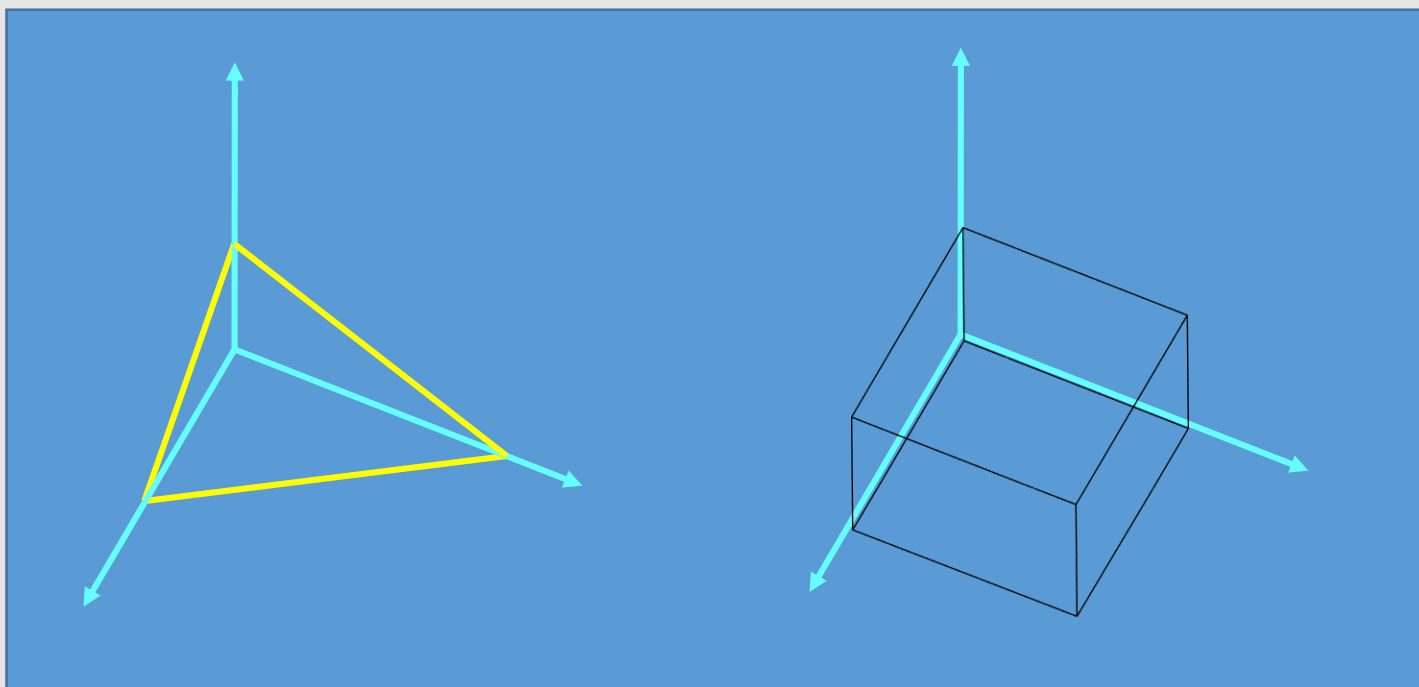
# 1. 正平行投影 — 正轴侧投影

- **正二侧**：投影平面与**两个坐标轴**的交点到坐标原点的距离**都相等**。沿两个轴线具有相同的变形系数。



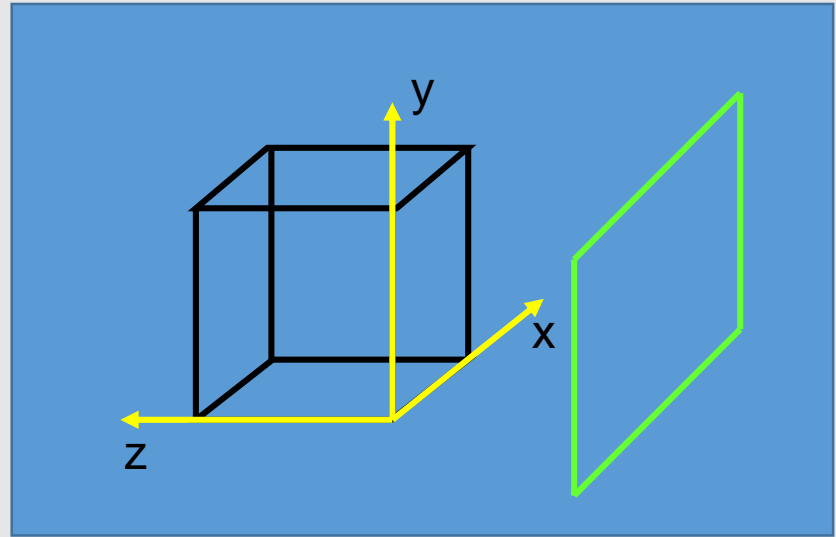
# 1. 正平行投影 — 正轴侧投影

- **正三侧：** 投影平面与三个坐标轴的交点到坐标原点的距离都不相等。沿三个轴线具有各不相同的变形系数。

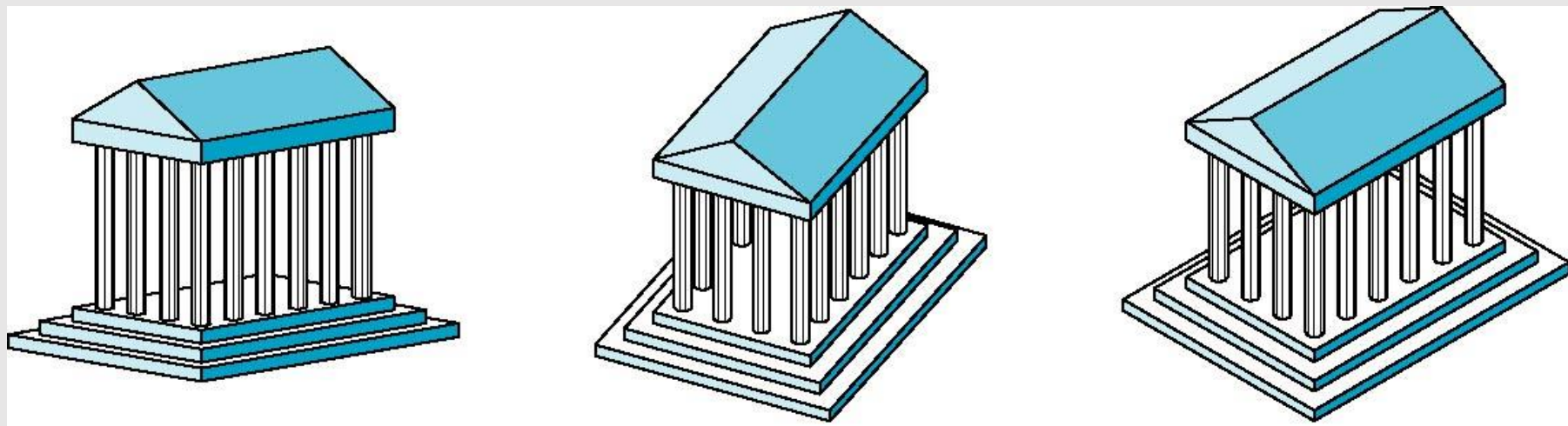


# 1. 正平行投影 — 正轴侧投影

- 如何获得正轴侧投影的变换矩阵？
  - 利用三视图。先将物体绕坐标轴旋转，再沿 $z$ 轴方向进行投影。



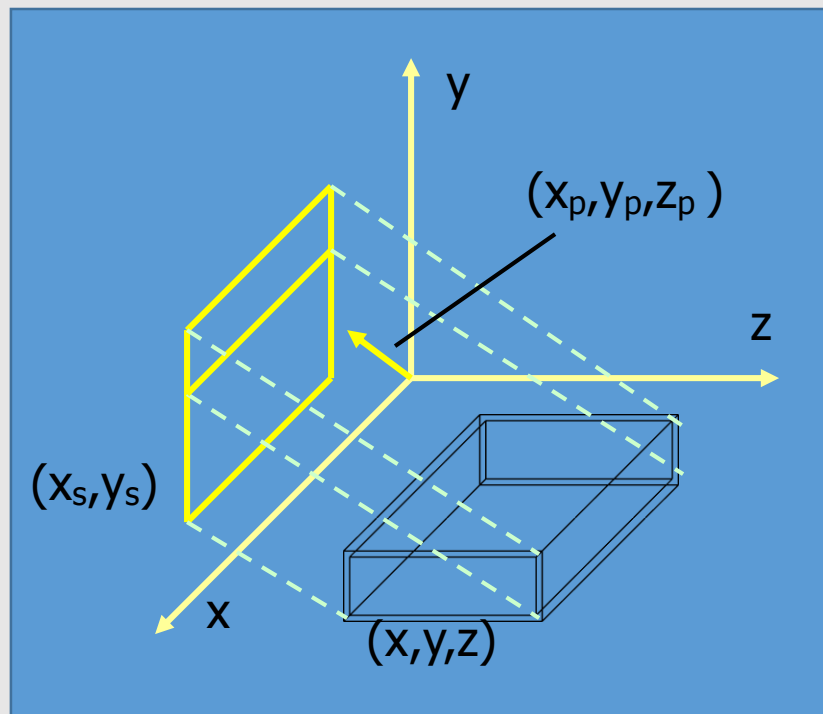
# 1. 正平行投影 — 正轴侧投影



- 由于这种投影的投影平面不与立体的轴线垂直，同时可见到物体的多个面，因而可产生立体效果。
- 经过正轴侧投影变换后，物体线间的平行性不变，但角度有变化。线条比例发生变化。
- 真实感差。（没有远小近大的效果）

## 2. 斜平行投影

- 如果**投影方向**不垂直于**投影平面**，则称为**斜平行投影**。
  - 斜等侧：投影平面与一坐标轴垂直；投影线与投影平面成  $45^\circ$  角。与投影平面垂直的线投影后长度不变。
  - 斜二侧：投影平面与一坐标轴垂直；投影线与该轴夹角成  $\text{arccotg}(1/2)$  角。该轴轴向变形系数为  $\frac{1}{2}$ 。即与投影平面垂直的线投影后长度变为原来的一半。
- 通过**不同角度强调**物体的某一面。
- 在斜平行投影中，投影平面一般取坐标平面。



## 2.1 三维空间变换

2.1.1 变换的数学基础

2.1.2 几何变换

2.1.3 坐标变换

2.1.4 投影变换

2.1.5 图形的显示流程

2.1.6 OpenGL中的变换

## 2.1.5 图形的显示流程

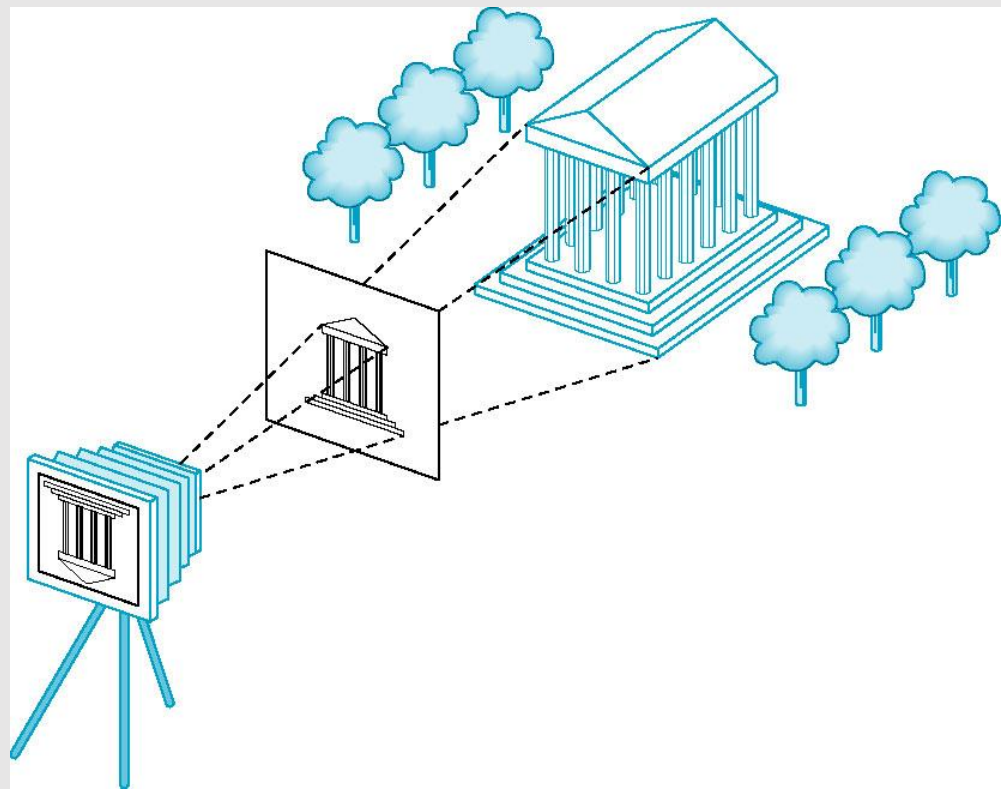
2.1.5.1    概 述

2.1.5.2    视变换

2.1.5.3    窗口－视区变换

## 2.1.5.1 概述

- 前面我们推导了各种经典视图的投影变换矩阵。
- 如何通过计算机构建虚拟场景，并生成投影图？

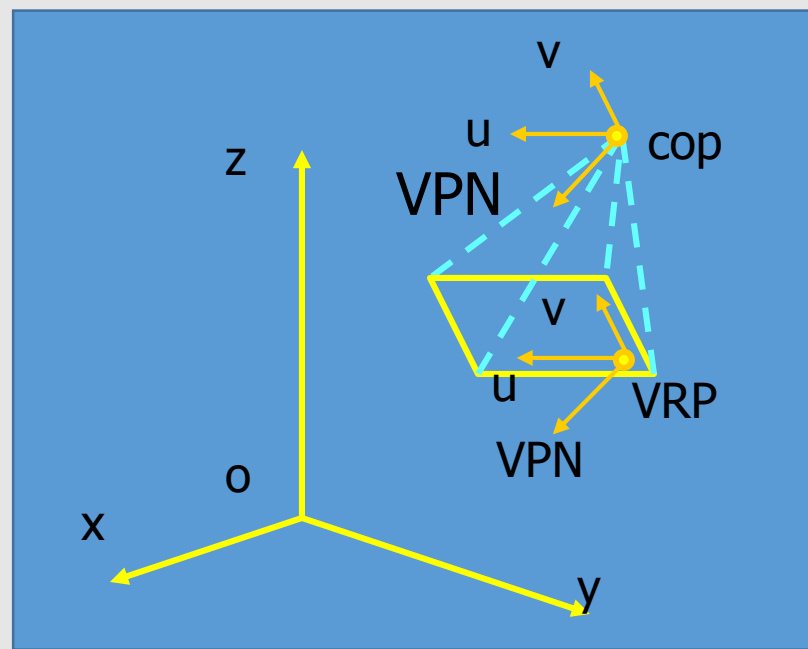
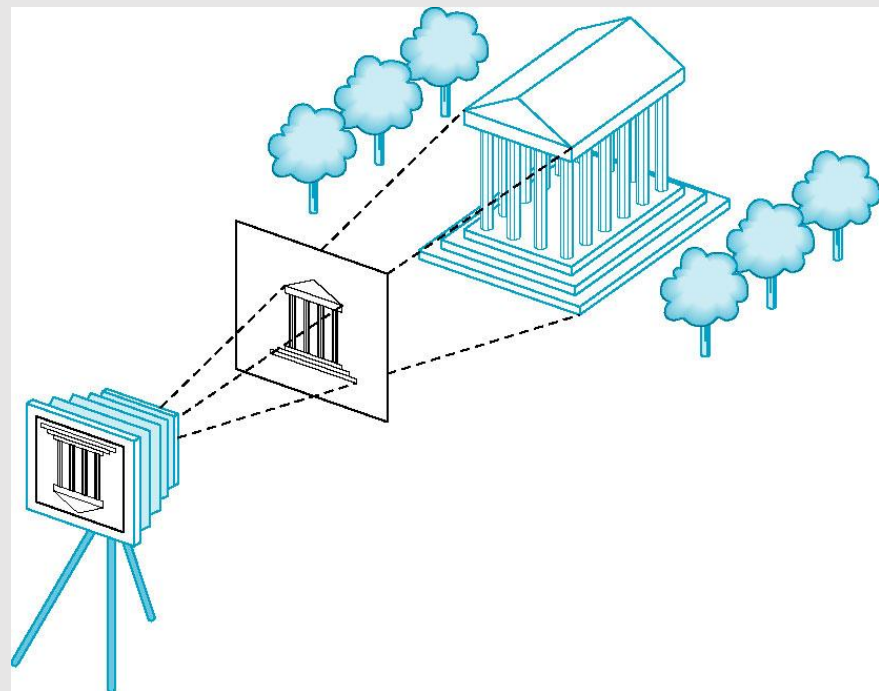


- 前面的推导中，**物体**和**相机**在同一个坐标系中定义；
- 如果需要**多角度**观察物体，每次都需要**重新推导**变换矩阵，太复杂！
- 将物体和相机用**各自的坐标系定义**(将对象和观察者分离)，利用坐标系的变换来生成不同的投影视图。



## 2.1.5.1 概述

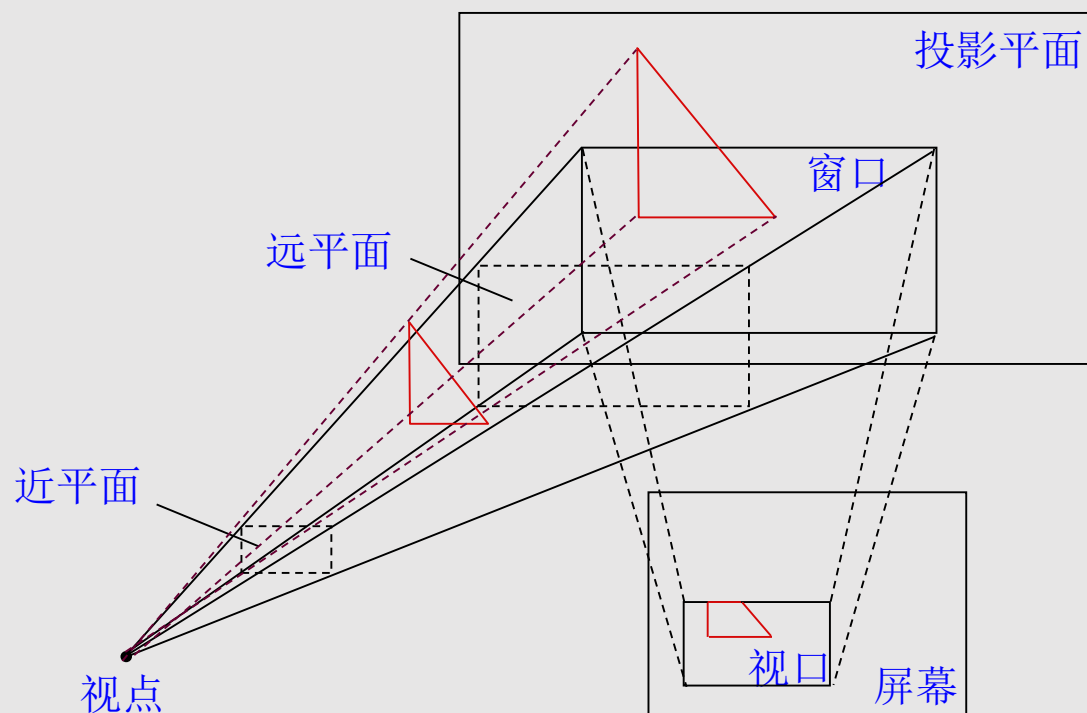
- 在xoy坐标系(世界坐标系)中建立场景;
- 根据需要, 建立一个视坐标系(观察坐标系);
- 显示过程主要就是将场景中的物体从世界坐标系变换到视坐标系, 然后再投影到视平面。



## 2.1.5.1 概 述

### 图形的显示流程:

模型变换→投影变换→裁减→视口变换



## 2.1.5 图形的显示流程

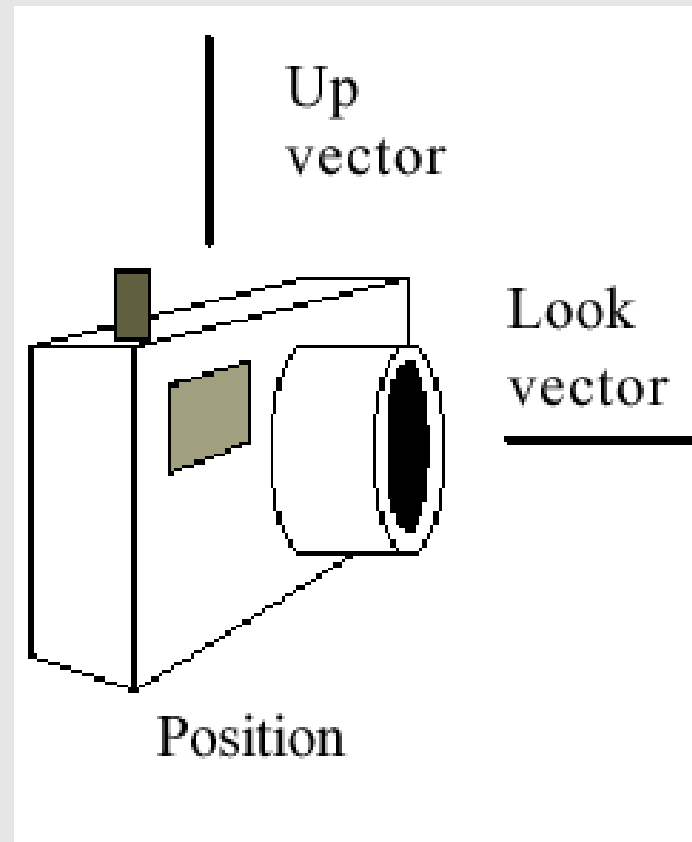
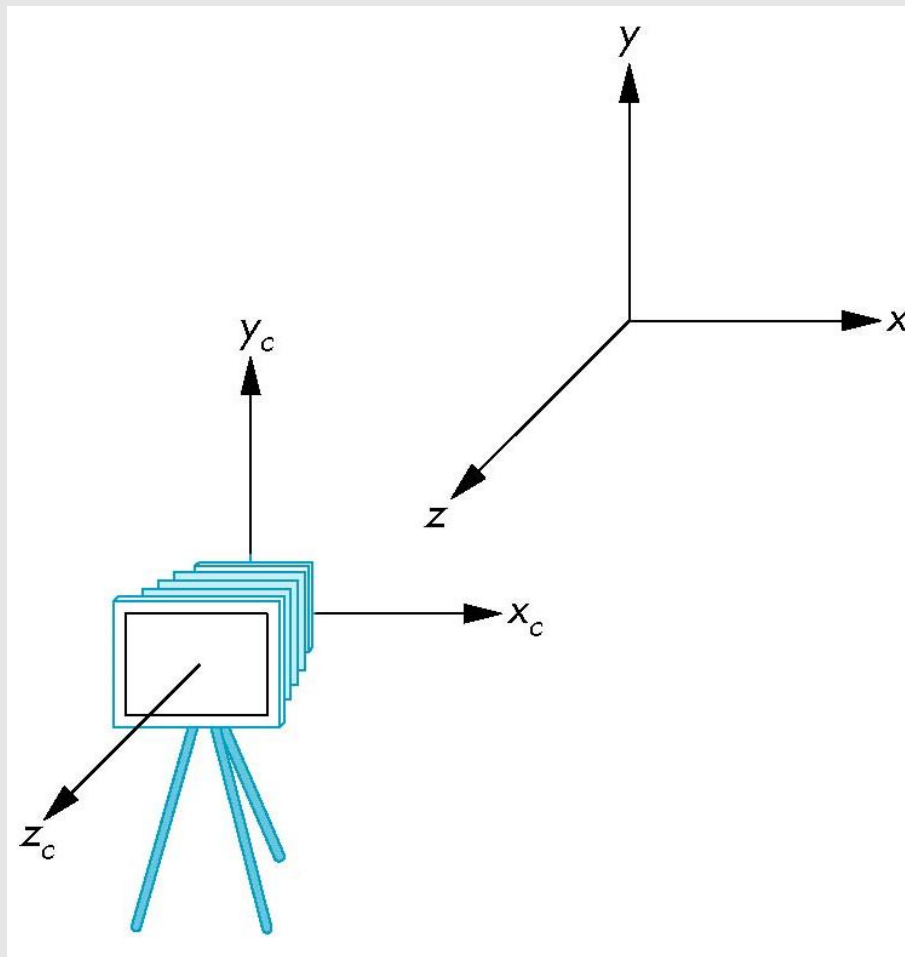
2.1.5.1 概 述

2.1.5.2 视变换

2.1.5.3 窗口－视区变换

## 2.1.5.2 视变换 — 相关参数

- 相机定位;

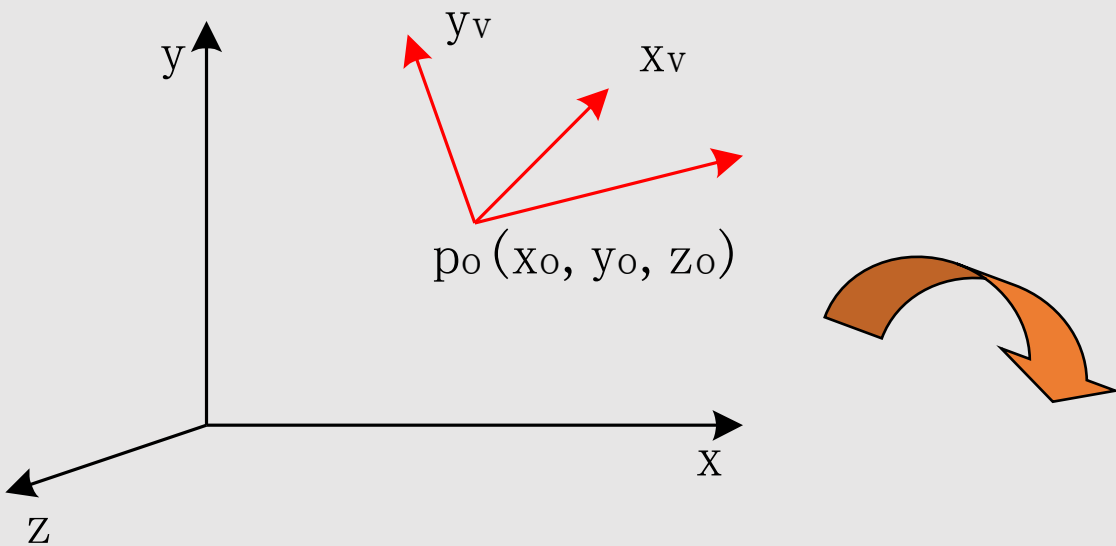


## 2.1.5.2 视变换 — 相关参数

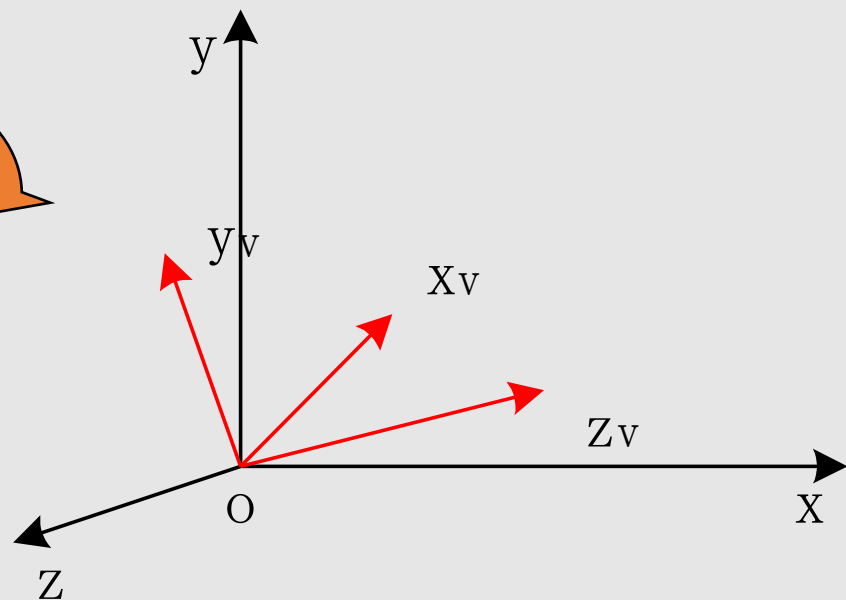
**视变换：世界坐标系到观察坐标系的变换**

具体变换步骤：

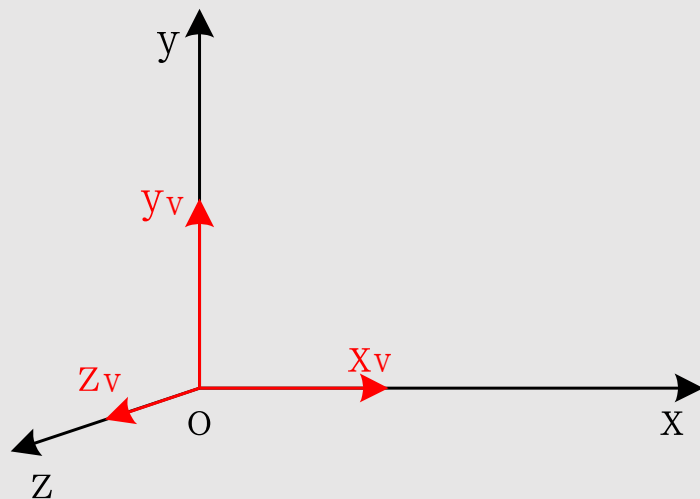
- (1) 平移观察参考点到世界坐标系原点
- (2) 进行旋转变换分别让 $x_v$ 、 $y_v$ 和 $z_v$ 轴对应到世界坐标系中的 $x$ 、 $y$ 和 $z$ 轴。



(a)世界坐标系与观察坐标系



(b)平移观察坐标系



(c)旋转观察坐标系

## 2.1.5 图形的显示流程

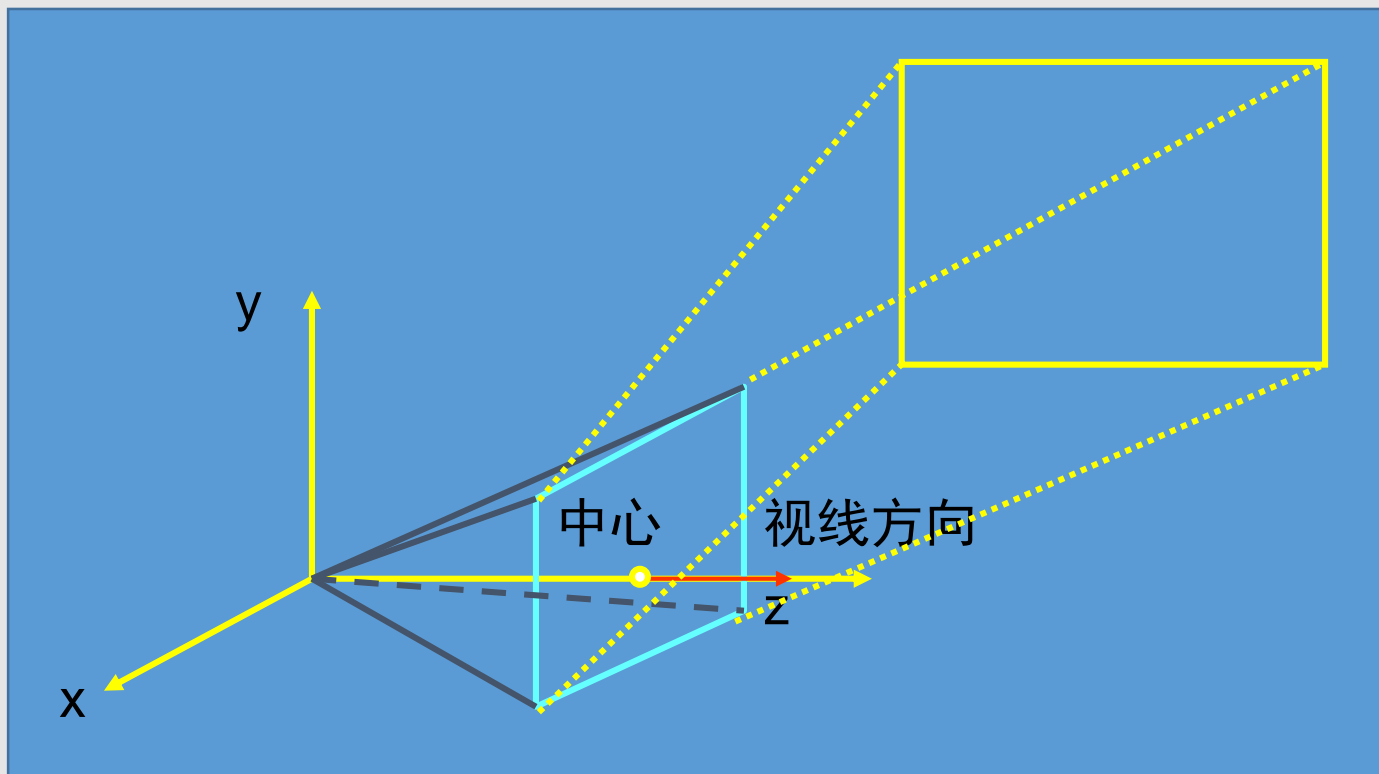
2.1.5.1 概 述

2.1.5.2 视变换

2.1.5.3 窗口－视区变换

## 2.1.5.3 窗口—视区变换

- 窗口(Window): 在WC中定义的一个(矩形)区域, 该区域内的对象将予以显示。
- 视区(Viewport): 在DC中定义一个(矩形)区域, 所有在窗口内的对象都将显示在该区域中。





## 2.1.5.3 窗口—视区变换

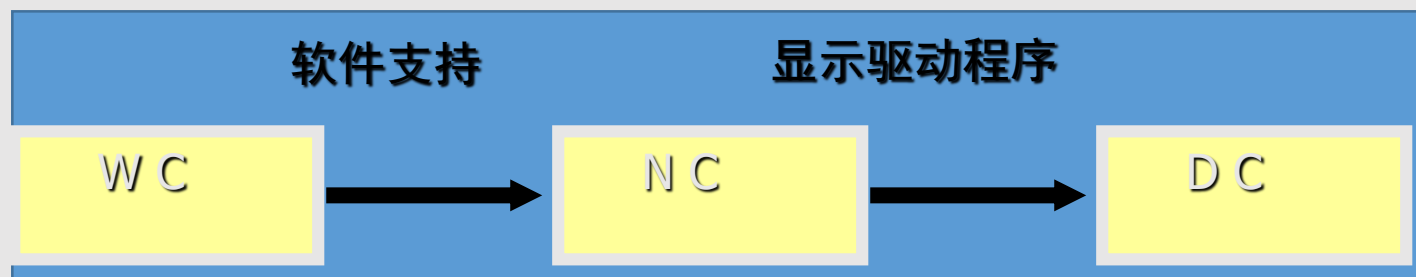
对图形的描述、图形的输入输出都是在坐标系中进行的。

- 1.世界坐标系 WC(World Coordinate System): 包括常用的直角坐标系、几何坐标系等各种坐标系, 用来直接描述对象。或称为用户坐标系 UC (User Coordinate System), 取值范围为整个实数域。
- 2.设备坐标系 DC(Device Coordinate System): 图形的显示是在设备上进行的, 在设备上描述图形的坐标系称为设备坐标系 DC(Device Coordinate System), 取值范围受设备的输入输出的精度以及画面有效范围的限制。屏幕上显示的图形均以其一个像素点单位为量化单位。

## 2.1.5.3 窗口—视区变换

对图形的描述、图形的输入输出都是在坐标系中进行的。

- 3.规范化坐标系 NC(Normalization Coordinate System): 二维, 取值范围 $[0, 1]$ 。
- 4.不同坐标系的转换: 将世界坐标系中的对象转换成设备坐标系中的图形有两种方式:
  - 直接转换。容易实现, 可移植性较差。
  - 规范化坐标系。

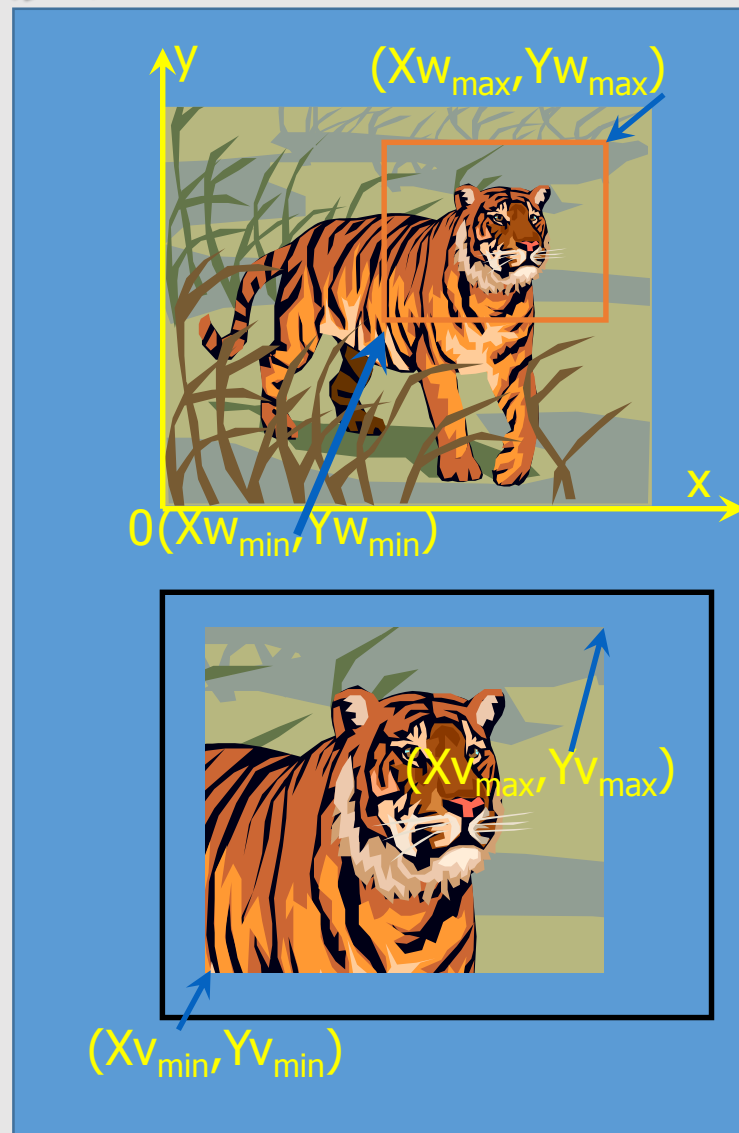


## 2.1.5.3 窗口—视区变换

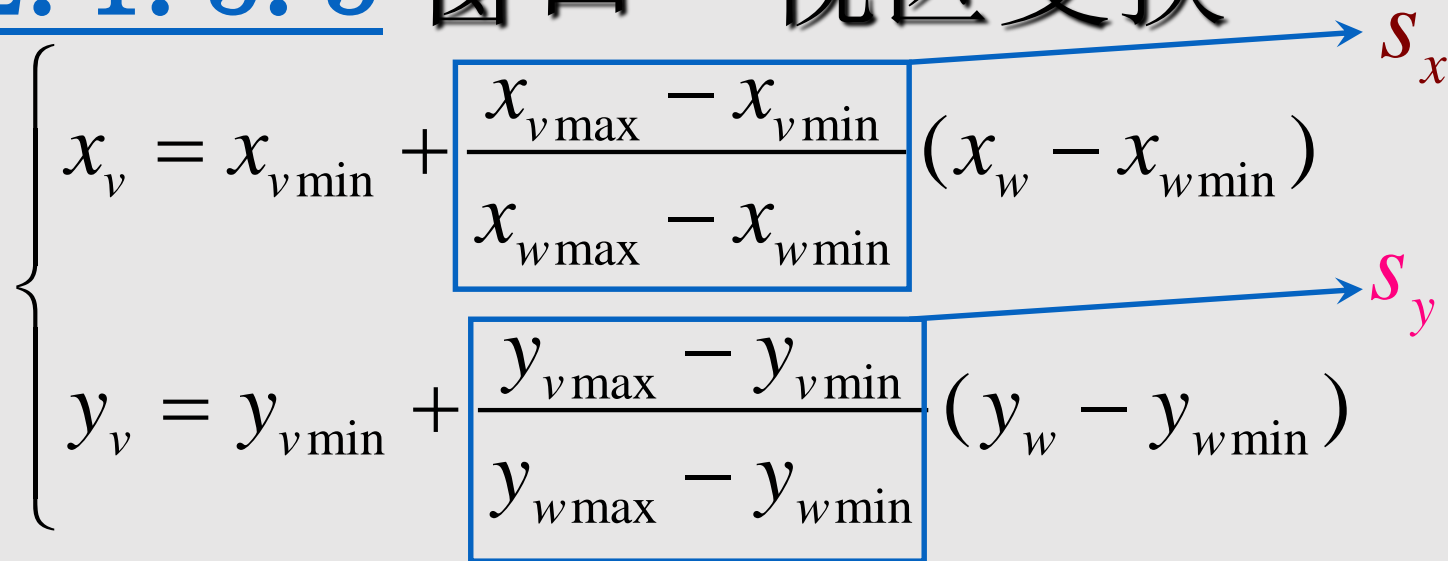
$$\frac{x_w - x_{w\min}}{x_{w\max} - x_{w\min}} = \frac{x_v - x_{v\min}}{x_{v\max} - x_{v\min}}$$

$$\frac{y_w - y_{w\min}}{y_{w\max} - y_{w\min}} = \frac{y_v - y_{v\min}}{y_{v\max} - y_{v\min}}$$

$$\begin{cases} x_v = x_{v\min} + \frac{x_{v\max} - x_{v\min}}{x_{w\max} - x_{w\min}} (x_w - x_{w\min}) \\ y_v = y_{v\min} + \frac{y_{v\max} - y_{v\min}}{y_{w\max} - y_{w\min}} (y_w - y_{w\min}) \end{cases}$$



## 2.1.5.3 窗口—视区变换

$$\begin{cases} x_v = x_{v\min} + \frac{x_{v\max} - x_{v\min}}{x_{w\max} - x_{w\min}} (x_w - x_{w\min}) \\ y_v = y_{v\min} + \frac{y_{v\max} - y_{v\min}}{y_{w\max} - y_{w\min}} (y_w - y_{w\min}) \end{cases}$$


The diagram shows two blue arrows pointing from the fraction parts of the equations to the variables  $s_x$  and  $s_y$ . The first arrow points from the fraction  $\frac{x_{v\max} - x_{v\min}}{x_{w\max} - x_{w\min}}$  to  $s_x$ . The second arrow points from the fraction  $\frac{y_{v\max} - y_{v\min}}{y_{w\max} - y_{w\min}}$  to  $s_y$ .

$$a = x_{v\min} - s_x x_{w\min}, \quad b = y_{v\min} - s_y y_{w\min}$$

则

$$\begin{aligned} x_v &= s_x x_w + a, \\ y_v &= s_y y_w + b \end{aligned}$$

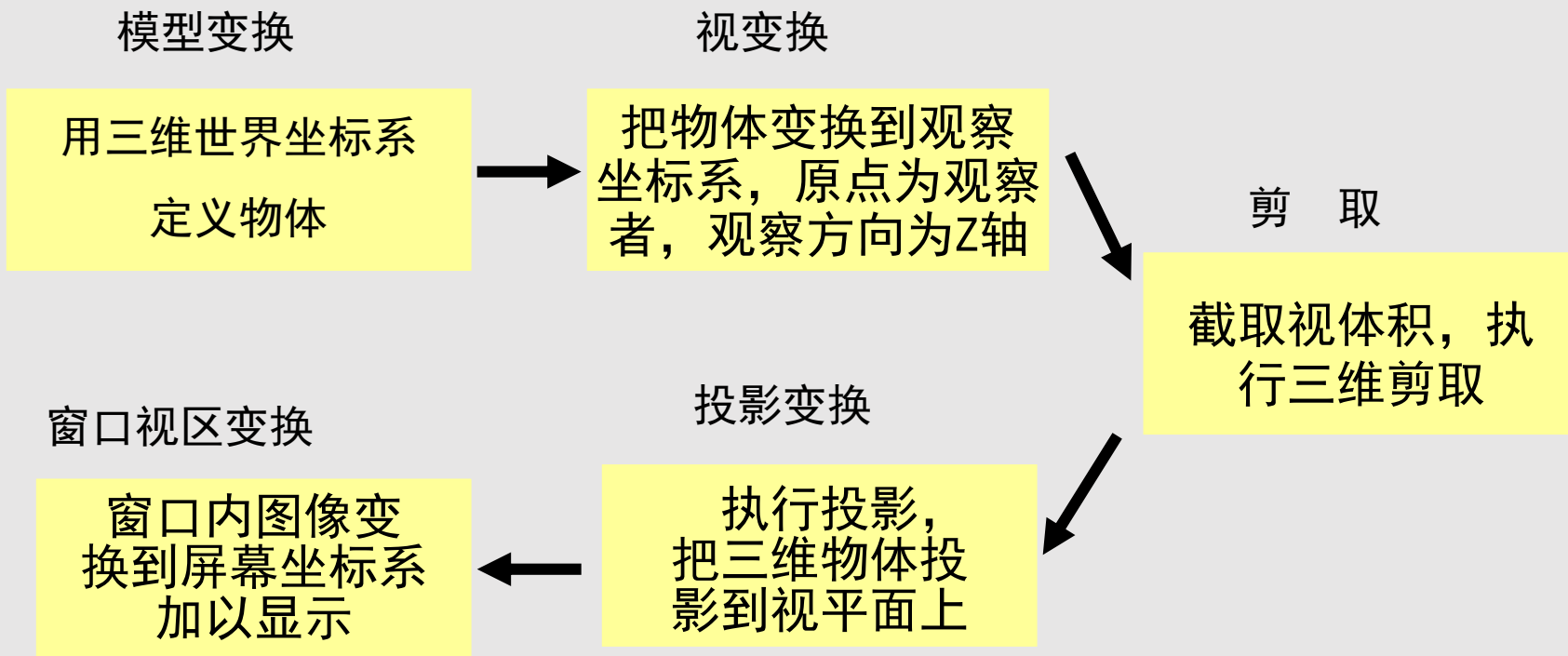
即,

$$\begin{bmatrix} x_v \\ y_v \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & a \\ 0 & s_y & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix}$$

## 2.1.5.3 窗口—视区变换

- 选择不同的Window，显示在同一Viewport中：
  - 窗口位置不同，显示出不同的图形；
  - 窗口位置不变，只变大小，出现变焦效果；
- 选择不同的Window，显示在不同Viewport中，显示图形有个调度问题。
- 如果Window的X:Y与Viewport的X:Y不等时，图形将变形。

## 2.1.5 三维图形的显示流程



何时裁剪？

## 2.1.5 三维图形的显示流程

### ■何时裁剪？

- 投影~~之前~~裁剪：三维裁剪
  - 优点：只对可见的物体进行投影变换
  - 缺点：三维裁剪相对复杂
- 投影~~之后~~裁剪：二维裁剪
  - 优点：二维裁剪相对容易
  - 缺点：需要对所有的物体进行投影变换

## 2.1.5 三维图形的显示流程

■选择在**投影之前**裁剪的理由：

- 三维物体的表面通常被离散表示成**多边形**或**折线**，而对这类简单图元，三维裁剪同样比较简单。
- 三维物体在显示过程中需要被**消隐**，做这个工作要有物体的**深度信息**，所以必须在投影之前完成。消隐很费时，如果在此之前裁剪（或部分裁剪）掉不可见的图形，可使需要消隐的图形减至最小。



## 2.1 三维空间变换

2.1.1 变换的数学基础

2.1.2 几何变换

2.1.3 坐标变换

2.1.4 投影变换

2.1.5 图形的显示流程

2.1.6 OpenGL中的变换

## 2.1.6 OpenGL中的变换

- 变换种类
- 模型视图矩阵
- 矩阵操作
- 矩阵堆栈
- 投影变换
- 高级矩阵操作

# 变换种类

- 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；

# 模型视图矩阵

- 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);
```

# 模型视图矩阵

- 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  
);
```

# 矩阵操作

**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

**GL\_TEXTURE:** 纹理矩阵堆栈

## 矩阵操作——单位矩阵

```
glTranslatef(10.0f, 0.0f, 0.0f);
```

```
glutSolidSphere(1.0f, 15, 15);
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity();
```

```
glTranslatef(0.0f, 10.0f, 0.0f);
```

```
glutSolidSphere(1.0f, 15, 15);
```

# 矩阵堆栈

- OpenGL为模型视图矩阵和投影矩阵各维护着一个“矩阵堆栈”，可以把当前矩阵压到堆栈中保存它，然后对当前矩阵进行修改。把矩阵弹出堆栈即可恢复。使用的函数如下：

```
void glPushMatrix(void);
```

```
void glPopMatrix(void);
```



# 投影变换

- OpenGL中只提供了两种投影方式，一种是平行投影（正射投影），另一种是透视投影。在投影变换之前必须指定当前处理的是投影变换矩阵：

```
glMatrixMode(GL_PROJECTION);
```

```
glLoadIdentity();
```

# 投影变换

- 平行投影：视景物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 );
```

# 投影变换

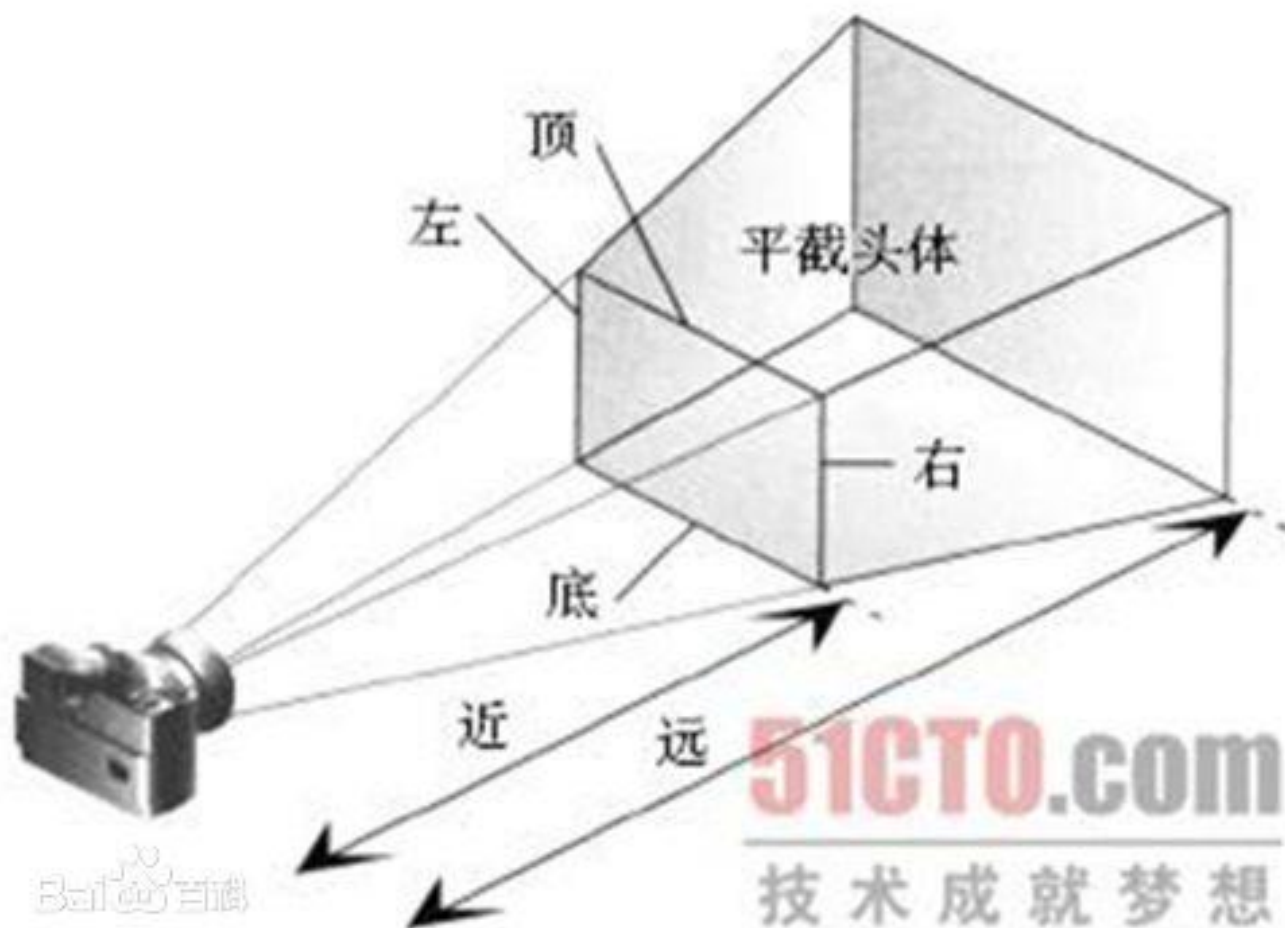
```
void gluOrtho2D (  
    GLdouble left,    GLdouble right,  
    GLdouble bottom, GLdouble top );
```

- 一个特殊的正射投影函数，主要用于二维图像到二维屏幕上的投影。其near和far缺省值分别为-1.0和1.0，所有二维物体的Z坐标都为0.0。因此它的裁剪面是一个左下角点为（left, bottom）、右上角点为（right, top）的矩形。

## □ 透视投影

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

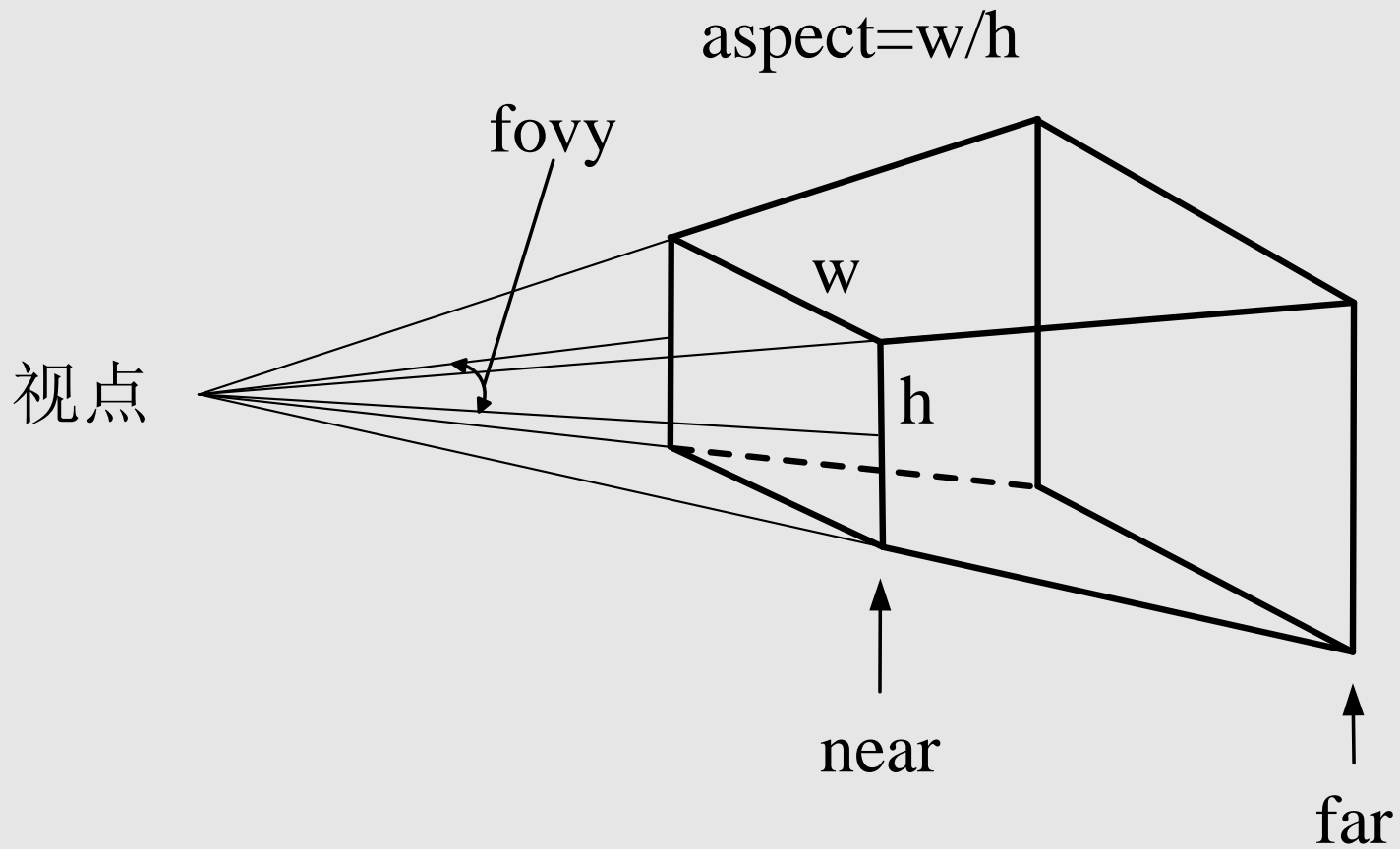
此函数创建一个透视投影矩阵，并且用这个矩阵乘以当前矩阵。它的参数只定义近裁剪平面的左下角点和右上角点的三维空间坐标，即（**left**, **bottom**, -**near**）和（**right**, **top**, -**near**）；最后一个参数**far**是远裁剪平面的Z负值，其左下角点和右上角点空间坐标由函数根据透视投影原理自动生成。



```
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。

# 投影变换



# 高级矩阵操作

```
GLfloat m[] = { 1.0f, 0.0f, 3.0f, 0.0f,  
                0.0f, 1.0f, 0.0f, 1.0f,  
                0.0f, 0.0f, 1.0f, 1.0f,  
                0.0f, 0.0f, 0.0f, 1.0f  
                };  
glMatrixMode(GL_MODELVIEW);  
glLoadMatrixf(m);  
glMultiMatrixf(m);
```



## 思考题:

- 请总结如下变换矩阵与各种变换对应关系的特点。

$$T_{3D} = \left[ \begin{array}{ccc|c} a & b & c & p \\ d & e & f & q \\ g & h & i & r \\ \hline l & m & n & s \end{array} \right]$$

# 思考题：

- 请总结三维图形的显示流程。

模型变换→投影变换→裁减→视口变换

