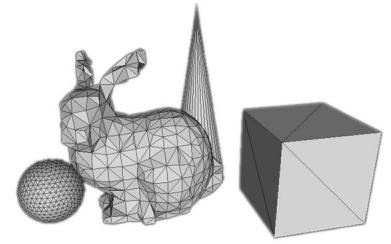


Introduction to Computer Graphics



Transformation 2

本节内容

- 顶点变换流程
- 模型视图变换
 - 模型变换
 - 平移
 - 旋转
 - 缩放
 - 视图变换
 - 相机变换gluLookAt
- 投影变换
 - 正交投影
 - 透视投影
- 视口变换

向量

- 向量

- $\mathbf{a} = (a_1, a_2, a_3)^T$

- \mathbf{a} 的模长: $|\mathbf{a}| = \sqrt{a_1^2 + a_2^2 + a_3^2}$

- 归一化: $\frac{\mathbf{a}}{|\mathbf{a}|} = \frac{1}{\sqrt{a_1^2 + a_2^2 + a_3^2}} (a_1, a_2, a_3)^T$

向量

- 向量点乘（点积）

- $\mathbf{a} = (a_1, a_2, a_3)^T$

- $\mathbf{b} = (b_1, b_2, b_3)^T$

- $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$
 $= |\mathbf{a}| |\mathbf{b}| \cos \angle \mathbf{a}, \mathbf{b}$

- $\mathbf{a} \cdot \frac{\mathbf{b}}{|\mathbf{b}|}$ 几何意义：向量 \mathbf{a} 在 \mathbf{b} 方向上的投影

向量

- 向量叉乘（叉积）

- $\mathbf{a} = (a_1, a_2, a_3)^T$

- $\mathbf{b} = (b_1, b_2, b_3)^T$

- $\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$

- $= (a_2b_3 - a_3b_2)\mathbf{i} + (a_3b_1 - a_1b_3)\mathbf{j} + (a_1b_2 - a_2b_1)\mathbf{k}$

- $\mathbf{i} = (1, 0, 0)^T, \mathbf{j} = (0, 1, 0)^T, \mathbf{k} = (0, 0, 1)^T,$

矩阵

- 矩阵

- 矩阵与向量的乘法
- 投影观点

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} - & \mathbf{r}_1 & - \\ - & \mathbf{r}_2 & - \\ - & \mathbf{r}_3 & - \end{bmatrix} \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix}$$

- 加权观点

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$
$$\mathbf{y} = x_1 \mathbf{c}_1 + x_2 \mathbf{c}_2 + x_3 \mathbf{c}_3.$$

矩阵

- 矩阵

- 矩阵与矩阵的乘法

- $\mathbf{AB}=\mathbf{P}$

$$\begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{i1} & \dots & a_{im} \\ \vdots & & \vdots \\ a_{r1} & \dots & a_{rm} \end{bmatrix} \begin{bmatrix} b_{11} & \dots & b_{1j} & \dots & b_{1c} \\ \vdots & & \vdots & & \vdots \\ b_{m1} & \dots & b_{mj} & \dots & b_{mc} \end{bmatrix} = \begin{bmatrix} p_{11} & \dots & p_{1j} & \dots & p_{1c} \\ \vdots & & \vdots & & \vdots \\ p_{i1} & \dots & p_{ij} & \dots & p_{ic} \\ \vdots & & \vdots & & \vdots \\ p_{r1} & \dots & p_{rj} & \dots & p_{rc} \end{bmatrix}$$

- $\mathbf{P}=\mathbf{AB}$

- $\mathbf{P}^T=\mathbf{B}^T\mathbf{A}^T$

- $\mathbf{AB}=\mathbf{BA}$ (一般不成立!)

矩阵

- 正交基的构建、正交矩阵
 - 单位正交矩阵 $A^T A = A A^T = I$
 - 正交：各向量点积为0
 - 单位：各向量模长为1
- 给定三个不共面的向量，如何构建正交基？
 - 叉乘

顶点变换的流程



顶点变换的流程

模视变换

$$\begin{pmatrix} X_{eye} \\ Y_{eye} \\ Z_{eye} \\ W_{eye} \end{pmatrix} = M_{modelView} * \begin{pmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ W_{obj} \end{pmatrix} = M_{view} * M_{model} * \begin{pmatrix} X_{obj} \\ Y_{obj} \\ Z_{obj} \\ W_{obj} \end{pmatrix}$$



投影变换

$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = M_{projection} * \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix}$$



透视除法

$$\begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{pmatrix}$$



视口变换

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} \frac{w}{2} x_{ndc} + (x + \frac{w}{2}) \\ \frac{h}{2} y_{ndc} + (y + \frac{h}{2}) \\ \frac{f-n}{2} z_{ndc} + \frac{f+n}{2} \end{pmatrix}$$

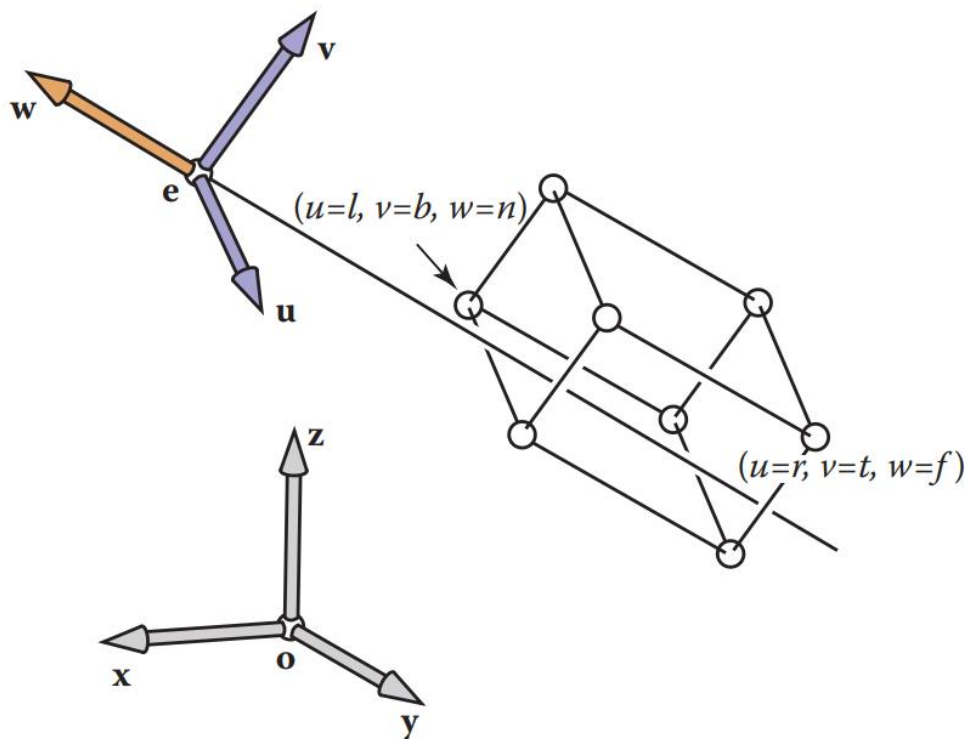
视图变换

- 设置相机位置、方位
- 属于模型视图矩阵
 - gluLookAt设置相机的位置和方位
 - 其实可以等价于反方向设置物体的位置和方位
 - 因此：gluLookAt可以有glRotate和glTranslate等函数联合实现
- 如何求gluLookAt所对应的变换矩阵

gluLookAt

- 相机变换（视点变换）

- the eye position e ,
- the gaze direction g ,
- the view-up vector t .



模型视图变换

- 相机变换（视点变换）

- the eye position \mathbf{e} ,
- the gaze direction \mathbf{g} ,
- the view-up vector \mathbf{t} .

$$\mathbf{w} = -\frac{\mathbf{g}}{\|\mathbf{g}\|},$$

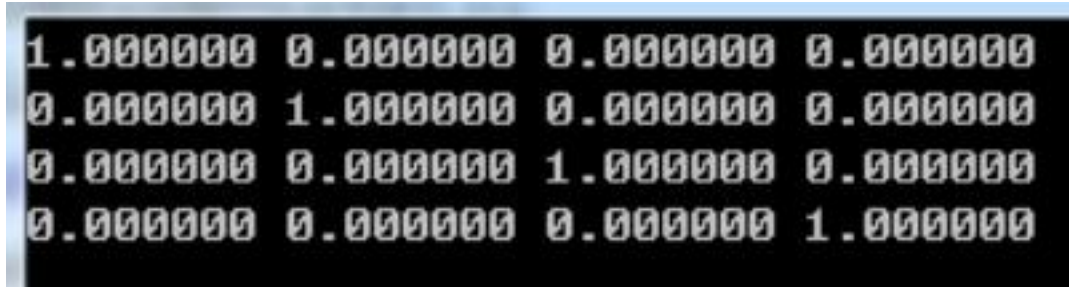
$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{\|\mathbf{t} \times \mathbf{w}\|},$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}.$$

$$\mathbf{M}_{\text{cam}} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

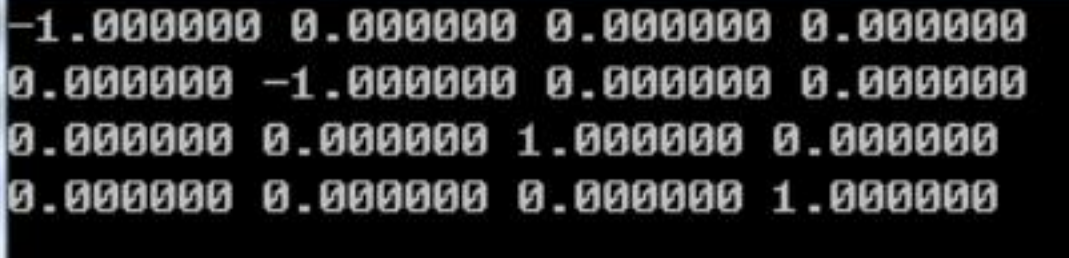
实例计算

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0, 0, 0, 0, 0, -1, 0, 1, 0);
float arr[16];
glGetFloatv(GL_MODELVIEW_MATRIX, arr);
for(int i=0; i<4; ++i)
{
    for(int j=0; j<4; ++j)
    {
        printf("%f ", arr[j*4+i]);
    }
    printf("\n");
}
```



```
1.000000 0.000000 0.000000 0.000000
0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 1.000000
```

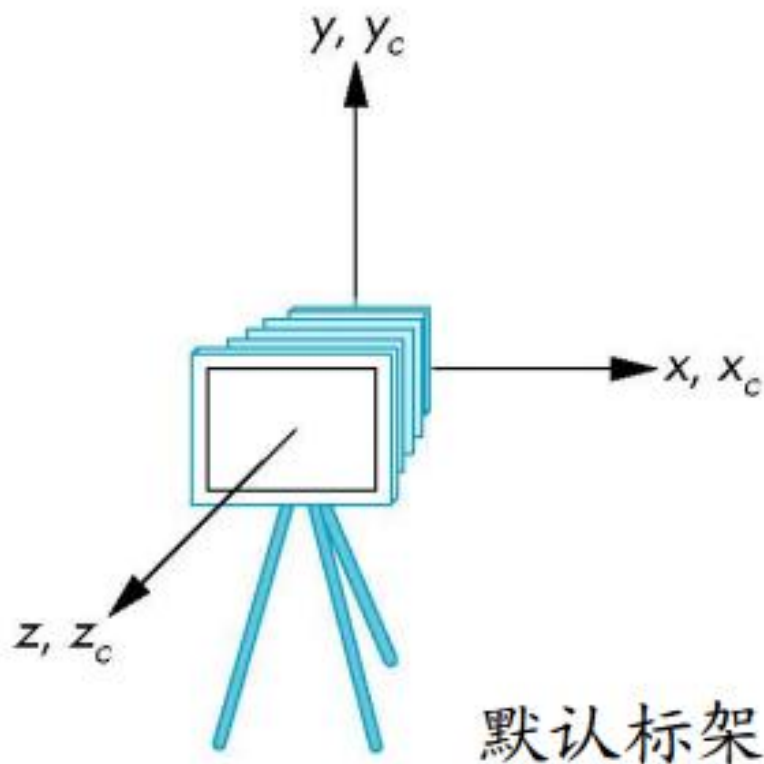
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0, 0, 0, 0, 0, -1, 0, -1, 0);
float arr[16];
glGetFloatv(GL_MODELVIEW_MATRIX, arr);
for(int i=0; i<4; ++i)
{
    for(int j=0; j<4; ++j)
    {
        printf("%f ", arr[j*4+i]);
    }
    printf("\n");
}
```



```
-1.000000 0.000000 0.000000 0.000000
0.000000 -1.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 1.000000
```

注意相机标架

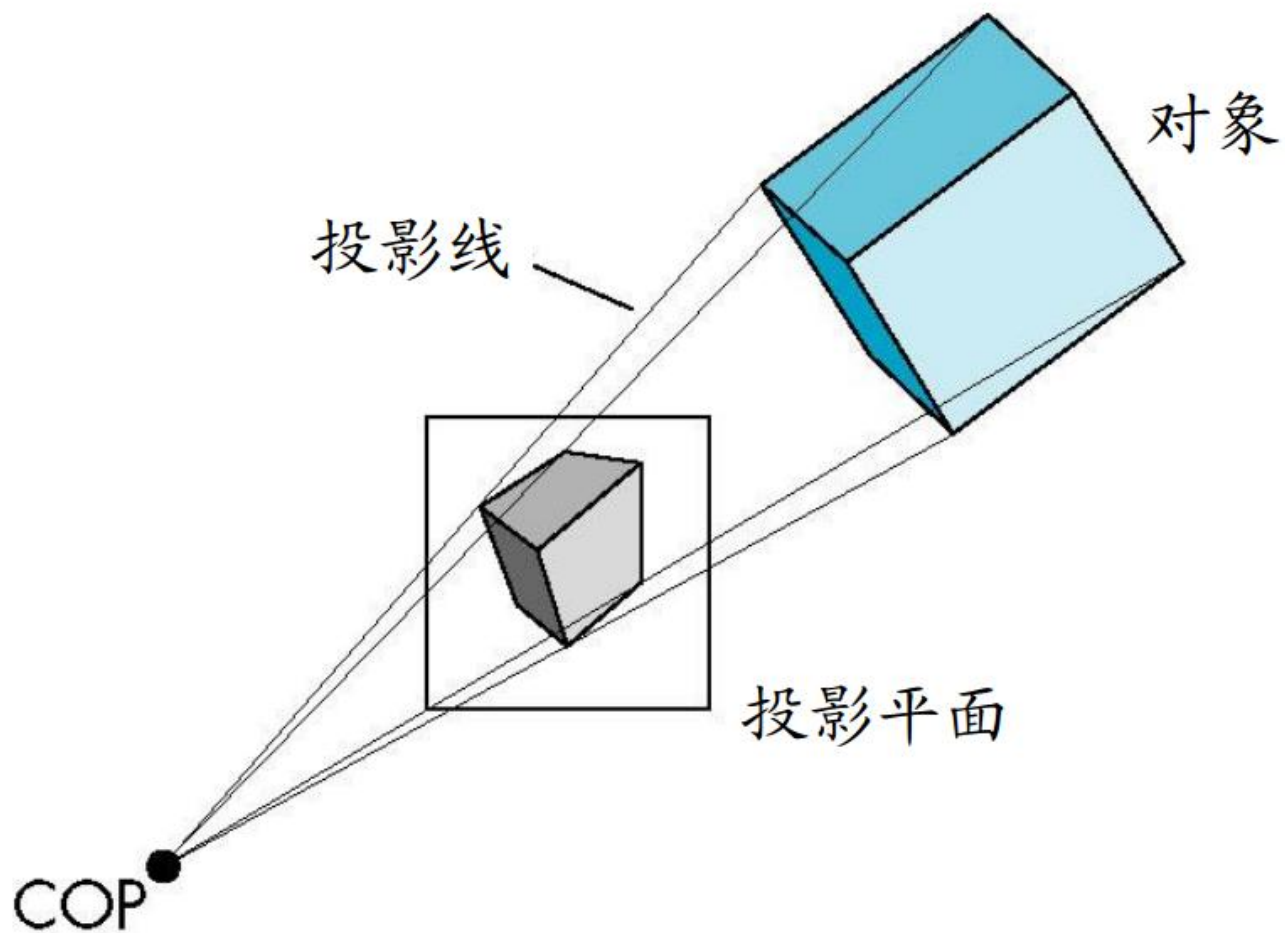
- 默认相机标架是与世界标架重合
- 但是相机是朝z负方向看的



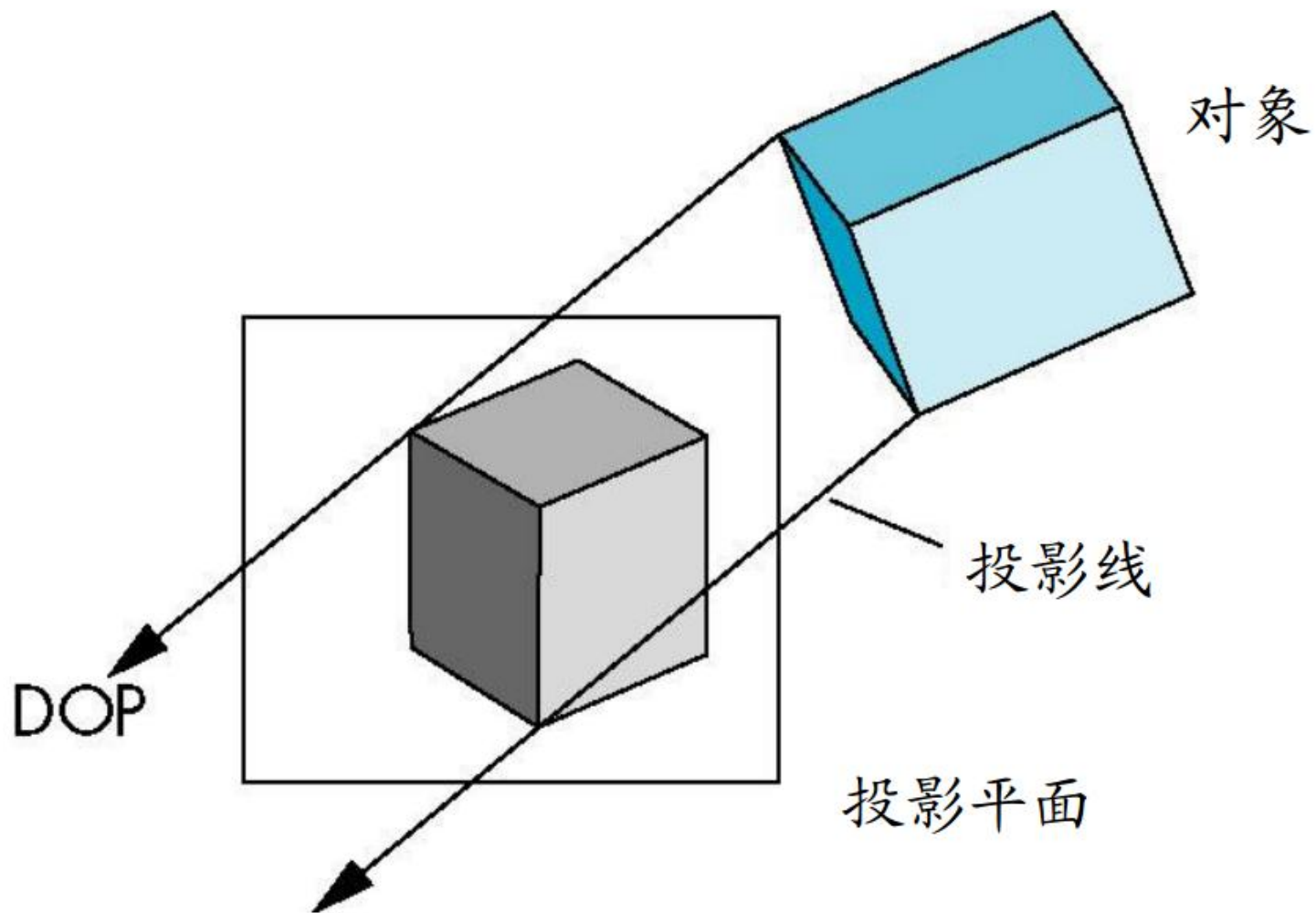
透视投影 vs 平行投影

- 计算机图形学中把所有的投影用同样的方法处理，用一个流水线体系实现它们
- 在经典视图中为了绘制不同类型的投影，发展出来不同的技术
- 平行投影和透视投影有基本区别，虽然从数学上说，平行投影是透视投影的极限状态

透视投影

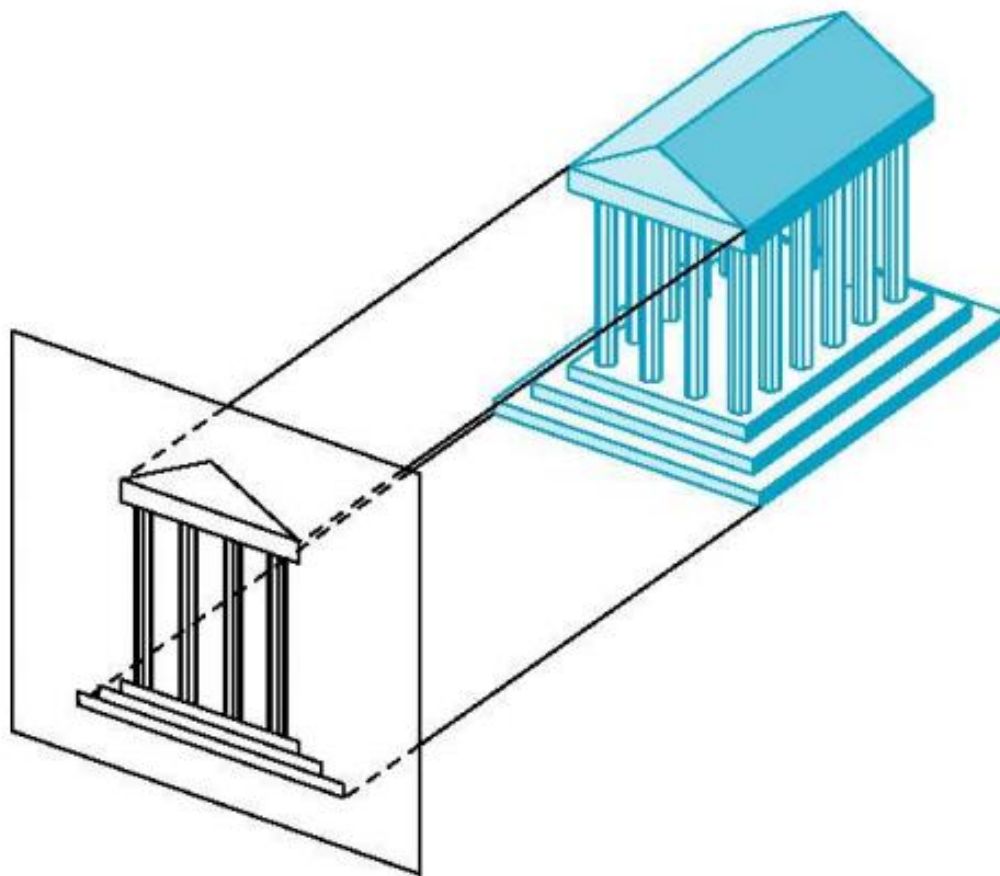


平行投影



正交投影

- 投影线垂直于投影平面



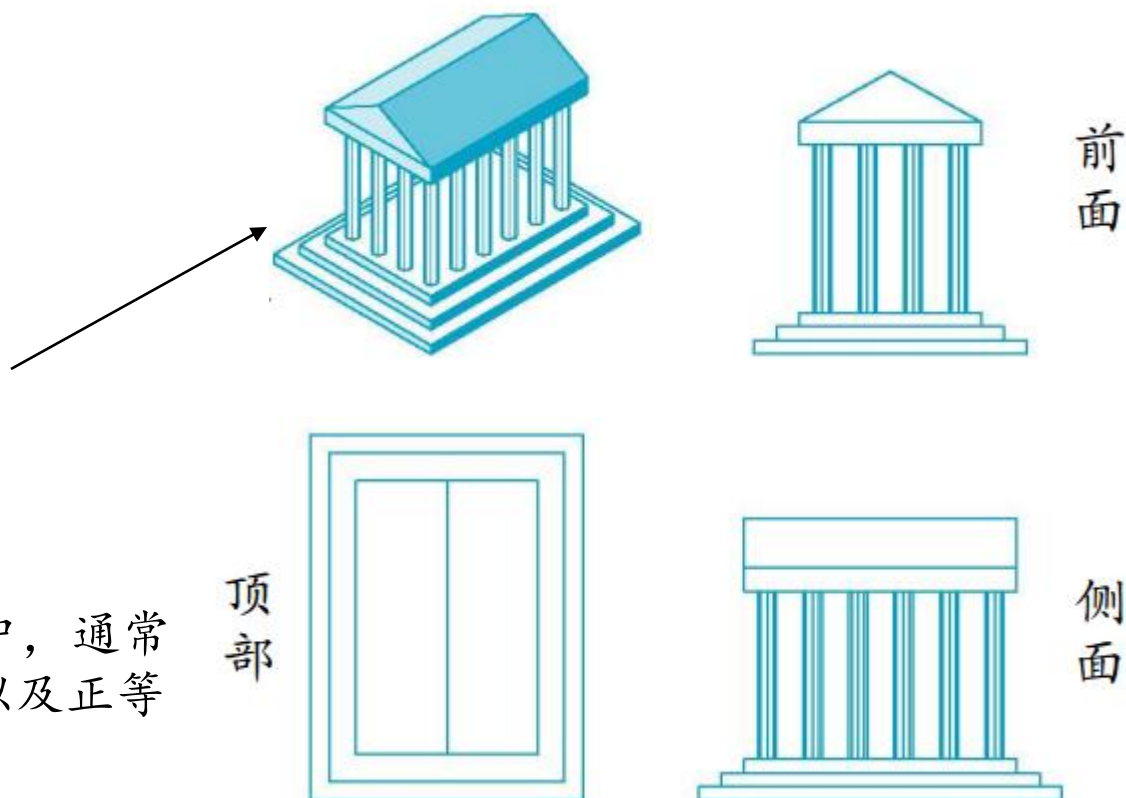
多视点正交投影

- 投影平面平行于某个主视面
- 通常从前面、和侧面进行投影

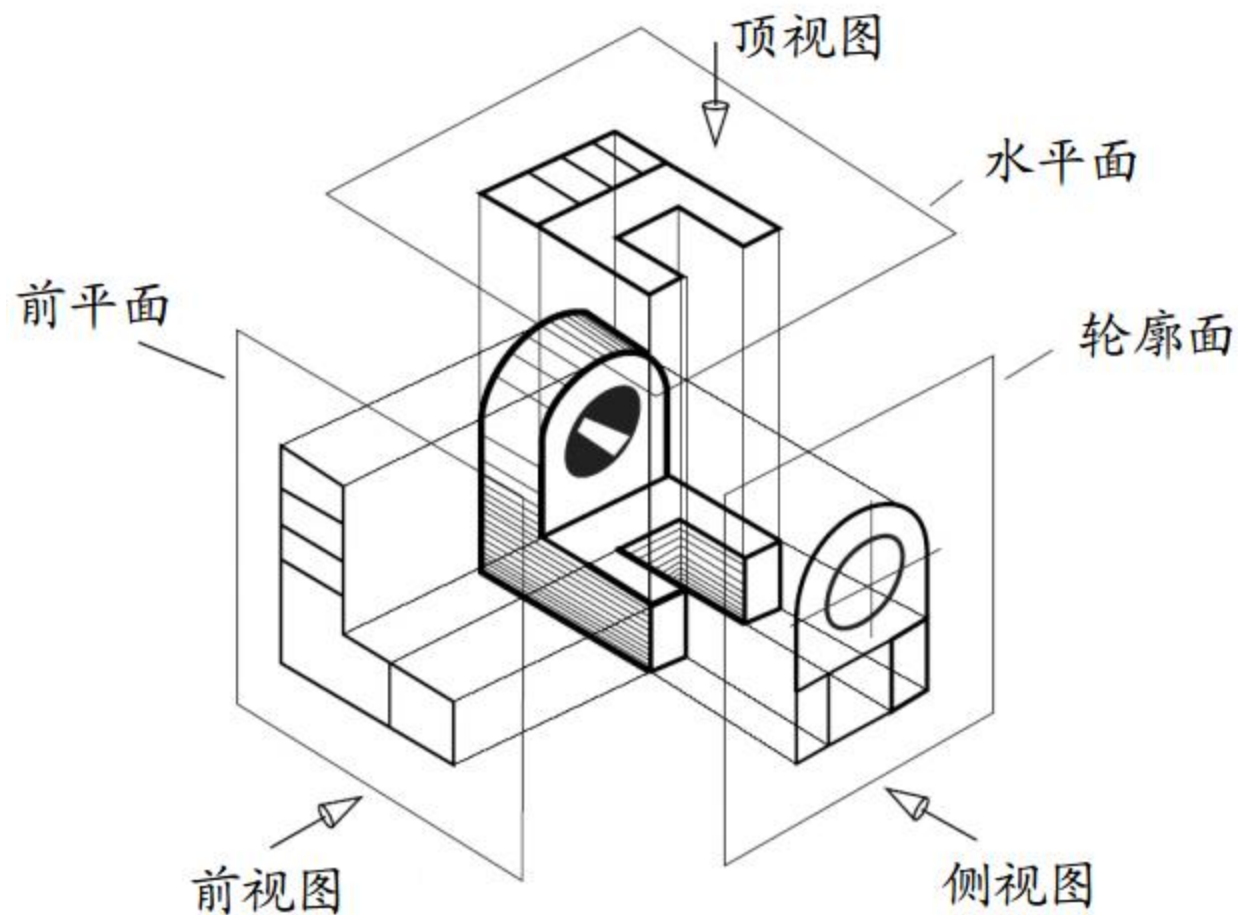
- 三视图
- 正视图（主视图）
- 俯视图
- 侧视图（左视图）

- 正等轴测图（不是多视图正交视图的一部分）

- 在CAD和建筑行业中，通常显示出来三个视图以及正等轴测图



机器零件的三视图

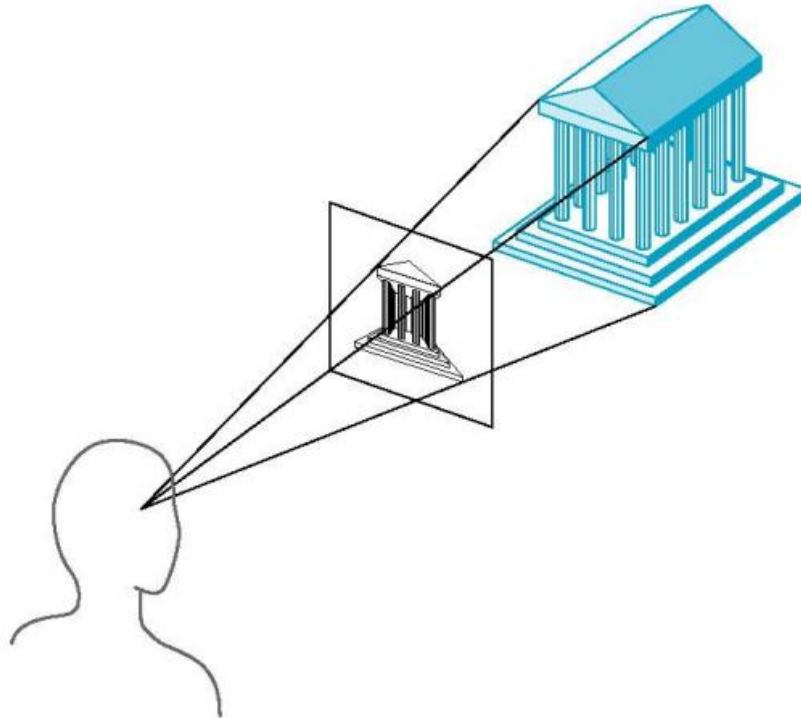


优缺点

- 保持了距离与角度
 - 保持形状
 - 可以用来测量
 - 建筑设计图
 - 手册
- 看不到对象真正的全局形状，因为许多面在视点中不可见

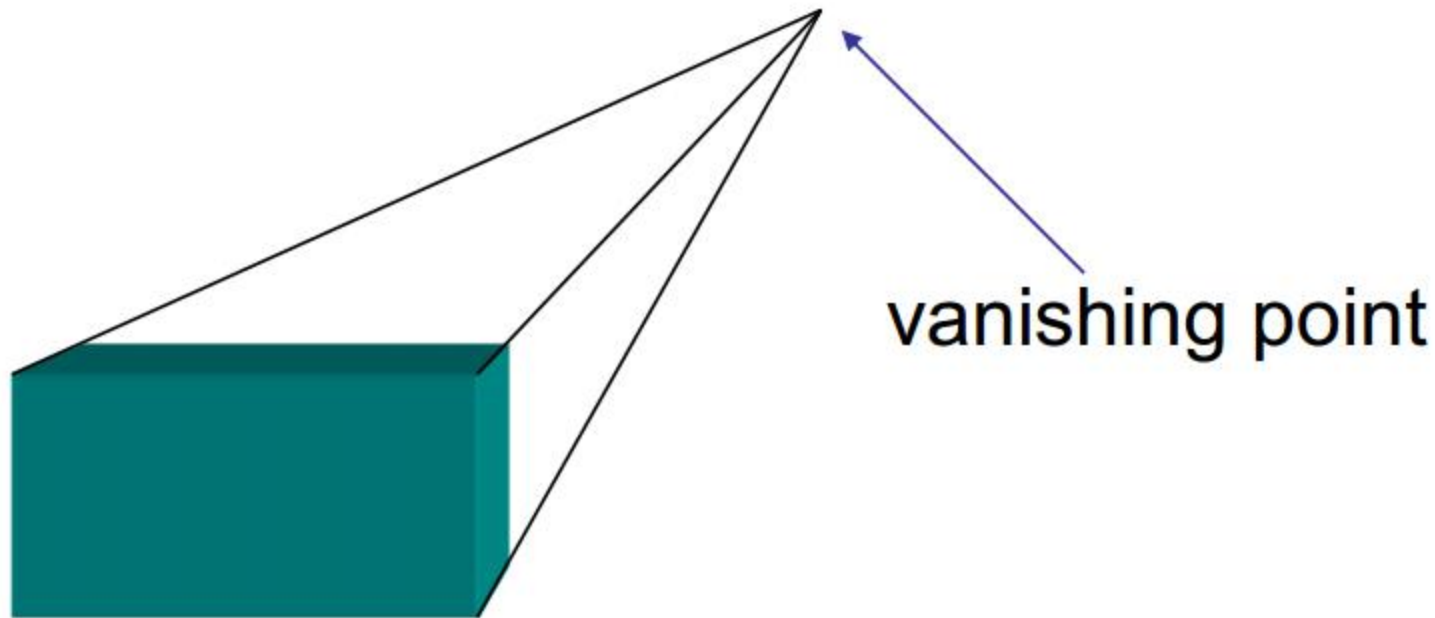
透视投影

- 投影线会聚于投影中心(COP): 尺寸缩小
- 经典视图中, 观察者相对于投影平面对称
— 投影中心和投影窗口确定一个对称的正棱锥

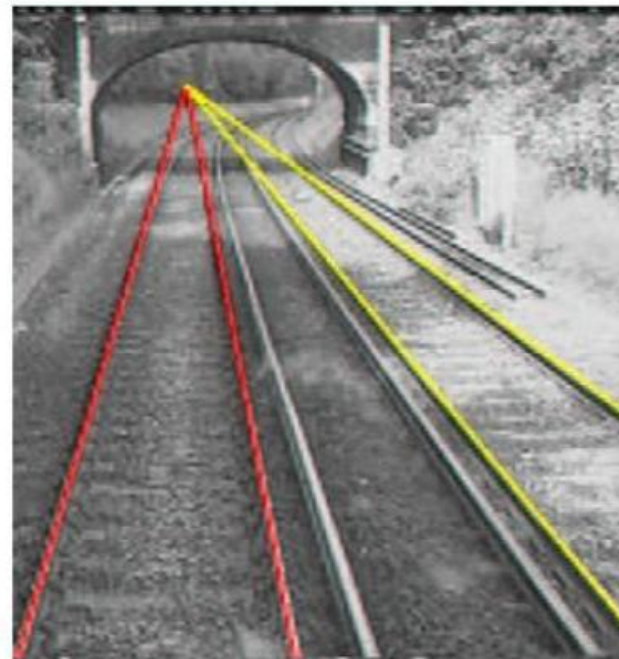
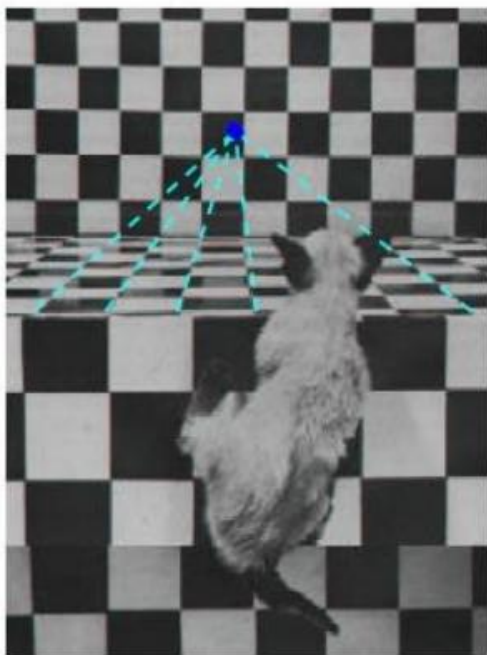


灭点 (vanishing point)

- 对象上（不平行于投影面）的平行线在投影后交于一个灭点(vanishing point)
- 手工绘制简单透视投影时要利用这些灭点

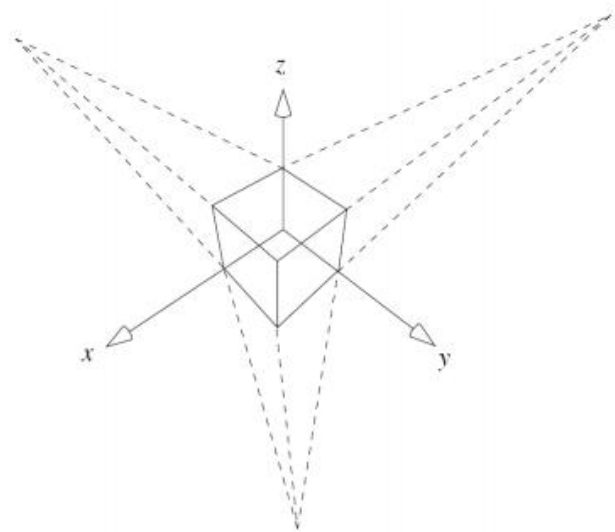


示例



三点透视

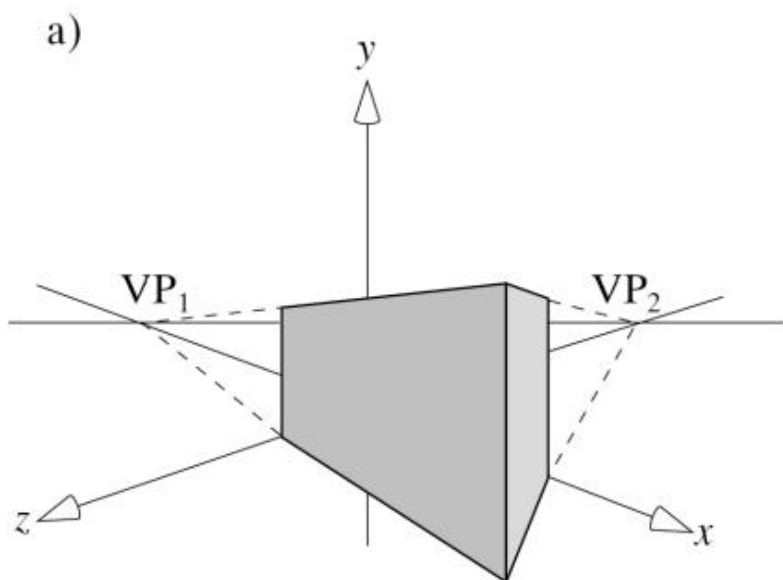
- 立方体的投影中有三个灭点



(a)

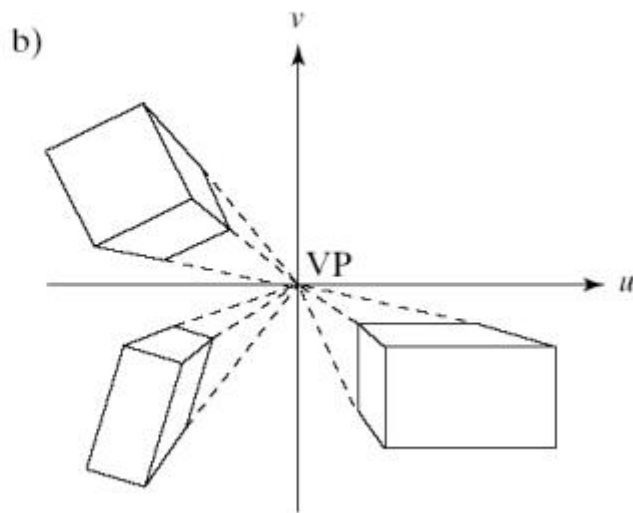
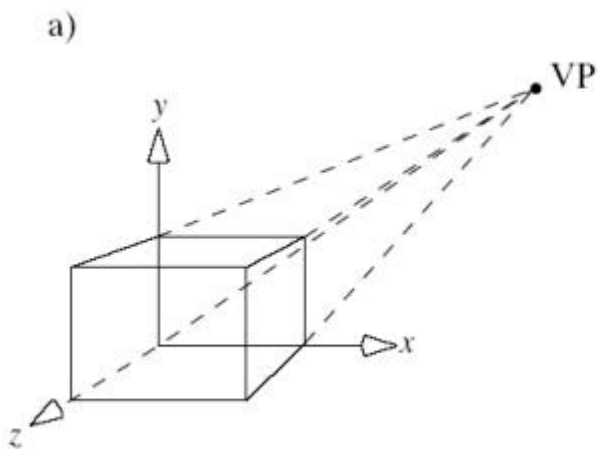
两点透视

- 立方体的投影中有两个灭点



单点透视

- 立方体的投影中有一个灭点

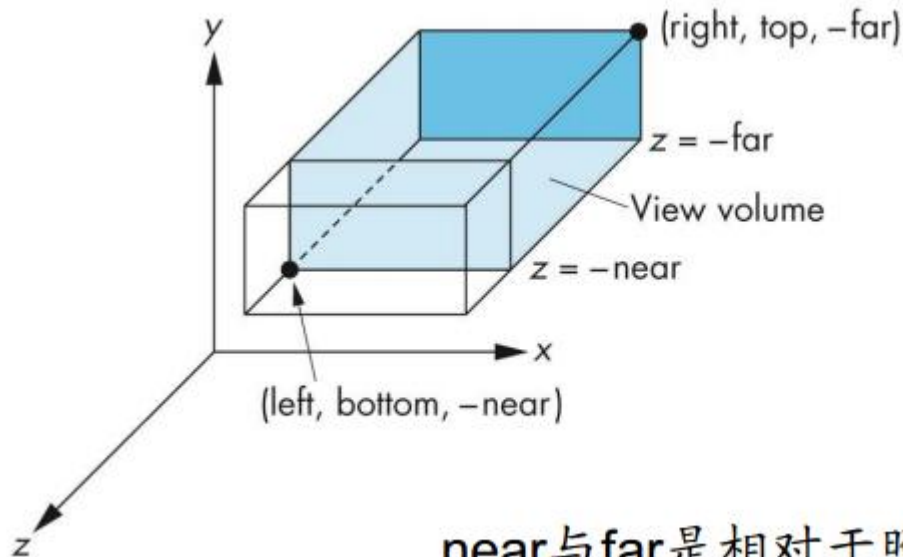


优缺点

- 同样大小的对象，离视点越远，投影结果就越小(diminution)
 - 看起来更自然
- 直线上等距的几点投影后不一定等距——非均匀缩短(nonuniform foreshortening)
 - 借助透视投影图测量尺寸较平行投影困难
- 只有在平行于投影面的平面上角度被保持
- 相对于平行投影而言，更难用手工进行绘制（但对计算机而言，没有增加更多的困难）
- 主要应用在动画等真实感图形领域

正交投影

- `glMatrixMode(GL_PROJECTION)`
- `void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);`
 - 视景体是与坐标轴平行的长方体
 - `near`和`far`可取正值、零或负值，但`near`和`far`不应相同



`near`与`far`是相对于照相机而言的

规范视景体

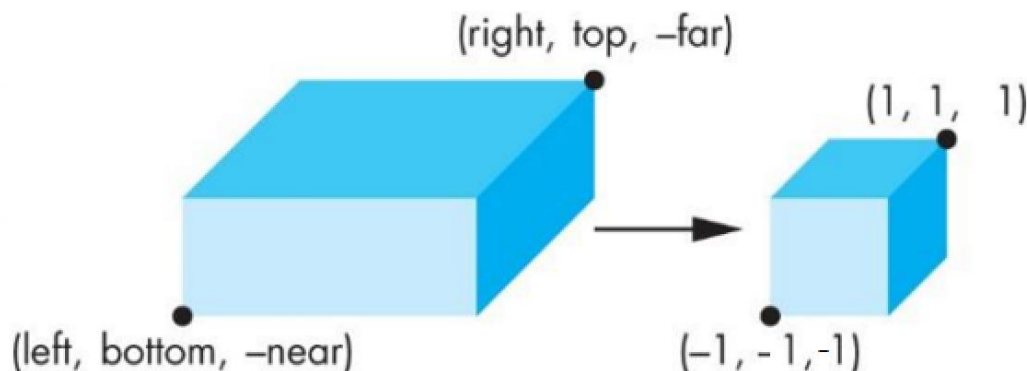
- OpenGL缺省的视景体是中心在原点，边长为2的立方体，相当于调用
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
称这个视景体为规范视景体（canonical view volume）
- 采用规范视景体的优点：
 - 同一流水线支持平行投影和透视投影
 - 简化了裁剪过程

正交规范化

- 规范化 -> 求出把指定裁剪体转化为默认裁剪体的变换

`glOrtho(left, right, bottom, top, near, far)`

- 作用：把视景体内的坐标规范化到-1和1之间



近裁剪面 $z = -\text{near}$ 映射到 $z = -1$ 平面，
远裁剪面 $z = -\text{far}$ 映射到 $z = 1$ 平面

正交规范化矩阵

分两步

近裁剪面 $z=-near$ 映射到 $z=-1$ 平面，
远裁剪面 $z=far$ 映射到 $z=1$ 平面

– 平移：把中心移到原点，对应的变换为

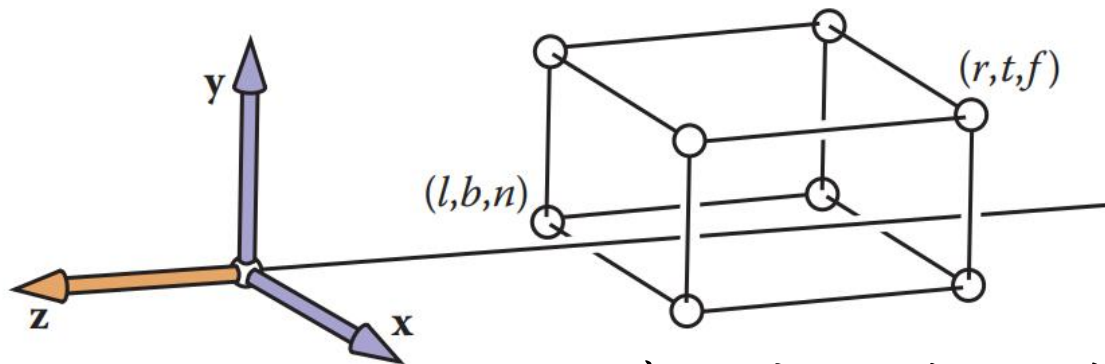
$$T(-(left+right)/2, -(bottom+top)/2, (near+far)/2))$$

– 缩放：进行放缩从而使视景体的边长为2

$$S(2/(right-left), 2/(top - bottom), -2/(far-near))$$

$$P = ST = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

正交投影矩阵公式

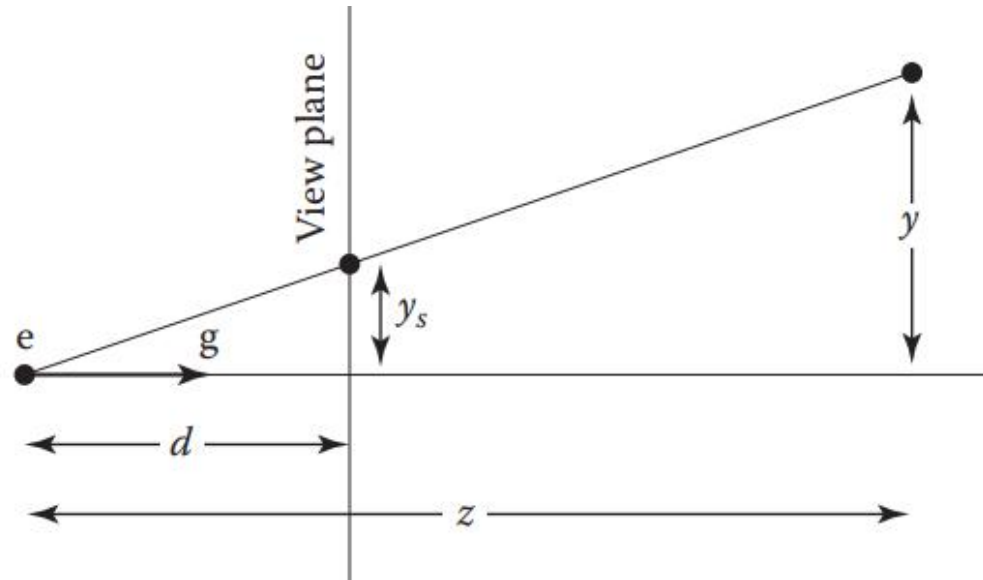


注意：这里n和f取的是绝对值

$$M_{orth} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

透视变换

- 思考 z 的作用？



$$y_s = \frac{d}{z}y$$

透视变换

- 如何将下面的式子写成矩阵乘法的形式？

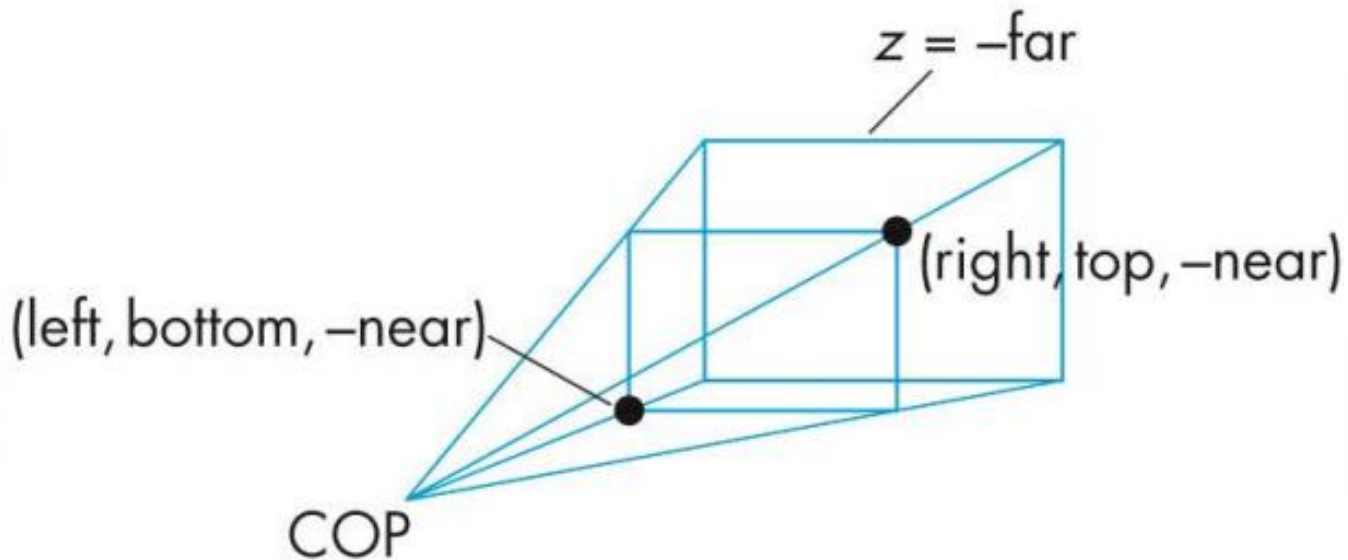
$$x_s = \frac{d}{z} y$$

$$y_s = \frac{d}{z} x$$

$$\begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

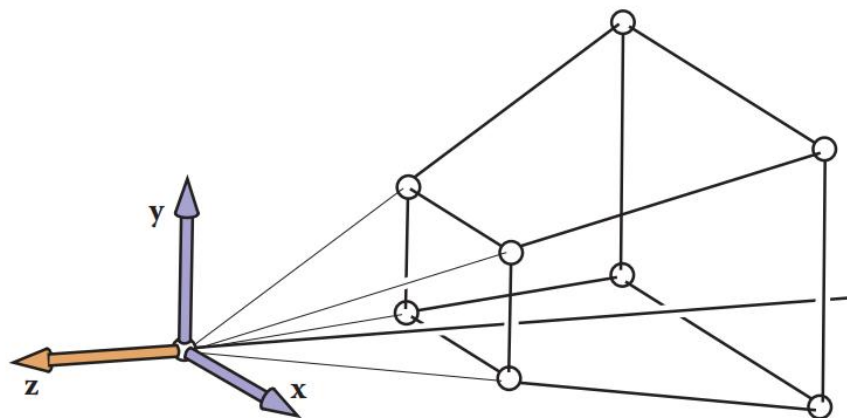
透视投影

- `glFrustum` 可以定义非对称视景体，而 `gluPerspective` 只能定义对称视景体



透视投影

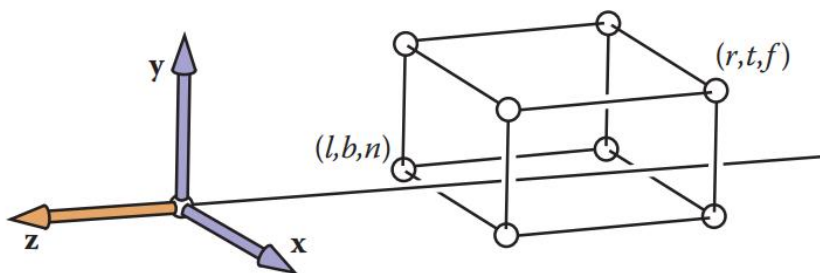
- 透视变换转成正交变换



$$z = -n \rightarrow -n$$

$$z = -f \rightarrow -f$$

$$P = \begin{bmatrix} -n & 0 & 0 & 0 \\ 0 & -n & 0 & 0 \\ 0 & 0 & -(n+f) & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



$$P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} -nx \\ -ny \\ -(n+f)z - fn \\ z \end{bmatrix} \sim \begin{bmatrix} -\frac{nx}{z} \\ -\frac{ny}{z} \\ -(n+f) - \frac{fn}{z} \\ 1 \end{bmatrix}$$

透视投影

- 透视变换的投影矩阵

$$\mathbf{M}_{\text{per}} = \mathbf{M}_{\text{orth}} \mathbf{P}$$

$$= \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

glFrustum对应的矩阵

$$\mathbf{P} = \mathbf{N} \mathbf{S} \mathbf{H} = \begin{bmatrix} \frac{2near}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & -\frac{2far * near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

视口变换glViewport

- glViewport(Ox, Oy, width, height)
 - 把规范化视景体中的内容“填入”指定的范围内
 - 规范化视景体坐标(x,y,z)
 - $WinX = Ox + (x+1)/2*width$
 - $WinY = Oy + (y+1)/2*height$
 - $WinZ = (z+1)/2$ (规范化到[0,1]之间)
 - 写成矩阵形式?
 - 注意：这里的WinY坐标和窗口系统的坐标（例如鼠标获取）朝向是反的

gluProject

- 学会此函数的用法
- 要求自行实现顶点变换的结果应与此函数结果一致