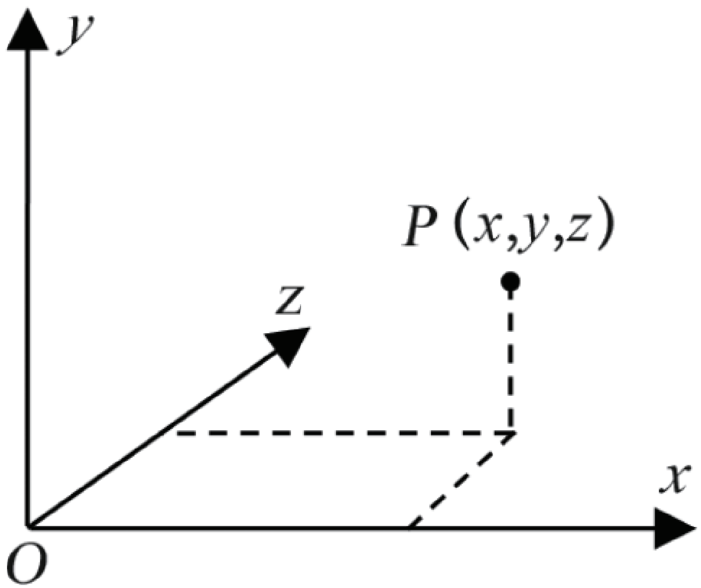
XIAMEN UNIVERSITY

# COMPUTER GRAPHICS

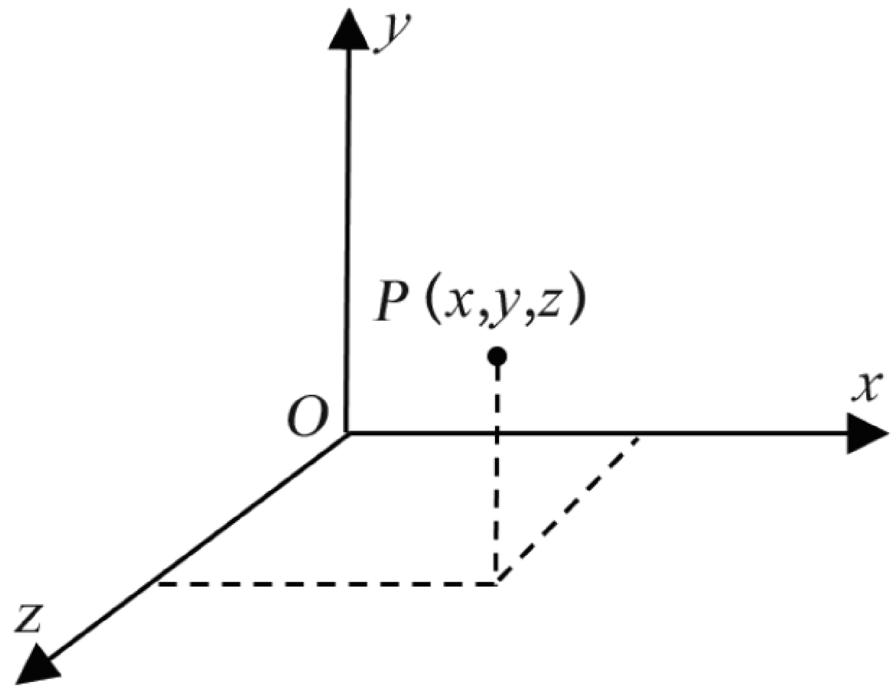
## 第三章 数学基础

陈中贵
厦门大学信息学院
http://graphics.xmu.edu.cn

# 3D 坐标系




左手坐标系




右手坐标系

# 点、矩阵

- 3D点和向量
  - 齐次坐标
  - GLSL和GLM库中的数据类型vec4

$$[\,x\ y\ z\ w\,] = [\,x/w\ \ y/w\ \ z/w\ \ 1\,]$$

- 3D图形中用到的矩阵通常是4x4:

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{bmatrix}$$

# 矩阵乘法

- 点和矩阵相乘

$$\begin{pmatrix} AX+BY+CZ+D \\ EX+FY+GZ+H \\ IX+JY+KZ+L \\ MX+NY+OZ+P \end{pmatrix} = \begin{bmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ M & N & O & P \end{bmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- 矩阵和矩阵相乘

$$\begin{bmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ M & N & O & P \end{bmatrix} * \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

$$= \begin{bmatrix} Aa+Be+Ci+Dm & Ab+Bf+Cj+Dn & Ac+Bg+Ck+Do & Ad+Bh+Cl+Dp \\ Ea+Fe+Gi+Hm & Eb+Ff+Gj+Hn & Ec+Fg+Gk+Ho & Ed+Fh+Gl+Hp \\ Ia+Je+Ki+Lm & Ib+Jf+Kj+Ln & Ic+Jg+Kk+Lo & Id+Jh+Kl+Lp \\ Ma+Ne+Oi+Pm & Mb+Nf+Oj+Pn & Mc+Ng+Ok+Po & Md+Nh+Ol+Pp \end{bmatrix}$$

# 矩阵乘法的结合律

- 考虑如下运算序列：

$$\text{NewPoint} = \mathbf{Matrix}_1 \times [\mathbf{Matrix}_2 \times (\mathbf{Matrix}_3 \times \text{Point})]$$

$$= (\mathbf{Matrix}_1 \times \mathbf{Matrix}_2 \times \mathbf{Matrix}_3) \times \text{Point}$$

- 将前三个矩阵合并：

$$\mathbf{Matrix}_C = \mathbf{Matrix}_1 \times \mathbf{Matrix}_2 \times \mathbf{Matrix}_3$$

$$\text{NewPoint} = \mathbf{Matrix}_C \times \text{Point}$$

- 矩阵求逆： mat4.inverse()

# 矩阵变换

- 平移矩阵
  - glm::translate(x,y,z)
  - mat4*vec4

$$\begin{pmatrix} X+T_x \\ Y+T_y \\ Z+T_z \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- 缩放矩阵
  - glm::scale(x, y, z)
  - mat4*vec4

$$\begin{pmatrix} X\ S_x \\ Y\ S_y \\ Z\ S_z \\ 1 \end{pmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- 旋转矩阵
  - glm::rotate(mat4, θ, x, y, z)，用于构建绕轴(x, y, z)旋转 θ 度的矩阵
  - mat4*vec4

# 旋转矩阵

- **欧拉定理：** 围绕任何轴的旋转都可以表示为绕 *x* 轴、 *y* 轴、 *z* 轴旋转的组合。
  - 围绕这 3 个轴的旋转角度被称为欧拉角

当在 3D 空间中旋转轴不穿过原点时：
（1）平移旋转轴以使它经过原点；
（2）绕 *x* 轴、 *y* 轴、 *z* 轴旋转适当的欧拉角；
（3）复原步骤（1）中的平移。

绕$x$轴旋转$\theta$度：

$$\begin{pmatrix} X \\ Y' \\ Z' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

绕$y$轴旋转$\theta$度：

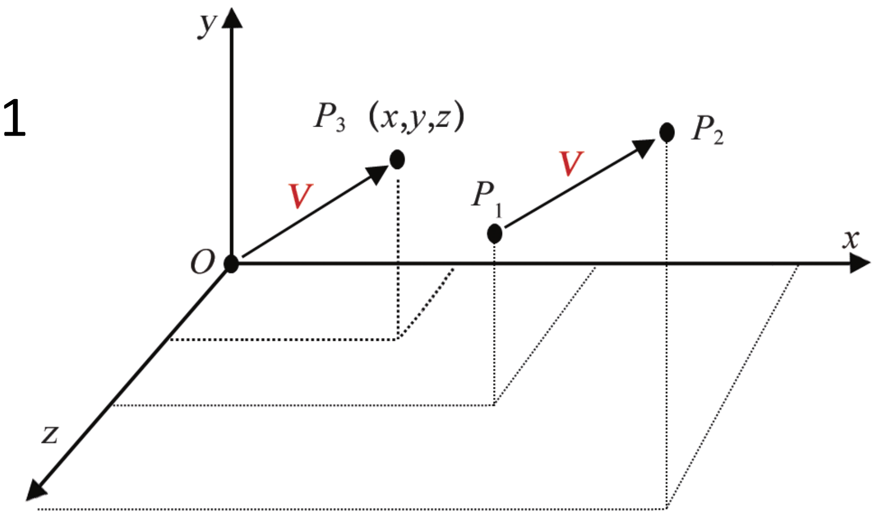
$$\begin{pmatrix} X' \\ Y \\ Z' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

绕$z$轴旋转$\theta$度：

$$\begin{pmatrix} X' \\ Y' \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

# 向量

- 为了方便起见，向量用空间中的单个点表示，P3 = P2-P1
- vec4 类型既存储点也存储向量
- 加法和减法




$\boldsymbol{A} \pm \boldsymbol{B} = (u \pm x,\ v \pm y,\ w \pm z)$

GLM: vec3±vec3

GLSL: vec3±vec3

归一化（将长度变为 1）

$\hat{\boldsymbol{A}} = \boldsymbol{A}/|\boldsymbol{A}| = \boldsymbol{A}/\sqrt{u^2+v^2+w^2}$，其中$|\boldsymbol{A}|$为向量 $\boldsymbol{A}$ 的长度

GLM: normalize(vec3) 或 normalize(vec4)

GLSL: normalize(vec3) 或 normalize(vec4)

点积

$\boldsymbol{A} \cdot \boldsymbol{B} = ux + vy + wz$

GLM: dot(vec3,vec3) 或 dot(vec4,vec4)

GLSL: dot(vec3,vec3) 或 dot(vec4,vec4)

叉积

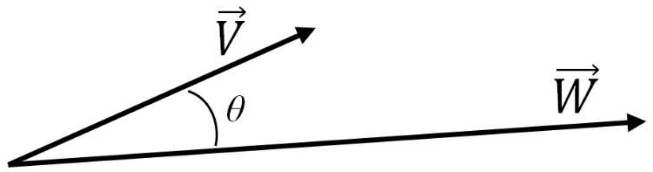
$\boldsymbol{A} \times \boldsymbol{B} = (vz - wy,\ wx - uz,\ uy - vx)$

GLM: cross(vec3,vec3)

GLSL: cross(vec3,vec3)

# 点积

- 设V=(a,b,c) , W=(d,e,f)
  V·W = ad + be + cf



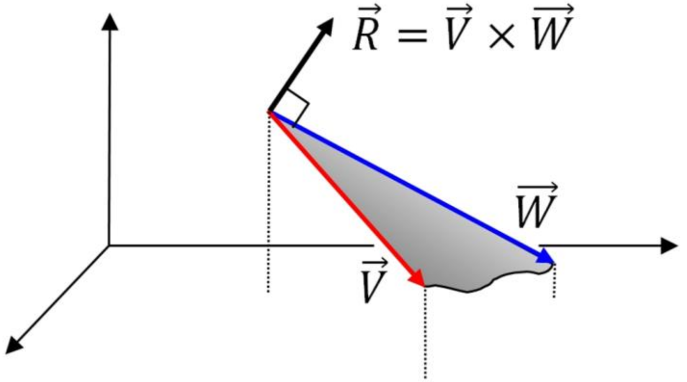

- 其夹角 θ $\boldsymbol{V}\cdot\boldsymbol{W} = |\boldsymbol{V}||\boldsymbol{W}|\cos(\theta)$

$$\cos(\theta) = \frac{\boldsymbol{V}\cdot\boldsymbol{W}}{|\boldsymbol{V}||\boldsymbol{W}|}$$

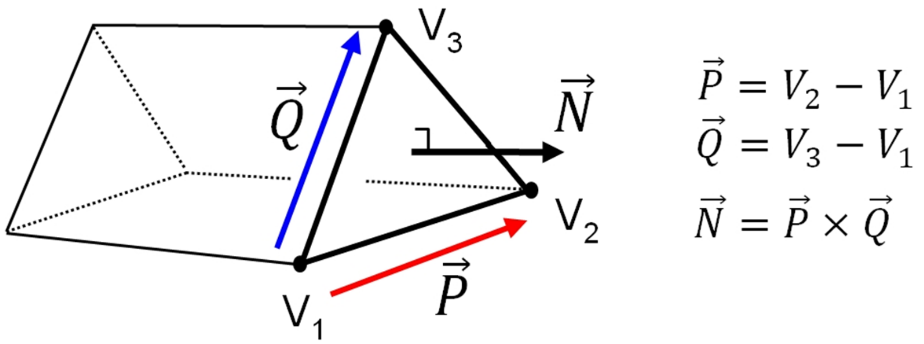
- 求点 $P=(x, y, z)$到平面 $S=(a, b, c, d)$的最小有符号距离。由垂直于 $S$ 的单位法向量 $\hat{\boldsymbol{n}} = \left(\frac{a}{\sqrt{a^2+b^2+c^2}}, \frac{b}{\sqrt{a^2+b^2+c^2}}, \frac{c}{\sqrt{a^2+b^2+c^2}}\right)$ 和从原点到平面的最短距离 $D = \frac{d}{\sqrt{a^2+b^2+c^2}}$，有 $P$ 到 $S$ 的最小有符号距离为 $(\hat{\boldsymbol{n}}\cdot P)+D$，符号由 $P$ 与 $S$ 的相对位置决定。

# 叉乘

- 设V=(a,b,c) , W=(d,e,f) ,    R = V × W = (bf-ce, cd-af, ae-bd)
- 遵循右手定则， 即将右手手指从 V 向 W 卷曲会使得大拇指指向法向量 R 的方向

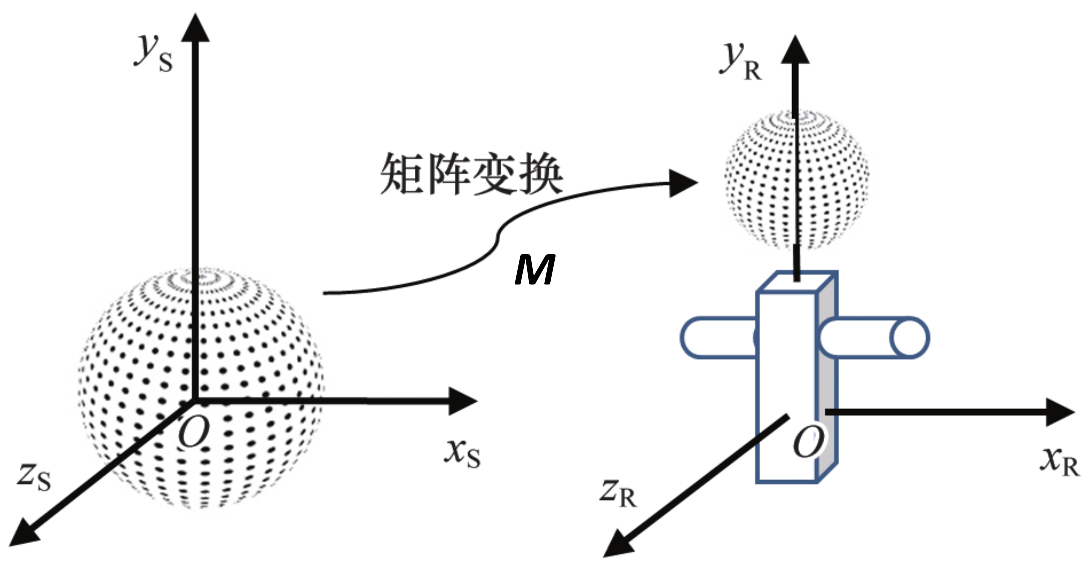



- 使用叉积计算来获得面的外向法向量



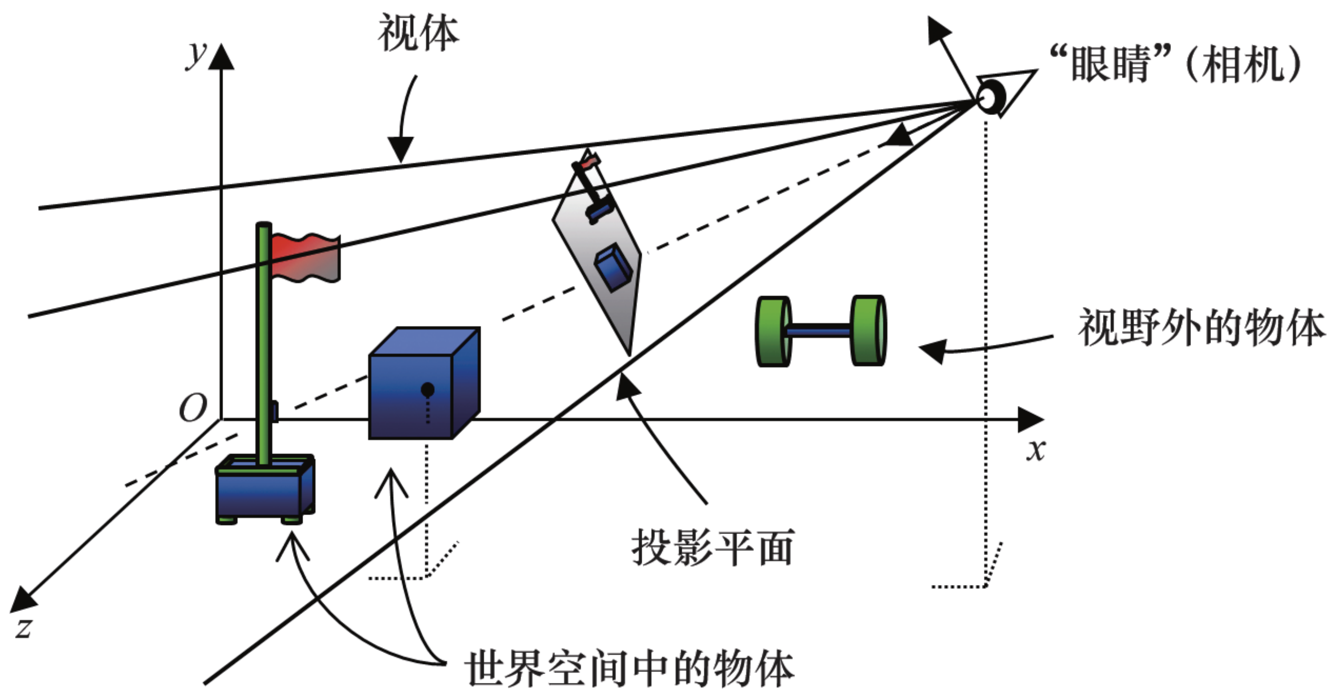
# 局部空间和世界空间

- 模型定义的空间叫作局部空间（local space）或模型空间（model space）或物体空间（object space）
- 通过设定物体在模拟世界中的朝向和大小，可以将物体放在模拟世界的空间中，这个空间叫作世界空间。在世界空间中为对象定位及定向的矩阵称为**模型矩阵**，通常记为 ***M***



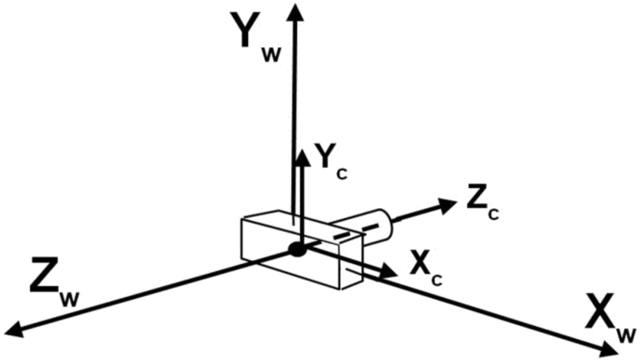
# 视觉（相机）空间和合成相机

- 观察 3D 世界需要：
  （a）将相机放入世界的某个位置；
  （b）调整相机的角度，通常需要一套它自己的直角坐标轴 u、 v、 n（由向量 U， V， N 构成）；
  （c）定义一个视体（view volume）；
  （d）将视体内的对象投影到投影平面（projection plane）上

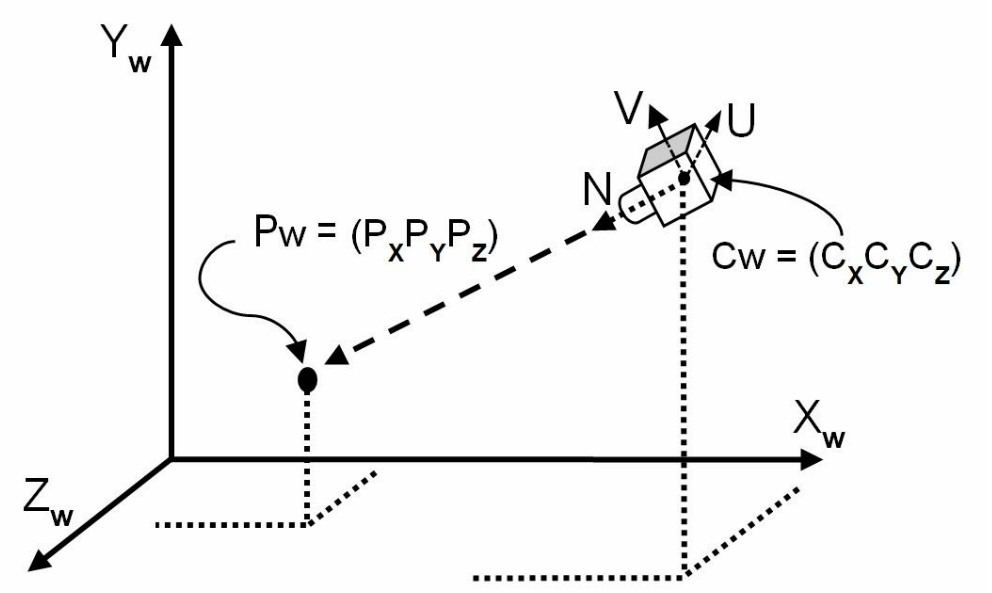


# OpenGL相机

- OpenGL 相机位置永远固定在点(0,0,0)并朝向 z 轴负方向的相机

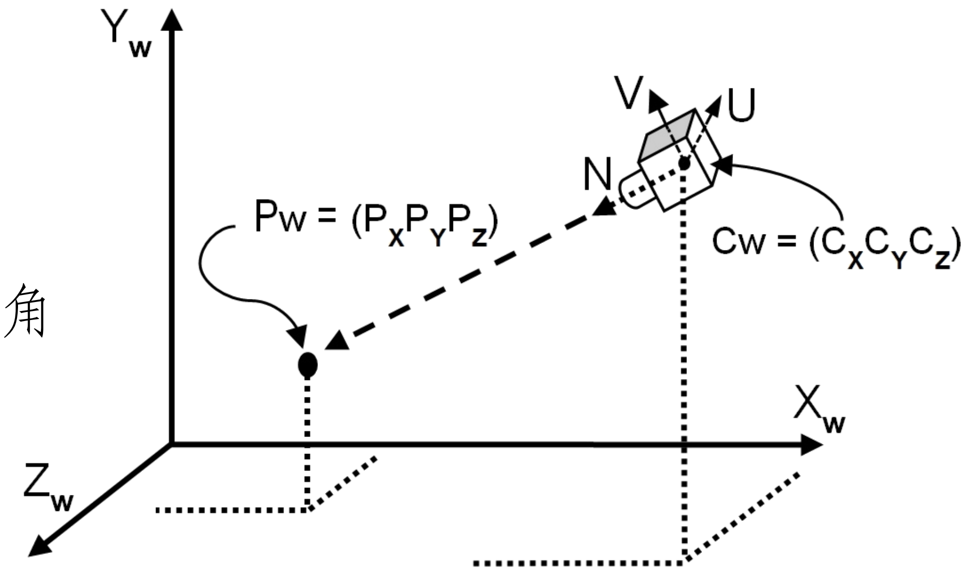



- 给定世界空间中的点 $P_W$，如何让它看起来好像是从我们期望的相机位置 $C_W$ 看到的样子?



# 构建视图变换矩阵

- 需要做的变换如下：
  （1）将 $P_W$ 平移，
  其向量为负的期望相机位置
  （2）将 $P_W$ 旋转，
  其角度为负的期望相机欧拉角




- 构建视图变换矩阵：

$$
\underset{\substack{\text{视觉空间}\\ \text{中的点}P_C}}{\begin{pmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{pmatrix}} = \underbrace{\overset{\text{负相机欧拉角}}{\underset{\boldsymbol{R}\text{（旋转）}}{\begin{bmatrix} \hat{U}_x & \hat{U}_y & \hat{U}_z & 0 \\ \hat{V}_x & \hat{V}_y & \hat{V}_z & 0 \\ -\hat{N}_x & -\hat{N}_y & -\hat{N}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}} \overset{\text{负相机位置}}{\underset{\boldsymbol{T}\text{（平移）}}{\begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}}}_{\boldsymbol{V}\text{（视图变换）}} \underset{\substack{\text{世界空间}\\ \text{中的点}P_W}}{\begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}}
$$

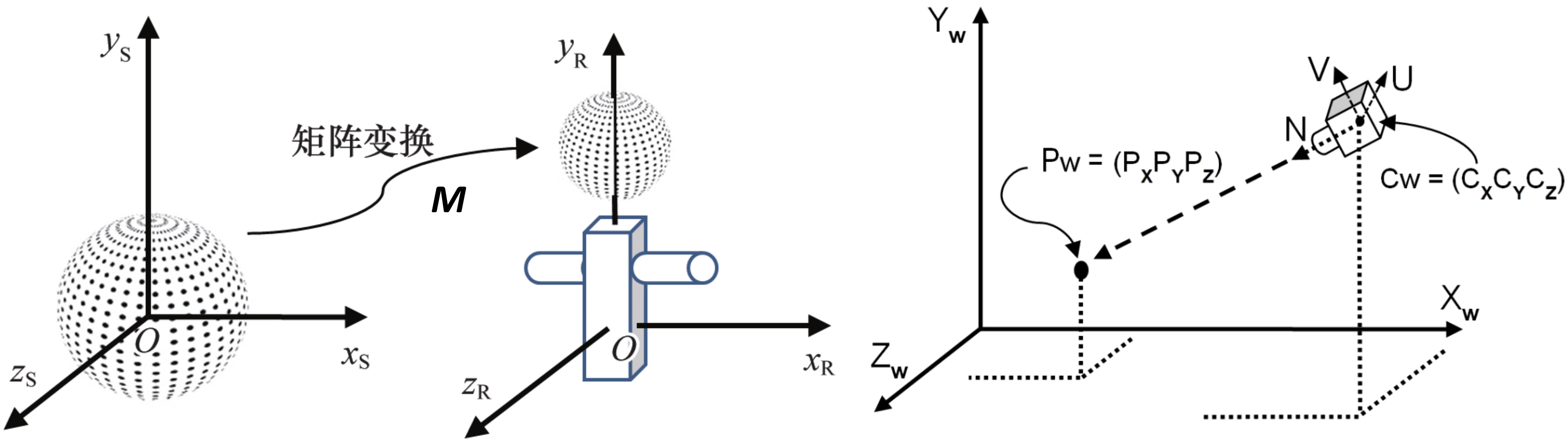
# 模型-视图矩阵

- 将视图 ***V*** 矩阵与模型矩阵 ***M*** 的积定义为模型-视图（Model-View，MV）矩阵，记作 **MV**：

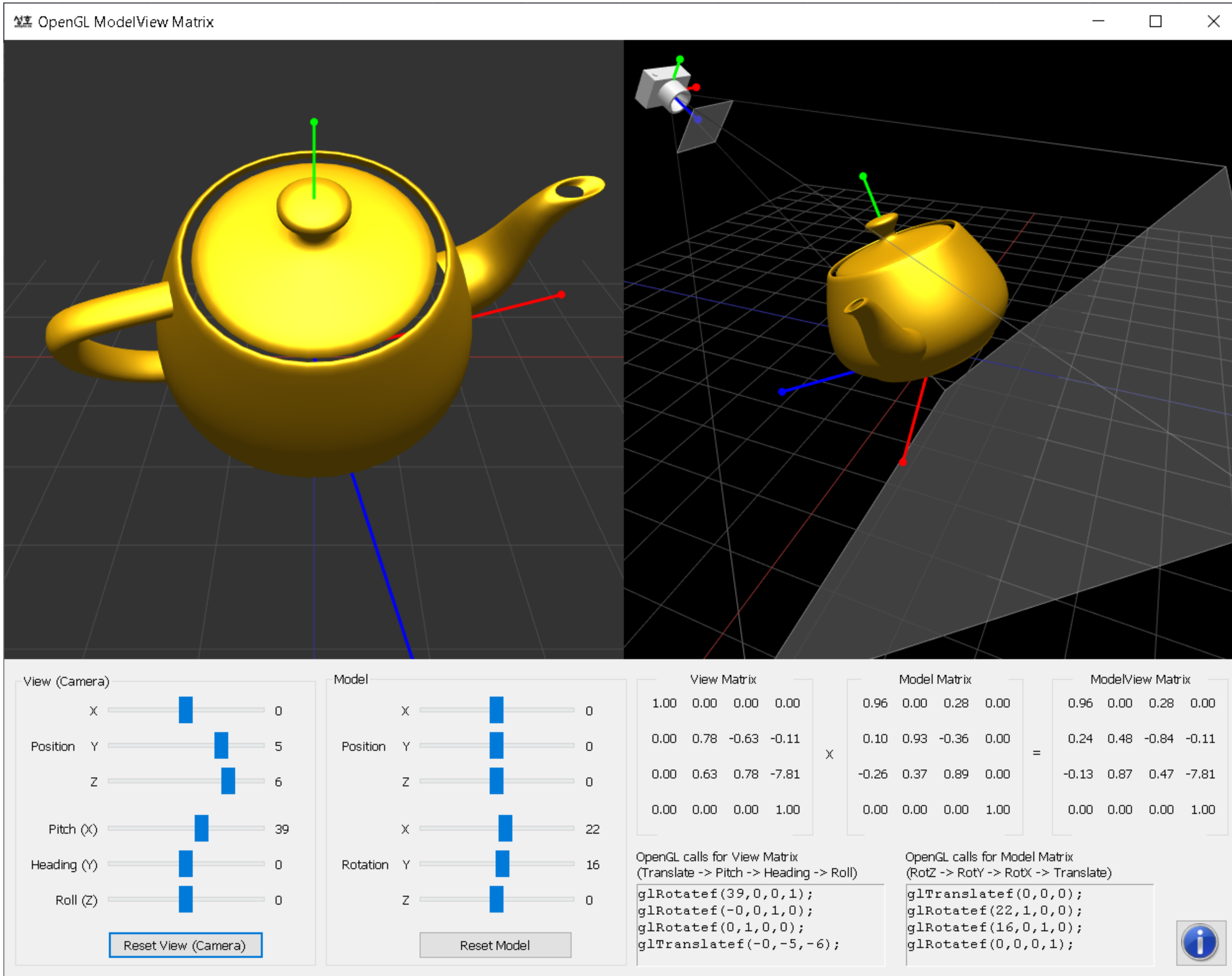
$$\mathbf{MV} = \boldsymbol{VM}$$

- 点 $P_M$ 从自己的模型空间直接转换至相机空间：

$$P_C = \mathbf{MV}P_M$$

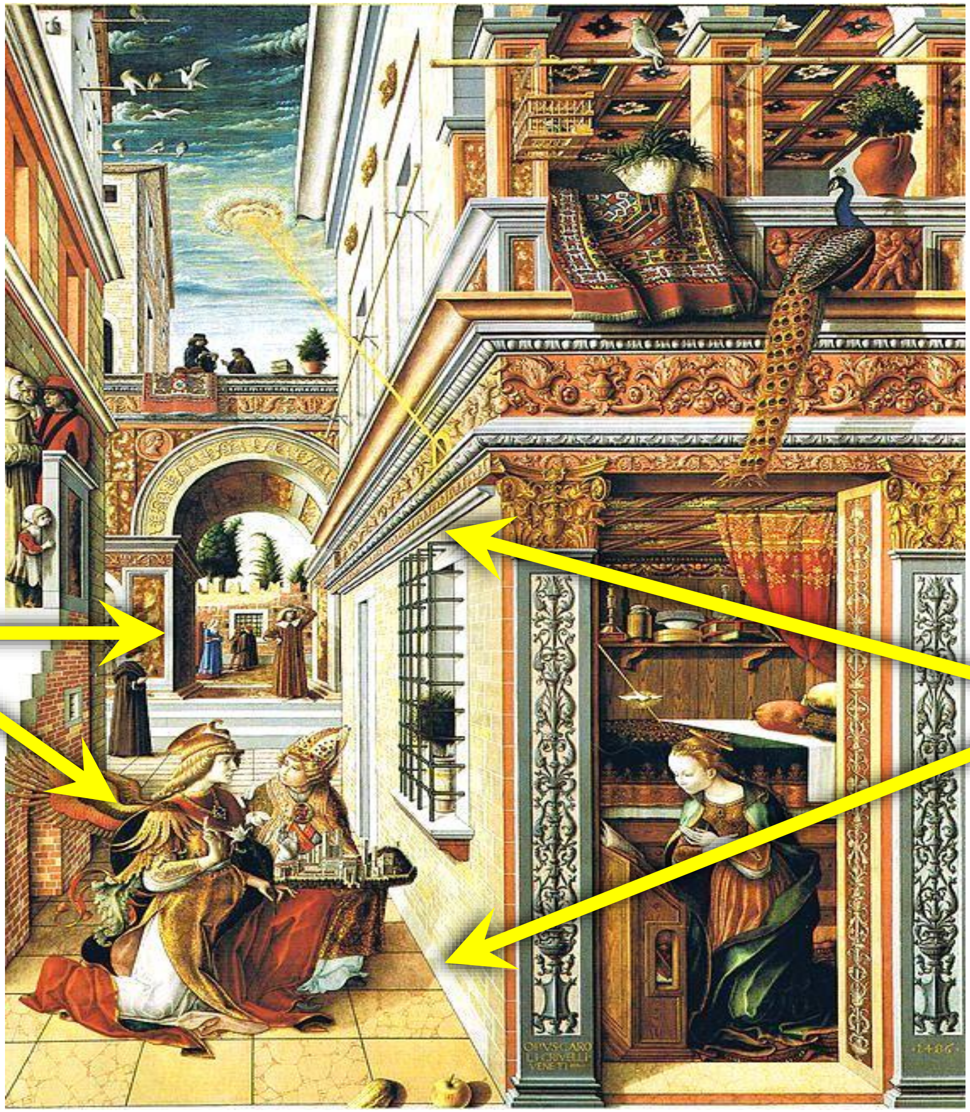


# 模型-视图矩阵


OpenGL ModelView Matrix
View (Camera)
Position X 0
Y 5
Z 6
Pitch (X) 39
Heading (Y) 0
Roll (Z) 0
Reset View (Camera)
Model
Position X 0
Y 0
Z 0
Rotation X 22
Y 16
Z 0
Reset Model
View Matrix
1.00 0.00 0.00 0.00
0.00 0.78 -0.63 -0.11
0.00 0.63 0.78 -7.81
0.00 0.00 0.00 1.00
X
Model Matrix
0.96 0.00 0.28 0.00
0.10 0.93 -0.36 0.00
-0.26 0.37 0.89 0.00
0.00 0.00 0.00 1.00
=
ModelView Matrix
0.96 0.00 0.28 0.00
0.24 0.48 -0.84 -0.11
-0.13 0.87 0.47 -7.81
0.00 0.00 0.00 1.00
OpenGL calls for View Matrix
(Translate -> Pitch -> Heading -> Roll)
glRotatef(39,0,0,1);
glRotatef(-0,0,1,0);
glRotatef(0,1,0,0);
glTranslatef(-0,-5,-6);
OpenGL calls for Model Matrix
(RotZ -> RotY -> RotX -> Translate)
glTranslatef(0,0,0);
glRotatef(22,1,0,0);
glRotatef(16,0,1,0);
glRotatef(0,0,0,1);

# 投影矩阵

- 透视投影矩阵

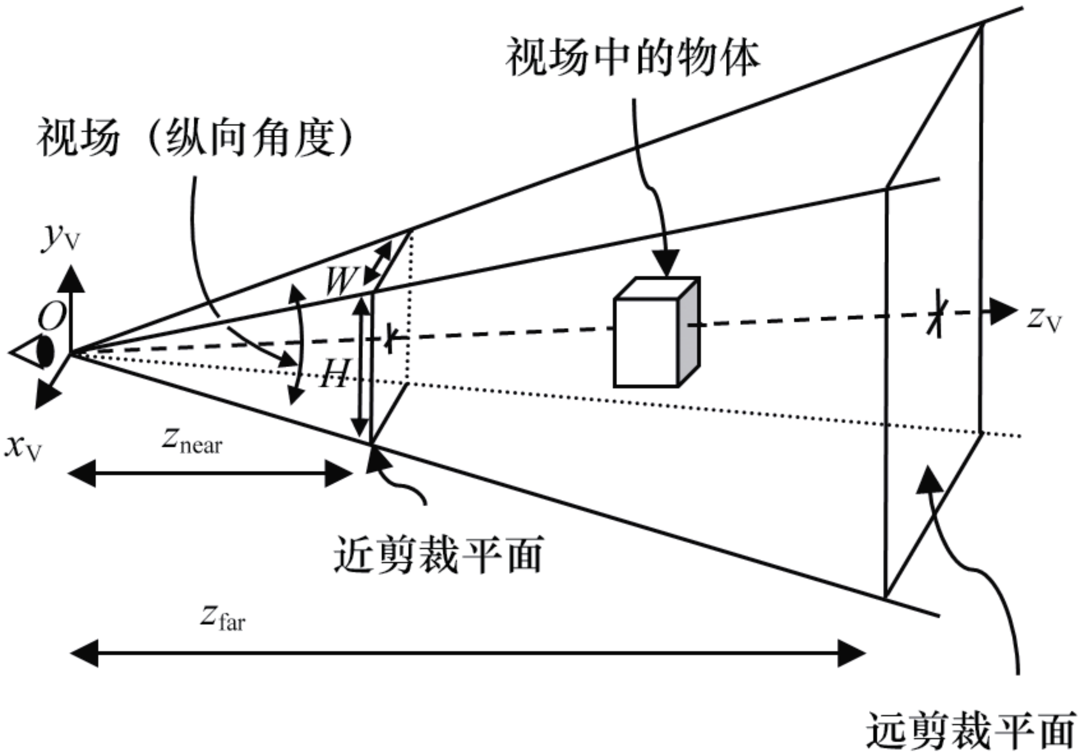


远小
近大

平行线

# 投影矩阵

- 透视投影




$$q = \frac{1}{\tan\left(\frac{\text{视场}}{2}\right)}$$

$$A = \frac{q}{\text{纵横比}}$$

$$B = \frac{z_{near} + z_{far}}{z_{near} - z_{far}}$$

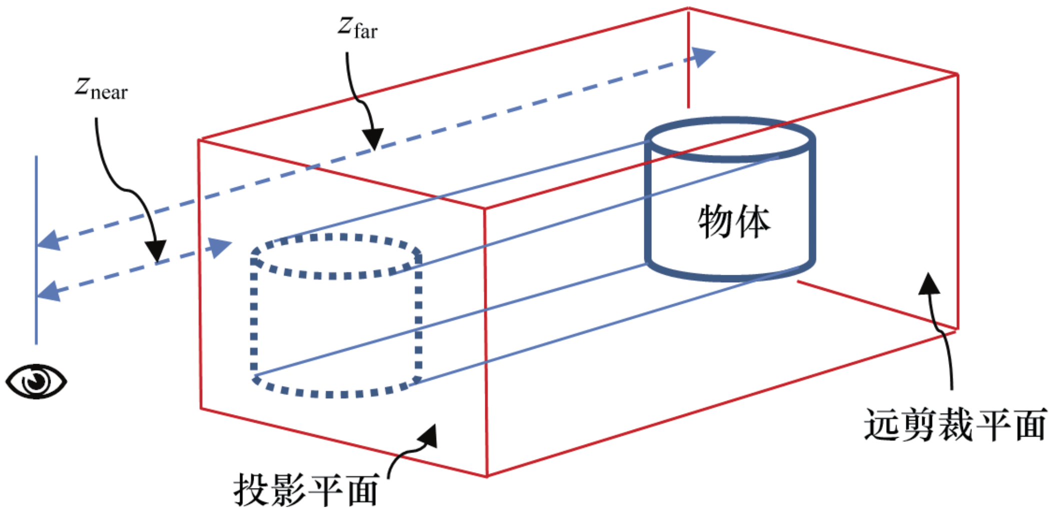
$$C = \frac{2\, z_{near}\, z_{far}}{z_{near} - z_{far}}$$

透视矩阵：

$$\begin{bmatrix} A & 0 & 0 & 0 \\ 0 & q & 0 & 0 \\ 0 & 0 & B & C \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

# 正射投影

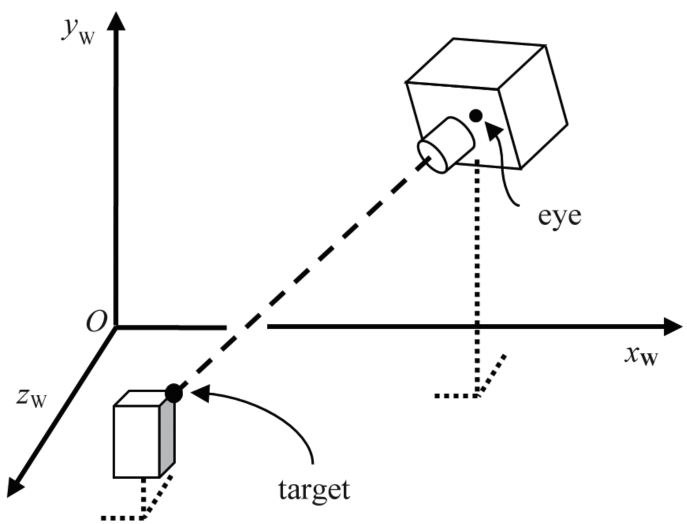
- 正射投影是一种平行投影，其中所有的投影过程都沿与投影平面垂直的方向进行




$$\begin{bmatrix} \frac{2}{R-L} & 0 & 0 & -\frac{R+L}{R-L} \\ 0 & \frac{2}{T-B} & 0 & -\frac{T+B}{T-B} \\ 0 & 0 & \frac{-2}{z_{\text{far}} - z_{\text{near}}} & -\frac{z_{\text{far}} + z_{\text{near}}}{z_{\text{far}} - z_{\text{near}}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# LookAt 矩阵

- 由相机位置（点 eye）、目标位置（点 target）、初始向上向量 ***Y*** 构建 LookAt 矩阵




$$\mathbf{fwd} = \mathrm{normalize}(\mathrm{eye} - \mathrm{target})$$

$$\mathbf{side} = \mathrm{normalize}(-\mathbf{fwd} \times \boldsymbol{Y})$$

$$\mathbf{up} = \mathrm{normalize}(\mathbf{side} \times (-\mathbf{fwd}))$$

LookAt矩阵:

$$\begin{bmatrix} \mathbf{side}_x & \mathbf{side}_y & \mathbf{side}_z & -(\mathbf{side} \cdot \mathrm{eye}) \\ \mathbf{up}_x & \mathbf{up}_y & \mathbf{up}_z & -(\mathbf{up} \cdot \mathrm{eye}) \\ -\mathbf{fwd}_x & -\mathbf{fwd}_y & -\mathbf{fwd}_z & -(-\mathbf{fwd} \cdot \mathrm{eye}) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- GLM 中已经有一个用来构建LookAt 矩阵的函数 glm::lookAt()