



中山大學
SUN YAT-SEN UNIVERSITY



国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

计算机图形学 观察

陶钧

taoj23@mail.sysu.edu.cn

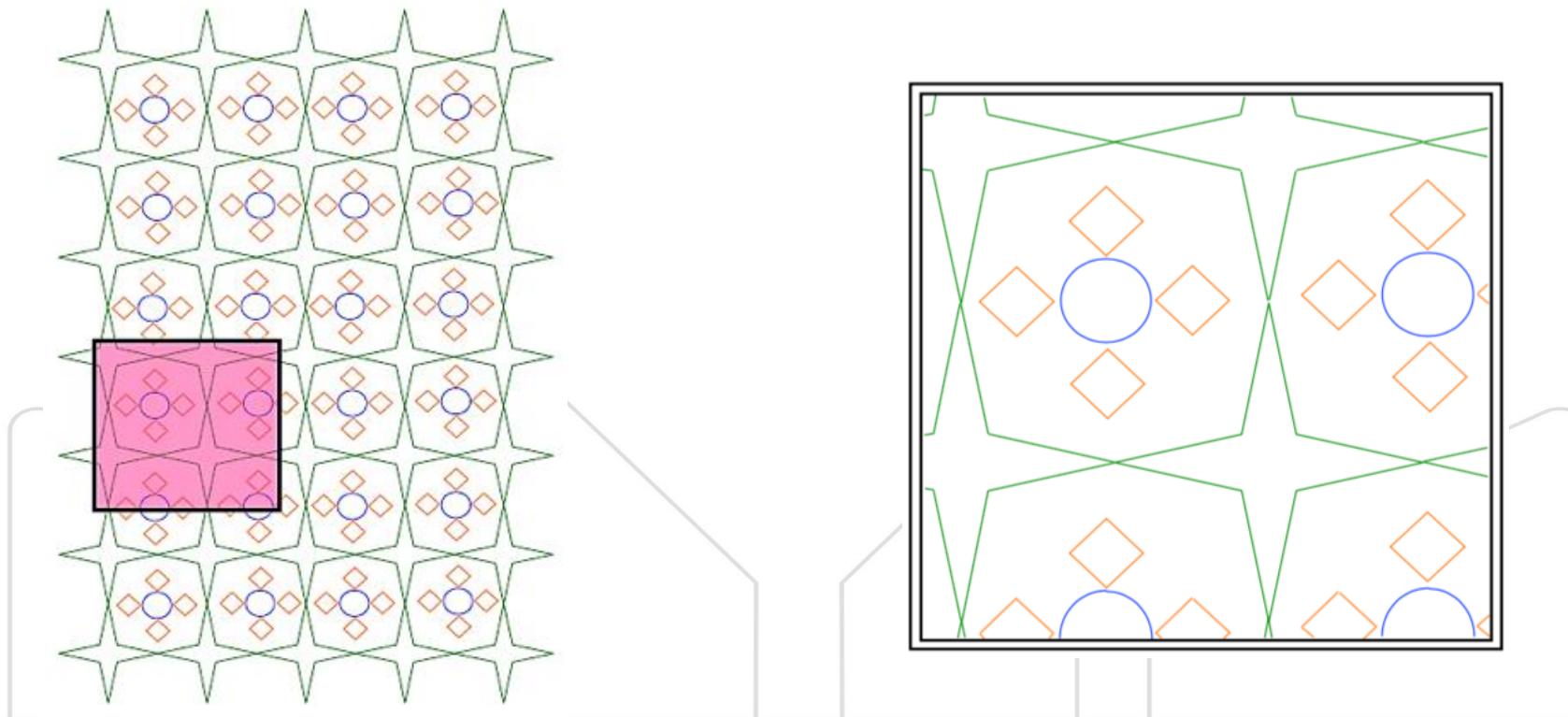
中山大学 数据科学与计算机学院
国家超级计算广州中心

- 二维观察
- 三维观察
 - 经典观察
 - 计算机观察
 - 定位照相机
 - 投影



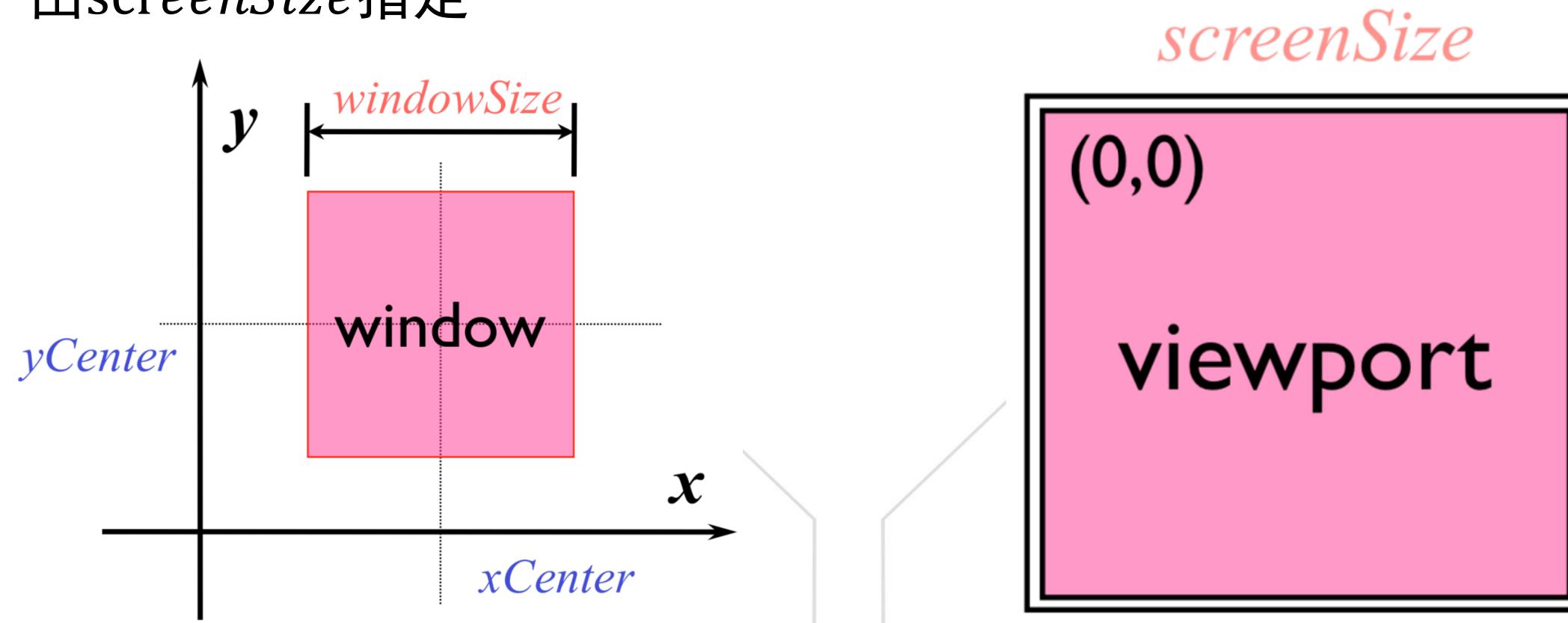
● 二维观察 (2D Viewing)

- 对二维平面图形进行绘制
- 世界可以是无限的，而屏幕一定是有有限的
- 依据需要观察的内容及细节程度，截取二维世界中的一个视窗
 - 将二维世界中的内容通过适当的变换映射至屏幕中的一部分



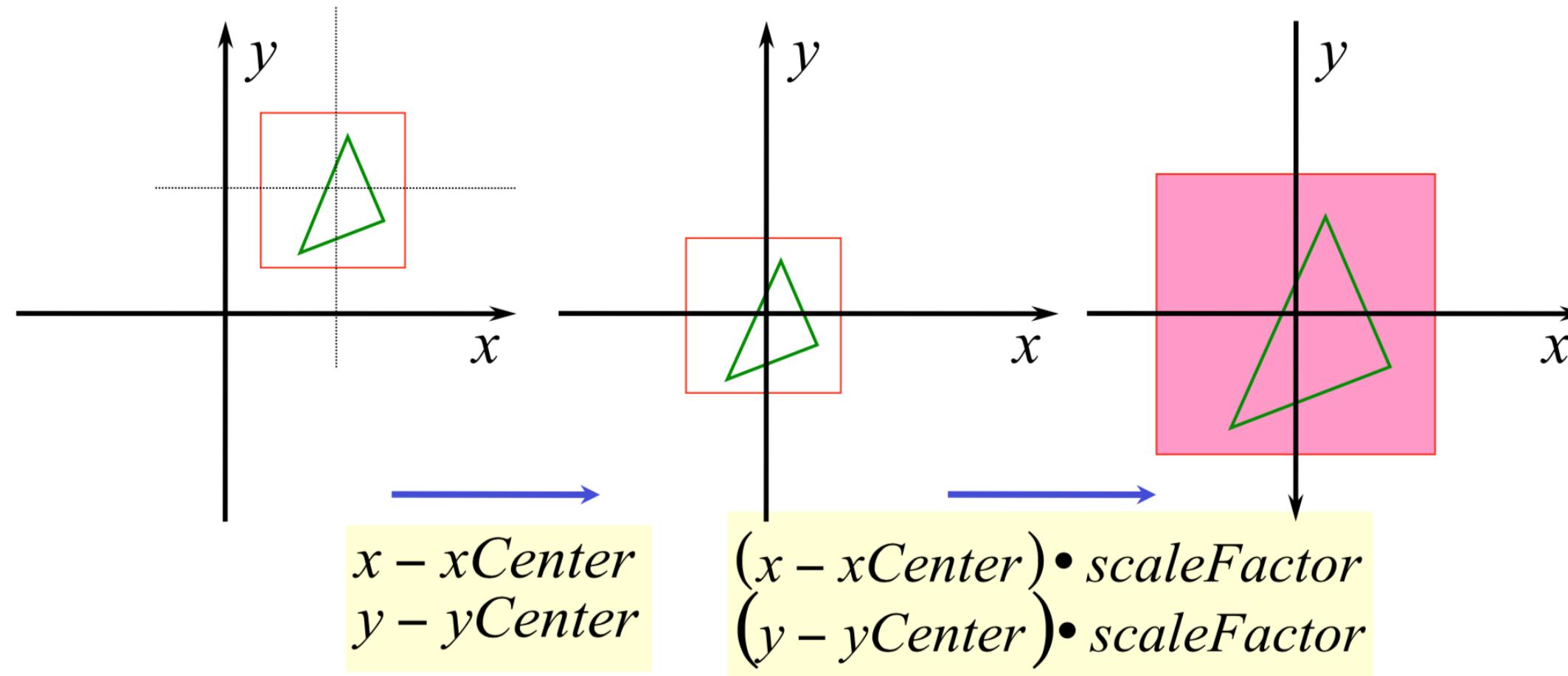
● 视窗 (window) 与视口 (viewport)

- Window是二维世界中的一个矩形区域（某种意义上连续）
 - 由 $center = (x, y)$ 及 $windowSize$ 指定
- Viewport是屏幕中由像素组成的矩阵（离散化）
 - 由 $screenSize$ 指定



● 二维观察变换 (2D viewing transformation)

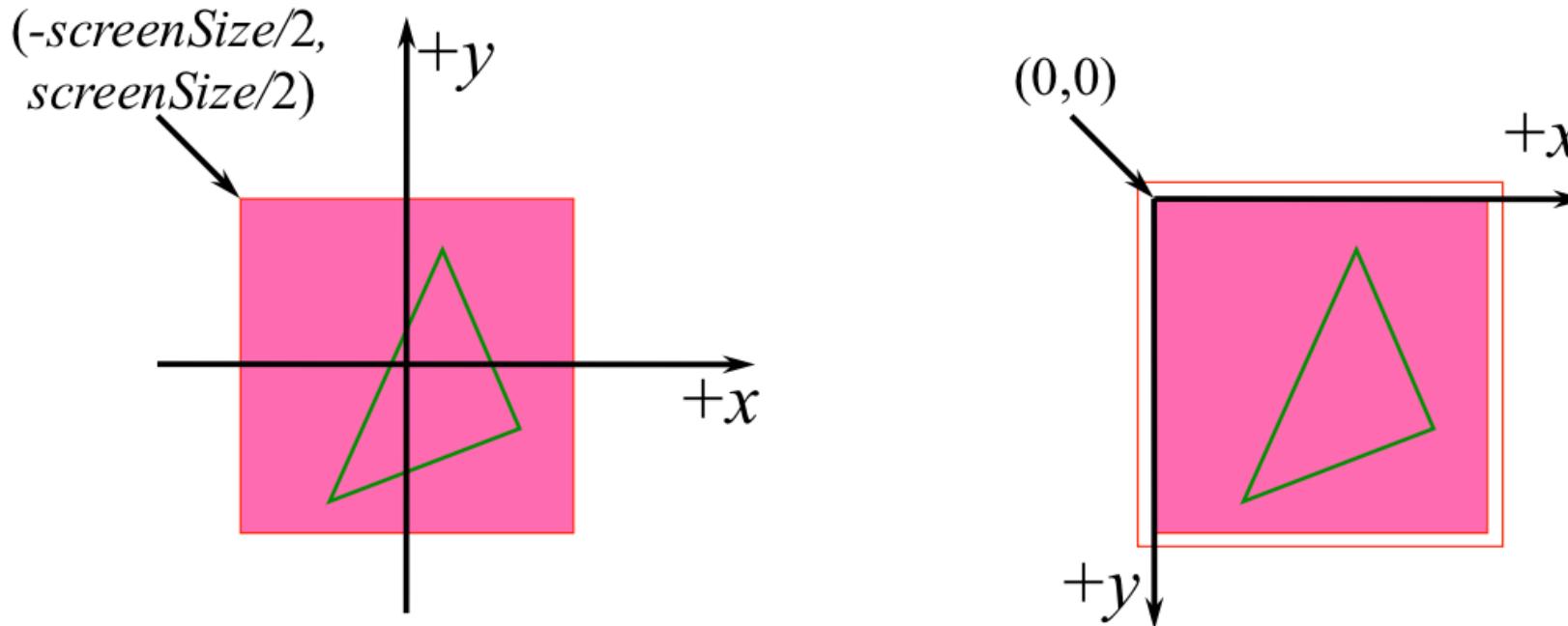
- 将二维世界中的window映射至屏幕中的viewport



$$\text{where, } scaleFactor = \frac{\text{screenSize}}{\text{windowSize}}$$

● 二维观察变换 (2D viewing transformation)

- 将二维世界中的window映射至屏幕中的viewport
 - 注意屏幕空间原点在左上角、y轴指向下方

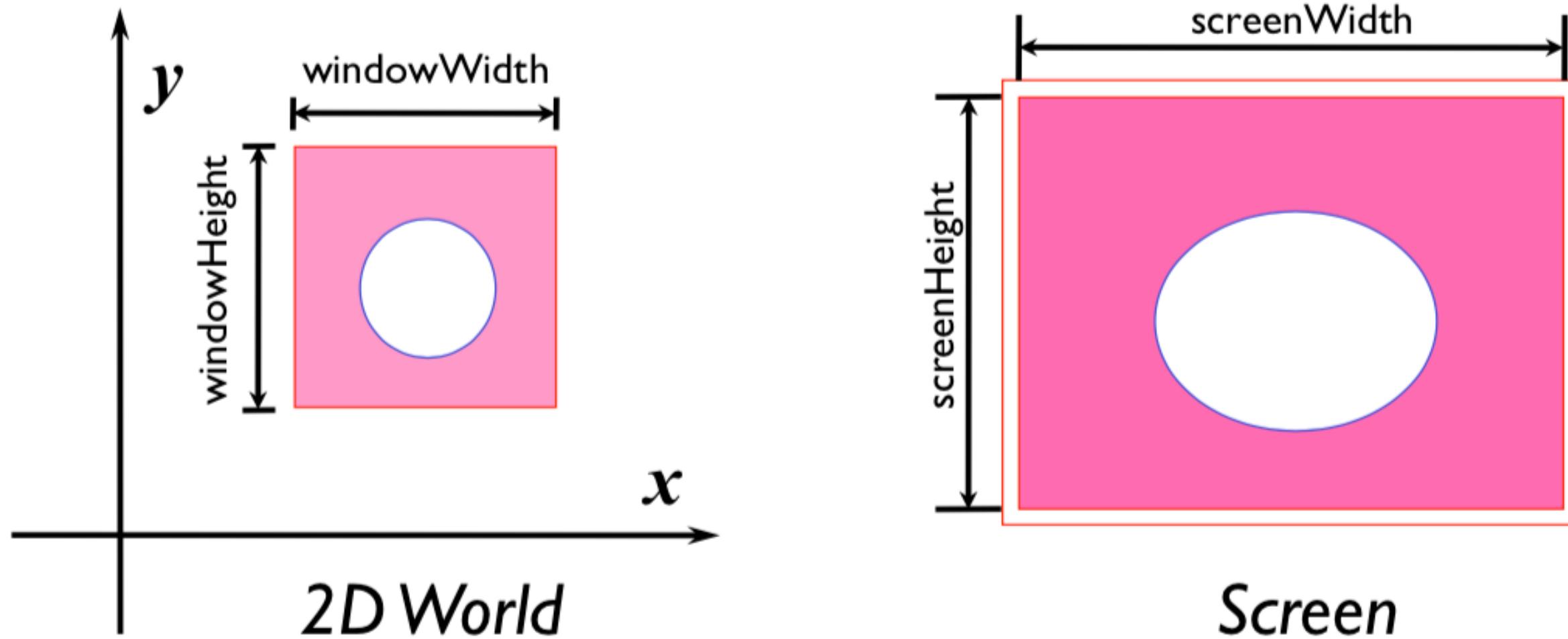


$$\frac{screenSize}{2} + (x - xCenter) \cdot scaleFactor$$
$$\frac{screenSize}{2} - (y - yCenter) \cdot scaleFactor$$

● 二维观察变换 (2D viewing transformation)

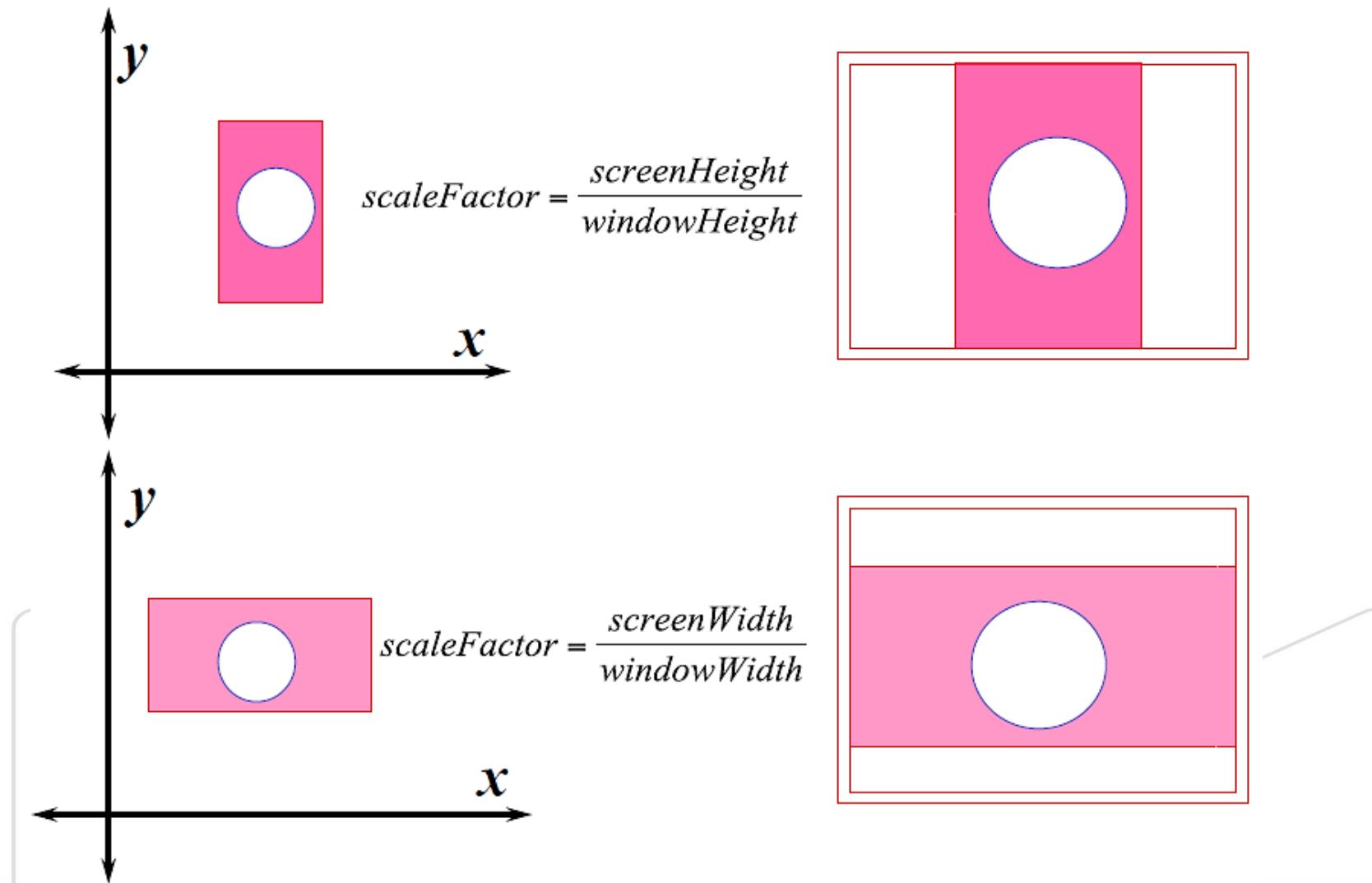
- 将二维世界中的window映射至屏幕中的viewport

- 为保持纵横比 (aspect ratio) 不变, x与y方向上的scaleFactor需一致



● 二维观察变换 (2D viewing transformation)

- 将二维世界中的window映射至屏幕中的viewport
 - 当纵横比不一致时，需选取合适的scaleFactor



● 二维观察变换 (2D viewing transformation)

- 将二维世界中的window映射至屏幕中的viewport
 - 当纵横比不一致时，需选取合适的scaleFactor



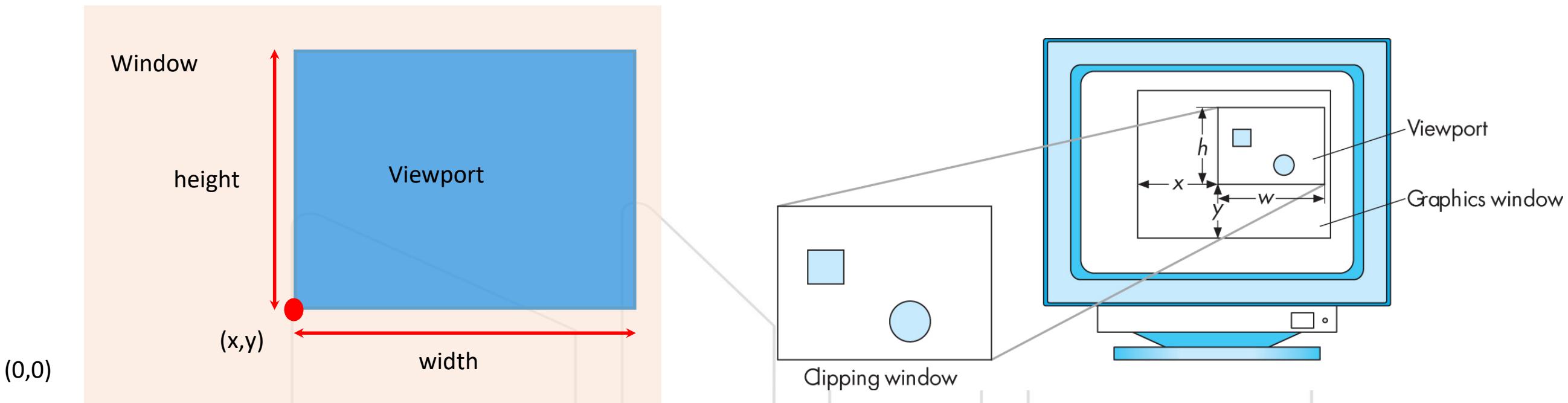
● OpenGL命令

– **gluOrtho2D(left, right, bottom, top)**

- 创建投影矩阵，将camera space中坐标变换为screen space中坐标

– **glViewport(x, y, width, height)**

- 定义最终绘制图像的像素矩阵。x, y指明viewport左下角位置，width, height指明viewport大小

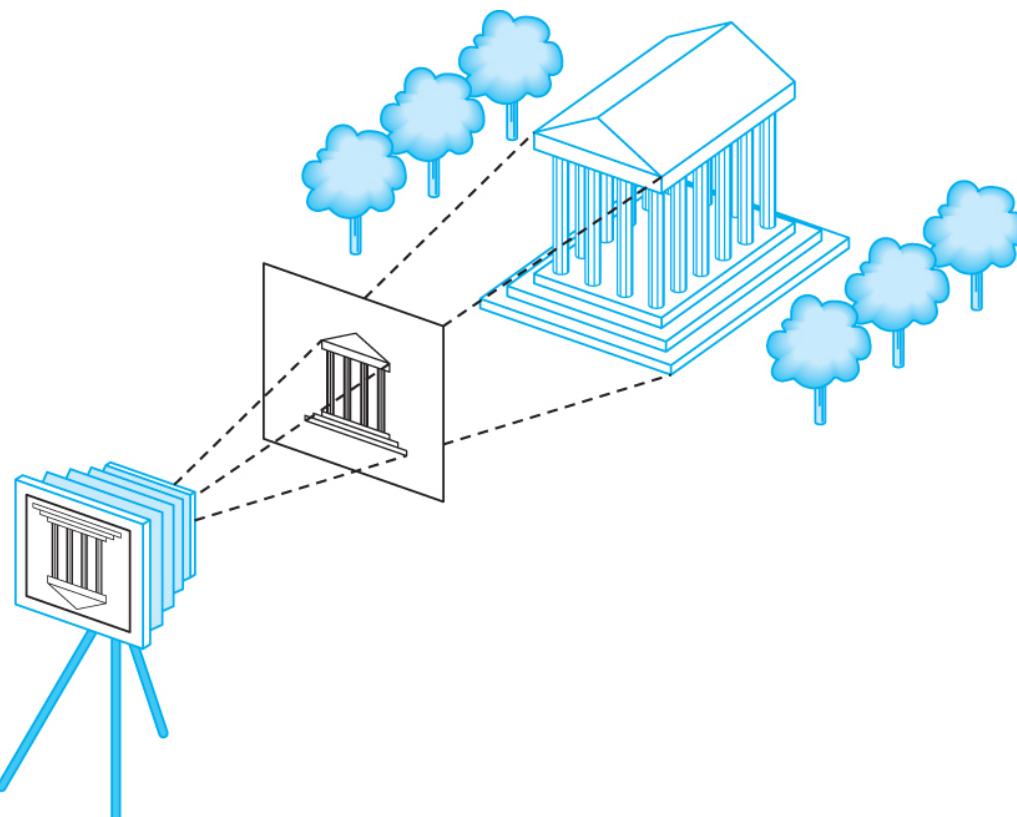


- 二维观察
- 三维观察
 - 经典观察
 - 计算机观察
 - 定位照相机
 - 投影



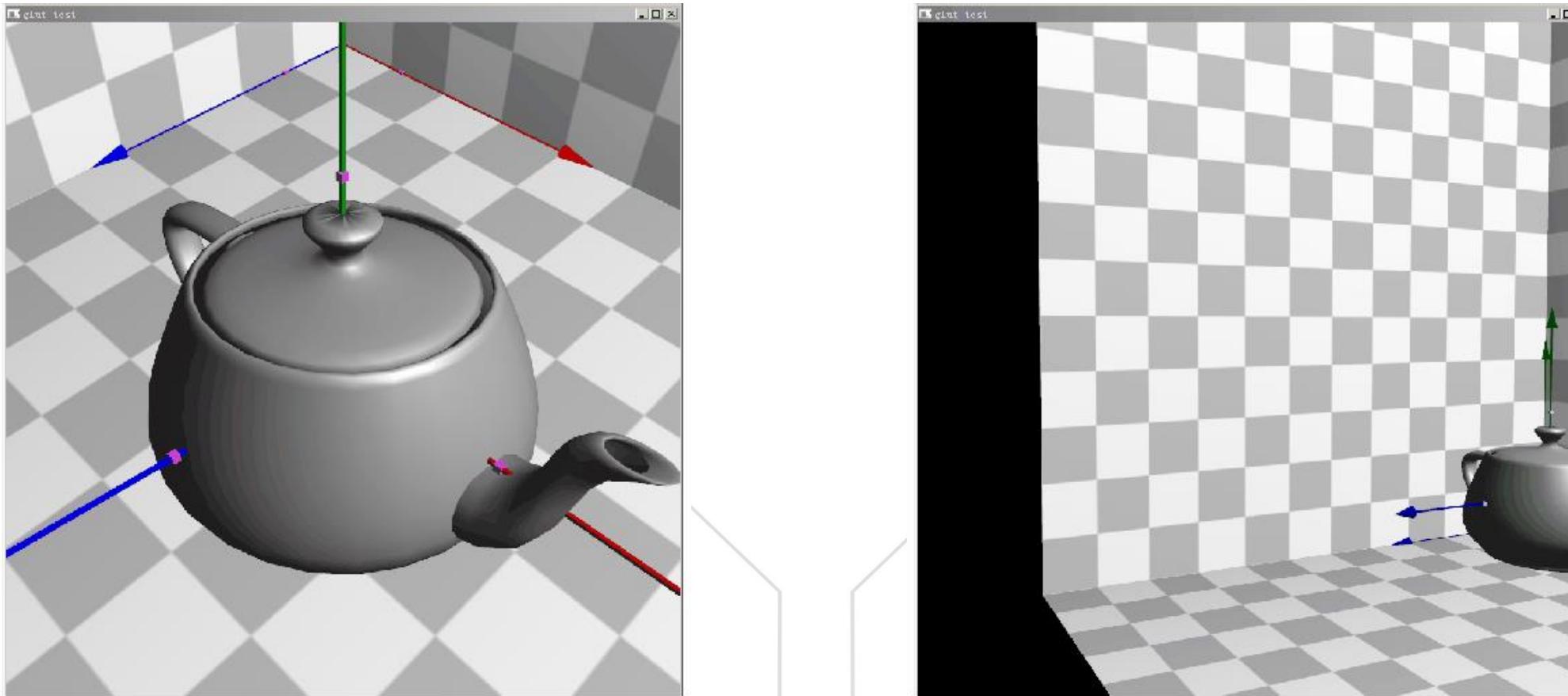
● 三维观察 (3D Viewing) 概述

- 将三维世界投影至二维屏幕
- 指明三维观察视角比二维更为复杂
 - 需要控制的参数更多
- 需要进行降维 (3D→2D)
 - 二维中只需要进行截取
- 三维观察类似于通过相机拍照
 - 一个或多个物体
 - 观察者与投影平面
 - 投影变换：从物体至投影平面



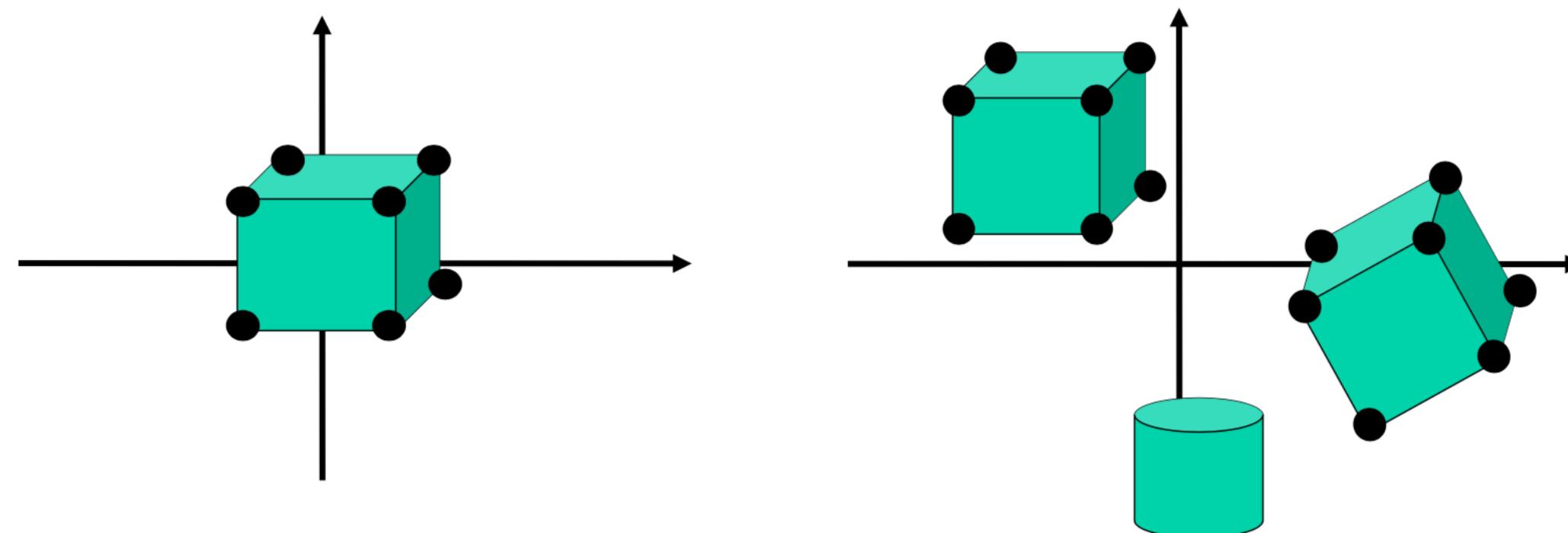
- 建模 (modeling) 与观察 (viewing)

- 建模变换改变物体在世界坐标中的位置
- 观察变换至改变从世界坐标到屏幕的投影过程
 - 观察变换不改变“世界”中的任何物体



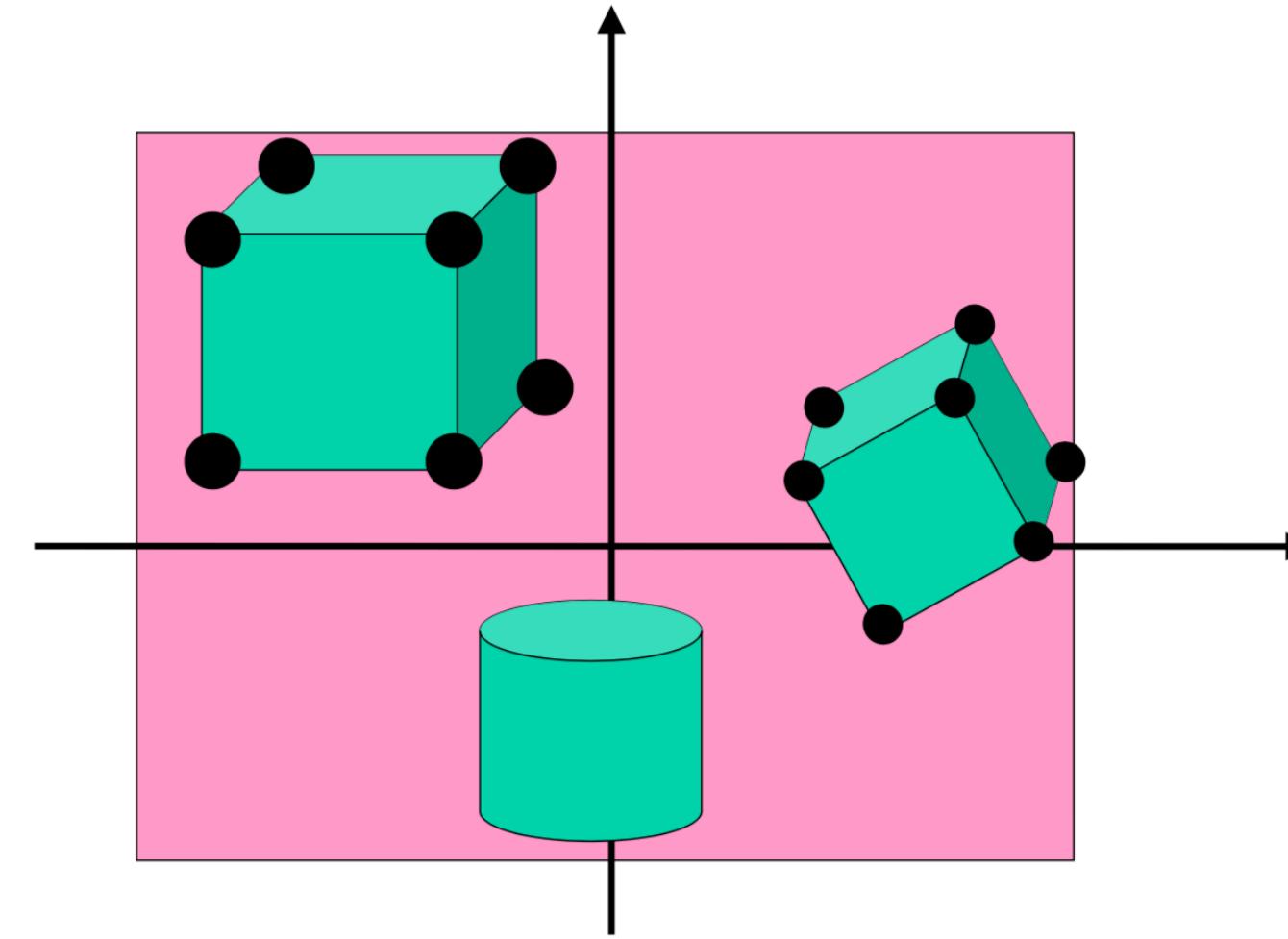
● 建模 (modeling) 与观察 (viewing)

- 物体通常在其局部坐标系统 (local coordinate system) 中定义
 - Instance transformation
- 将物体置于同一个场景 (world coordinate)
 - Modeling transformation



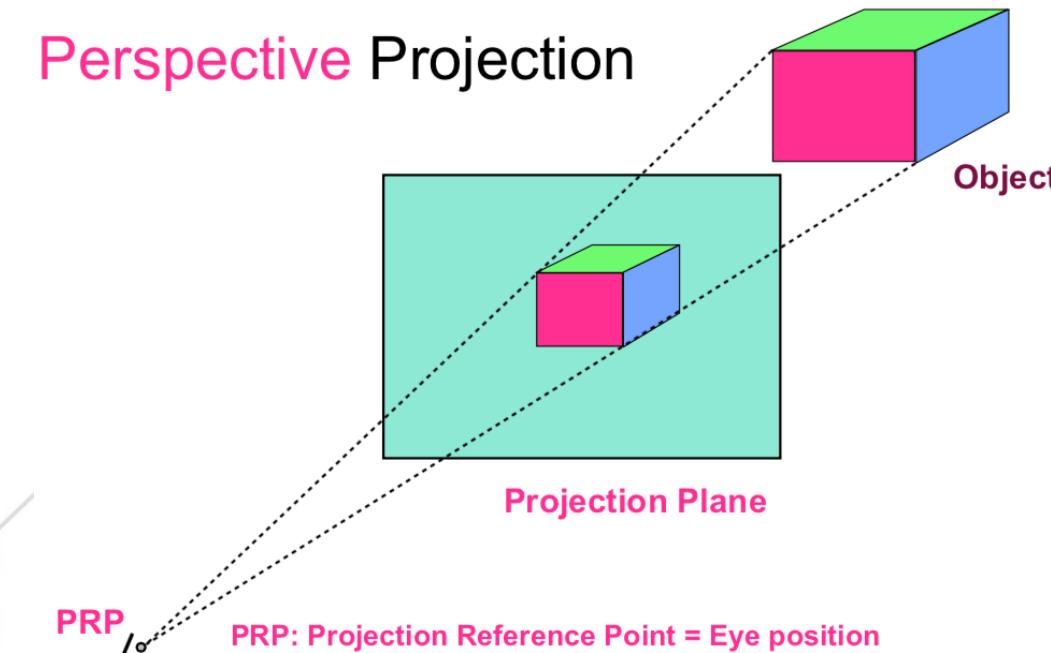
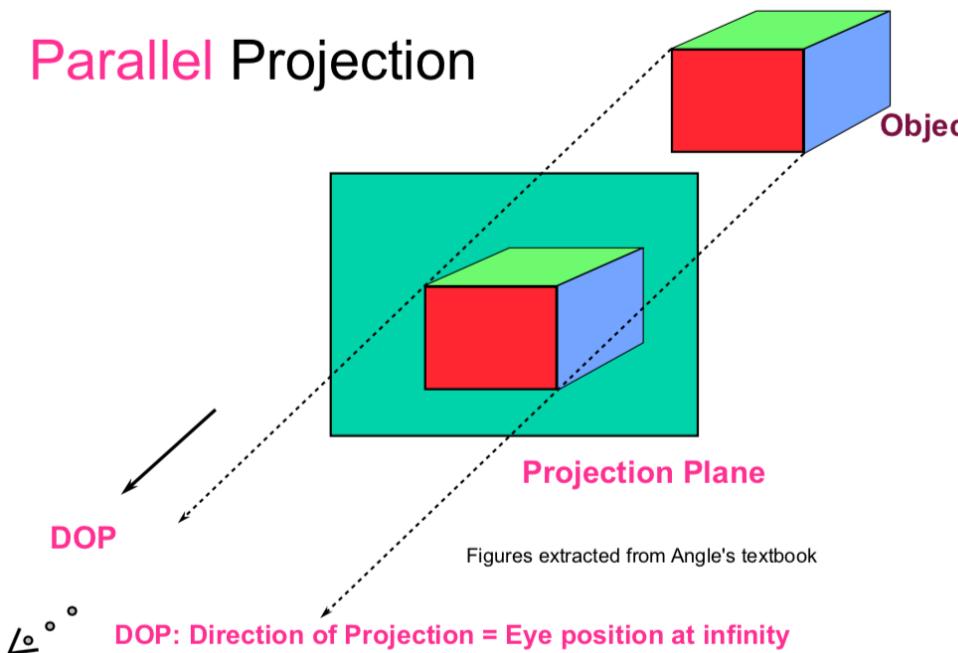
- 建模 (modeling) 与 观察 (viewing)

- 最终，通过观察变换将物体投影至屏幕
 - Viewing transformation



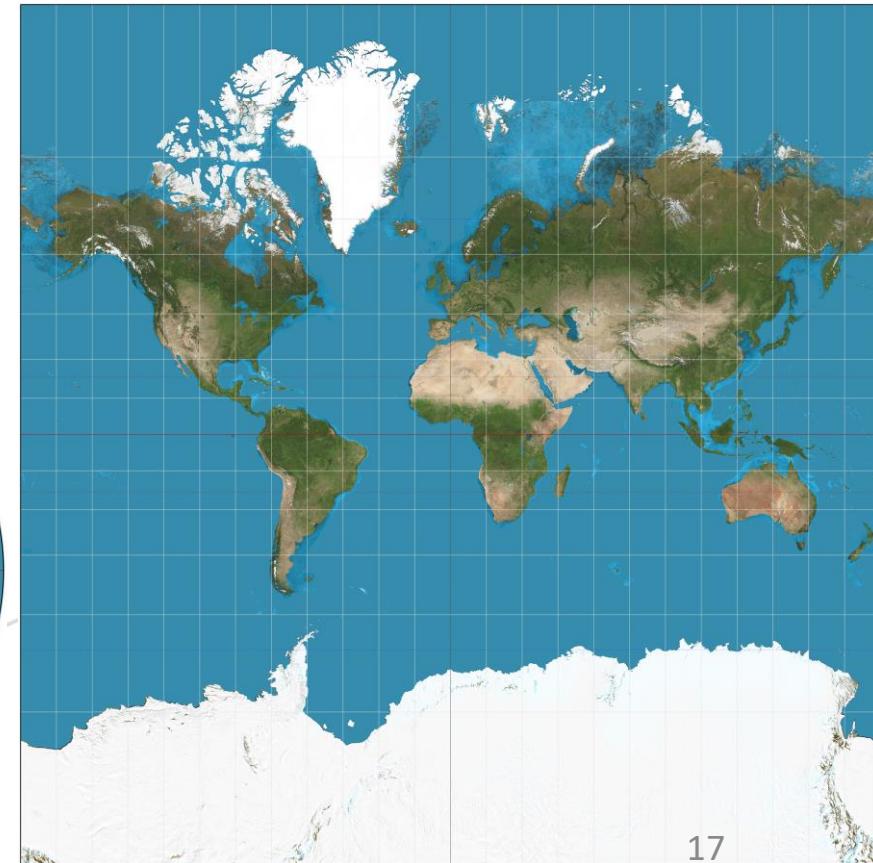
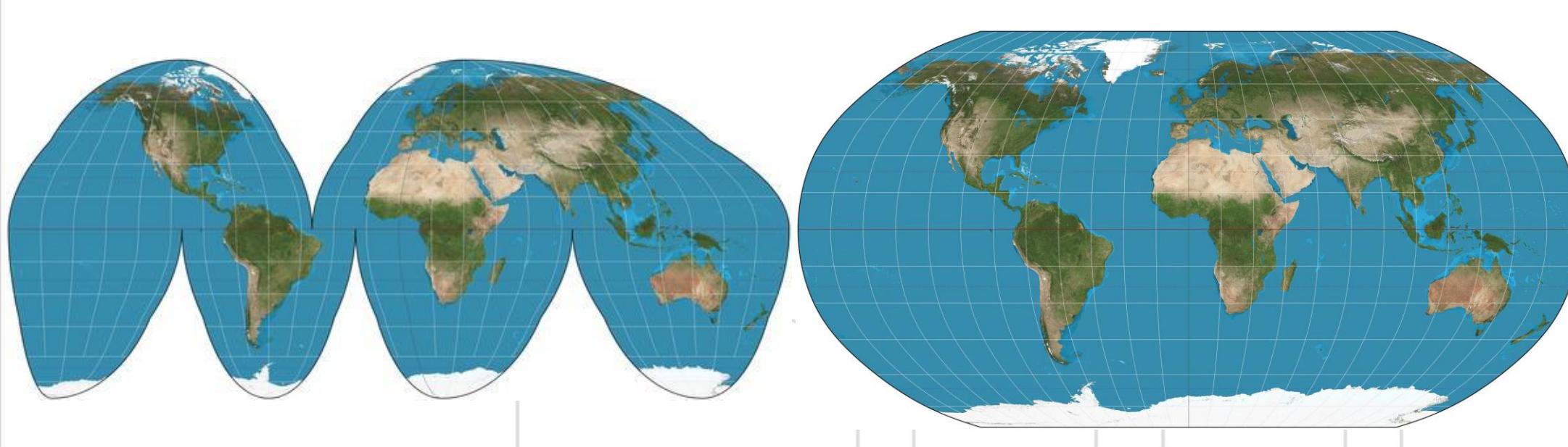
● 三维投影类型

- 三维投影的基本元素大体相同
 - 对象（物体）、观察者、投影线、投影平面
- 三维投影的方式并非唯一的
 - 平行投影：投影线为平行线，其方向称为DOP (direction of projection)
 - 透视投影：投影线相交于投影中心 (COP, center of projection)



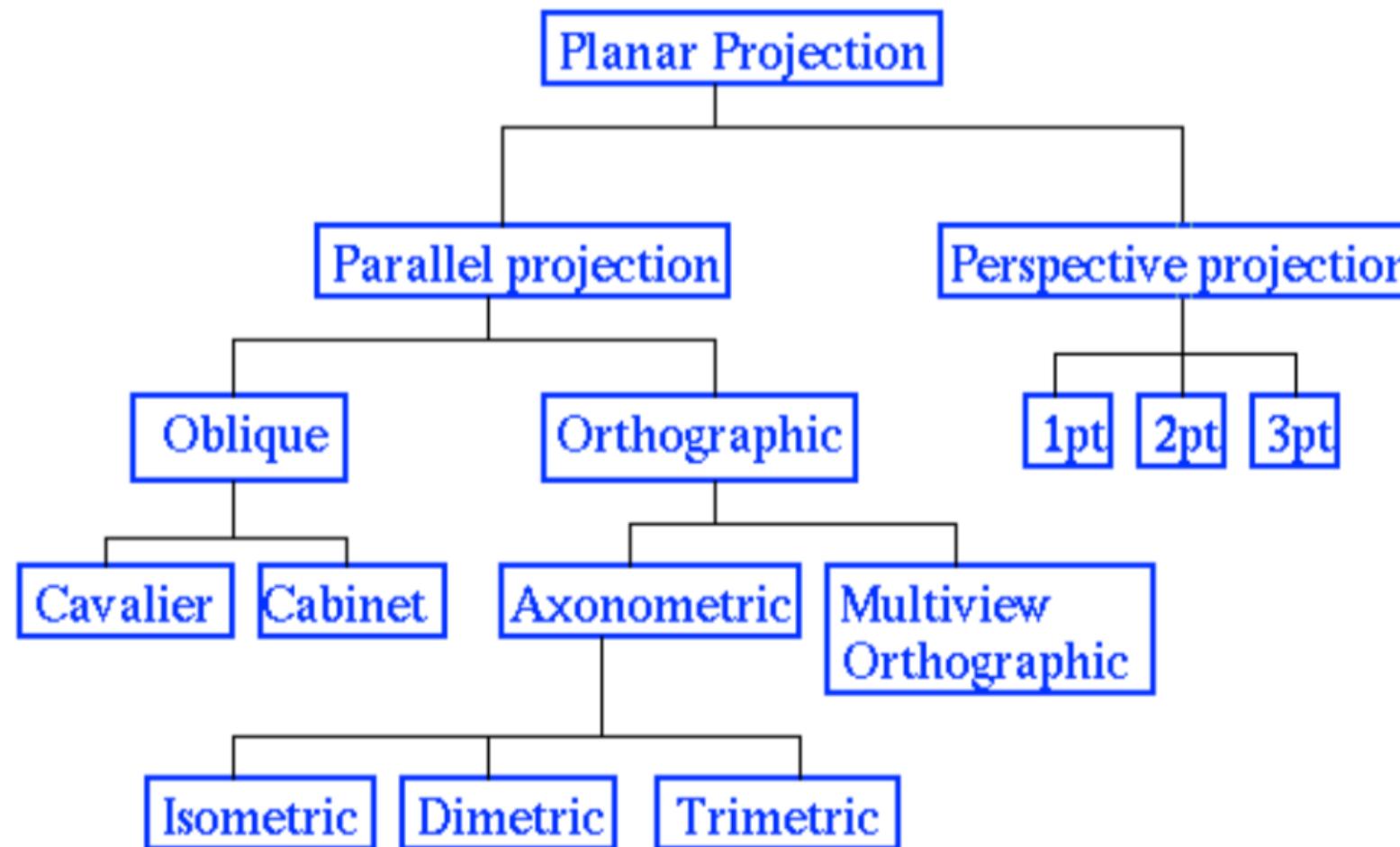
● 三维投影类型

- 三维投影的方式并非唯一的
 - 平行投影与透视投影均为平面投影，投影过程中保持共线性
 - 直线仍为直线，尽管平行线未必仍然平行
 - 存在非平面投影，如地图映射
 - 球面→平面：不存在理想的投影，保持所有几何性质不变
 - » 面积、角度（conformal）



● 三维投影类型

- 三维投影的方式并非唯一的
 - 在此，我们只讨论平面投影（仍然具有多种方式！）



● 经典观察

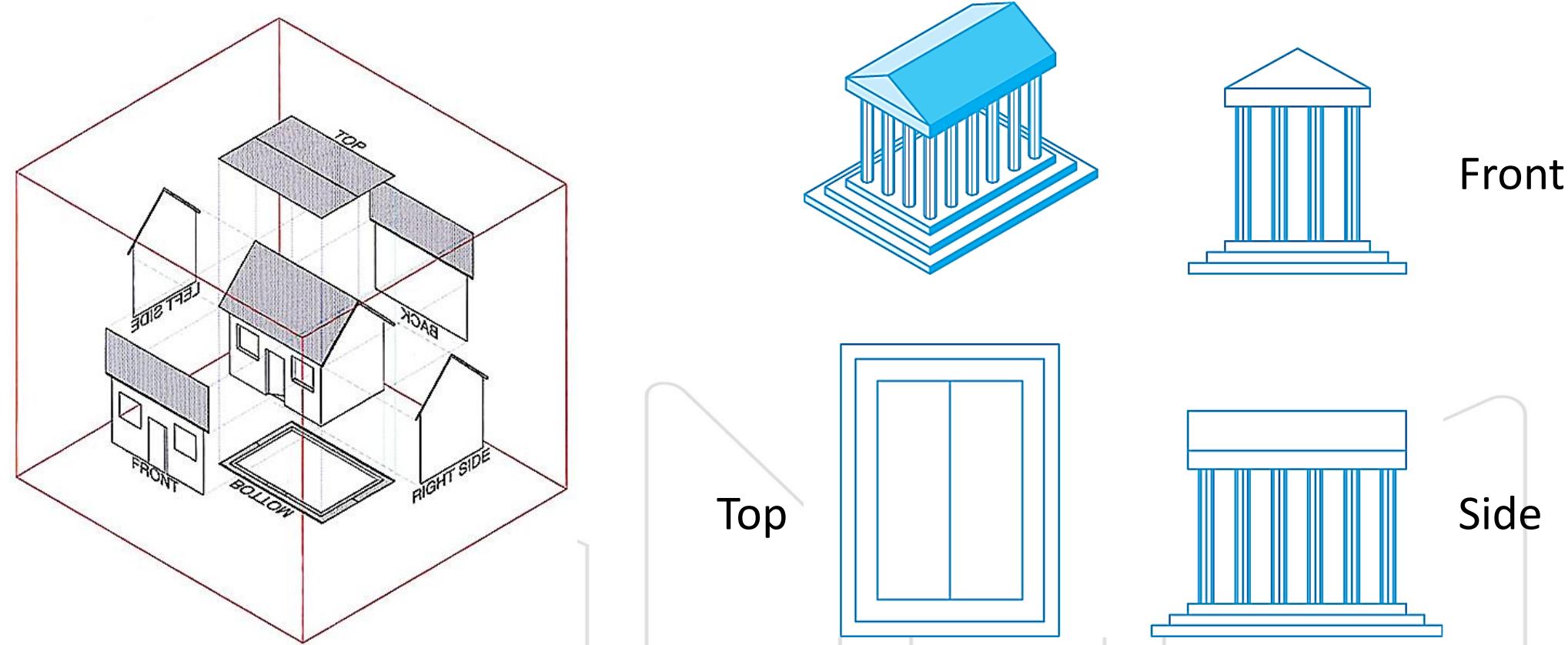
- 在计算机出现前，投影就已经广泛应用
 - 绘画、建筑设计、工业设计
 - 文艺复兴时期的绘画已经出现较为系统的透视投影



● 经典观察

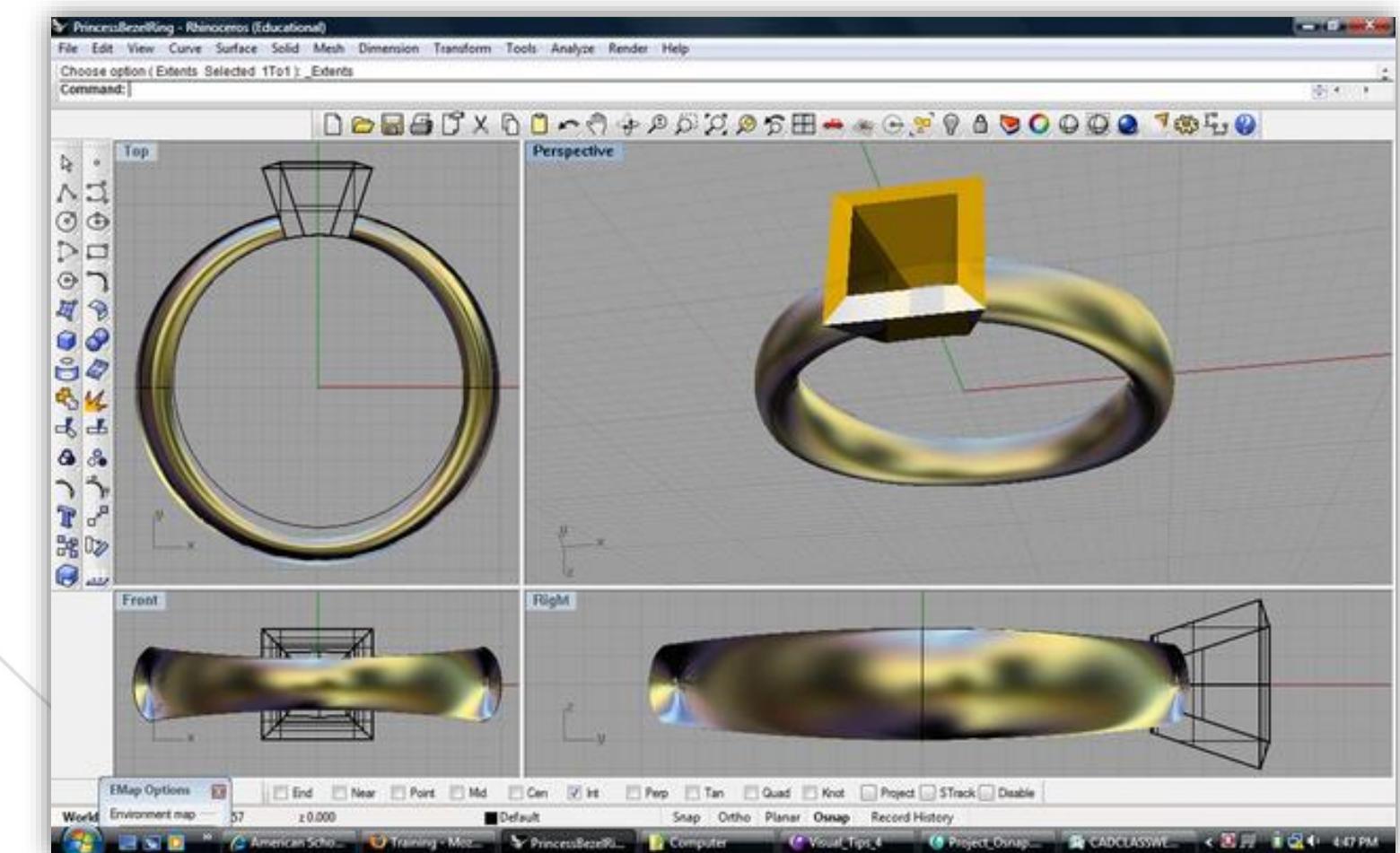
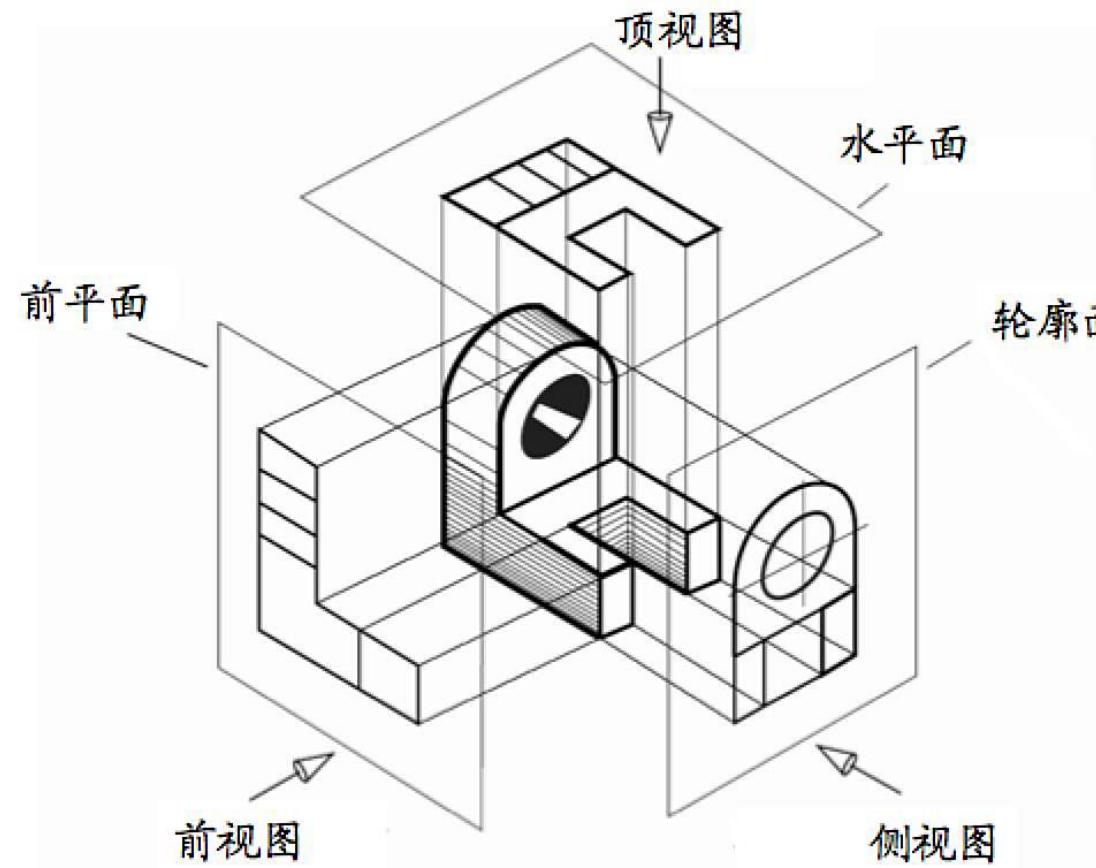
- 多角度正交投影 (multi-view orthographic projection)

- 投影平面平行于参照平面（对象某个主面），投影线垂直于投影平面
- 通常从前视，顶视，与侧视三个角度观察（投影）



● 经典观察

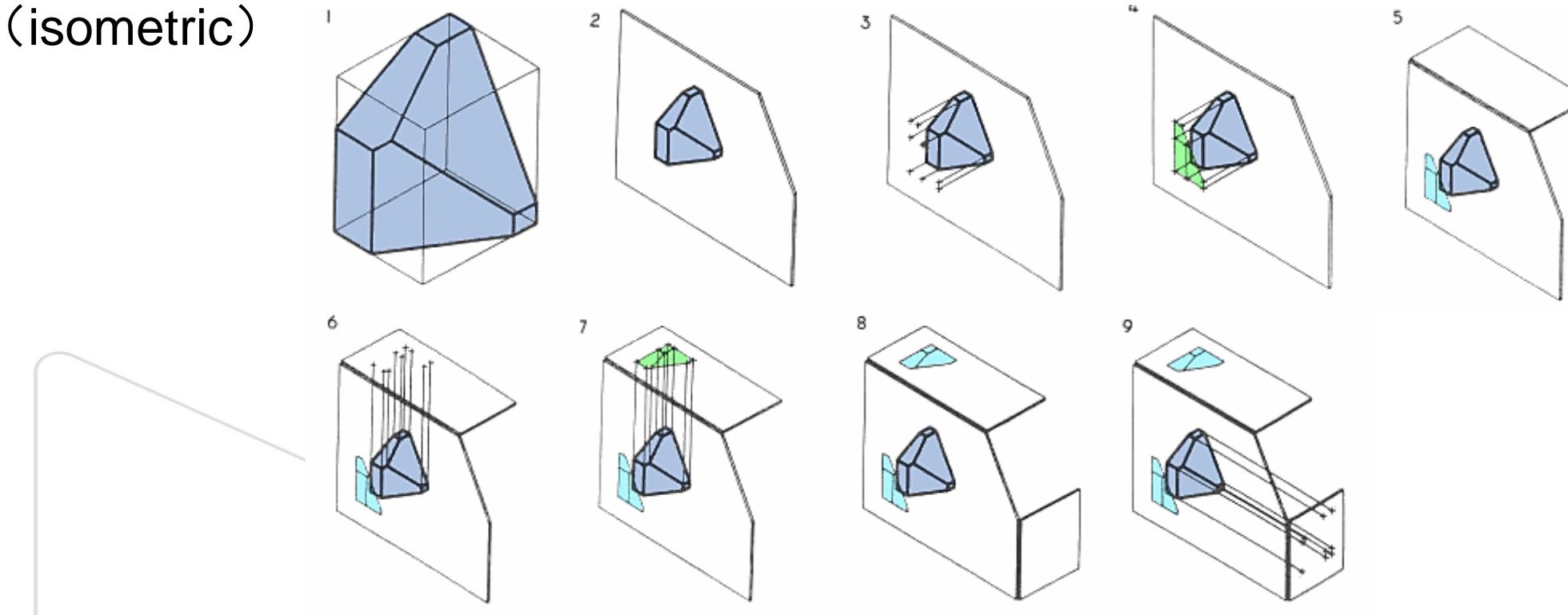
- 多角度正交投影 (multi-view orthographic projection)
 - 在CAD与建筑行业中，常显示三角度正交投影图及等角投影图



● 经典观察

- 多角度正交投影 (multi-view orthographic projection)

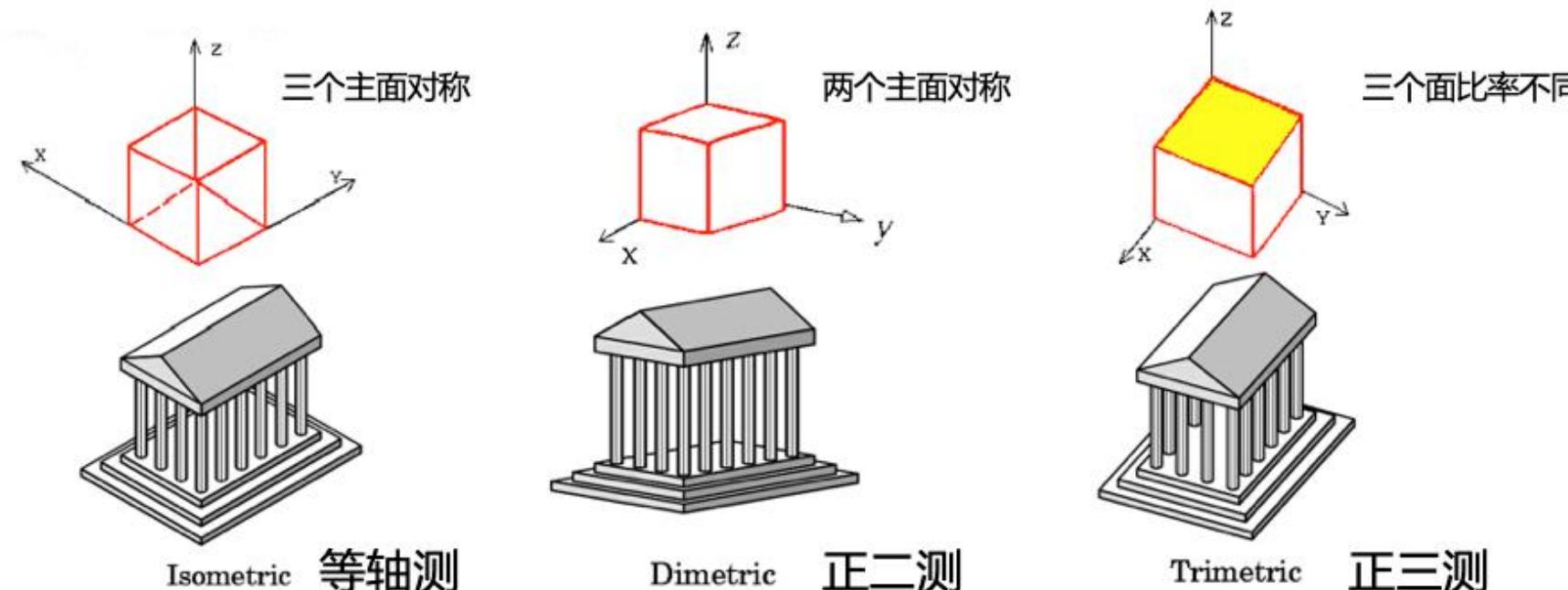
- 优点：保持距离和角度不变
 - 形状不发生改变、可用于测量，适用于施工图纸
- 缺点：无法观察到物体的整体形状，许多面在这三个视角中无法观察
 - 从正交投影中想象物体的形状往往需要观察者具备一定技能，因此，通常添加等角图 (isometric)



● 经典观察

– 轴测投影 (axonometric projection)

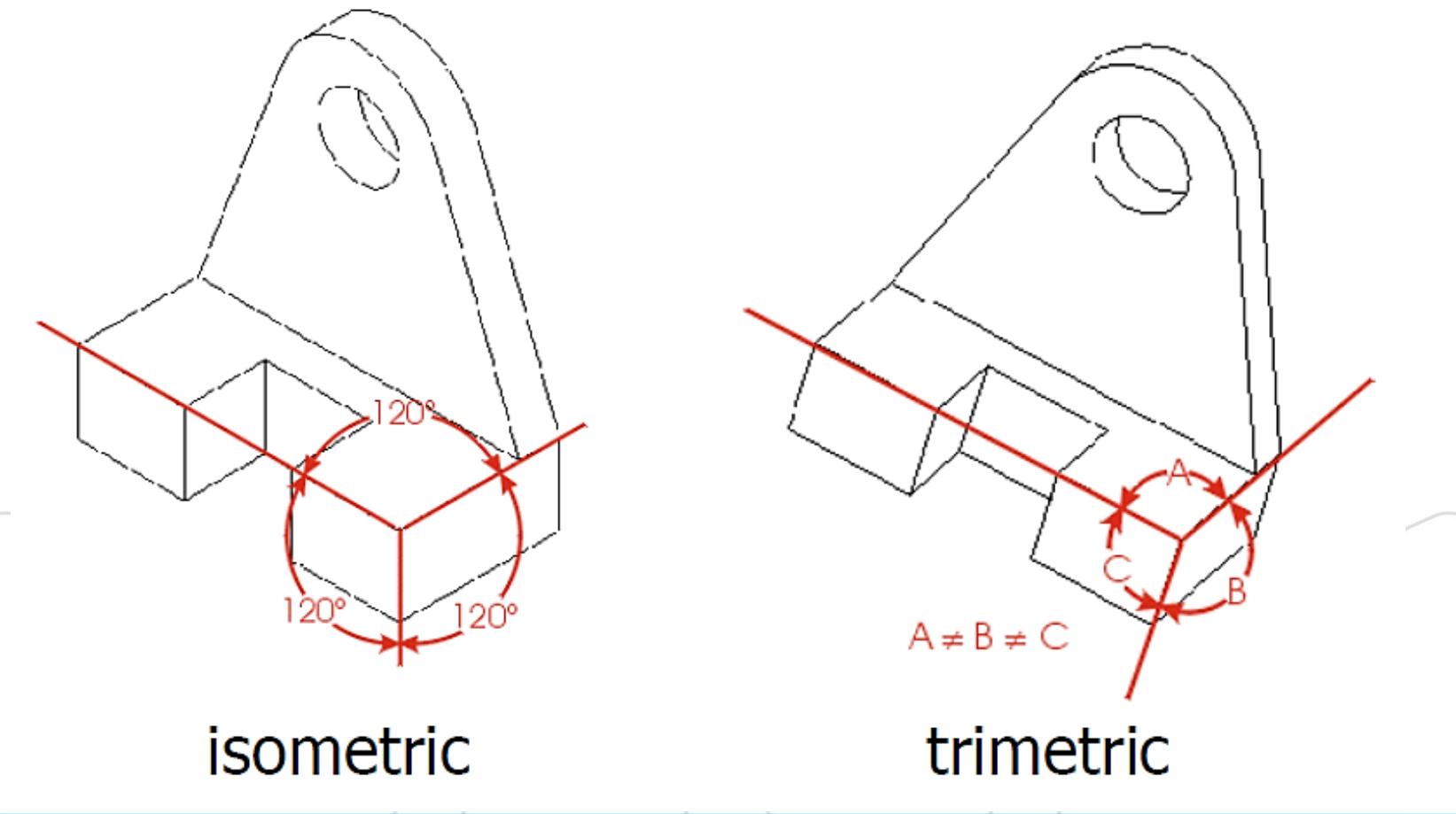
- 投影线仍垂直于投影平面，但投影平面相对于对象的方向可任意
- 存在投影缩短 (foreshortening)
 - 线段在图像空间里长度比对象空间中长度短
 - 不同类型投影在不同方向上缩短的比例不一
 - 平行线仍平行，但角度可能会改变；圆投影后成为椭圆



● 经典观察

– 轴测投影 (axonometric projection)

- 投影线仍垂直于投影平面，但投影平面相对于对象的方向可任意
- 存在投影缩短 (foreshortening)



● 经典观察

- 斜平行投影 (oblique projection)

- 最为一般的平行投影
- 投影线与投影平面成任意角度
- 平行线仍平行
 - 常用折叠暗箱照相机生成建筑物图像



折叠暗箱照相机



cavalier

斜等测

Cavalier

Angle between projectors and projection plane is 45° . Perpendicular faces are projected at full scale



cabinet

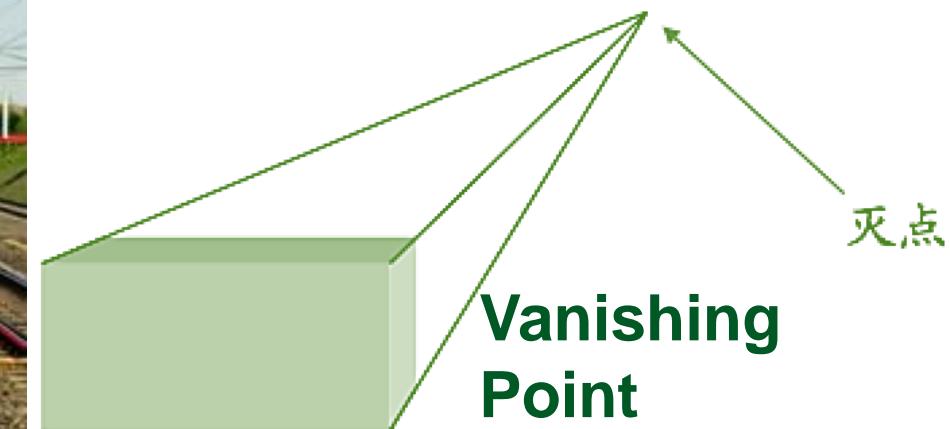
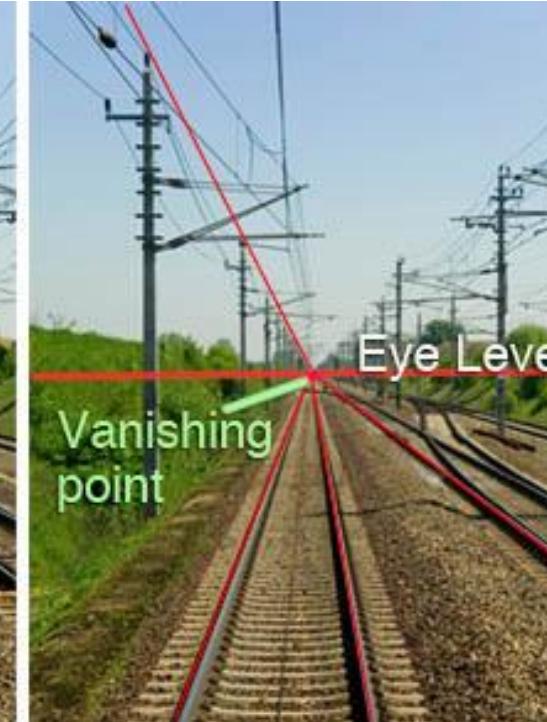
斜二测

Cabinet

Angle between projectors and projection plane is 63.4° . Perpendicular faces are projected at 50% scale

● 经典观察

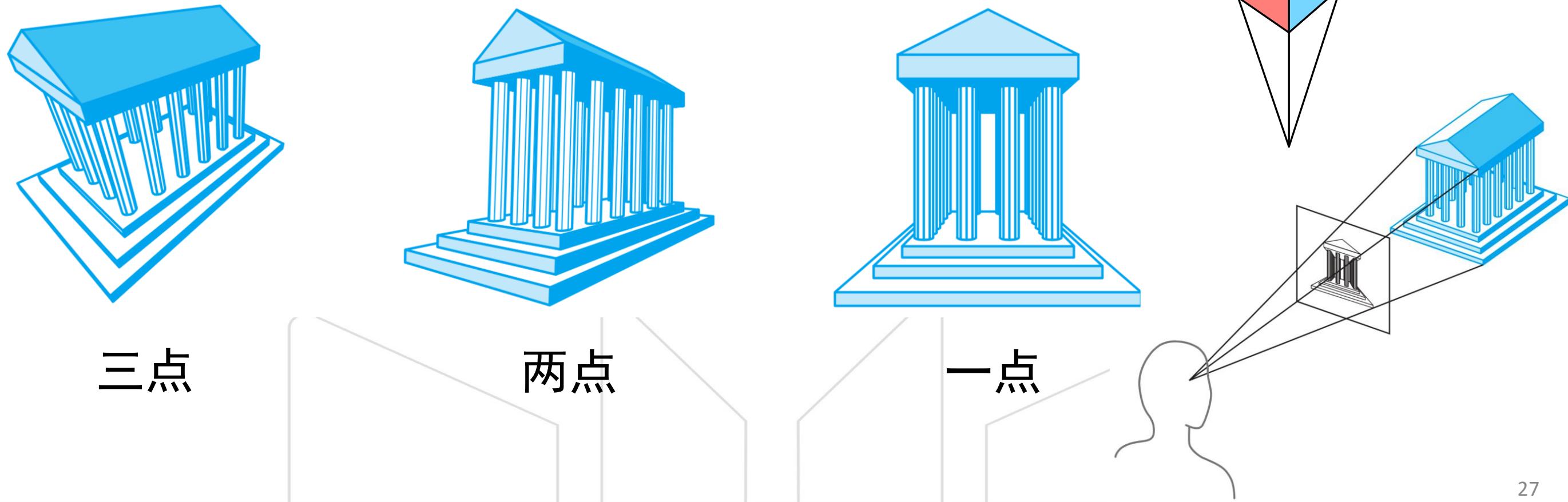
- 透视投影 (perspective projection)
 - 投影线汇于一点，不垂直于投影平面
 - 近大远小，存在灭点 (vanishing point)
 - 不保持平行，不能用于测量



● 经典观察

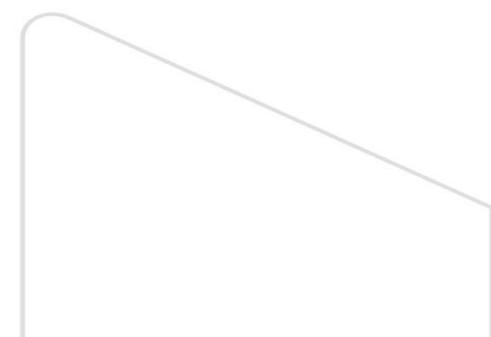
- 透视投影 (perspective projection)

- 对象的三个主方向中平行于投影平面的数量



- 二维观察
- 三维观察
 - 经典观察
 - 计算机观察

- 定位照相机
- 投影



● 经典观察与计算机观察

- 计算机观察建立在虚拟照相机模型上，理论上能生成任何一种经典投影图
- 但两者间有**本质区别**
 - 经典投影图依赖于对象，观察者（虚拟照相机），和投影线之间的特定位置关系
 - 计算机图形学强调对象定义和照相机参数设置之间的独立性

- 计算机观察的三个主要步骤（实现于渲染管线中）

- 设置照相机位置

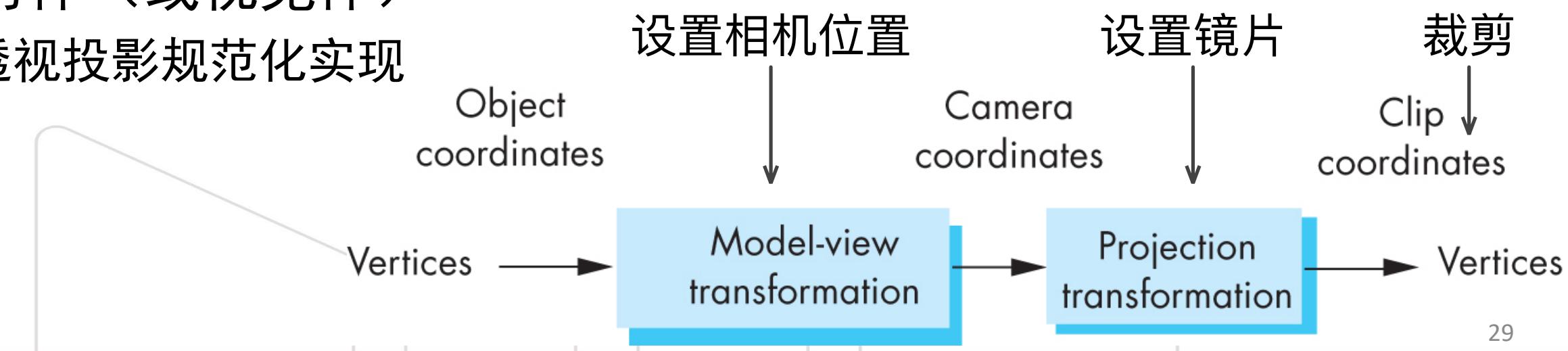
- 该设置方式应独立于对象定义
 - 通过修改modelview矩阵实现

- 设置透镜

- 执行平行投影或透视投影
 - 通过修改projection矩阵实现

- 设置裁剪体（或视见体）

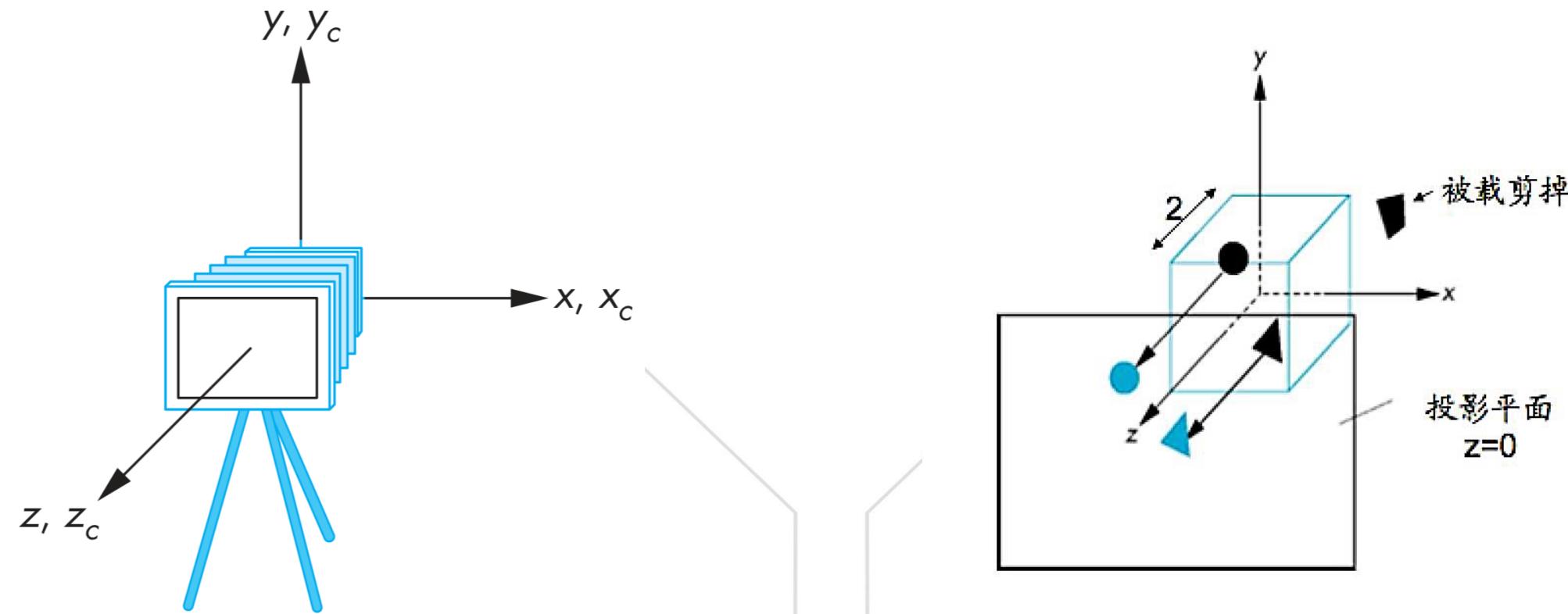
- 通过透视投影规范化实现



● OpenGL中的虚拟照相机

– 初始状态下，OpenGL中的对象标架与相机标架一致

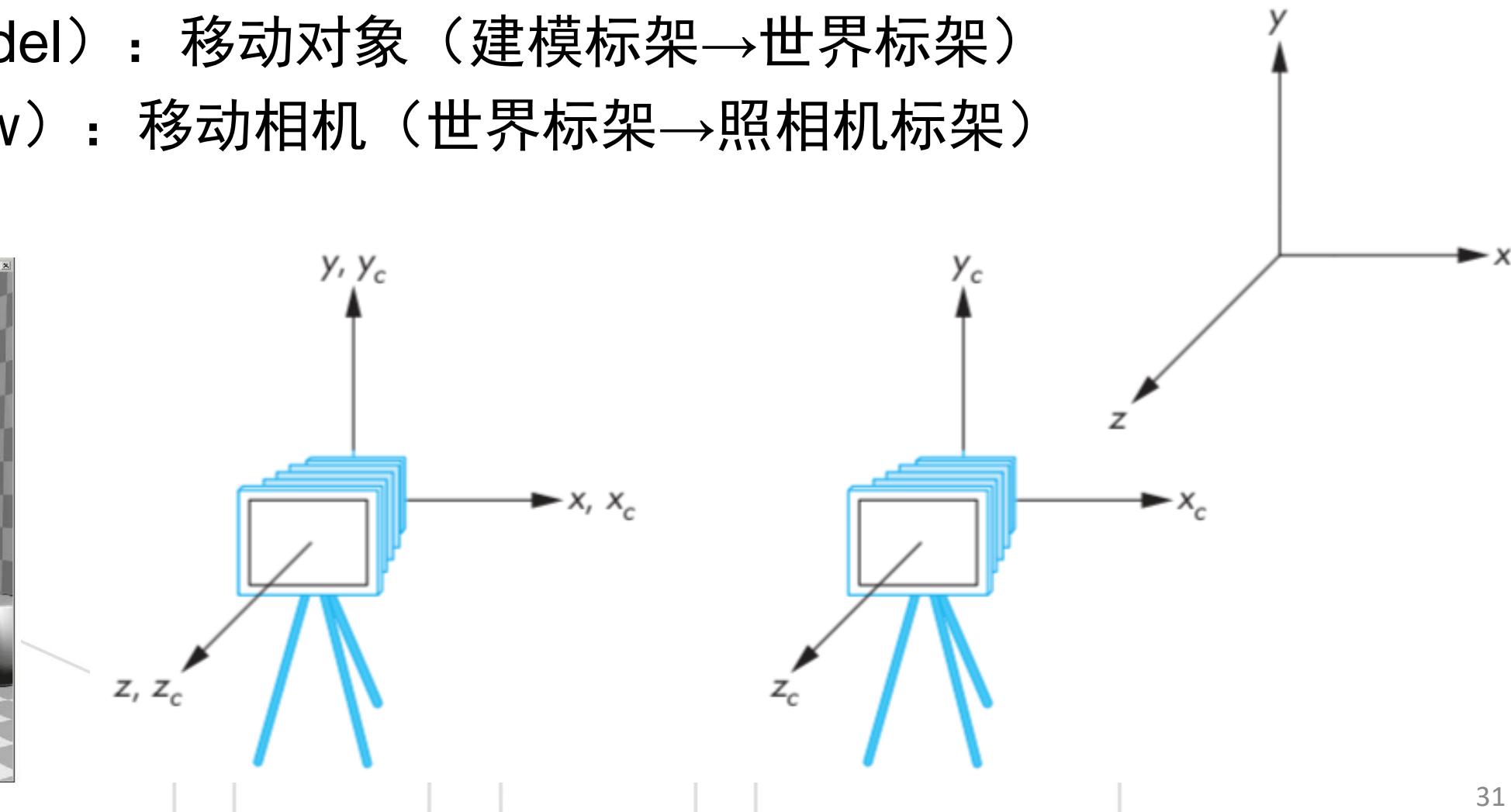
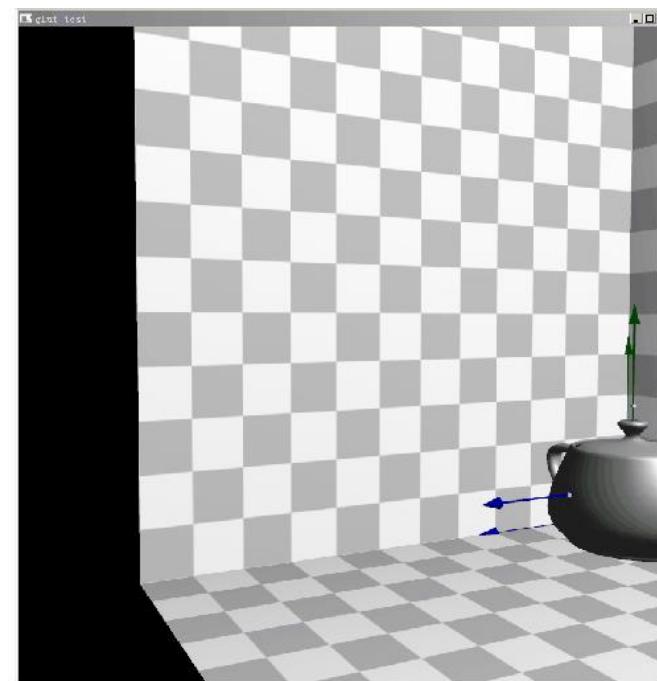
- $[x, y, z] = [x_c, y_c, z_c]$
- 照相机位于原点，朝向z轴负方向
- 默认frustum为以原点为中心，宽度为2的立方体（正交投影）



● OpenGL中的虚拟相机

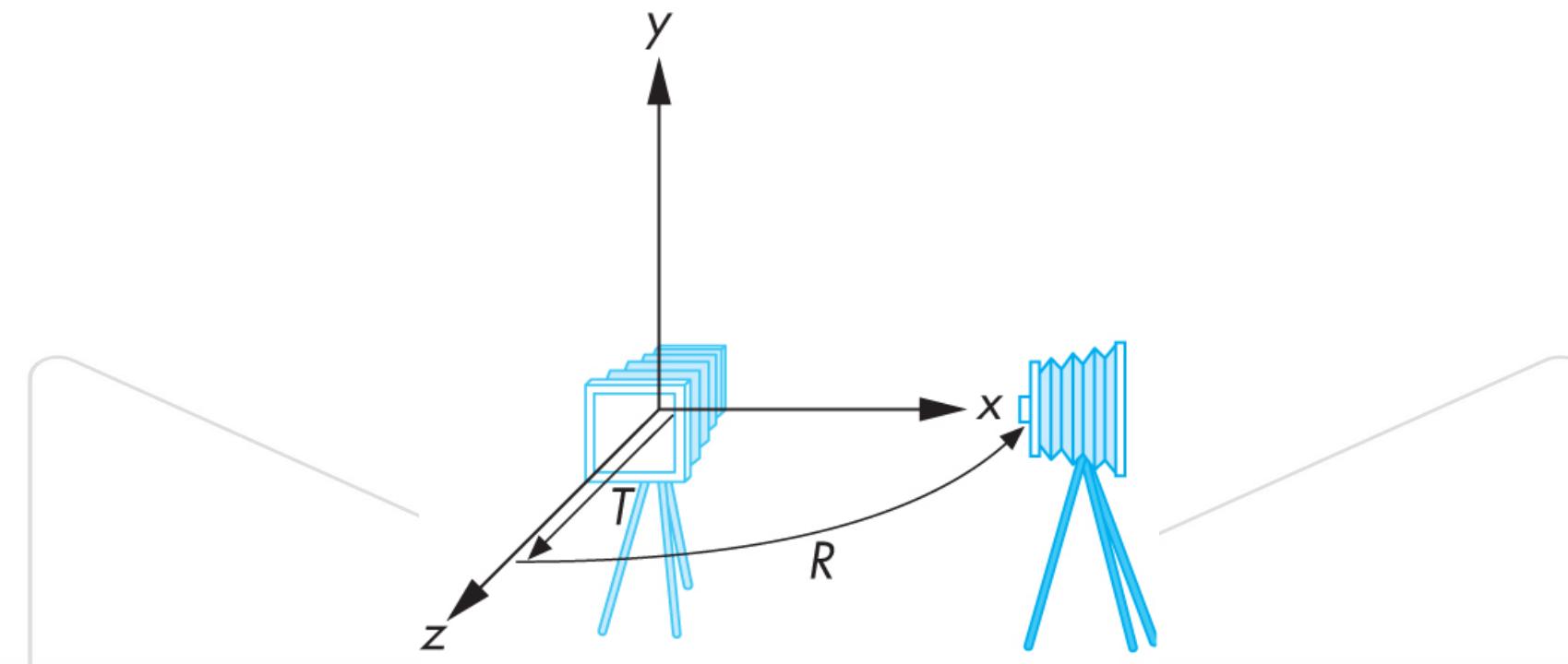
– Modelview变换指明了对象与照相机间的相对运动

- 将物体移动至照相机的frustum进行显示
- 建模矩阵 (model) : 移动对象 (建模标架 \rightarrow 世界标架)
- 观察矩阵 (view) : 移动相机 (世界标架 \rightarrow 照相机标架)



● OpenGL中的虚拟照相机

- Modelview变换指明了对象与照相机间的相对运动
 - 移动对象与移动照相机是等价的
 - 因此, OpenGL使用modelview矩阵进行变化, 而不是分别使用建模和观察矩阵
 - 如, 将照相机沿z轴正方向移动等价于将物体沿z轴负方向移动
 - 如, 为得到物体的侧视图, 可将物体旋转, 或将照相机旋转后平移

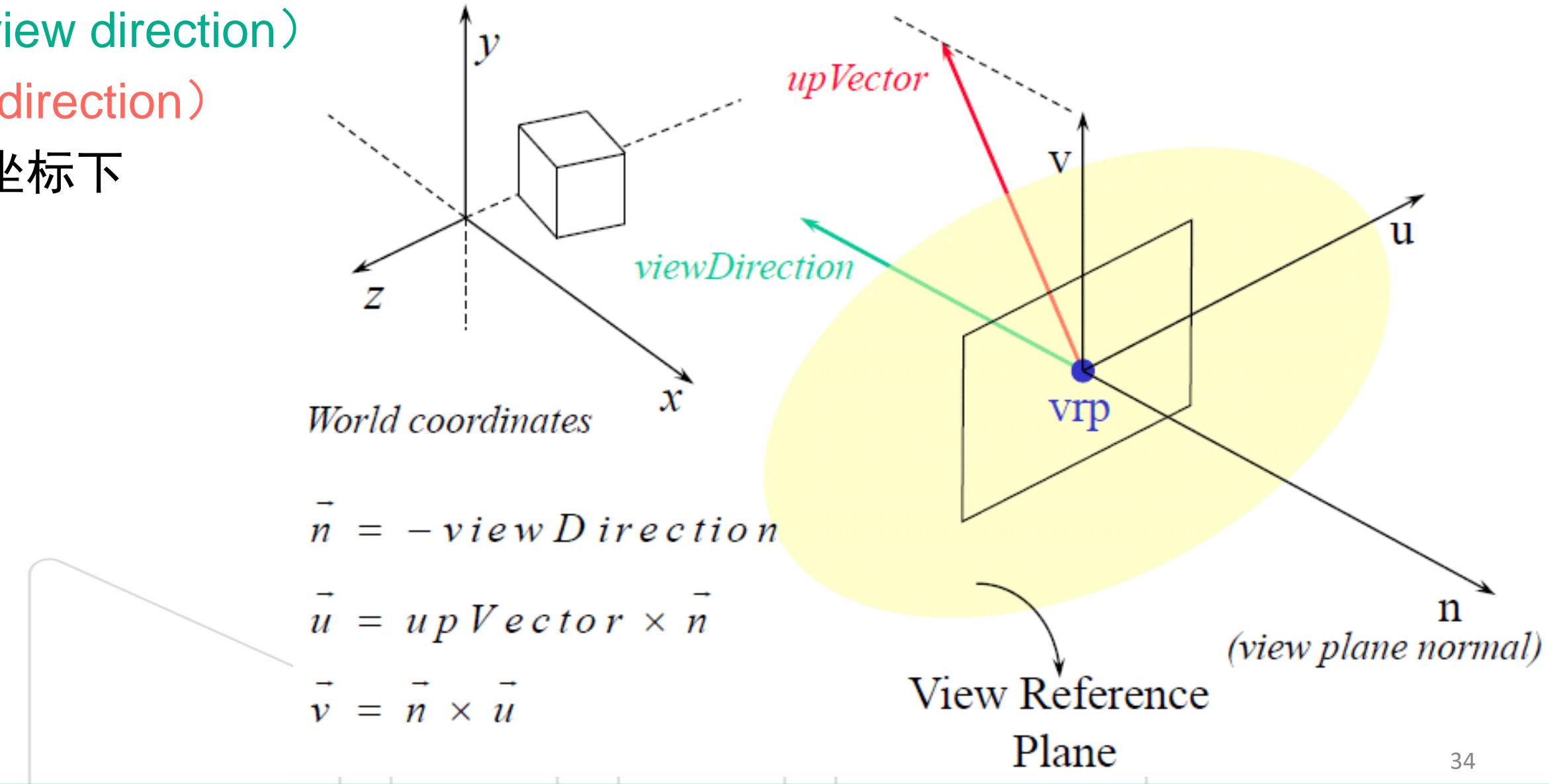


- 二维观察
- 三维观察
 - 经典观察
 - 计算机观察
 - 定位照相机
 - 投影



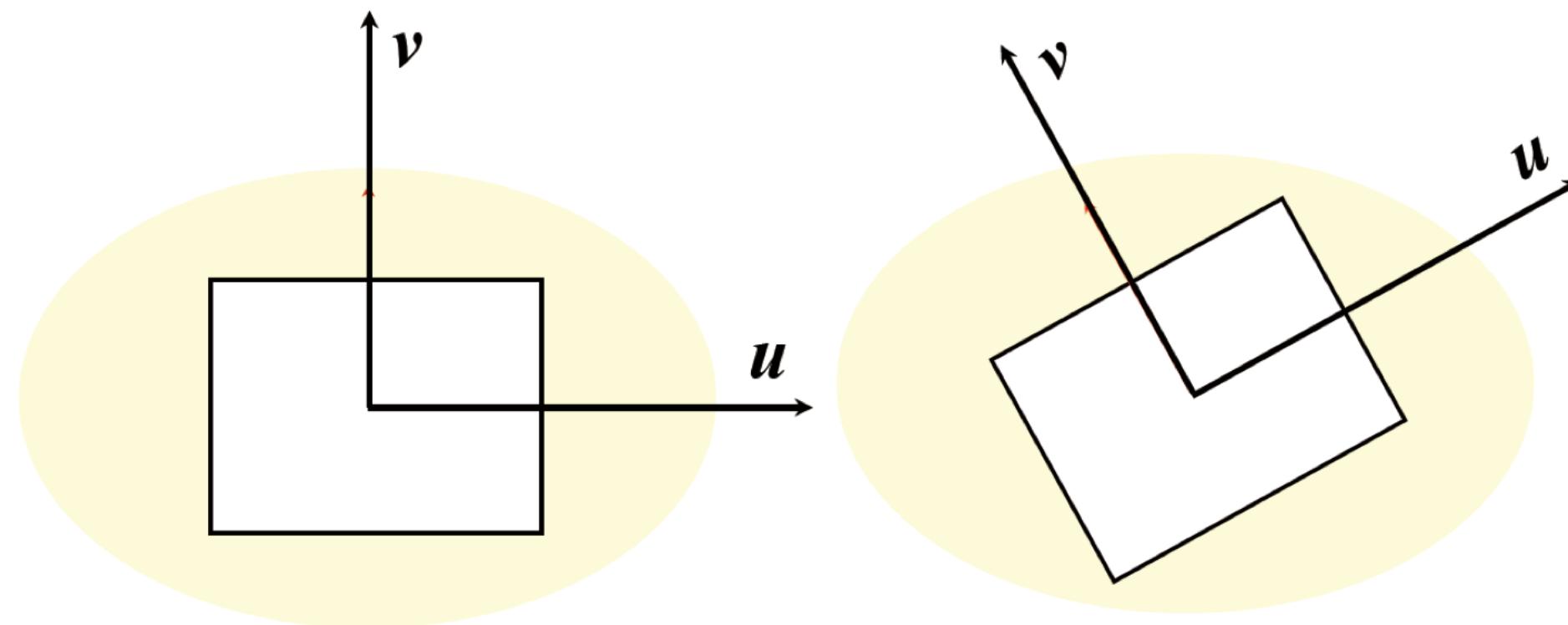
● 定位照相机的三要素

- 观察参考点 (view reference point, vrp)
- 观察方向 (view direction)
- 正方向 (up direction)
- 定义在世界坐标下



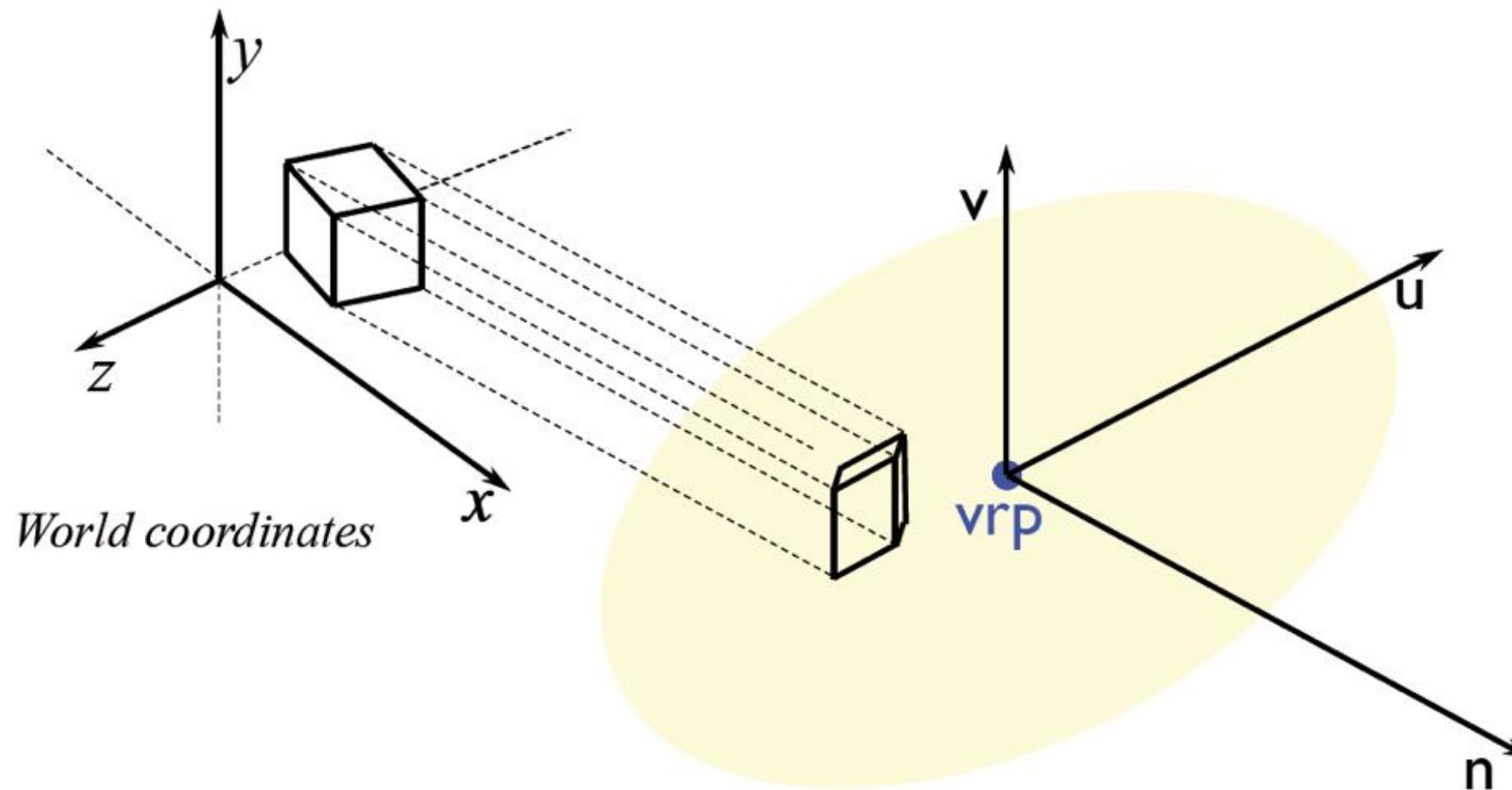
- 定位照相机的三要素

- 观察参考点 (view reference point, vrp)
- 观察方向 (view direction)
- 正方向 (up direction)
 - 正方向决定了视窗在观察参考平面上的角度



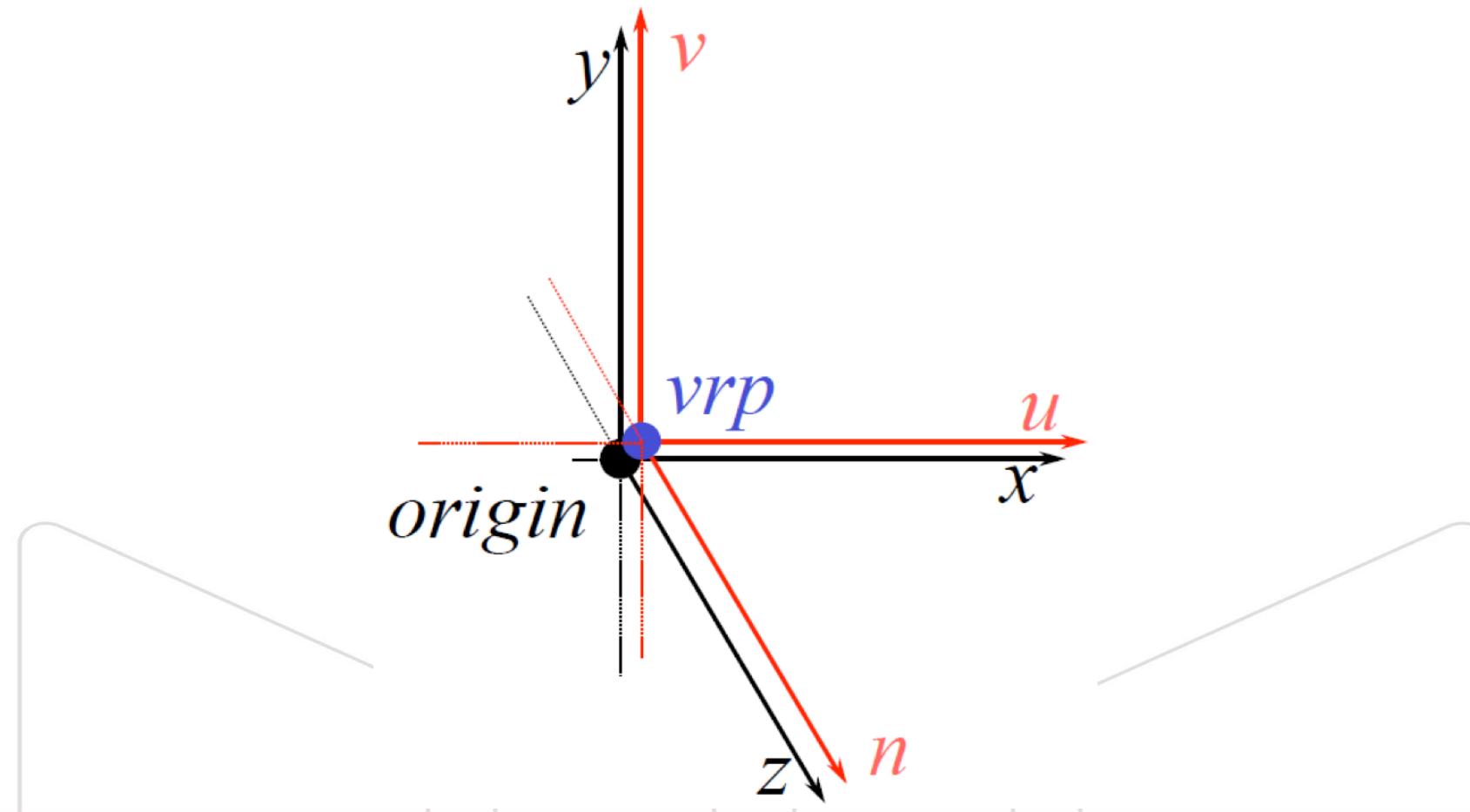
● 观察坐标系

- 观察坐标系一旦确定，就可以将三维世界的物体投影至观察参考平面



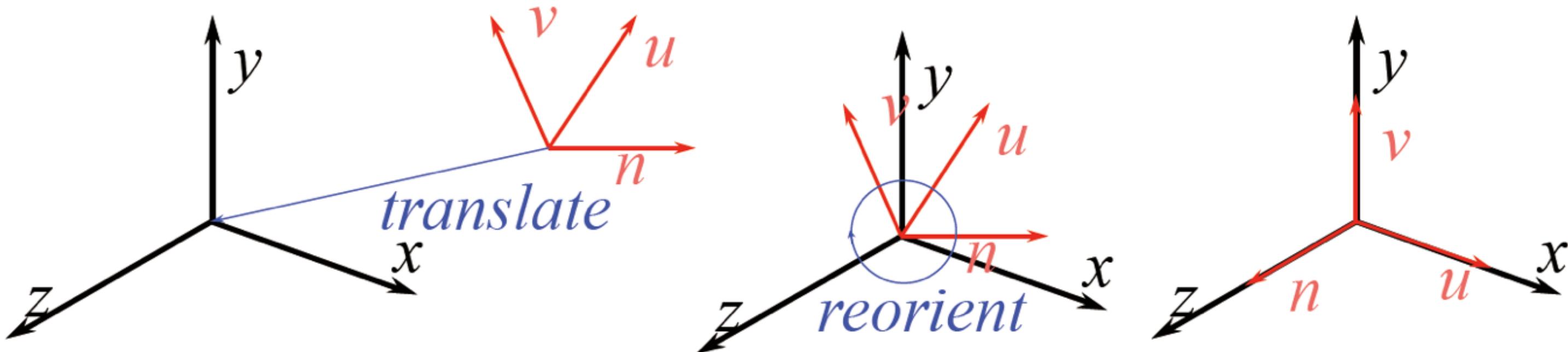
● 观察坐标系

- 初始观察坐标系与世界坐标系一致
 - $[u, v, n]$ 与 $[x, y, z]$ 重合
 - 正交投影过程可通过忽略三维点的z坐标完成



● 观察坐标系

- 对照相机定位后，需要确定物体在观察坐标系中的位置
- 世界坐标→观察坐标系的变换
 - 其变换矩阵为view orientation matrix
 - 通过变换使世界坐标系与观察坐标系重合
 - $V = RT$



● 观察坐标系

- 世界坐标→观察坐标的变换 $\mathbf{V} = \mathbf{RT}$
- $\mathbf{T} = \mathbf{T}(-\mathbf{P}_{vvp})$

- \mathbf{uvn} 坐标系中向量在原坐标系下的表示 $\mathbf{A} = \begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- $\mathbf{R} = \mathbf{A}^{-1} = \mathbf{A}^T = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{V} = \mathbf{RT} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{P}_{vvp} \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{P}_{vvp} \\ n_x & n_y & n_z & -\mathbf{n} \cdot \mathbf{P}_{vvp} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

● OpenGL中定位照相机

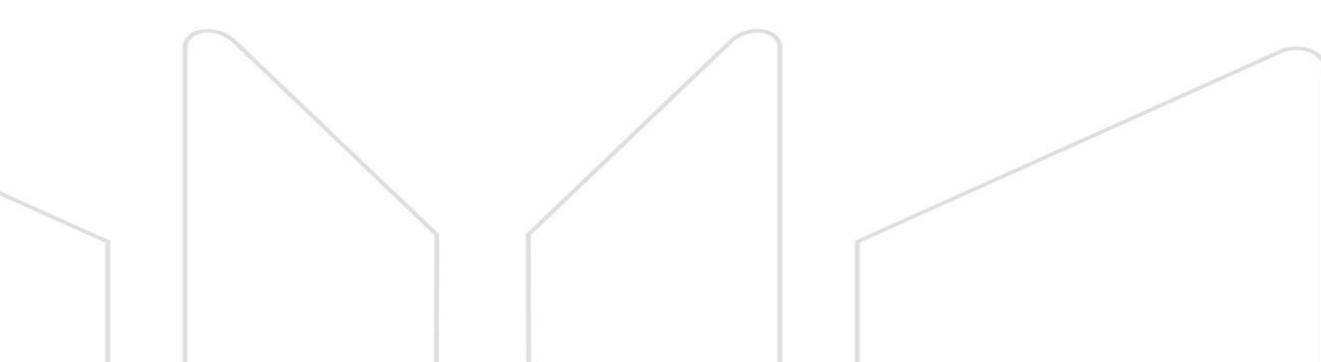
```
gluLookAt( eyex, eyey, eyez, // eye point  
           atx, aty, atz, //look-at point  
           upx, upy, upz); //up vector
```

- 同样，所有坐标均为世界坐标
- uvn坐标可由上述参数给出，从而决定view orientation matrix

$$\bullet \mathbf{n} = \frac{\mathbf{P}_{eye} - \mathbf{P}_{look}}{|\mathbf{P}_{eye} - \mathbf{P}_{look}|}$$

$$\bullet \mathbf{u} = \frac{\mathbf{up}}{|\mathbf{up}|}$$

$$\bullet \mathbf{v} = \frac{\mathbf{n} \times \mathbf{u}}{|\mathbf{n} \times \mathbf{u}|}$$



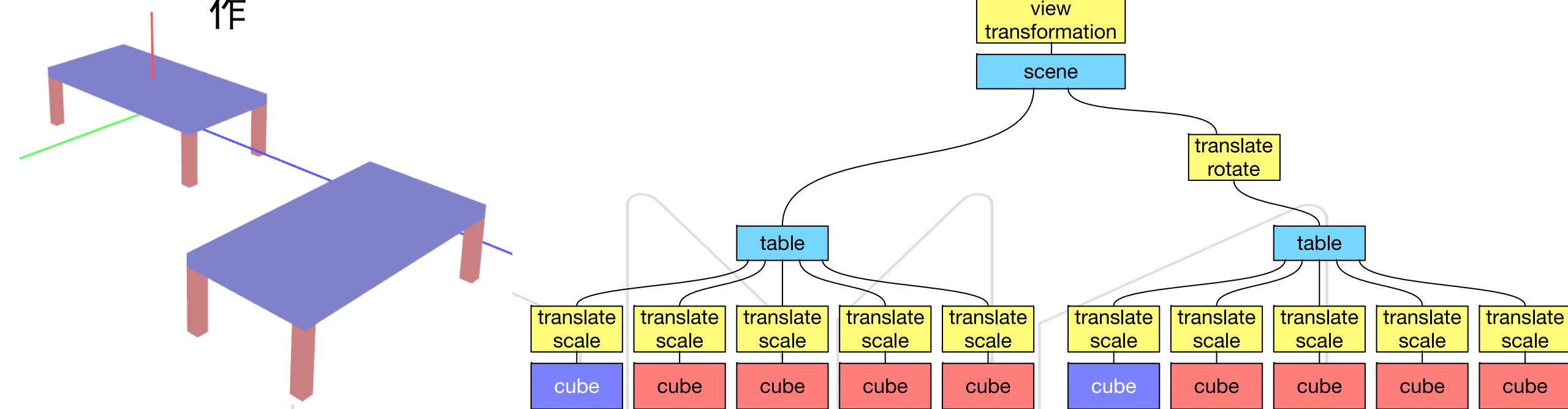
● OpenGL定位照相机举例

```
void display(void){  
    glClear(GL_COLOR_BUFFER_BIT);  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
  
    //Setting up the view  
    gluLookAt(0.0f, 0.0f, 5.0f, //eye is at (0, 0, 5)  
              0.0f, 0.0f, 0.0f, //center is at (0, 0, 0)  
              0.0f, 1.0f, 0.0f); //Up is in positive Y direction  
  
    //Now we are using the world frame  
    //Draw object  
    glColor3f(0.0f, 0.0f, 0.0f);  
    glutWireCube(1.0f);  
    glutSwapBuffer();  
}
```

● OpenGL中定位照相机

– 建模变换与观察变换合并为一个modelview变换

- 节省计算时间（尤其是当vertex数目庞大时）
- 观察变换紧接在建模变换后，中间没有其他操作
 - 在程序中，观察变换最先调用，应用于场景中所有物体上
 - 从实现上，**gluLookAt**的功能也可由其他建模变换完成
 - 但从概念上，我们将**gluLookAt**视为对照相机的设置，而其他建模变换为对物体的操作



- 二维观察
- 三维观察
 - 经典观察
 - 计算机观察
 - 定位照相机
 - 投影

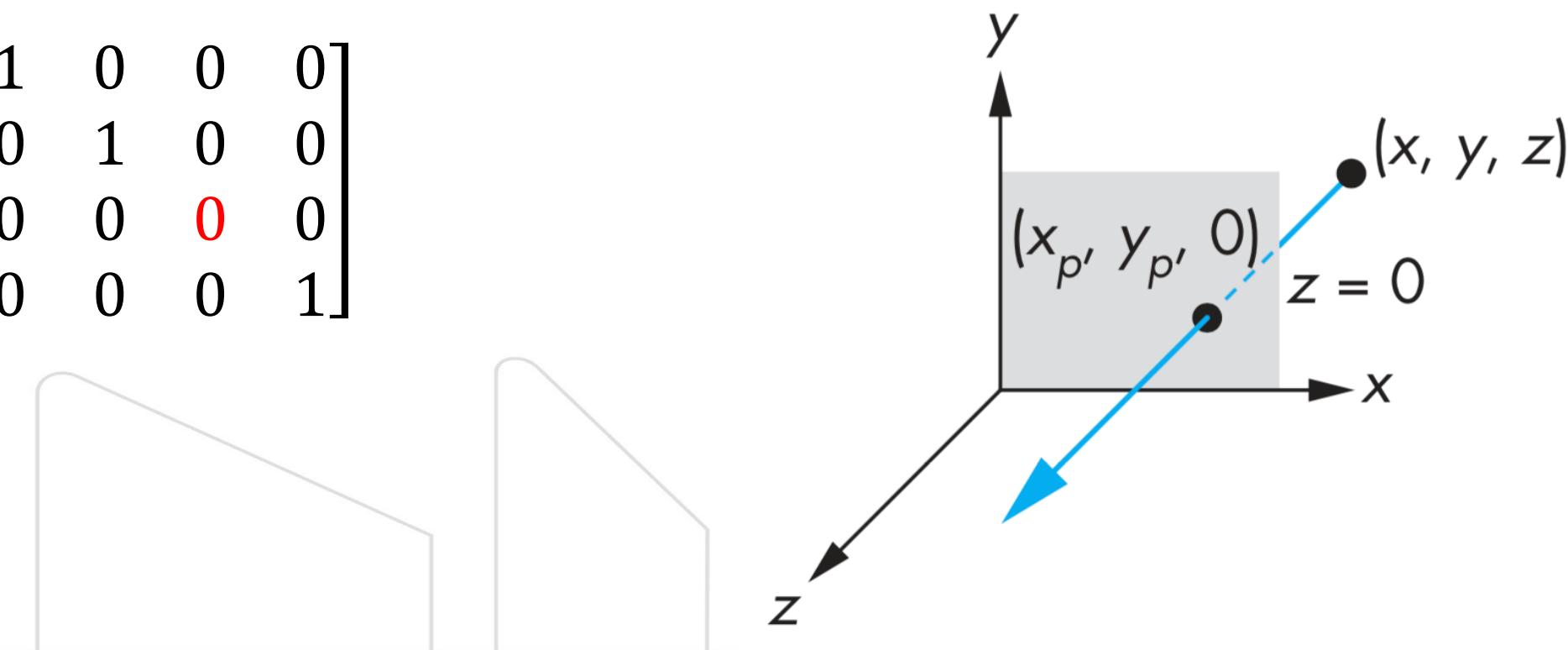


● 正投影 (orthographic projection)

– 正投影将场景中所有物体沿垂直方向投影于观察参考平面

- $P_p = \mathbf{MVP}$, 其中V观察变换, 在此我们只考虑观察变换后的点 (VP)
- 正投影将观察坐标下的点P的x坐标置零
- 即, 假设 $\mathbf{P} = [x, y, z, 1]^T$, 则投影的到的 $\mathbf{P}_p = [x, y, 0, 1]^T$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



● 投影规范化 (projection normalization)

- 将视见体变形为规范视见体 (canonical view volume)
 - $[-1, -1, -1]$ 与 $[1, 1, 1]$ 间边长为2的立方体
- OpenGL中平行投影由**glOrtho(...)**指定
 - **glOrtho(left, right, bottom, top, near, far)**
 - 默认投影变换为单位矩阵，等价于**glOrtho(-1, 1, -1, 1, -1, 1)**
 - 其规范化需要进行两次变换

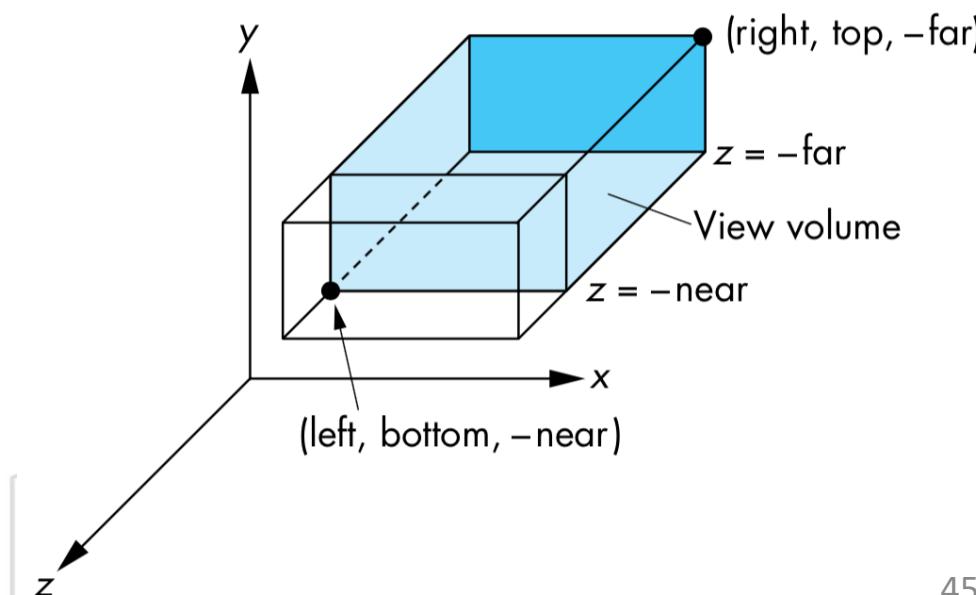
$$- \mathbf{T} = \mathbf{T}\left(-\frac{right+left}{2}, -\frac{top+bottom}{2}, -\frac{far+near}{2}\right)$$

$$- \mathbf{S} = \mathbf{S}\left(\frac{2}{right-left}, \frac{2}{top-bottom}, \frac{2}{near-far}\right)$$

- 注意，z轴上缩放为 $near - far$

» 将 $z = -near$ 映射为 $z = -1$

» 将 $z = -far$ 映射为 $z = 1$



● 投影规范化 (projection normalization)

– 将视见体变形为规范视见体 (canonical view volume)

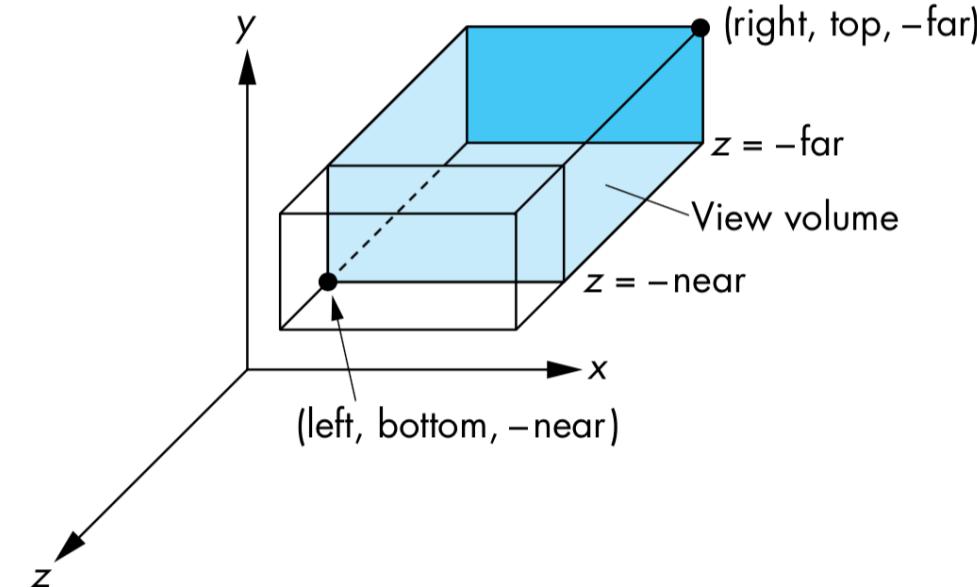
$$\bullet \mathbf{T} = \mathbf{T}\left(-\frac{right+left}{2}, -\frac{top+bottom}{2}, \frac{far+near}{2}\right)$$

$$\bullet \mathbf{S} = \mathbf{S}\left(\frac{2}{right-left}, \frac{2}{top-bottom}, \frac{2}{near-far}\right)$$

– 注意, z轴上缩放为 $near - far$

» 将 $z = -near$ 映射为 $z = -1$

» 将 $z = -far$ 映射为 $z = 1$



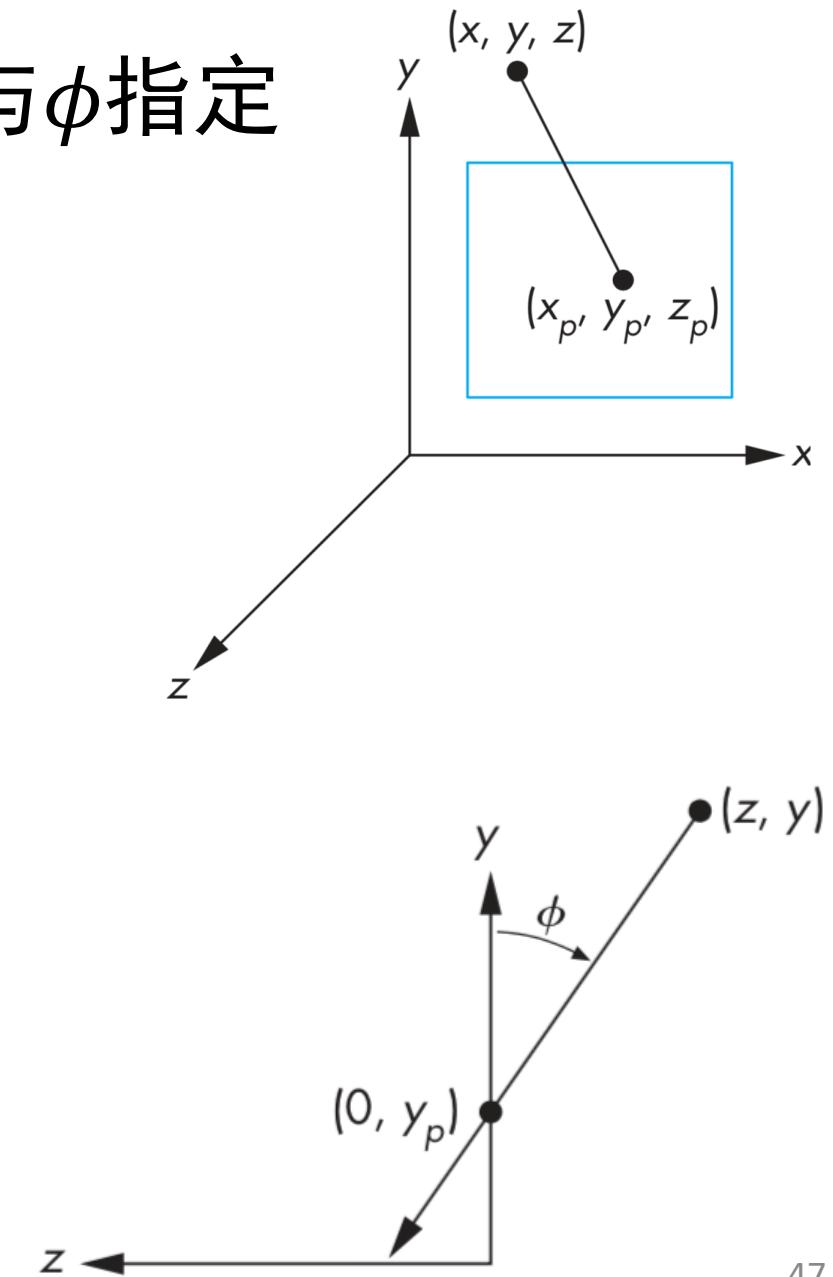
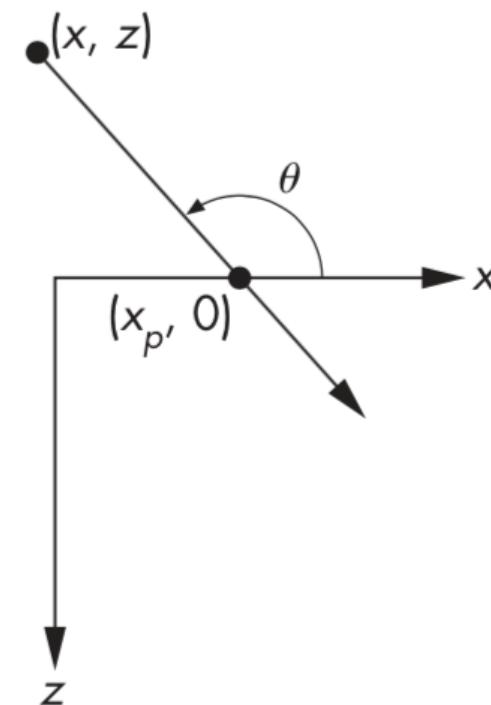
$$\bullet \mathbf{M} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 斜平行投影 (oblique projection)

– 斜平行投影可由其与投影平面的两个夹角 θ 与 ϕ 指定

- $x_p = x + z \cot \theta$
- $y_p = y + z \cot \phi$
- $z_p = 0$

- $P = \begin{bmatrix} 1 & 0 & \cot \theta & 0 \\ 0 & 1 & \cot \phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

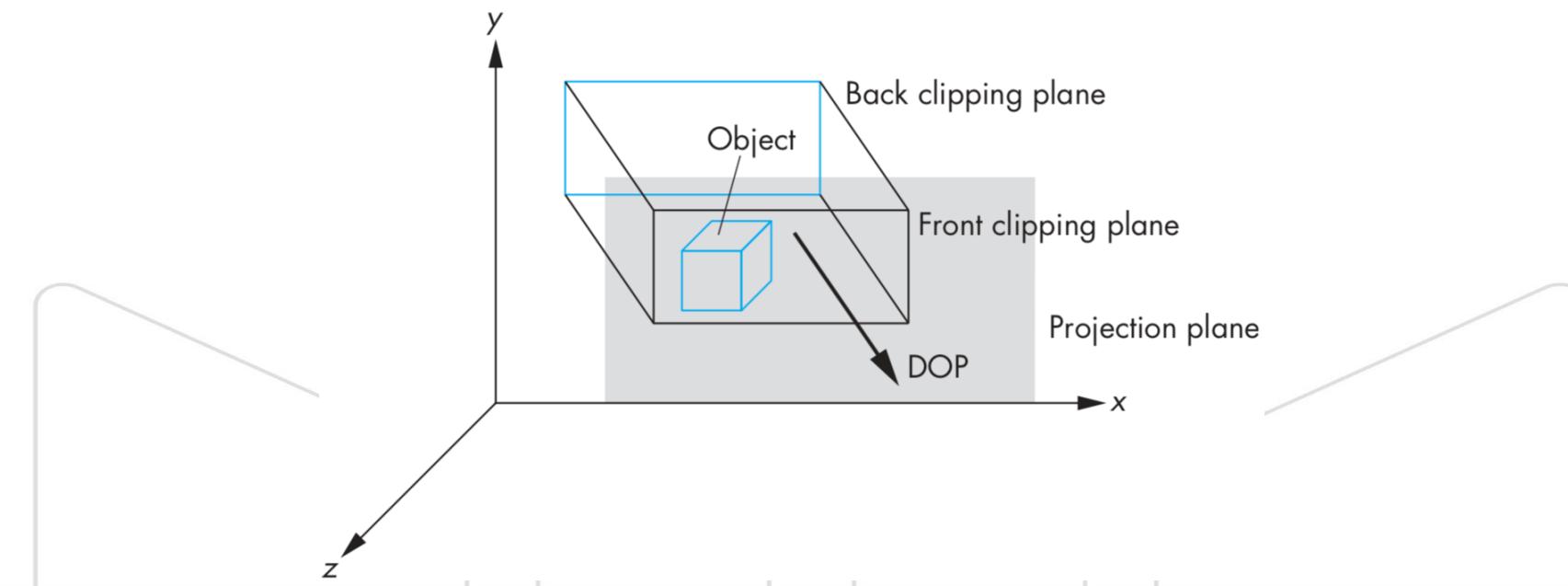


- 斜平行投影 (oblique projection)

– 斜平行投影可视为正投影与错切变换的合成

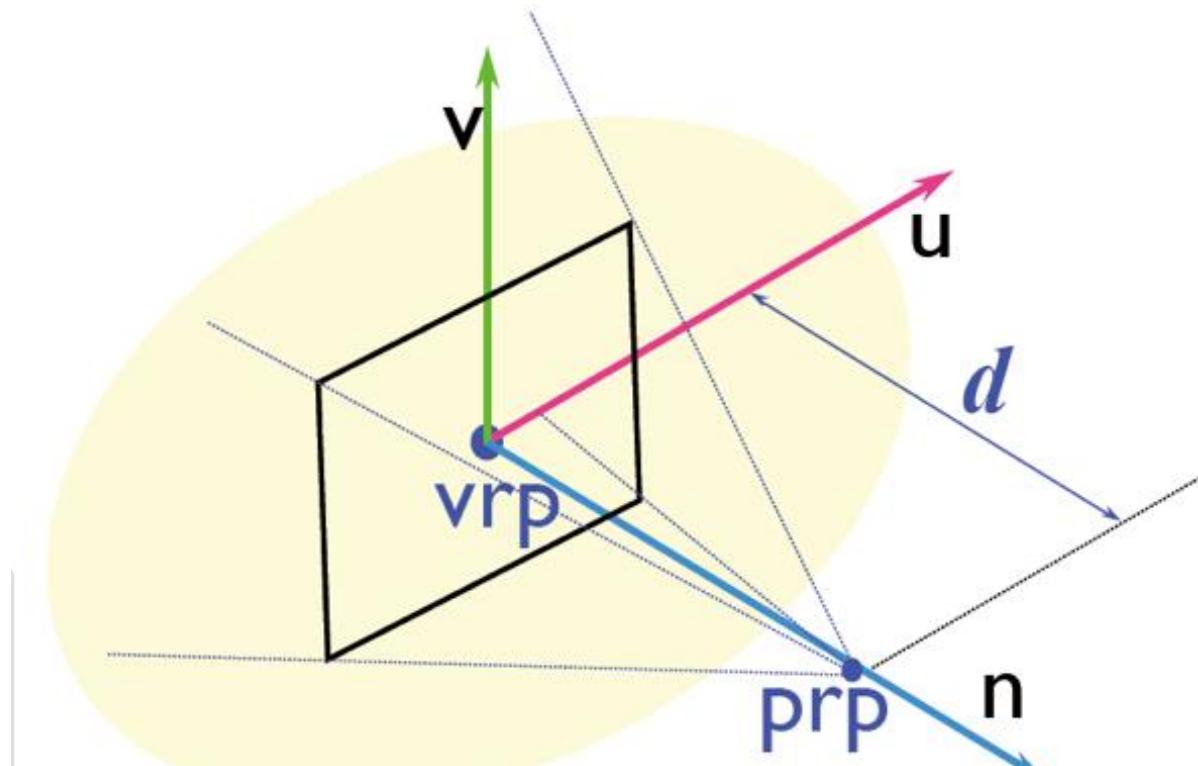
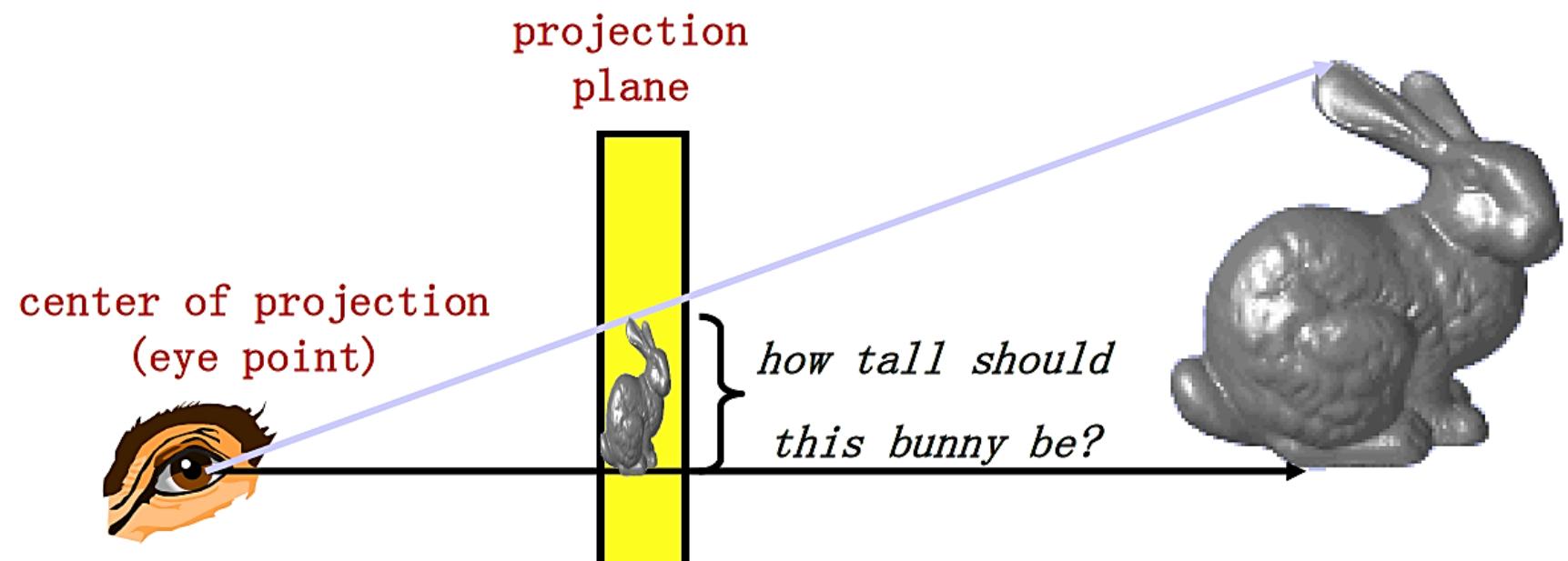
$$\bullet P = M_{orth} H(\theta, \phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \cot \theta & 0 \\ 0 & 1 & \cot \phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cot \theta & 0 \\ 0 & 1 & \cot \phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

• 正交坐标下的斜投影=“斜”坐标下的正投影



● 透视投影 (perspective projection)

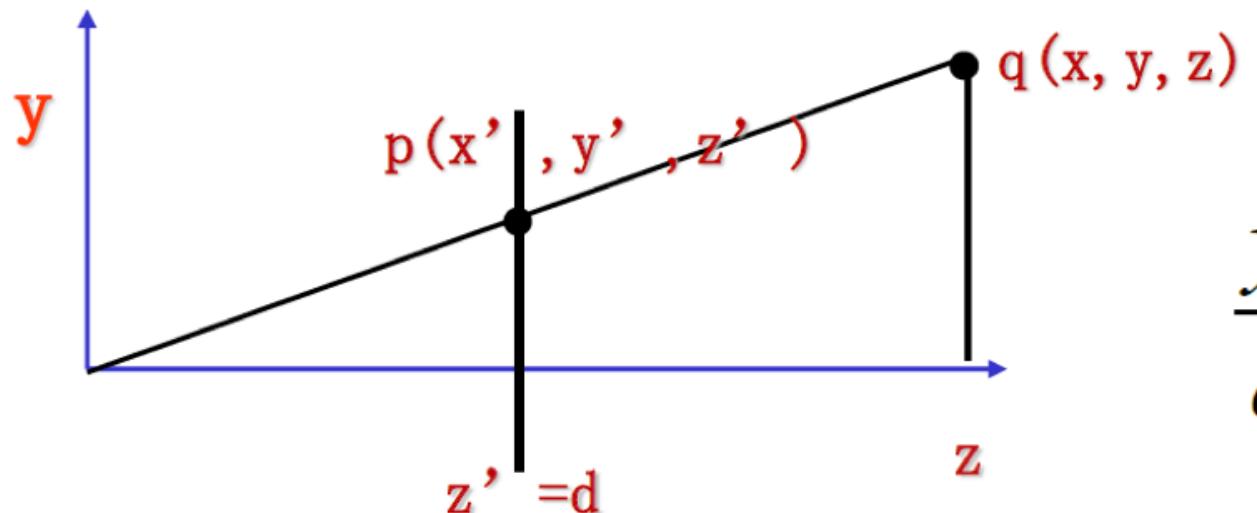
- 所有投影线汇聚于一点 (center of projection, COP)
 - 也称为投影参照点 (projection reference point, PRP)
 - COP即相机位置
- 近大远小



● 透视投影 (perspective projection)

- 将点 $q(x, y, z)$ 投影至 $z = d$ 平面上点 $p(x', y', z')$ 的计算过程
 - 如何表示为矩阵? (矩阵只能表示 x, y, z 的线性组合)

similar triangles



$$\frac{y'}{d} = \frac{y}{z} \rightarrow y' = \frac{y \cdot d}{z}$$

$$\frac{x'}{d} = \frac{x}{z} \rightarrow x' = \frac{x \cdot d}{z}$$

but

$$z' = d$$

- 透视投影 (perspective projection)

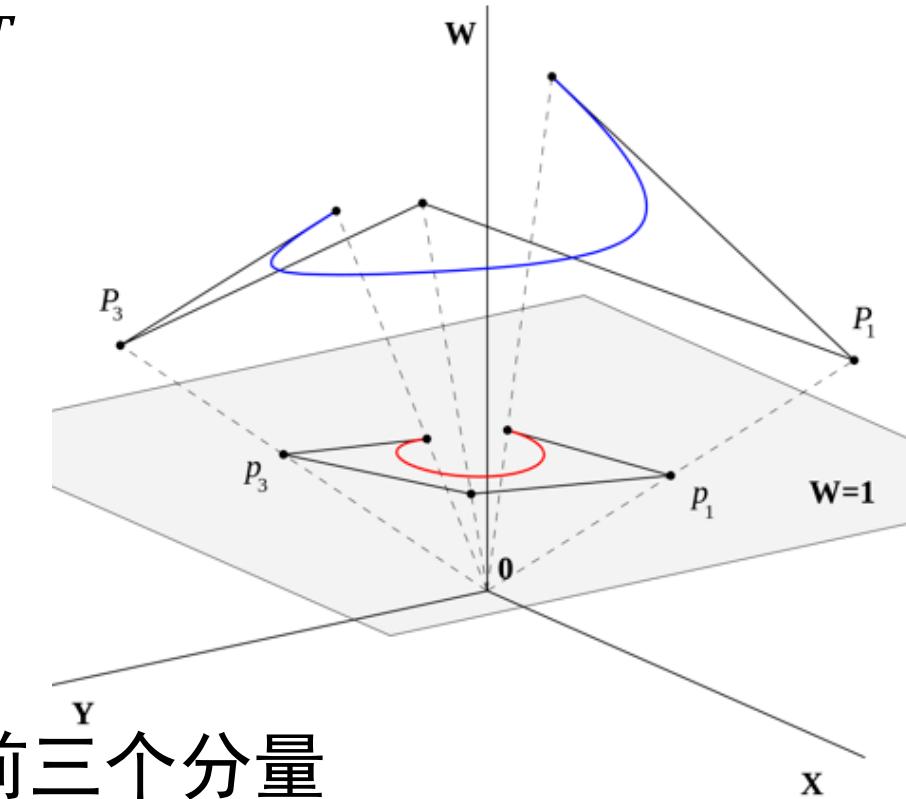
- 将齐次坐标用于投影变换

- 此前，我们将三维空间中的点 (x, y, z) 表示为四维坐标 $(x, y, z, 1)$
- 现在推广至更为一般的四维点 $\mathbf{P} = [wx, wy, wz, w]^T$
 - 从四维点恢复至三维点时，需保持 $w = 1$

- $\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

- \mathbf{M} 将点 $\mathbf{P} = [x, y, z, 1]^T$ 变换为点 $\mathbf{P}' = [x, y, z, z/d]^T$
- 当 \mathbf{P}' 回到三维空间时，需要使用第四个分量去除前三个分量

- $\mathbf{P}' = \left[\frac{x}{z/d}, \frac{y}{z/d}, \frac{z}{z/d}, 1 \right]^T = \left[\frac{x \cdot d}{z}, \frac{y \cdot d}{z}, d, 1 \right]^T$



● 透视投影 (perspective projection)

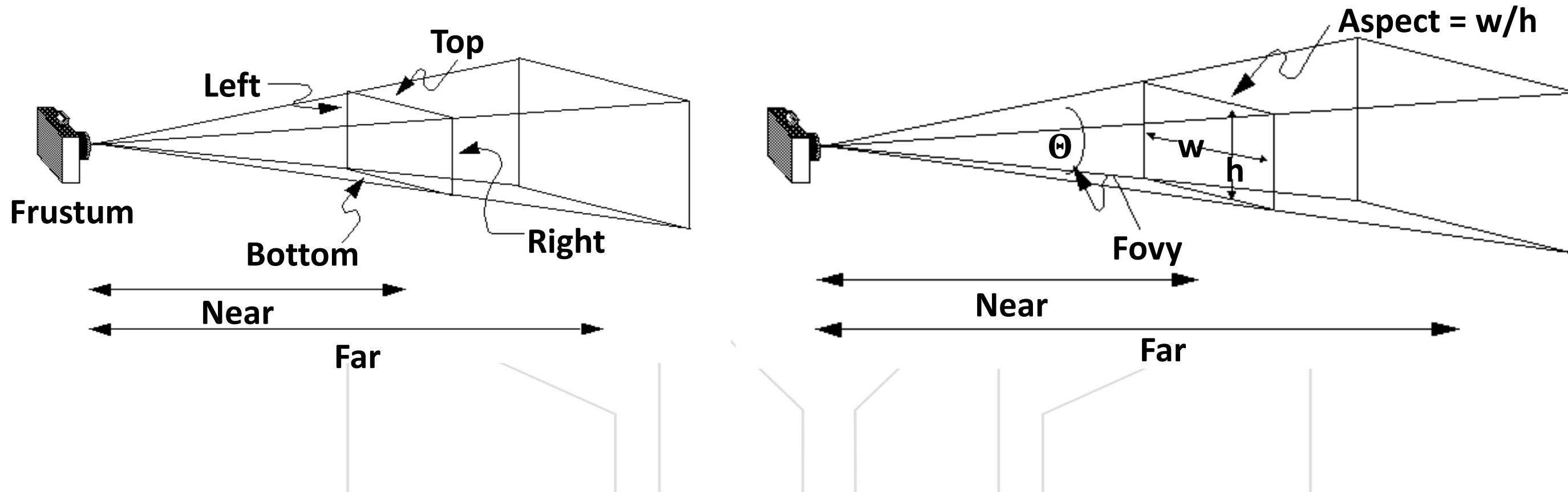
- 三维空间中的每一个点 $\mathbf{P} = [x, y, z]^T$ 都有对应的一族齐次坐标 $\mathbf{P} = [wx, wy, wz, w]^T$, 即四维空间中的一条线
- 当 $w \neq 1$ 时, 必须从齐次坐标中除以 w 得到所表示的点, 称为**透视除法**
 - 透视除法为非线性的, 导致非均匀缩短
 - 离投影中心 (COP) 越远, 尺寸缩短越大
- 透视变换是**保直线**的, 但**不是仿射变换**
- 透视变换是**不可逆**的, 因为沿投影线上所有点投影后的结果相同
 - 无法还原至投影前位置
 - 平行投影同样是不可逆的



● 透视投影 (perspective projection)

– OpenGL中的透视投影使用以下两个函数完成（参考课件4）

- **glFrustum(left, right, bottom, top, near, far)**
- **gluPerspective(fovy, aspect, near, far)**



Questions?

