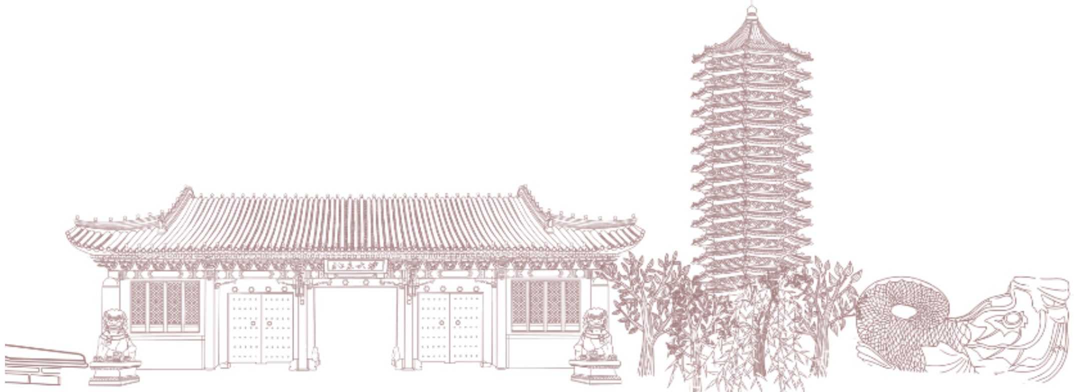
北京大學
PEKING UNIVERSITY

旋转
Rotation

阮良旺

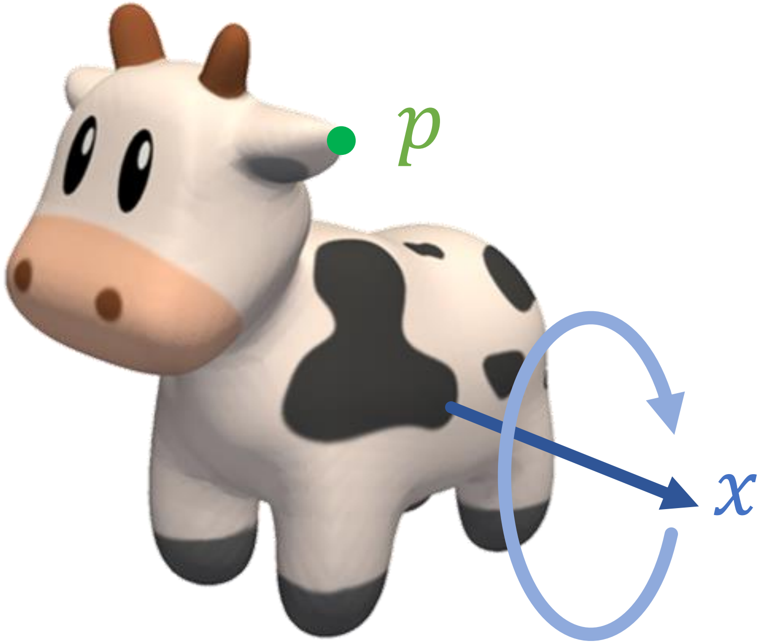
2024.3.25



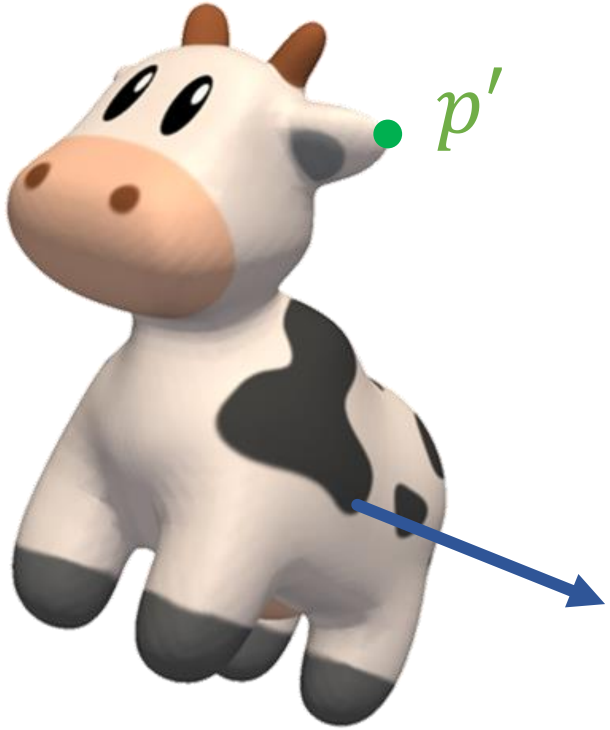
课程大纲

- Part I: 几何与代数
  - 1 线性代数基础
  - 2 计算几何
  - 3 旋转变换
  - 4 主成分分析与奇异值分解
- Part II: 数值方法
  - 5 插值、拟合与采样
  - 6 谱分析与傅里叶变换
  - 7 概率论 (I)
  - 8 概率论 (II)
- Part III: 微分方程求解
  - 9 场论初步
  - 10 古典微分几何
  - 11 微分方程
  - 12 线性系统
- Part IV: 优化与拓扑
  - 13 最优化
  - 14 机器学习 (I)
  - 15 机器学习 (II)
  - 16 拓扑

回顾旋转变换



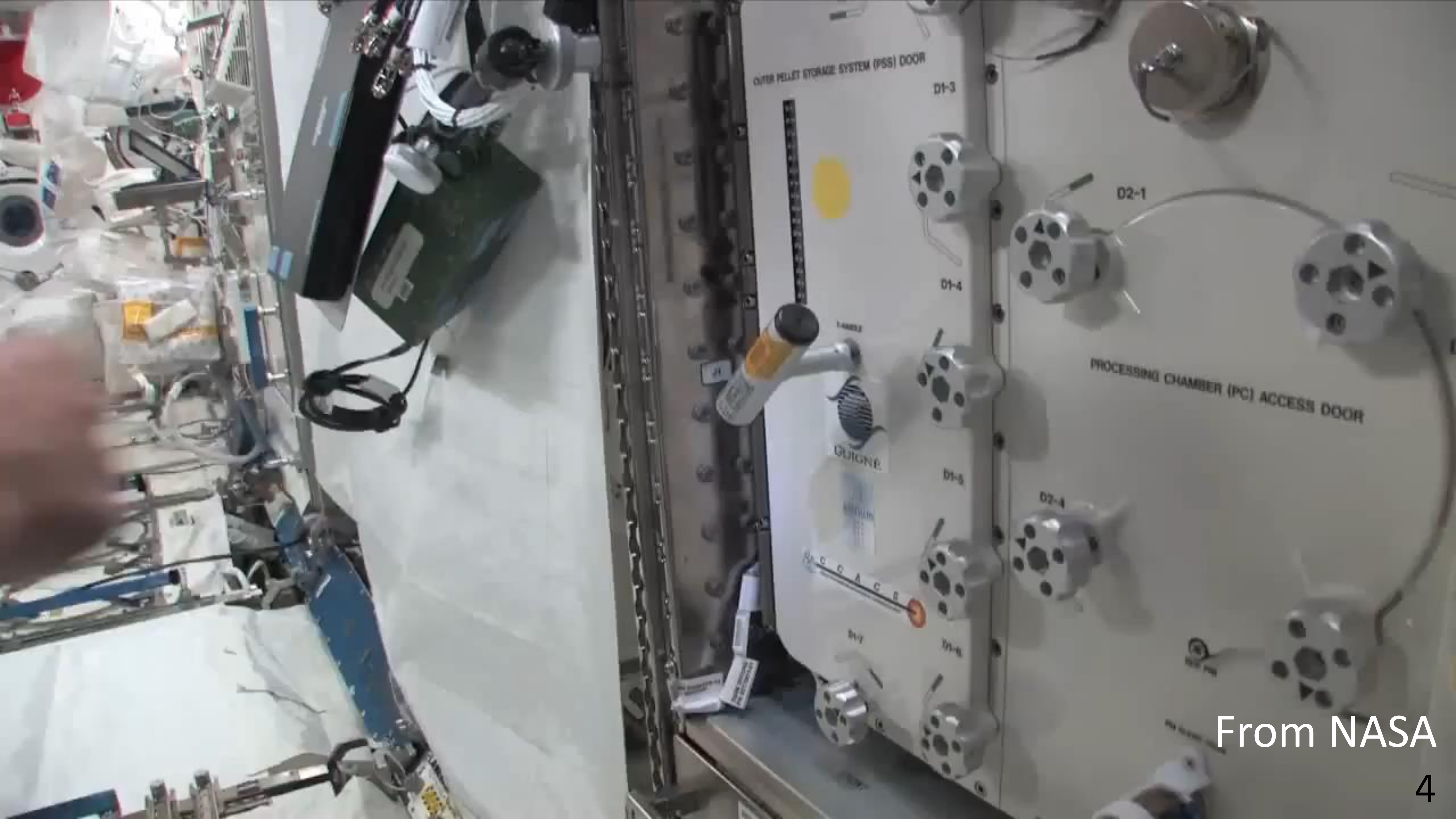
绕x轴顺时针旋转$30°$



$$p' = \mathrm{R}p$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \boxed{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(30°) & \sin(30°) \\ 0 & -\sin(30°) & \cos(30°) \end{pmatrix}} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

旋转矩阵


OUTER PELLET STORAGE SYSTEM (PSS) DOOR
D1-3
D1-4
D1-5
D1-6
D1-7
D2-1
D2-4
PROCESSING CHAMBER (PC) ACCESS DOOR
T-HANDLE

From NASA

旋转矩阵

考虑一个一般的三维线性变换：

$$p' = \mathrm{A}p, A \in \mathbb{R}^{3\times 3}$$

旋转不会改变相对距离：

$$p' - q' = \mathrm{A}(p - q), \|p' - q'\| = \|p - q\|$$

旋转矩阵

考虑一个一般的三维线性变换：

$$p' = \mathrm{A}p, A \in \mathbb{R}^{3\times 3}$$

旋转不会改变相对距离：

$$p' - q' = \mathrm{A}(p - q), \|p' - q'\| = \|p - q\|$$

于是下面的关系对任意的 p, q 都成立：

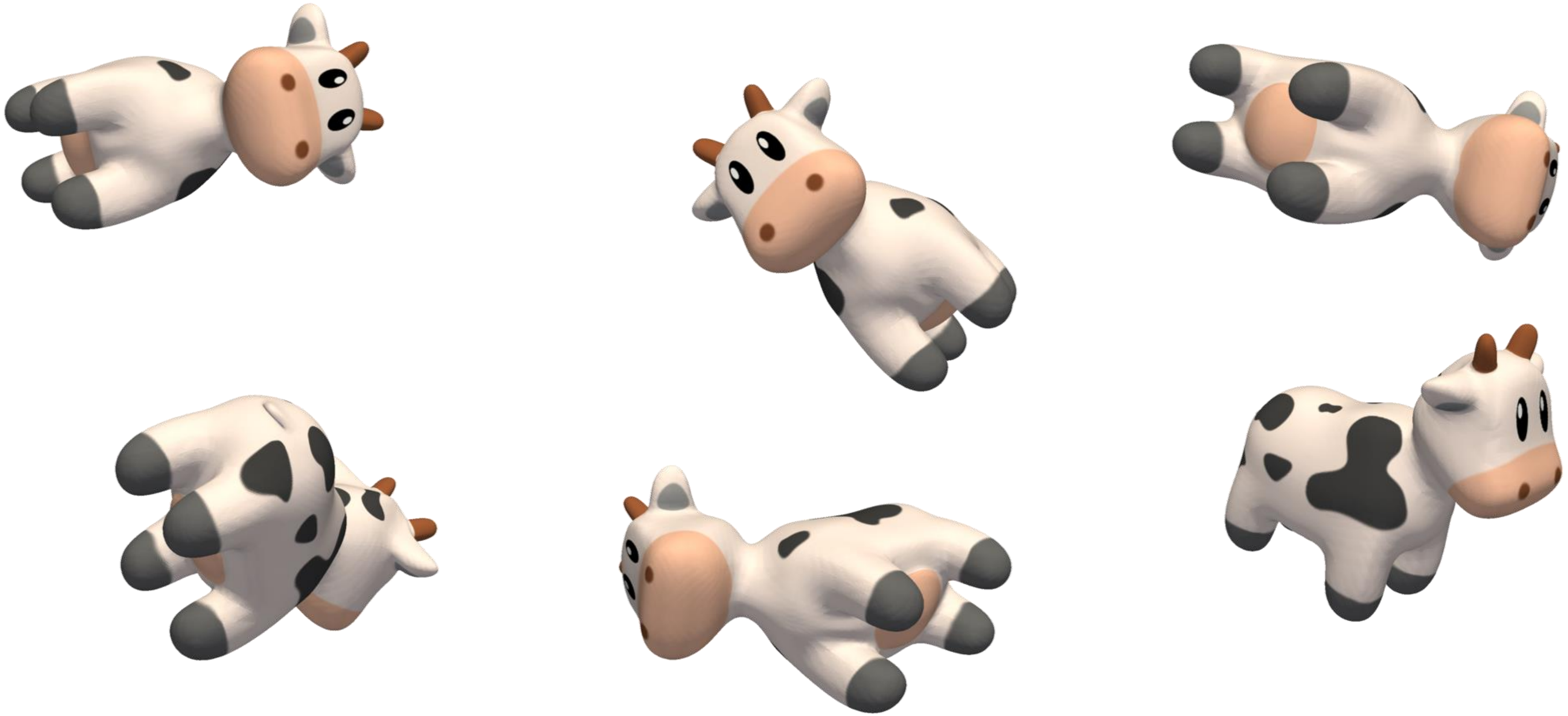
$$(p - q)^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}\mathrm{A}(p - q) = (p - q)^{\mathrm{T}}(p - q), \forall p, q$$

那么一定有：

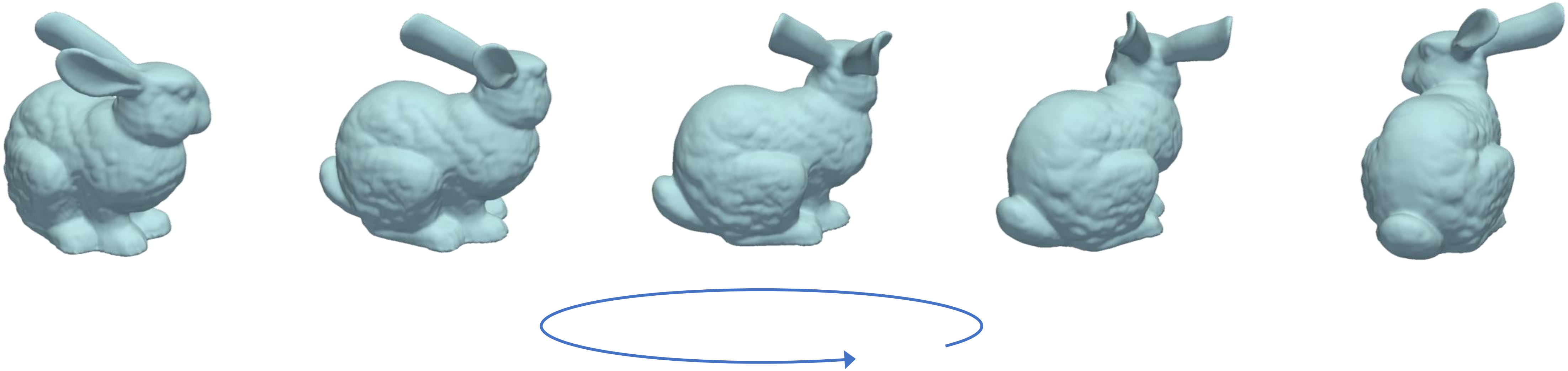
$$\mathrm{A}^{\mathrm{T}}\mathrm{A} = \mathrm{I}$$

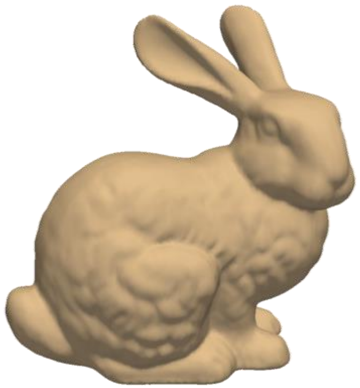
也就是说 A 是正交矩阵

随机采样正交矩阵



正交矩阵⊃旋转矩阵





$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

镜像无法通过旋转得到

旋转矩阵

旋转矩阵是特征值为+1的正交矩阵

$$R^T R = RR^T = I, \det R = 1$$

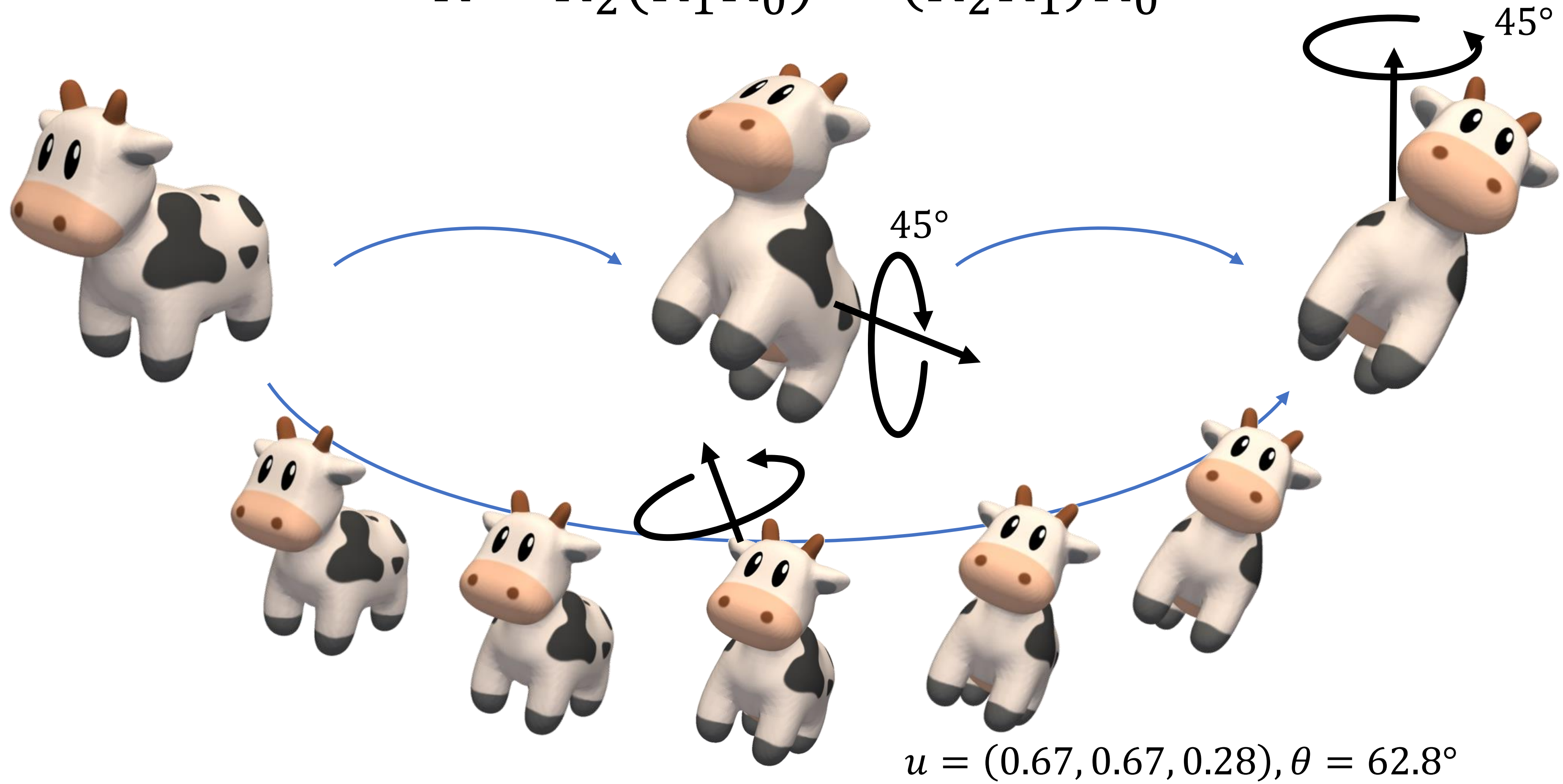
所有特征值为+1的n维正交矩阵集合称为特殊正交群 (Special Orthogonal Group)，记为 SO(n)

所以二维旋转矩阵的集合是SO(2)，三维旋转矩阵的集合是SO(3)

旋转的性质

- 任意两个旋转相乘依然是旋转，并且满足结合律：

$$R = R_2(R_1R_0) = (R_2R_1)R_0$$



旋转的性质

- 任意两个旋转相乘依然是旋转，并且满足结合律：

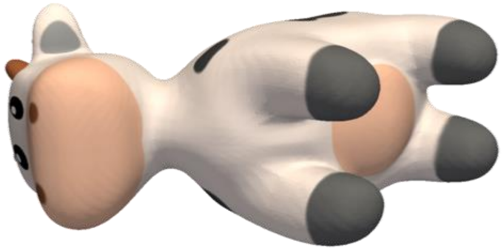
$$R = R_2(R_1R_0) = (R_2R_1)R_0$$

- 旋转是可逆的：

$$R^{-1} = R^T$$

$$R = \begin{pmatrix} 0.094 & 0.995 & 0.012 \\ 0.995 & -0.094 & -0.026 \\ 0.025 & 0.015 & 0.999 \end{pmatrix}$$

$$R^T = \begin{pmatrix} 0.094 & 0.995 & 0.025 \\ 0.995 & -0.094 & 0.015 \\ 0.012 & -0.026 & 0.999 \end{pmatrix}$$



旋转的性质

- 任意两个旋转相乘依然是旋转，并且满足结合律：

$$\mathrm{R} = \mathrm{R}_2(\mathrm{R}_1\mathrm{R}_0) = (\mathrm{R}_2\mathrm{R}_1)\mathrm{R}_0$$

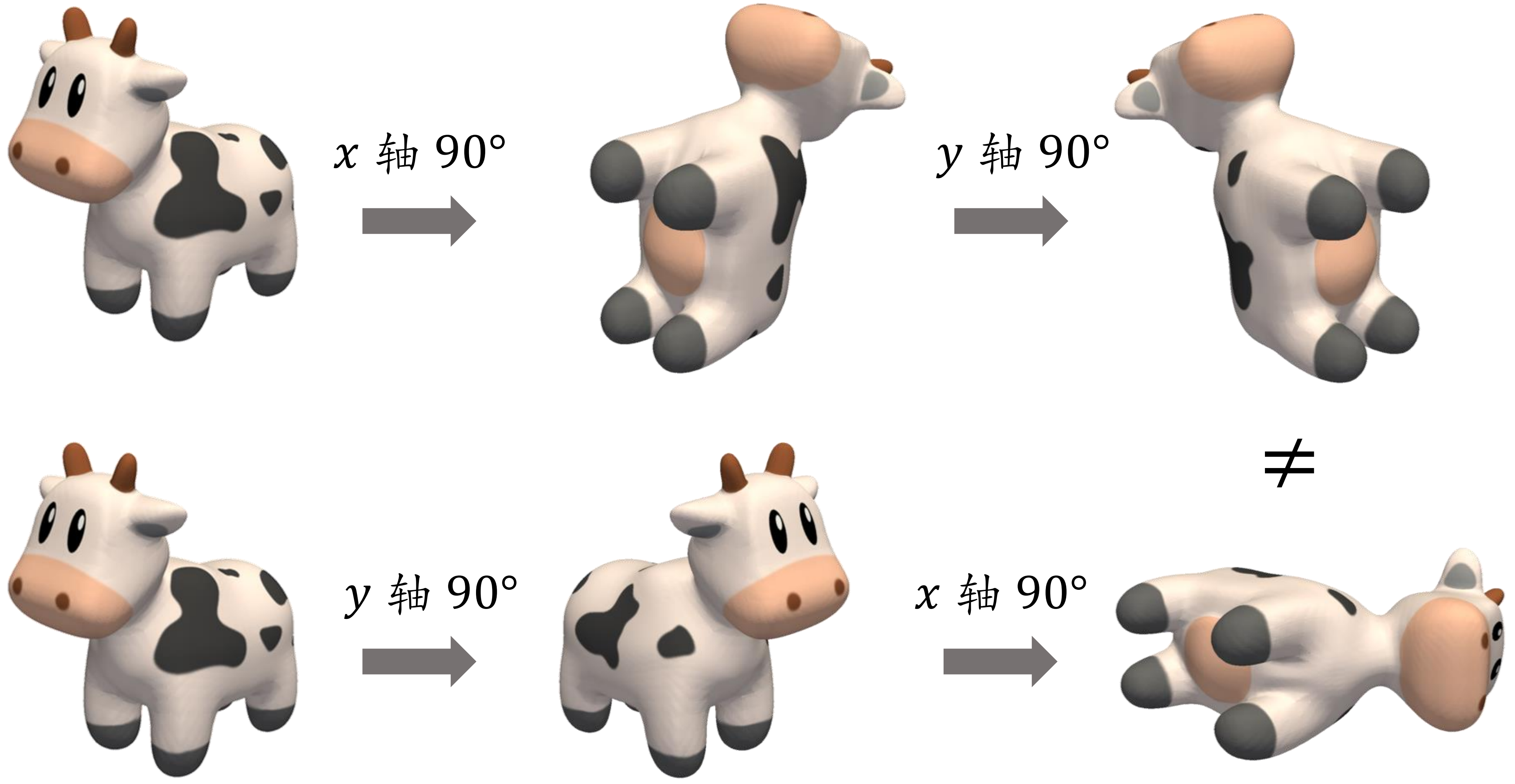
- 旋转是可逆的：

$$\mathrm{R}^{-1} = \mathrm{R}^{\mathrm{T}}$$

- 但是三维旋转不满足交换律：

$$R_1R_0 \neq R_0R_1$$

三维旋转不满足交换律



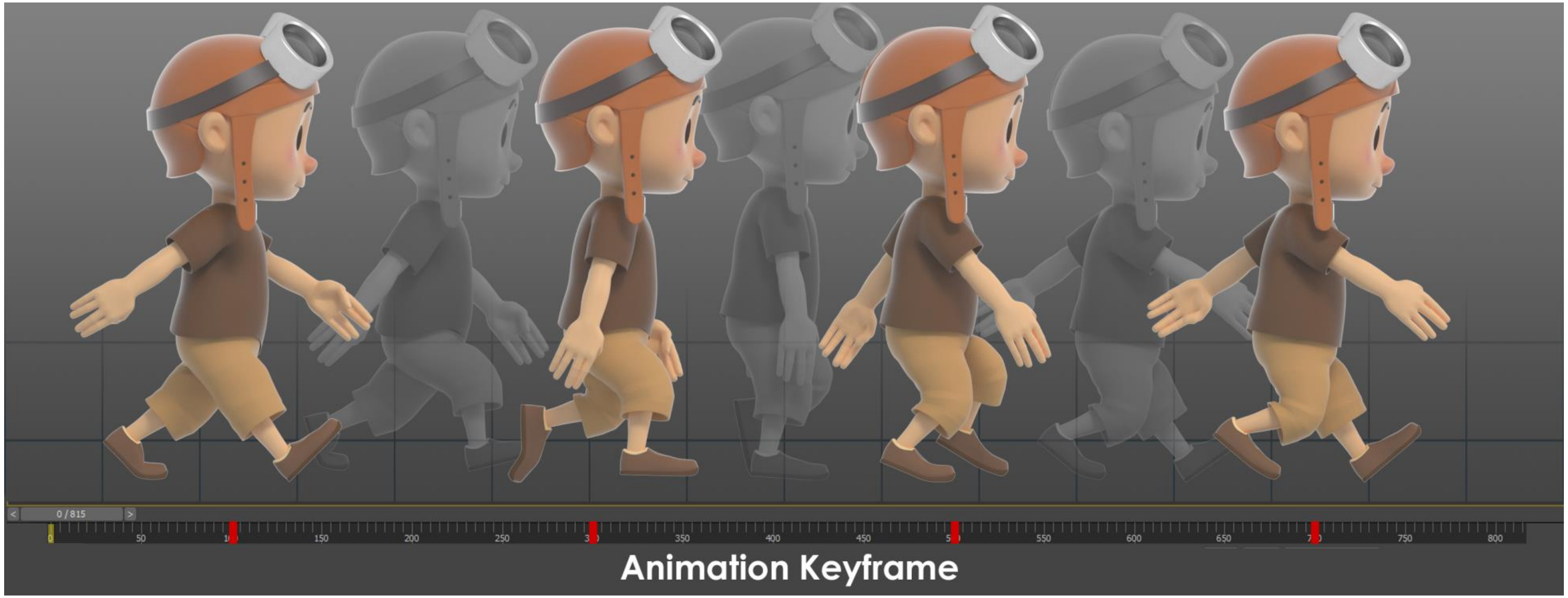
三维旋转表示
3D Rotation Representation

一些现实问题

$$R \leftarrow F(\Delta x, \Delta y)?$$

如何将屏幕空间鼠标的位移映射到相机视角的旋转?

一些现实问题



$$R_t \leftarrow F(R_0, R_1, t)?$$

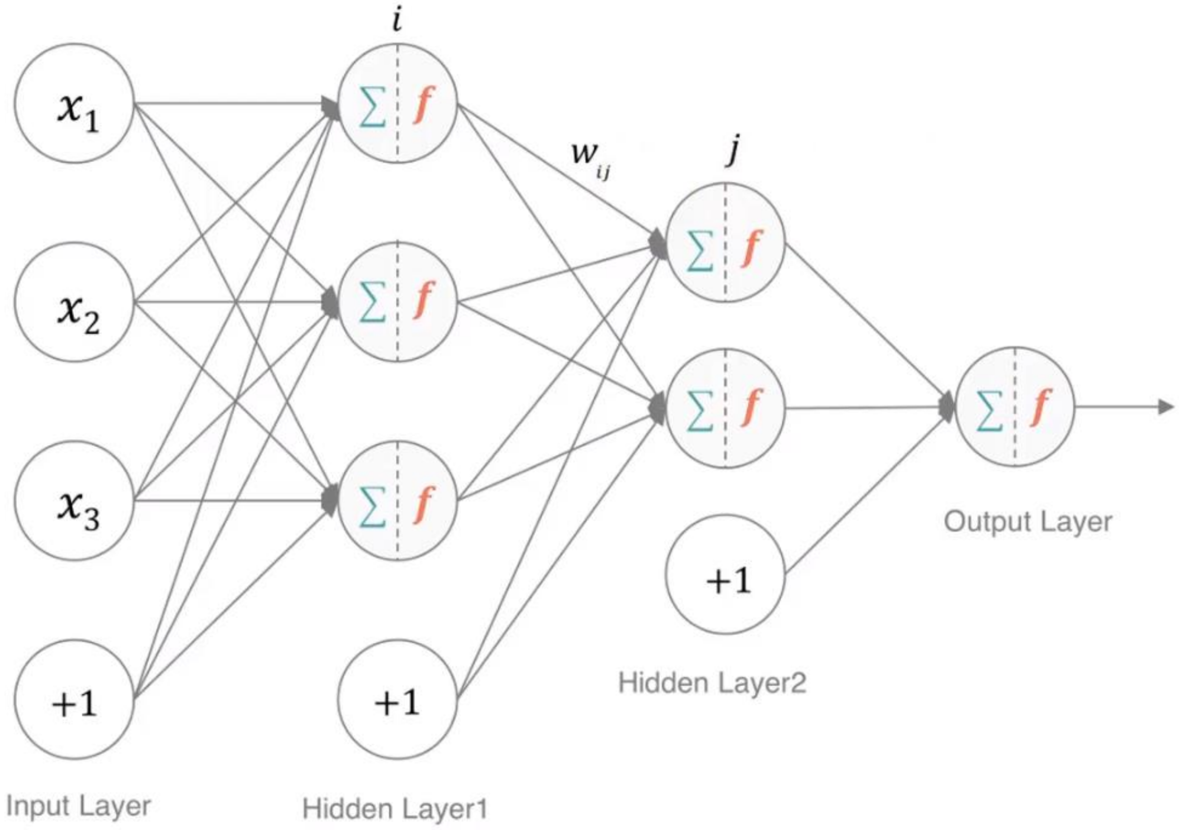
如何从关键帧的关节旋转插值出中间旋转?

一些现实问题

$$\omega \rightarrow R?$$

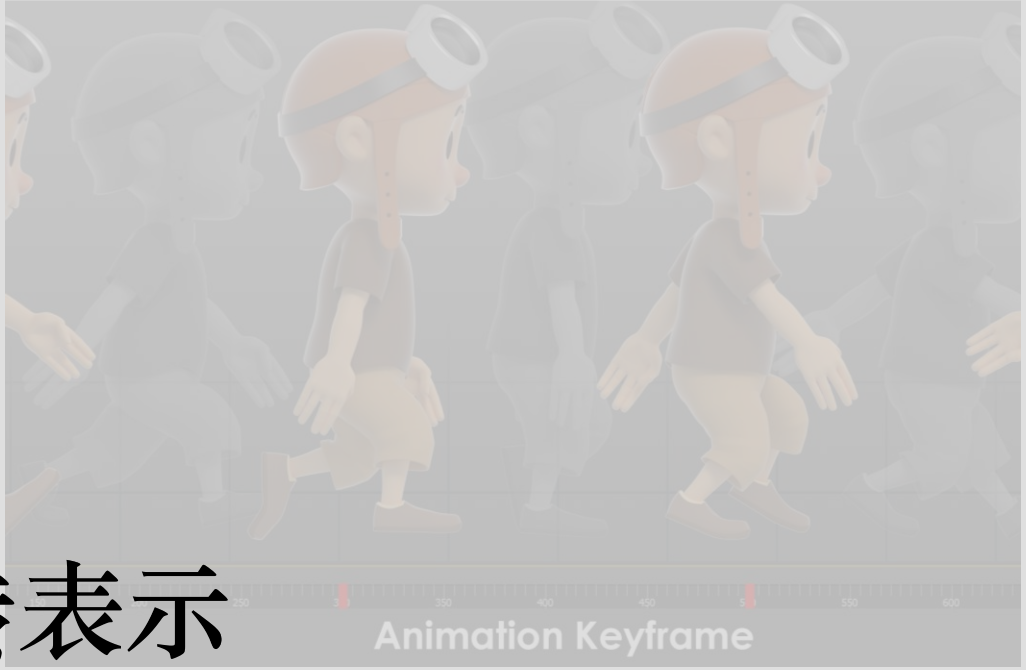
角速度（角动量）与刚体旋转的关系是什么？

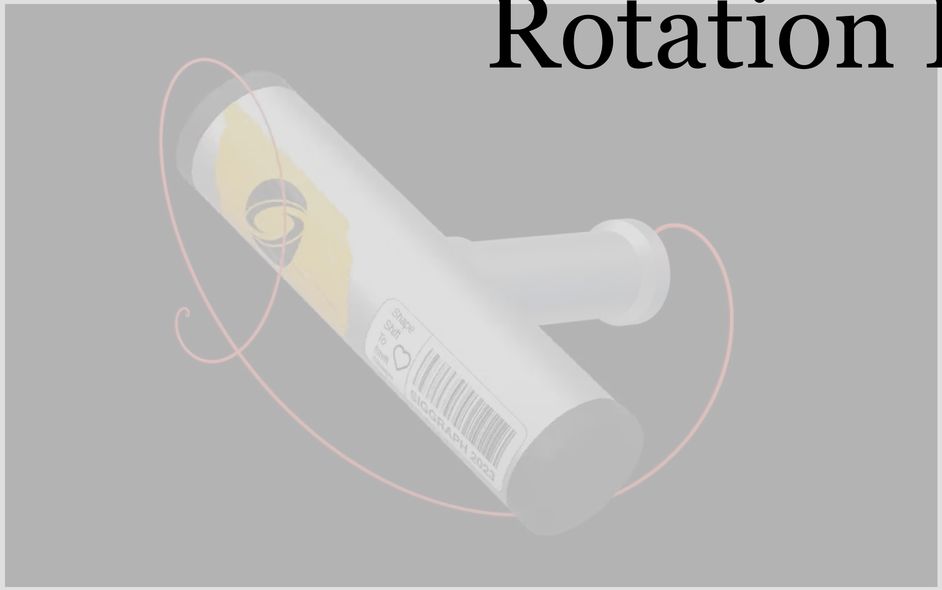
一些现实问题

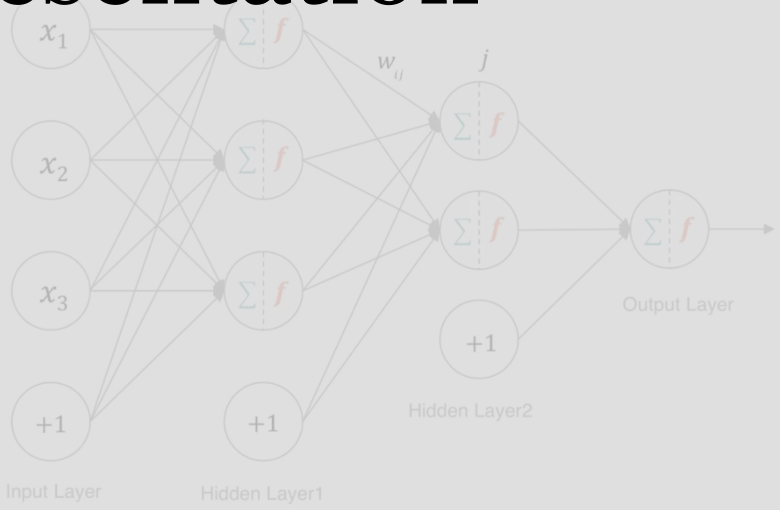


神经网络如何学习/估计/生成旋转?

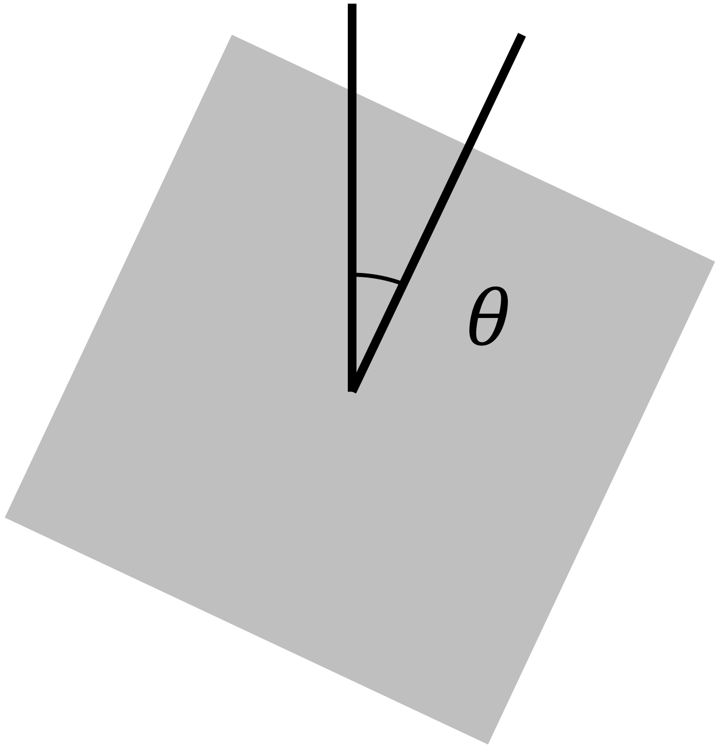
旋转表示
Rotation Representation



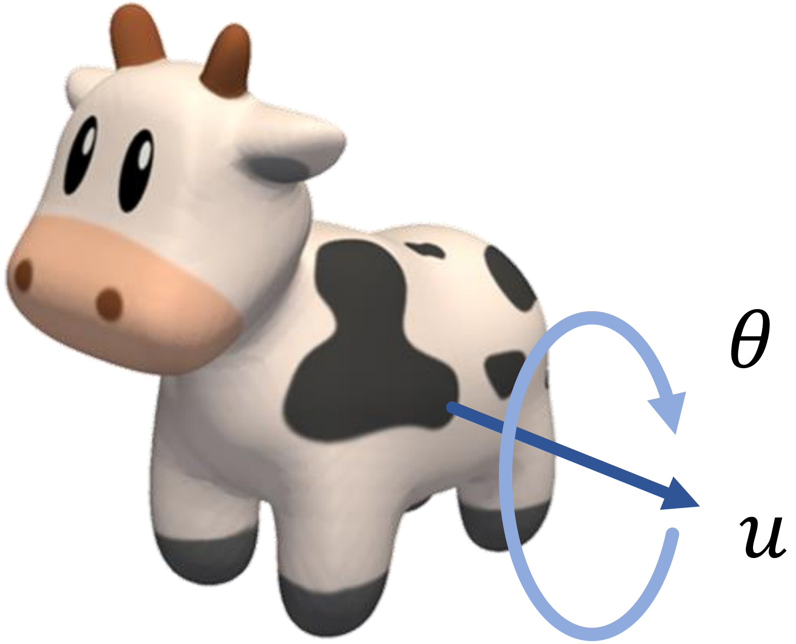




旋转自由度

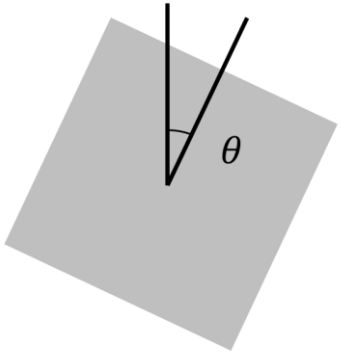
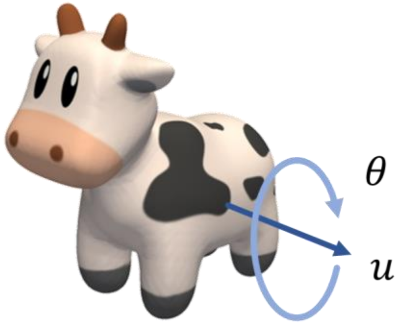


$\theta \in \mathbb{R}$
DoF = 1



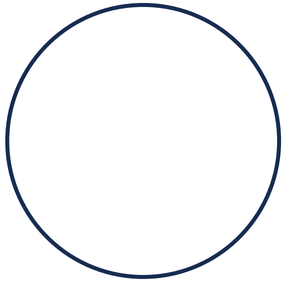
$u \in \mathbb{R}^3, \|u\| = 1, \theta \in \mathbb{R}$
DoF = 3

旋转自由度 (旋转矩阵角度)

	$\mathrm{R} \in \mathbb{R}^{2\times 2}$	$\mathrm{R}^{\mathrm{T}}\mathrm{R} = \mathrm{I} \Leftrightarrow \begin{cases} \mathrm{R}_{00}^2 + \mathrm{R}_{01}^2 = 1 \\ \mathrm{R}_{10}^2 + \mathrm{R}_{11}^2 = 1 \\ \mathrm{R}_{00}\mathrm{R}_{10} + \mathrm{R}_{01}\mathrm{R}_{11} = 0 \end{cases}$	$\mathrm{DoF} = 4 - 3 = 1$
	$\mathrm{R} \in \mathbb{R}^{3\times 3}$	$\mathrm{R}^{\mathrm{T}}\mathrm{R} = \mathrm{I} \Leftrightarrow \begin{cases} \mathrm{R}_{00}^2 + \mathrm{R}_{01}^2 + \mathrm{R}_{02}^2 = 1 \\ \mathrm{R}_{10}^2 + \mathrm{R}_{11}^2 + R_{12}^2 = 1 \\ \mathrm{R}_{20}^2 + \mathrm{R}_{21}^2 + R_{22}^2 = 1 \\ \mathrm{R}_{00}\mathrm{R}_{10} + \mathrm{R}_{01}\mathrm{R}_{11} + \mathrm{R}_{02}\mathrm{R}_{12} = 0 \\ \mathrm{R}_{00}\mathrm{R}_{20} + \mathrm{R}_{01}\mathrm{R}_{21} + \mathrm{R}_{02}\mathrm{R}_{22} = 0 \\ \mathrm{R}_{20}\mathrm{R}_{10} + \mathrm{R}_{21}\mathrm{R}_{11} + \mathrm{R}_{22}\mathrm{R}_{12} = 0 \end{cases}$	$\mathrm{DoF} = 9 - 6 = 3$

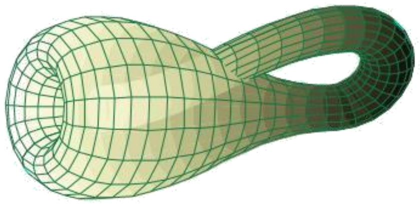
流形 (Manifold)

$\mathbb{R}^2$



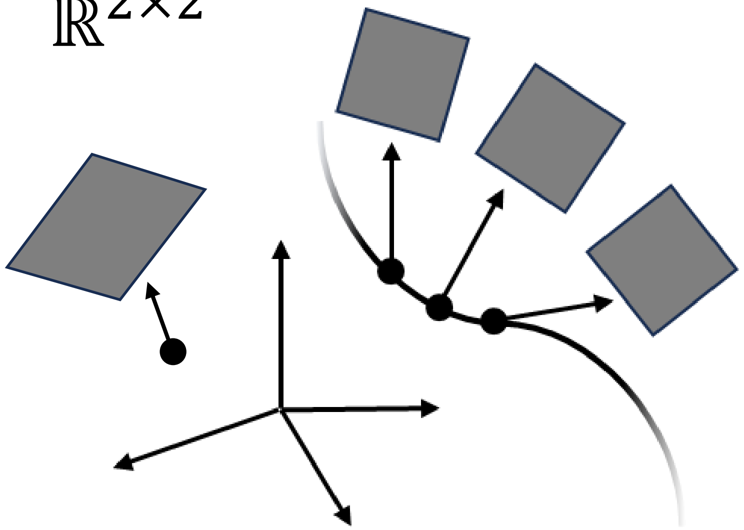
2维空间中的1维流形

$\mathbb{R}^3$



3维空间中的2维流形

$\mathbb{R}^{2\times2}$

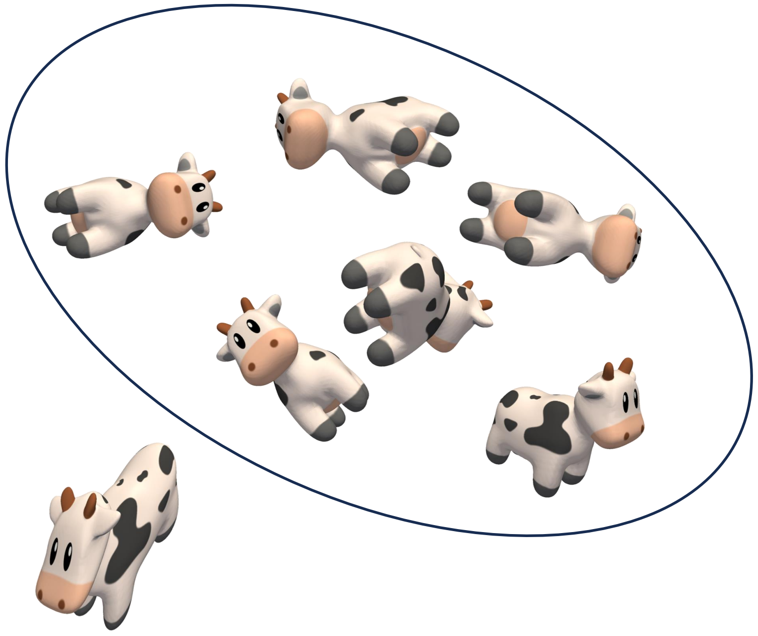


2维空间中的旋转

$R \in \mathbb{R}^{2\times2}, \mathrm{DoF} = 1$

↓

4维空间中的1维流形



3维空间中的旋转

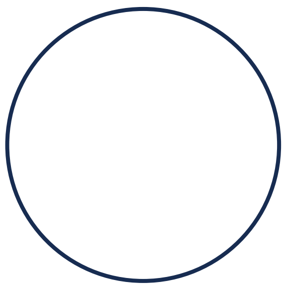
$R \in \mathbb{R}^{3\times3}, \mathrm{DoF} = 3$

↓

9维空间中的3维流形

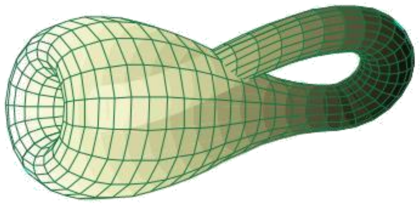
流形 (Manifold)

$\mathbb{R}^2$

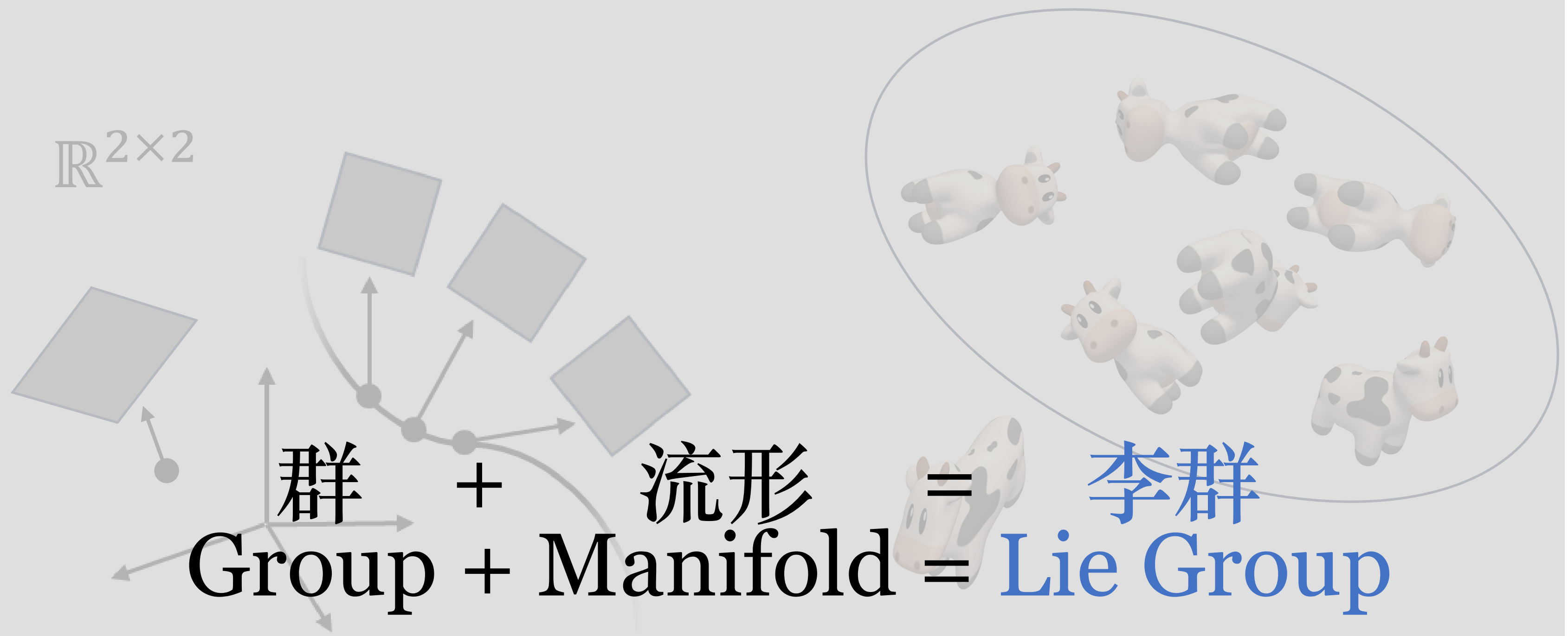


2维空间中的1维流形

$\mathbb{R}^3$



3维空间中的2维流形



群 + 流形 = 李群

Group + Manifold = Lie Group

2维空间中的旋转

$R \in \mathbb{R}^{2\times2}, \mathrm{DoF} = 1$

↓

4维空间中的1维流形

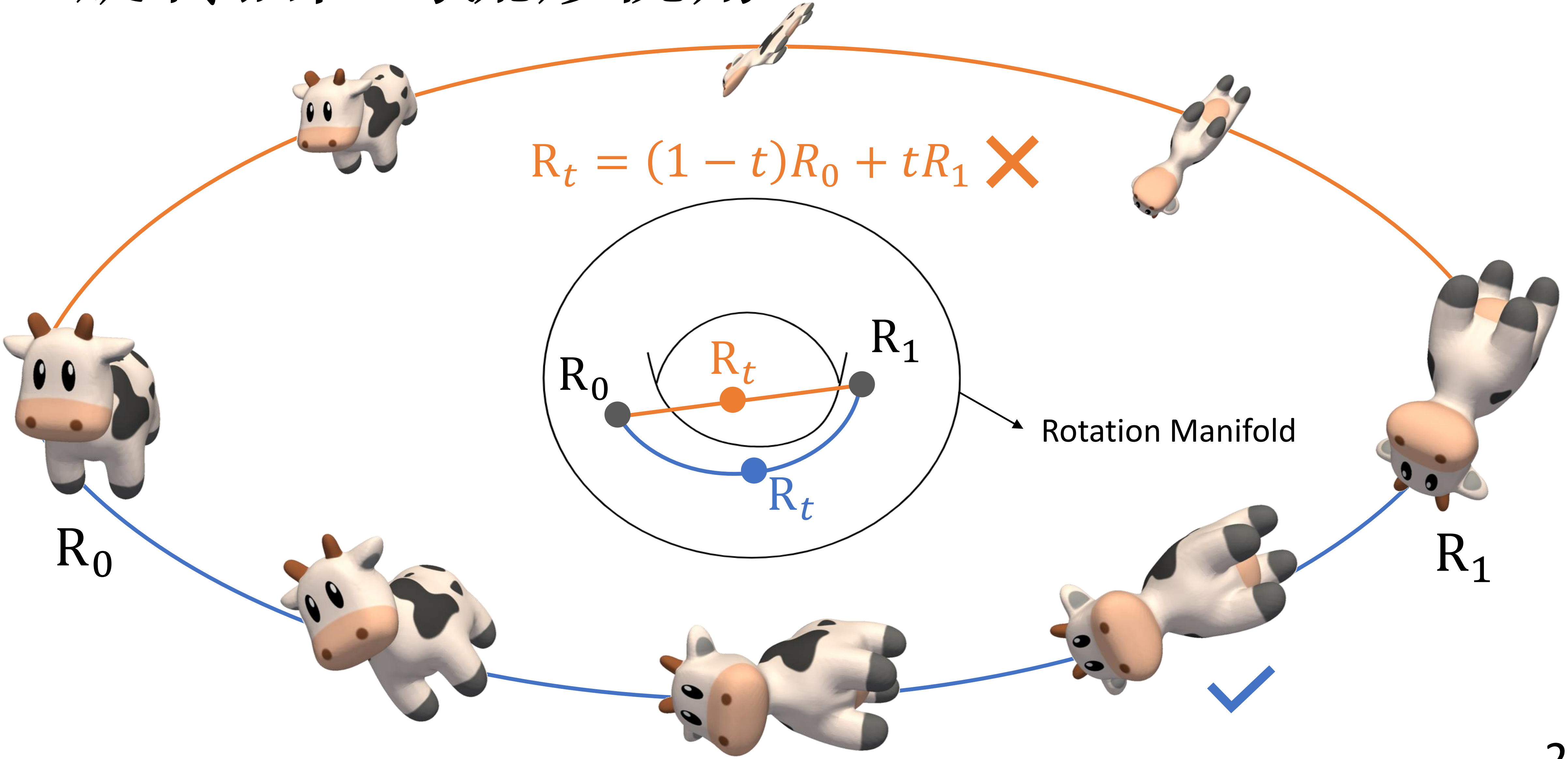
3维空间中的旋转

$R \in \mathbb{R}^{3\times3}, \mathrm{DoF} = 3$

↓

9维空间中的3维流形

旋转插值的流形视角

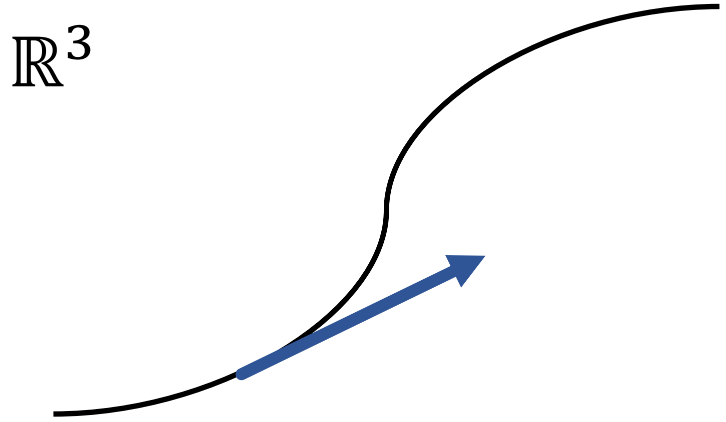


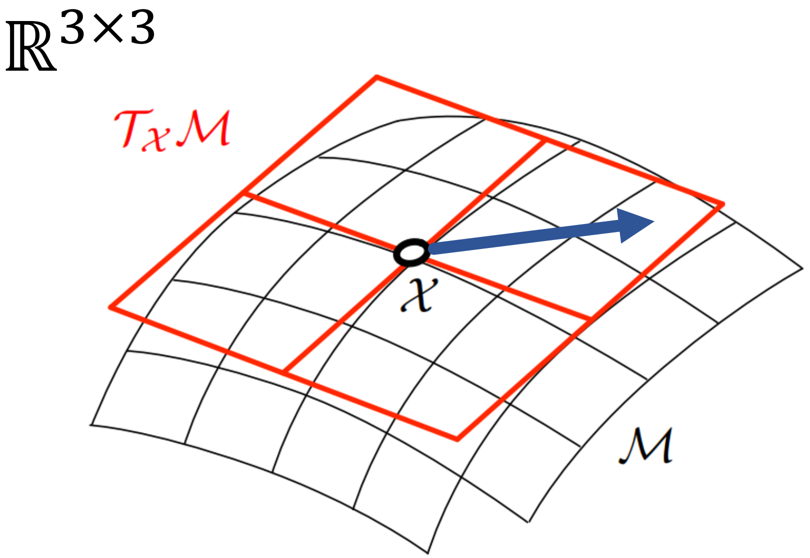
角速度与切空间

$$v = \frac{dx}{dt} \in \mathbb{R}^3$$

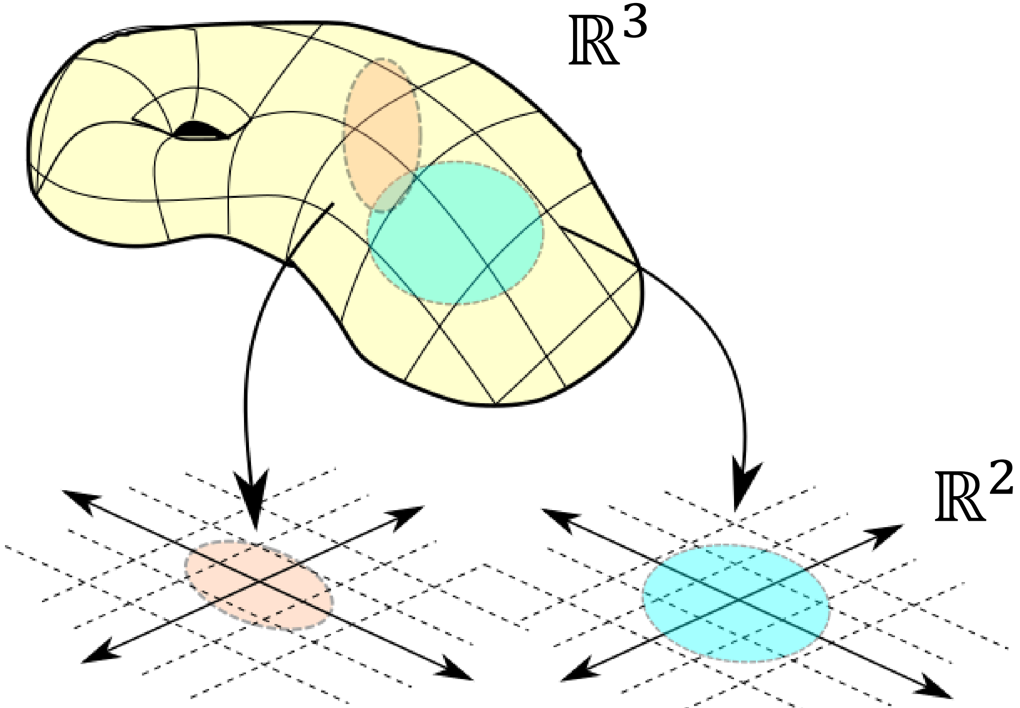
$$\omega \in \mathbb{R}^3 \neq \frac{d\mathrm{R}}{dt} \in \mathbb{R}^{3\times3}$$

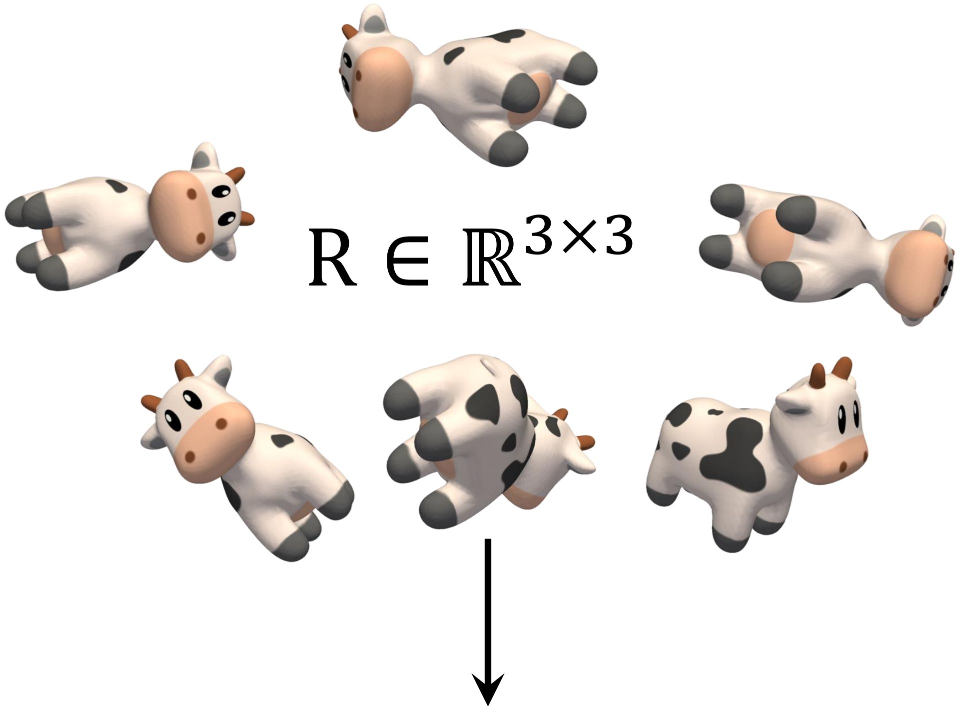
但是切空间$\mathcal{T}_x\mathcal{M} \cong \mathbb{R}^3$





旋转表示=流形参数化

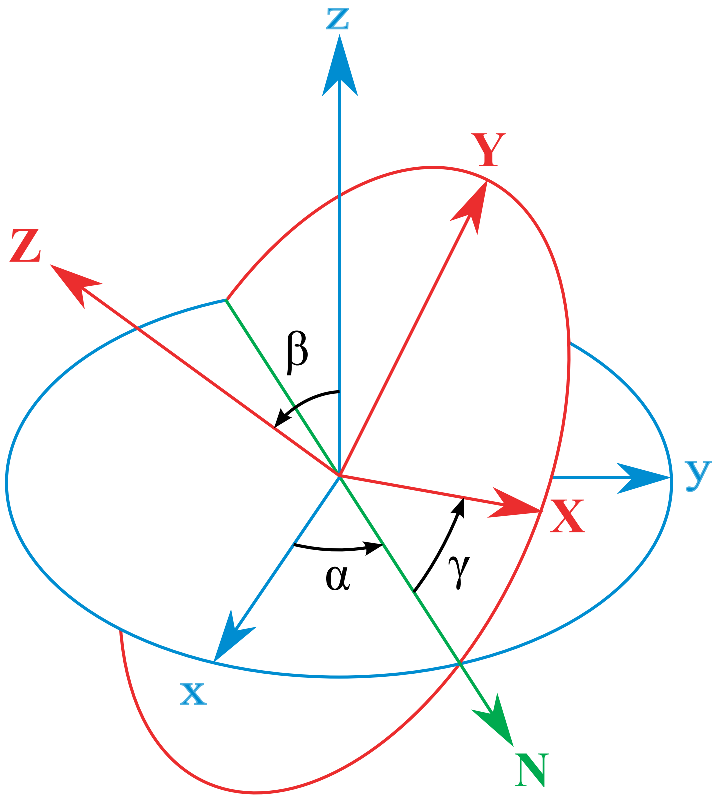


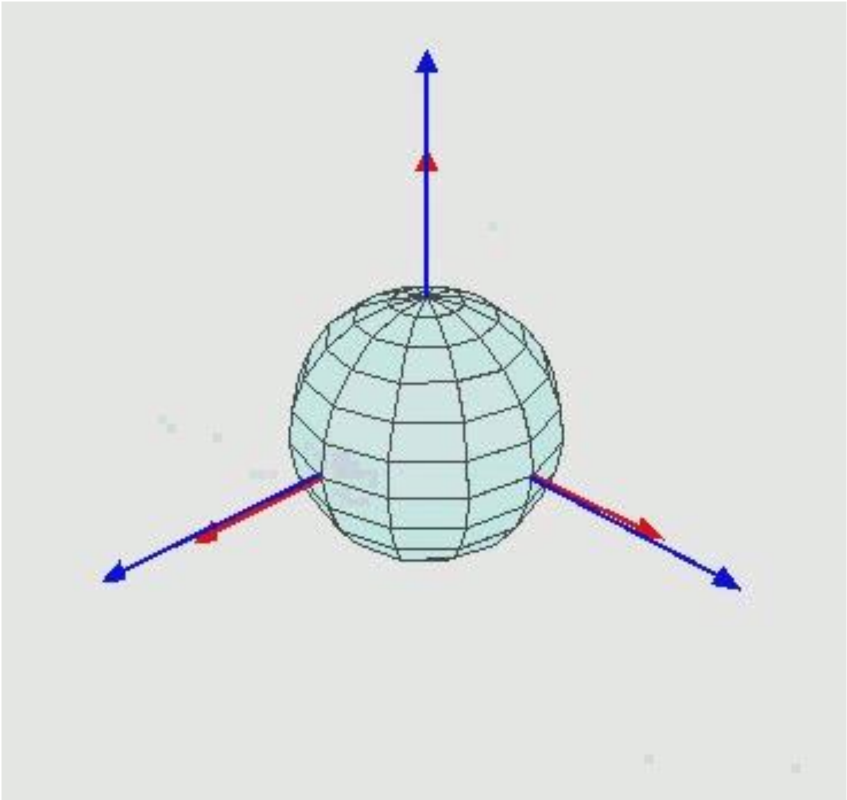


$\mathrm{DoF} = 3$

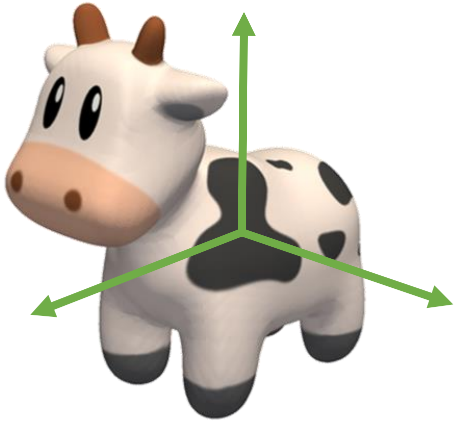
欧拉角, 轴角, 四元数, 指数⋯

欧拉角 (Euler Angle)





旋转与坐标变换



对于旋转前的一个点p，旋转后有两种表示方法：

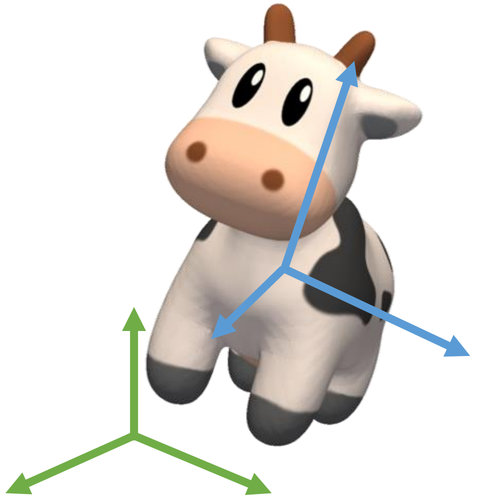
- 在世界坐标系下的坐标为p'_w
- 在局部坐标系下的坐标为$p'_l = p$ (局部坐标系下点没动)

只在世界坐标系下考虑，有

$$p'_w = \mathrm{R}p$$

也可以理解为

$$p'_w = \mathrm{R}p'_l$$



表示的是从局部坐标系到世界坐标系的坐标变换

欧拉角 (Euler Angle)

$$(\alpha, \beta, \gamma) \in \mathbb{R}^3$$

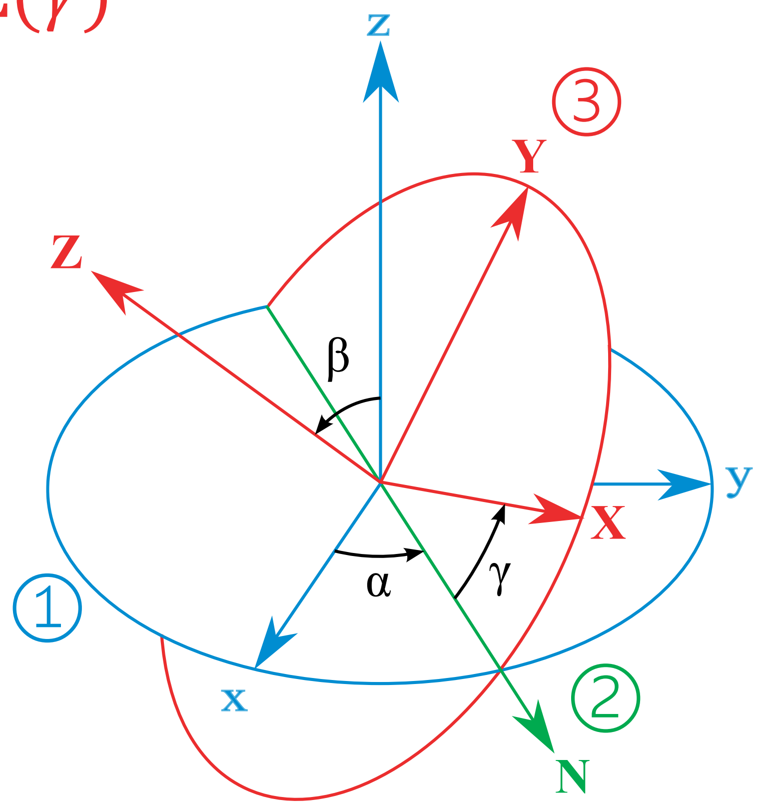
$$x_w = R_1 x_l^1 = R_1 R_2 x_l^2 = R_1 R_2 R_3 x_l^3$$

$$R = Z(\alpha) X(\beta) Z(\gamma)$$

$$Z(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$X(\beta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) \\ 0 & \sin(\beta) & \cos(\beta) \end{pmatrix}$$

$$Z(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



欧拉角 (Euler Angle)

可以自由选择三次旋转的顺序：XZX, XYX, XYZ, ZYX⋯

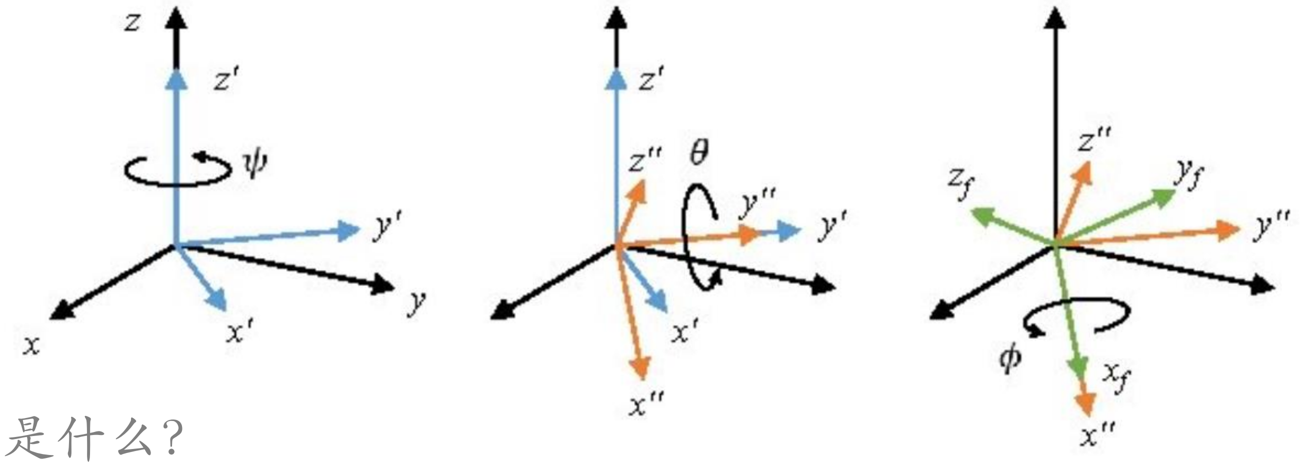
但是不同旋转顺序结果互不相同：**旋转群不满足交换律**

Proper Euler angles	Tait–Bryan angles
$X_1Z_2X_3 = \begin{bmatrix} c_2 & -c_3s_2 & s_2s_3 \\ c_1s_2 & c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 \\ s_1s_2 & c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$	$X_1Z_2Y_3 = \begin{bmatrix} c_2c_3 & -s_2 & c_2s_3 \\ s_1s_3 + c_1c_3s_2 & c_1c_2 & c_1s_2s_3 - c_3s_1 \\ c_3s_1s_2 - c_1s_3 & c_2s_1 & c_1c_3 + s_1s_2s_3 \end{bmatrix}$
$X_1Y_2X_3 = \begin{bmatrix} c_2 & s_2s_3 & c_3s_2 \\ s_1s_2 & c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 \\ -c_1s_2 & c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$	$X_1Y_2Z_3 = \begin{bmatrix} c_2c_3 & -c_2s_3 & s_2 \\ c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 & -c_2s_1 \\ s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 & c_1c_2 \end{bmatrix}$
$Y_1X_2Y_3 = \begin{bmatrix} c_1c_3 - c_2s_1s_3 & s_1s_2 & c_1s_3 + c_2c_3s_1 \\ s_2s_3 & c_2 & -c_3s_2 \\ -c_3s_1 - c_1c_2s_3 & c_1s_2 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$	$Y_1X_2Z_3 = \begin{bmatrix} c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 & c_2s_1 \\ c_2s_3 & c_2c_3 & -s_2 \\ c_1s_2s_3 - c_3s_1 & c_1c_3s_2 + s_1s_3 & c_1c_2 \end{bmatrix}$
$Y_1Z_2Y_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1s_2 & c_3s_1 + c_1c_2s_3 \\ c_3s_2 & c_2 & s_2s_3 \\ -c_1s_3 - c_2c_3s_1 & s_1s_2 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$	$Y_1Z_2X_3 = \begin{bmatrix} c_1c_2 & s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 \\ s_2 & c_2c_3 & -c_2s_3 \\ -c_2s_1 & c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 \end{bmatrix}$
$Z_1Y_2Z_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 & c_1s_2 \\ c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 & s_1s_2 \\ -c_3s_2 & s_2s_3 & c_2 \end{bmatrix}$	$Z_1Y_2X_3 = \begin{bmatrix} c_1c_2 & c_1s_2s_3 - c_3s_1 & s_1s_3 + c_1c_3s_2 \\ c_2s_1 & c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 \\ -s_2 & c_2s_3 & c_2c_3 \end{bmatrix}$
$Z_1X_2Z_3 = \begin{bmatrix} c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 & s_1s_2 \\ c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 & -c_1s_2 \\ s_2s_3 & c_3s_2 & c_2 \end{bmatrix}$	$Z_1X_2Y_3 = \begin{bmatrix} c_1c_3 - s_1s_2s_3 & -c_2s_1 & c_1s_3 + c_3s_1s_2 \\ c_3s_1 + c_1s_2s_3 & c_1c_2 & s_1s_3 - c_1c_3s_2 \\ -c_2s_3 & s_2 & c_2c_3 \end{bmatrix}$

Proper Euler angles		Tait–Bryan angles	
$X_1Z_2X_3$	$\alpha = \arctan\left(\frac{R_{31}}{R_{21}}\right)$ $\beta = \arccos(R_{11})$ $\gamma = \arctan\left(\frac{R_{13}}{-R_{12}}\right)$	$X_1Z_2Y_3$	$\alpha = \arctan\left(\frac{R_{32}}{R_{22}}\right)$ $\beta = \arcsin(-R_{12})$ $\gamma = \arctan\left(\frac{R_{13}}{R_{11}}\right)$
$X_1Y_2X_3$	$\alpha = \arctan\left(\frac{R_{21}}{-R_{31}}\right)$ $\beta = \arccos(R_{11})$ $\gamma = \arctan\left(\frac{R_{12}}{R_{13}}\right)$	$X_1Y_2Z_3$	$\alpha = \arctan\left(\frac{-R_{23}}{R_{33}}\right)$ $\beta = \arcsin(R_{13})$ $\gamma = \arctan\left(\frac{-R_{12}}{R_{11}}\right)$
$Y_1X_2Y_3$	$\alpha = \arctan\left(\frac{R_{12}}{R_{32}}\right)$ $\beta = \arccos(R_{22})$ $\gamma = \arctan\left(\frac{R_{21}}{-R_{23}}\right)$	$Y_1X_2Z_3$	$\alpha = \arctan\left(\frac{R_{13}}{R_{33}}\right)$ $\beta = \arcsin(-R_{23})$ $\gamma = \arctan\left(\frac{R_{21}}{R_{22}}\right)$
$Y_1Z_2Y_3$	$\alpha = \arctan\left(\frac{R_{32}}{-R_{12}}\right)$ $\beta = \arccos(R_{22})$ $\gamma = \arctan\left(\frac{R_{23}}{R_{21}}\right)$	$Y_1Z_2X_3$	$\alpha = \arctan\left(\frac{-R_{31}}{R_{11}}\right)$ $\beta = \arcsin(R_{21})$ $\gamma = \arctan\left(\frac{-R_{23}}{R_{22}}\right)$
$Z_1Y_2Z_3$	$\alpha = \arctan\left(\frac{R_{23}}{R_{13}}\right)$ $\beta = \arctan\left(\frac{\sqrt{1-R_{33}^2}}{R_{33}}\right)$ $\gamma = \arctan\left(\frac{R_{32}}{-R_{31}}\right)$	$Z_1Y_2X_3$	$\alpha = \arctan\left(\frac{R_{21}}{R_{11}}\right)$ $\beta = \arcsin(-R_{31})$ $\gamma = \arctan\left(\frac{R_{32}}{R_{33}}\right)$
$Z_1X_2Z_3$	$\alpha = \arctan\left(\frac{R_{13}}{-R_{23}}\right)$ $\beta = \arccos(R_{33})$ $\gamma = \arctan\left(\frac{R_{31}}{R_{32}}\right)$	$Z_1X_2Y_3$	$\alpha = \arctan\left(\frac{-R_{12}}{R_{22}}\right)$ $\beta = \arcsin(R_{32})$ $\gamma = \arctan\left(\frac{-R_{31}}{R_{33}}\right)$

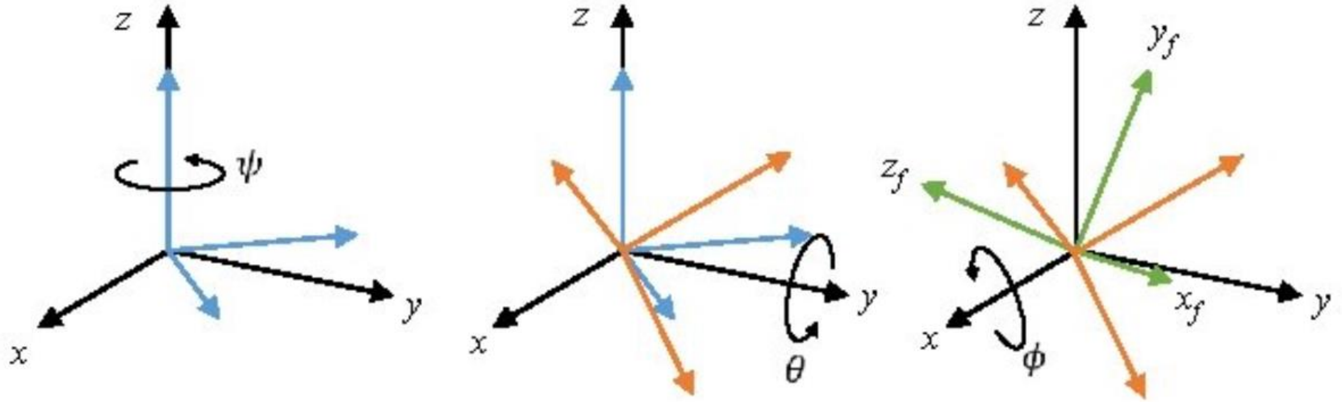
欧拉角 (Euler Angle)

内旋 (intrinsic rotations)



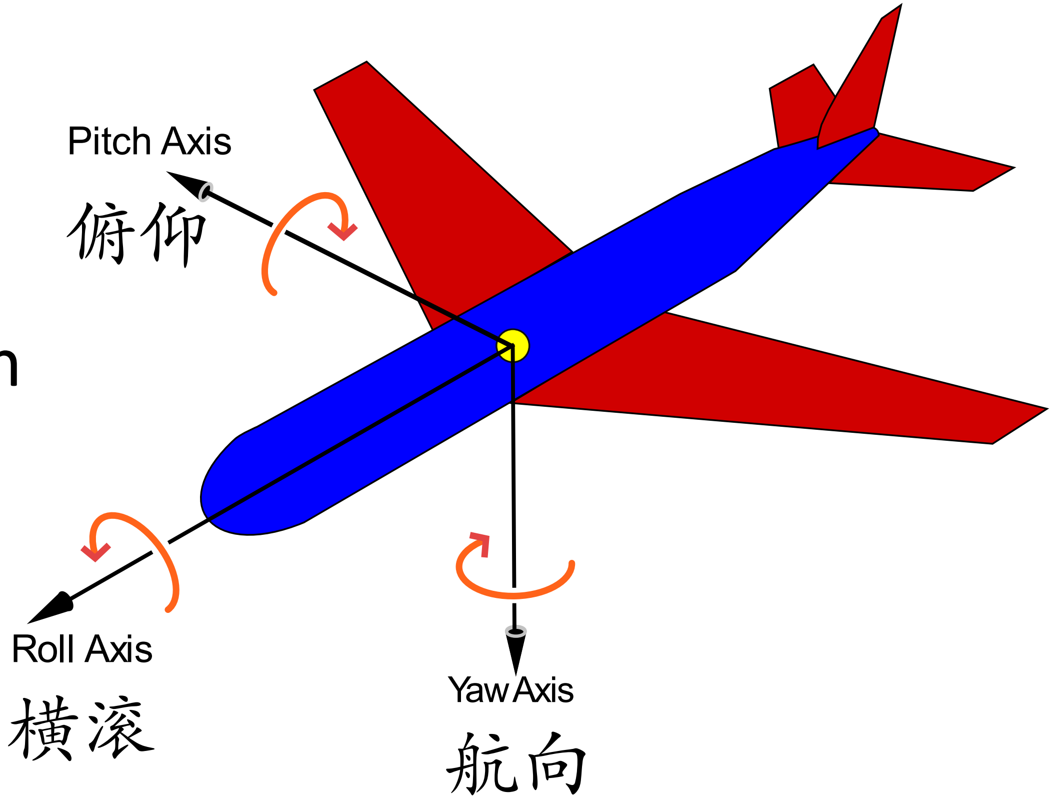
作业：内旋表示与外旋表示的关系是什么？

外旋 (extrinsic rotations)



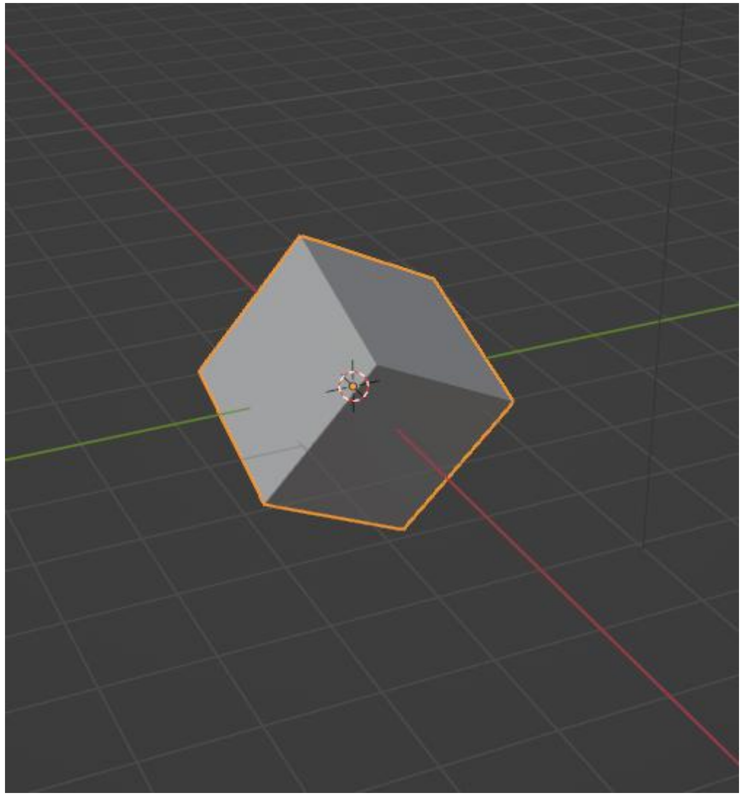
欧拉角 (Euler Angle)

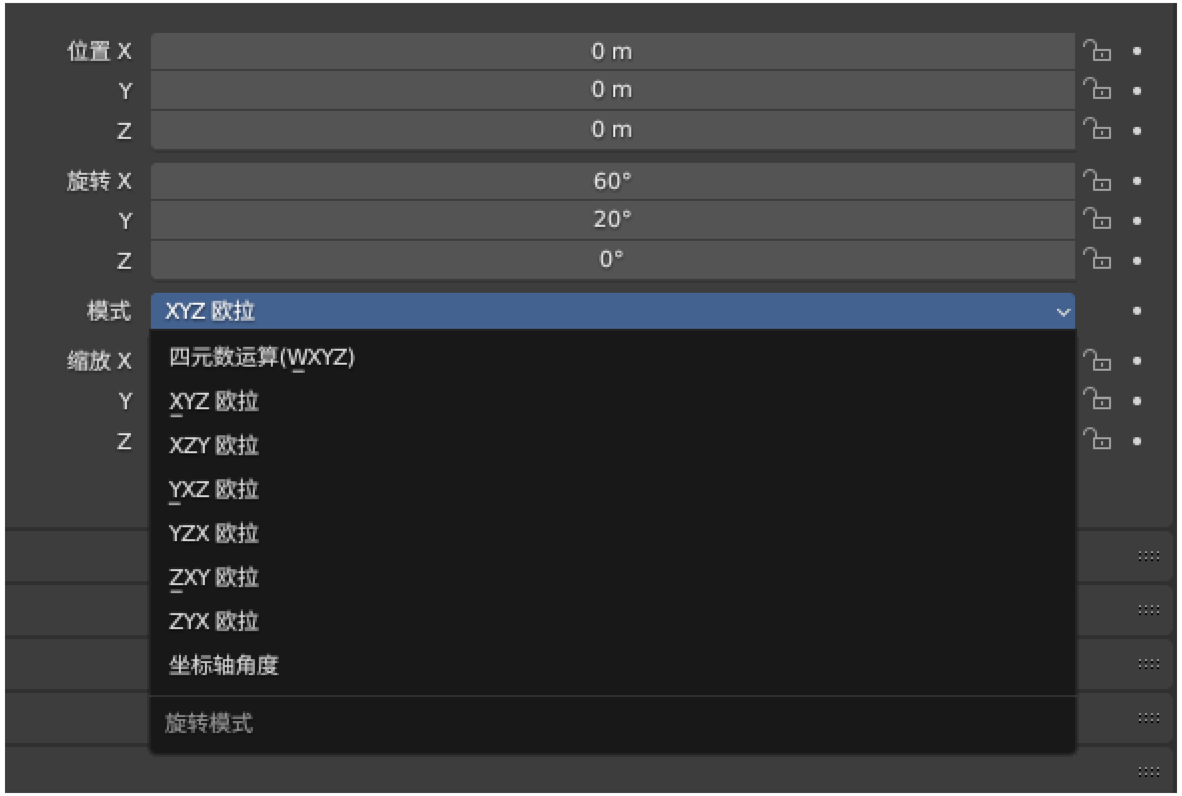
Pitch
Yaw



欧拉角 (Euler Angle)

Blender

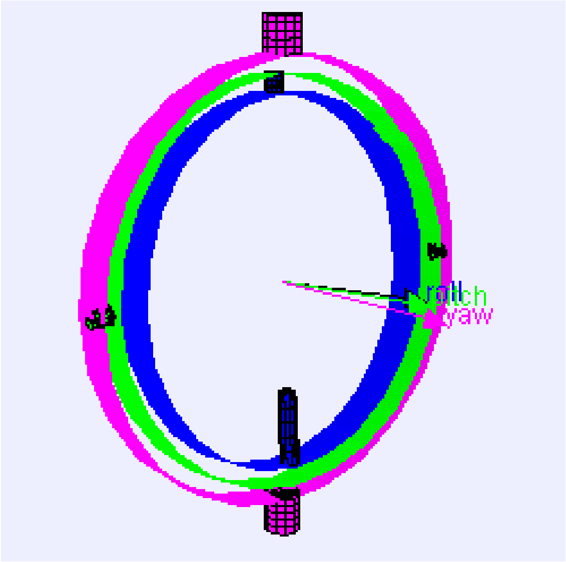


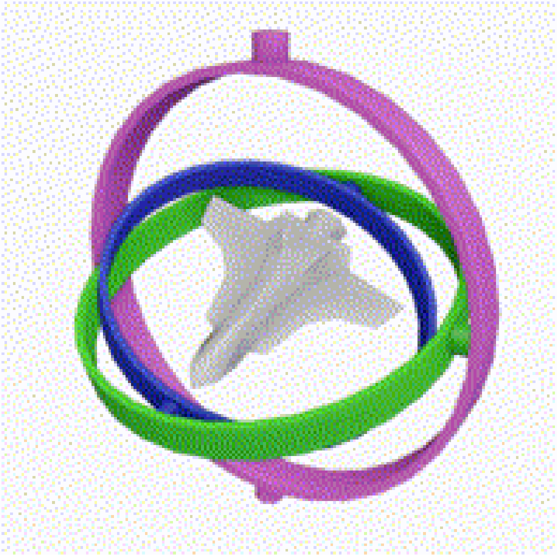


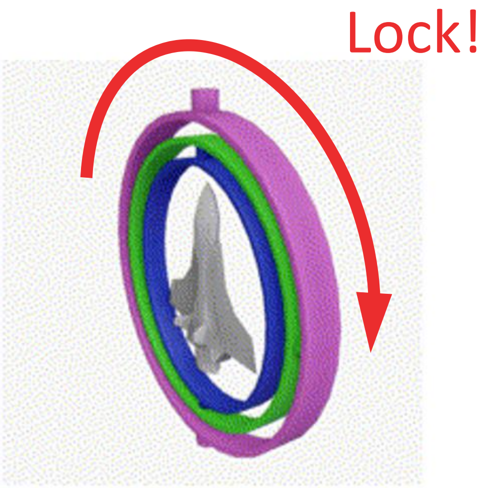
问题：万向锁 (Gimbal Lock)

当航向 (yaw)轴与俯仰 (pitch)轴平行时，改变横滚角(row)的效果与改变航向角相同在这个特殊的点上，欧拉角只能调节2个自由度的旋转，**有1个自由度被锁住了**

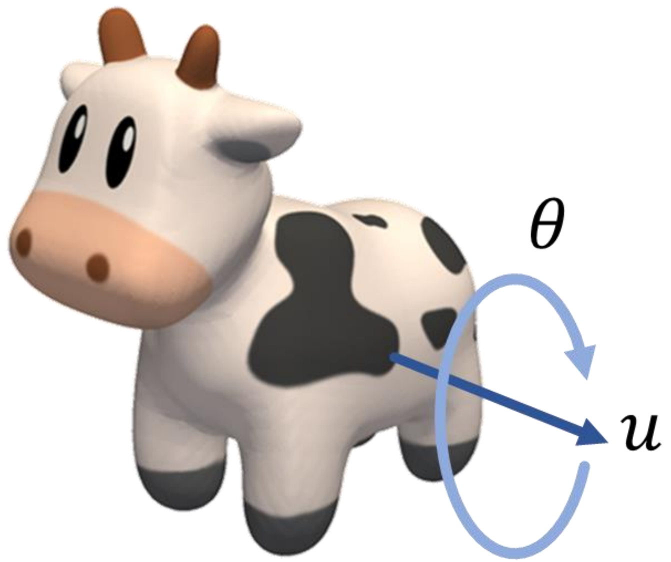
→俯仰角等于$\pi/2$是一个奇点 (singularity)

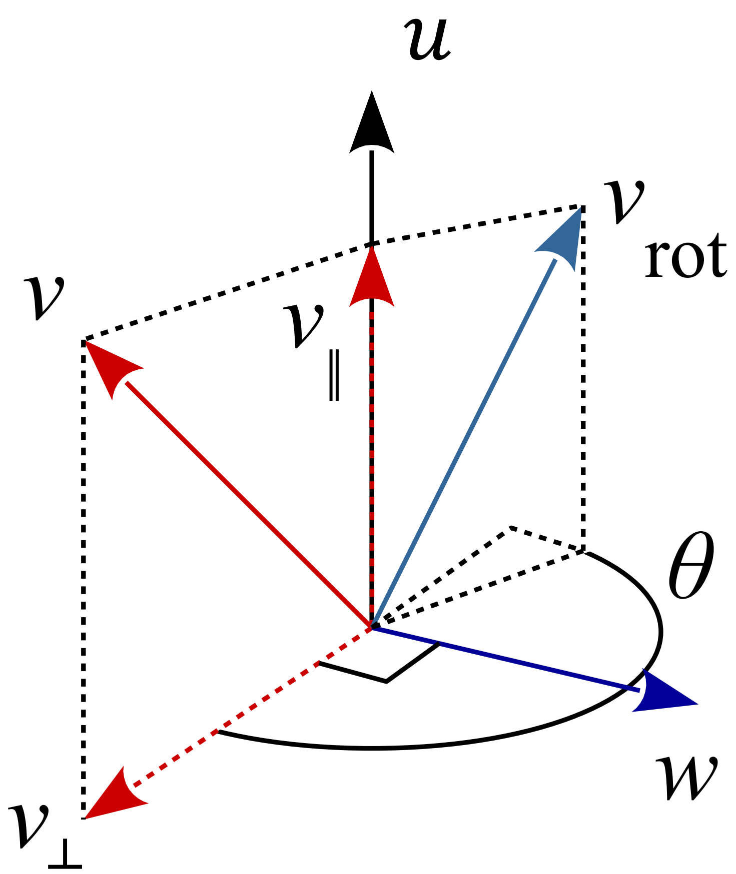






轴角表示 (Axis Angle)



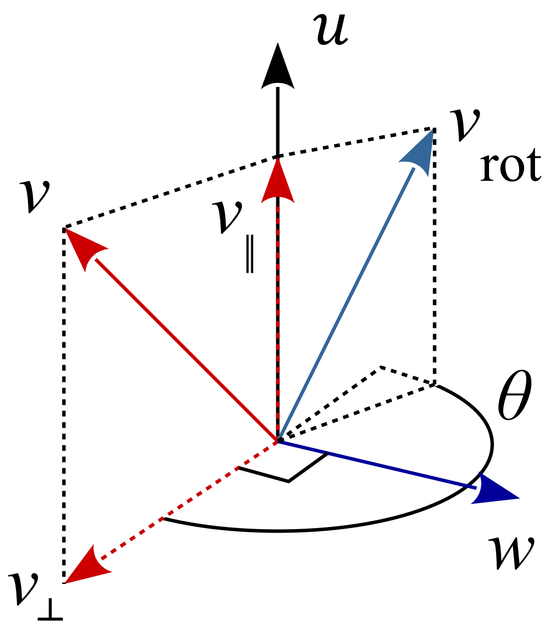


旋转轴：$u \in \mathbb{R}^3, \|u\| = 1$，旋转角度：$\theta \in \mathbb{R}$

旋转向量：$\boldsymbol{\theta} = \theta u \in \mathbb{R}^3$

轴角表示 (Axis Angle)

$$\begin{cases} v_{rot} = v_{\parallel} + v_{\perp rot} = (v \cdot u)u + v_{\perp rot} \\ v_{\perp rot} = \cos\theta\, v_{\perp} + \sin\theta\, u \times v_{\perp} \\ v_{\perp} = v - v_{\parallel} = v - (v \cdot u)u \\ u \times (u \times v) = (u \cdot v)u - v \end{cases}$$



$$v_{rot} = v + (1 - \cos\theta)u \times (u \times v) + \sin\theta\, u \times v$$

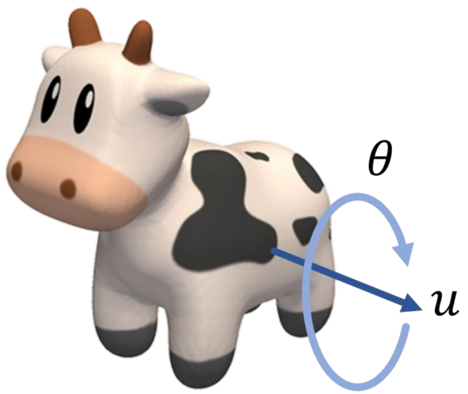
$$v_{rot} = \cos\theta\, v + (1 - \cos\theta)(v \cdot u)u + \sin\theta\, u \times v$$

罗德里格旋转公式

Rodrigues' rotation formula

$$\mathrm{R}(u, \theta) = \mathrm{I} + (1 - \cos\theta)[u]^2 + \sin\theta\, [u]$$

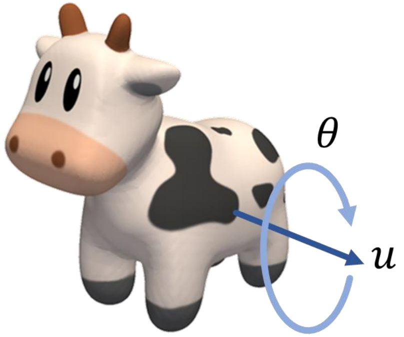
$$[u] = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}$$



轴角表示 (Axis Angle)

$$[u] = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}$$

$$\mathrm{R}(u,\theta) = \mathrm{I} + (1-\cos\theta)[u]^2 + \sin\theta\,[u]$$



$$\mathrm{R}(u,\theta)$$

$$tr(\mathrm{R}) = tr(\mathrm{I}) + (1-\cos\theta)tr([u]^2) = 3 - 2(1-\cos\theta) = 1 + 2\cos\theta$$
$$[u]^2 v = u \times (u \times v) = (u \cdot v)u - (u \cdot u)v = \left(uu^{\mathrm{T}} - \mathrm{I}\right)v$$
$$[u]^2 = uu^{\mathrm{T}} - \mathrm{I}$$
$$\mathrm{R} - \mathrm{R}^{\mathrm{T}} = 2\sin\theta\,[u]$$

$$u = \frac{1}{2\sin\theta}\begin{pmatrix} R_{21} - R_{12} \\ R_{02} - R_{20} \\ R_{10} - R_{01} \end{pmatrix}, \theta = \arccos\left(\frac{tr(\mathrm{R}) - 1}{2}\right)$$

指数形式

$$R(u, \theta) = \exp(\theta[u]) = \exp([\boldsymbol{\theta}])$$

$$[u] = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}$$

- $[u]^2 = uu^T - I, [u]^3 = -[u], [u]^4 = -[u]^2, [u]^5 = [u], \ldots$
- $\exp(\theta[u]) = I + \theta[u] + \frac{1}{2!}\theta^2[u]^2 + \frac{1}{3!}\theta^3[u]^3 + \cdots$

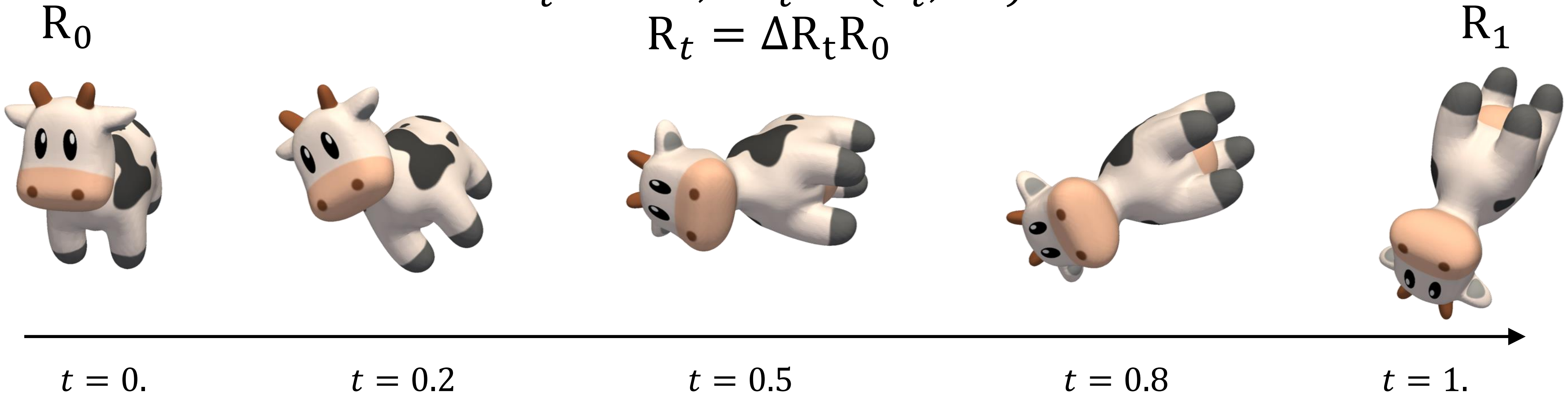
$$= I + \left(\theta - \frac{1}{3!}\theta^3 + \frac{1}{5!}\theta^5 - \cdots\right)[u] + \left(\frac{1}{2!}\theta^2 - \frac{1}{4!}\theta^4 + \cdots\right)[u]^2$$

$$= I + \sin\theta\,[u] + (1 - \cos\theta)[u]^2$$ ← 罗德里格旋转公式

这当然不是巧合，描述的是从李代数 (Lie Algebra) 到李群的指数映射 (Exponential Map)

轴角表示插值

$$\Delta \mathrm{R} = \mathrm{R}_1 \mathrm{R}_0^{\mathrm{T}}$$
$$(\Delta\theta, \Delta u) \leftarrow \Delta \mathrm{R}$$
$$\theta_t = t\Delta\theta, \Delta \mathrm{R}_t \leftarrow (\theta_t, \Delta u)$$
$$\mathrm{R}_t = \Delta \mathrm{R}_{\mathrm{t}} \mathrm{R}_0$$



旋转向量插值

$\boldsymbol{\theta}_0 = \frac{3\pi}{2}(1,0,0)$

旋转向量插值

$\boldsymbol{\theta}_t = (1-t)\boldsymbol{\theta}_0 + t\boldsymbol{\theta}_1$

✕

相对旋转插值

$(\Delta\theta, \Delta u) \leftarrow \Delta \mathrm{R}$

$\theta_t = t\Delta\theta, \Delta \mathrm{R}_t \leftarrow (\theta_t, \Delta u)$

✓

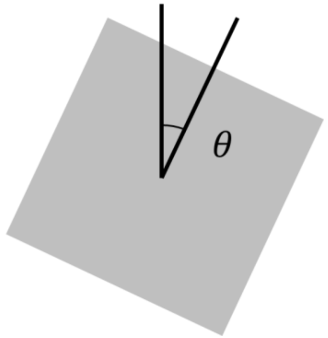
$\boldsymbol{\theta}_1 = \frac{3\pi}{2}(0,1,0)$

四元数 (Quaternion)

William Rowan Hamilton

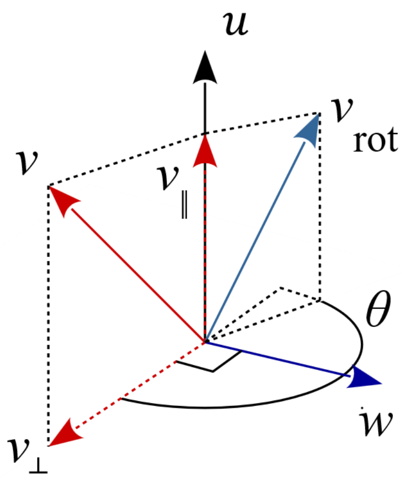
二维旋转

矩阵表示	复数表示
$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$	$z' = (\cos\theta + i\sin\theta)z$
$c = \begin{pmatrix} a \\ b \end{pmatrix}, z = \begin{pmatrix} x \\ y \end{pmatrix}$ $[c]z = \begin{pmatrix} a & -b \\ b & a \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax - by \\ bx + ay \end{pmatrix}$	$c = a + ib, z = x + iy$ $cz = (ax - by) + i(bx + ay)$
$[i] = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, [i]^2 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix},$ $[i]^3 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, [i]^4 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \ldots$	$i, i^2 = -1, i^3 = -i, i^4 = 1, \ldots$
$\exp(\theta[i]) = \sum \frac{1}{k!} \theta^k [i]^k$ $= \cos\theta \, \mathrm{I} + \sin\theta \, [i]$ $= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$	$e^{i\theta} = \cos\theta + i\sin\theta$



三维旋转

矩阵表示



$$R(u,\theta) = I + (1-\cos\theta)[u]^2 + \sin\theta\,[u]$$
$$R(u,\theta) = \exp(\theta[u]) = \exp([\boldsymbol{\theta}])$$

四元数表示

×	1	i	j	k
1	1	i	j	k
i	i	−1	k	$-j$
j	j	$-k$	−1	i
k	k	j	$-i$	−1

$$q = a + bi + cj + dk$$
$$i^2 = j^2 = k^2 = ijk = -1$$

四元数乘法

$$q_1q_2 = (a + bi + cj + dk)(e + fi + gj + \mathrm{h}k)$$
$$= (ae - bf - cg - dh) + (be + af - dg + ch)i$$
$$+ (ce + df + ag - bh)j + (de - cg + bg + ah)k$$

$$q_1q_2 = \begin{pmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{pmatrix} \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix}$$

$\times$	1	i	j	k
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1

Graßmann 积

- $q_1 = (a, \boldsymbol{v}), \boldsymbol{v} = b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$
- $q_2 = (e, \boldsymbol{u}), \boldsymbol{u} = f\boldsymbol{i} + g\boldsymbol{j} + \mathrm{h}\boldsymbol{k}$

$$q_1 q_2 \neq q_2 q_1$$

$$q_1 q_2 = (ae - \boldsymbol{v} \cdot \boldsymbol{u}, a\boldsymbol{u} + e\boldsymbol{v} + \boldsymbol{v} \times \boldsymbol{u})$$

- 如果 $a = e = 0$, 又称纯四元数

$$q_1 q_2 = (-\boldsymbol{v} \cdot \boldsymbol{u}, \boldsymbol{v} \times \boldsymbol{u})$$

$$q_1 q_2 = \begin{pmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{pmatrix} \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} a & -\boldsymbol{v}^{\mathrm{T}} \\ \boldsymbol{v} & a\mathrm{I} + [\boldsymbol{v}] \end{pmatrix} \begin{pmatrix} e \\ \boldsymbol{u} \end{pmatrix} \qquad [u] = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}$$

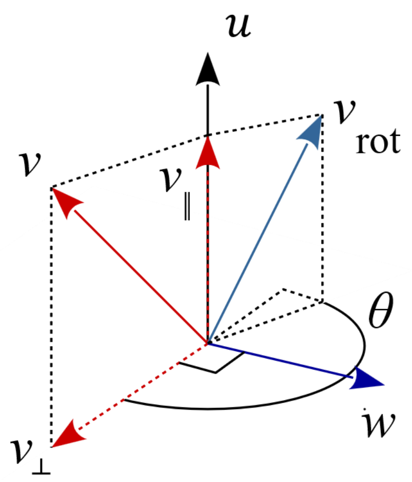
四元数的逆与共轭

	分量表示	向量表示	矩阵表示
	$q = a + bi + cj + dk$	$q = (a, \boldsymbol{v})$	$q = \begin{pmatrix} a & -\boldsymbol{v}^{\mathrm{T}} \\ \boldsymbol{v} & a\mathrm{I} + [\boldsymbol{v}] \end{pmatrix}$
共轭 (Conjugate)	$q^* = a - bi - cj - dk$	$q^* = (a, -\boldsymbol{v})$	$q^* = \begin{pmatrix} a & \boldsymbol{v}^{\mathrm{T}} \\ -\boldsymbol{v} & a\mathrm{I} - [\boldsymbol{v}] \end{pmatrix}$
	$qq^* = a^2 + b^2 + c^2 + d^2 = \|q\|^2$	$qq^* = (a^2 + \boldsymbol{v} \cdot \boldsymbol{v}, 0)$	$qq^* = \begin{pmatrix} a^2 + \boldsymbol{v}^{\mathrm{T}}\boldsymbol{v} & 0 \\ 0 & (a^2 + \boldsymbol{v}^{\mathrm{T}}\boldsymbol{v})\mathrm{I} \end{pmatrix}$

$$q^{-1} = \frac{1}{\|q\|^2} q^*$$

对于单位四元数 (Unit Quaternion) $\|q\| = 1$，有$q^{-1} = q^*$

四元数与旋转



纯虚四元数

- $\boldsymbol{v} \in \mathbb{R}^3 \rightarrow v = (0, \boldsymbol{v}) \in \mathbb{H}, q = (s, \boldsymbol{t}) \in \mathbb{H}$
- $qv = (-\boldsymbol{t} \cdot \boldsymbol{v}, s\boldsymbol{v} + \boldsymbol{t} \times \boldsymbol{v})$

$$\boldsymbol{v}_{rot} = \cos\theta\, \boldsymbol{v} + (1 - \cos\theta)(\boldsymbol{v} \cdot \boldsymbol{u})\boldsymbol{u} + \sin\theta\, \boldsymbol{u} \times \boldsymbol{v}$$

- $q = (\cos\theta, \sin\theta\, \boldsymbol{u})$
- $qv = (-\sin\theta\, \boldsymbol{u} \cdot \boldsymbol{v}, \cos\theta\, \boldsymbol{v} + \sin\theta\, \boldsymbol{u} \times \boldsymbol{v})$

看起来接近旋转变换，但是少了一项，并且有非零实部

四元数与旋转

- $q=(\cos\theta, \sin\theta\,\boldsymbol{u})$是单位四元数，$q^{-1}=q^{*}=(\cos\theta, -\sin\theta\,\boldsymbol{u})$

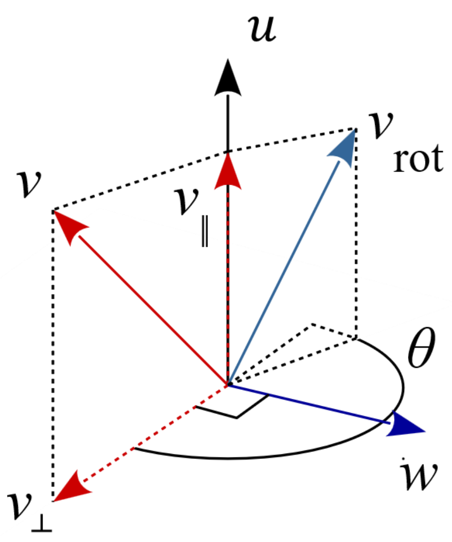
$$
\begin{aligned}
qvq^{-1} &= (c, s\boldsymbol{u})(0, \boldsymbol{v})(c, -s\boldsymbol{u}) \\
&= (c, s\boldsymbol{u})(s\boldsymbol{v}\cdot\boldsymbol{u}, c\boldsymbol{v}-s\boldsymbol{v}\times\boldsymbol{u}) \\
&= (cs\boldsymbol{v}\cdot\boldsymbol{u}-s\boldsymbol{u}\cdot(c\boldsymbol{v}-s\boldsymbol{v}\times\boldsymbol{u}), \\
&\quad c^{2}\boldsymbol{v}-cs\boldsymbol{v}\times\boldsymbol{u}+s^{2}(\boldsymbol{v}\cdot\boldsymbol{u})\boldsymbol{u}+s\boldsymbol{u}\times c\boldsymbol{v}-s^{2}\boldsymbol{u}\times(\boldsymbol{v}\times\boldsymbol{u})) \\
&= (0, \underbrace{(c^{2}-s^{2})}_{\cos 2\theta}\boldsymbol{v}+\underbrace{2sc}_{\sin 2\theta}\boldsymbol{u}\times\boldsymbol{v}+\underbrace{2s^{2}}_{1-\cos 2\theta}(\boldsymbol{u}\cdot\boldsymbol{v})\boldsymbol{u})
\end{aligned}
$$

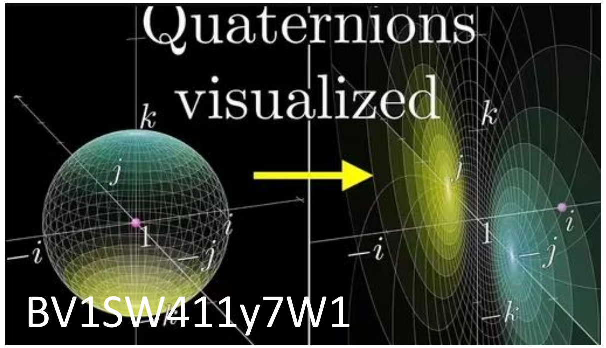
$$
\boldsymbol{v}_{rot}=\cos\theta\,\boldsymbol{v}+\sin\theta\,\boldsymbol{u}\times\boldsymbol{v}+(1-\cos\theta)(\boldsymbol{v}\cdot\boldsymbol{u})\boldsymbol{u}
$$

四元数与旋转

$$v = (0, \boldsymbol{v}), q = (\cos\frac{1}{2}\theta, \sin\frac{1}{2}\theta\, \boldsymbol{u})$$

$$v_{rot} = qvq^*$$





https://www.bilibili.com/video/BV1SW411y7W1

https://krasjet.github.io/quaternion

为什么需要4自由度的四元数表示旋转?

为什么是两次半角相乘?

…

四元数旋转性质

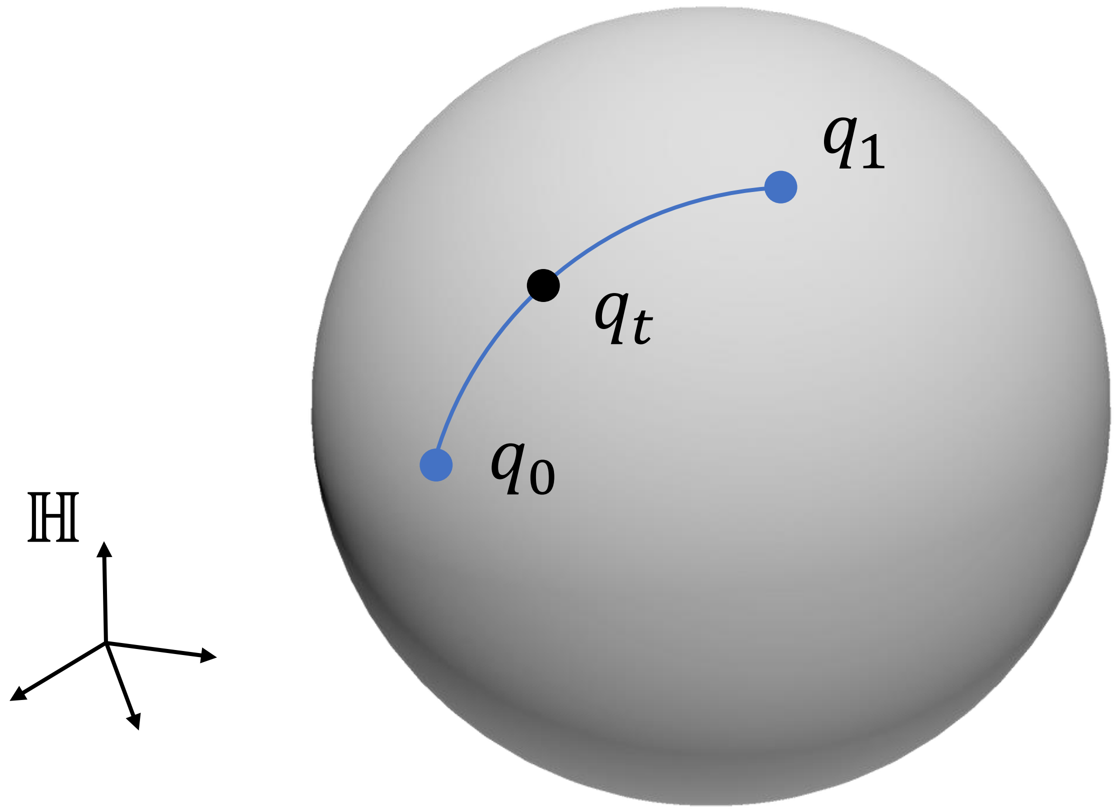
- 多次旋转 $q = q_n q_{n-1} \cdots q_1$
  - $v_1 = q_1 v q_1^*, v_2 = q_2 v_1 q_2^* = (q_2 q_1) v (q_2 q_1)^*, \cdots$
- q与$-q$表示的是相同的旋转
  - $(-q)v(-q)^* = qvq^*$
- 指数形式 $q = \left(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\boldsymbol{u}\right) = \exp\left(\frac{\theta}{2}u\right), u = (0, \boldsymbol{u})$
  - $v_{rot} = \exp\left(\frac{\theta}{2}u\right) v \exp\left(-\frac{\theta}{2}u\right)$

$$\mathrm{R}(u, \theta) = \exp(\theta[u]) = \exp([\boldsymbol{\theta}])$$

$$[u] = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}$$

×	1	i	j	k
1	1	i	j	k
i	i	−1	k	$-j$
j	j	$-k$	−1	i
k	k	j	$-i$	−1

四元数旋转流形

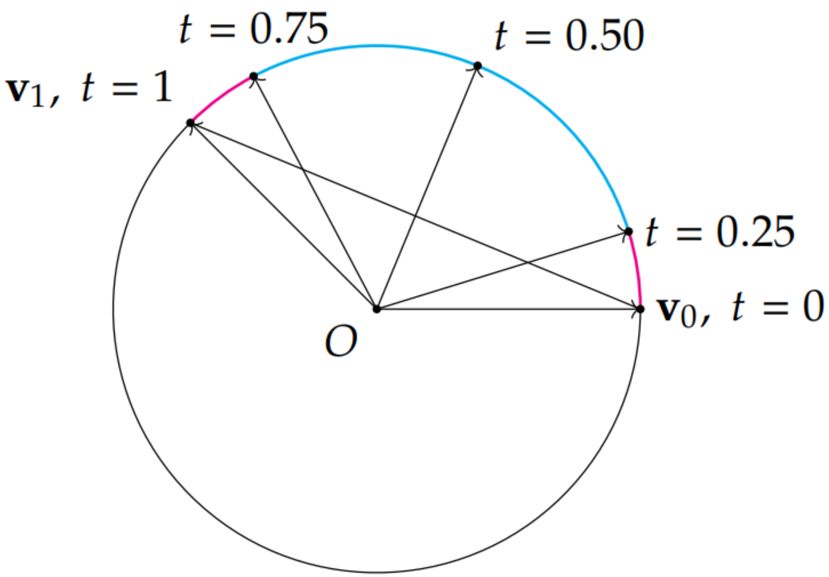


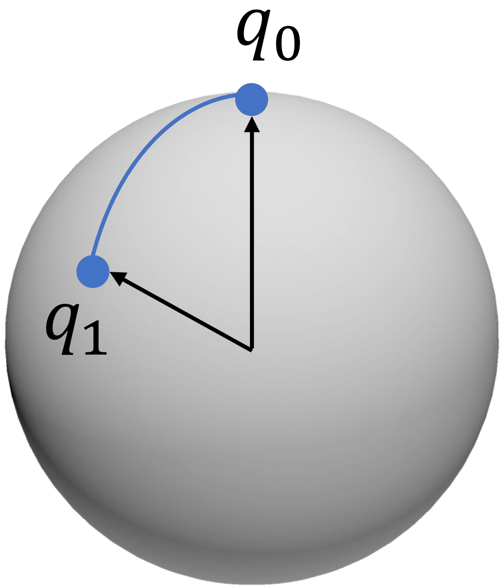
$\|q\| = 1$

四元数表示的旋转流形：四维空间中的球体

四元数插值：Nlerp

$$q_t = \mathrm{Normalize}((1-t)q_0 + tq_1)$$

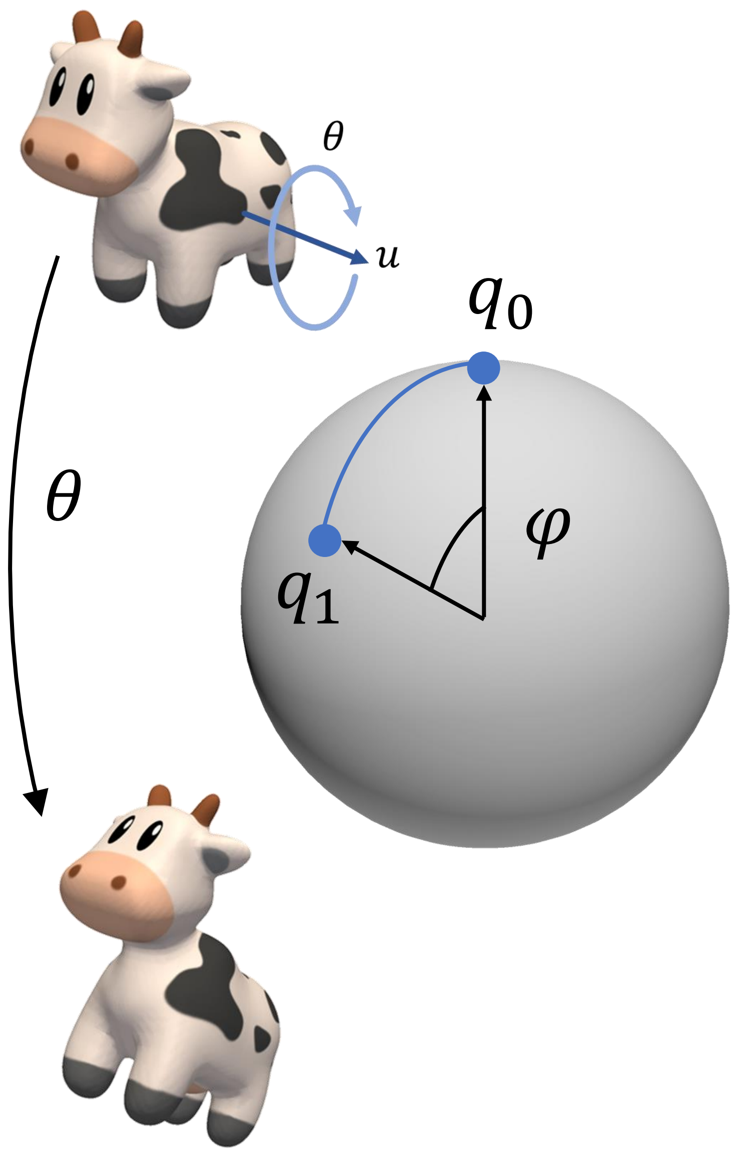




四元数夹角与旋转角度

- $q_0 = (1, \mathbf{0}), q_1 = \left(\cos\frac{1}{2}\theta, \sin\frac{1}{2}\theta\, \boldsymbol{u}\right)$
- $\cos\varphi = \frac{<q_0, q_1>}{\|q_0\|\|q_1\|} = \cos\frac{1}{2}\theta$

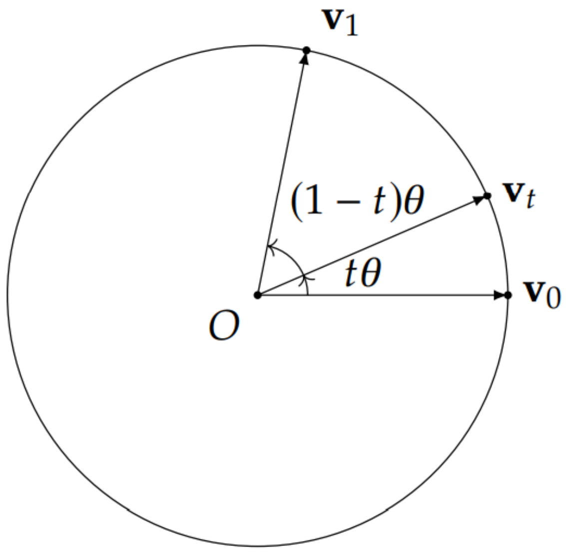
$$\varphi = \frac{1}{2}\theta$$



四元数插值：Slerp

$$q_t = \frac{\sin((1-t)\theta)}{\sin\theta} q_0 + \frac{\sin(t\theta)}{\sin\theta} q_1$$

$$\theta = \arccos(\langle q_0, q_1 \rangle)$$



Nlerp

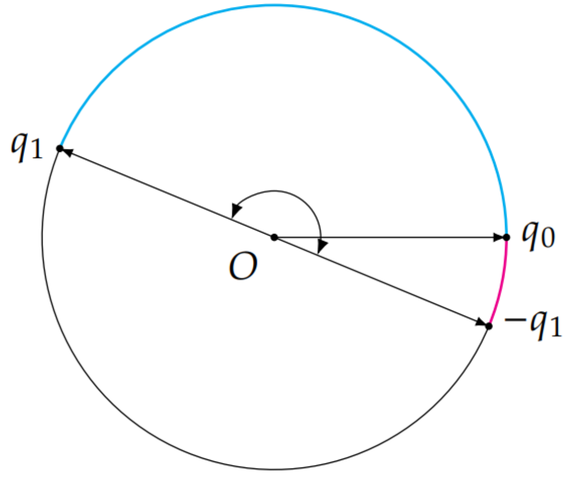
Slerp

最小弧

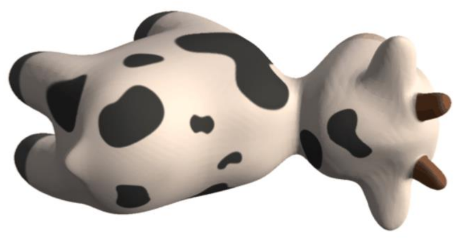
长弧

短弧

q_0

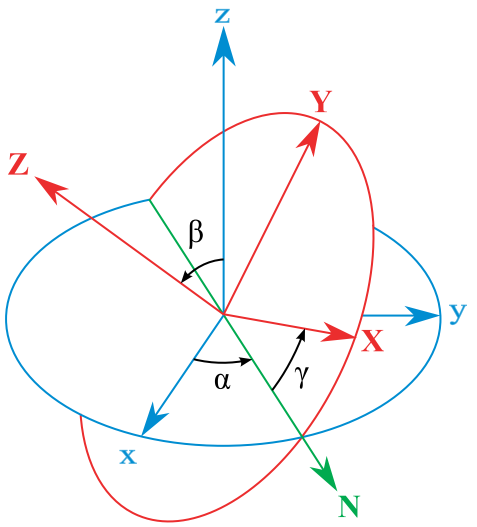


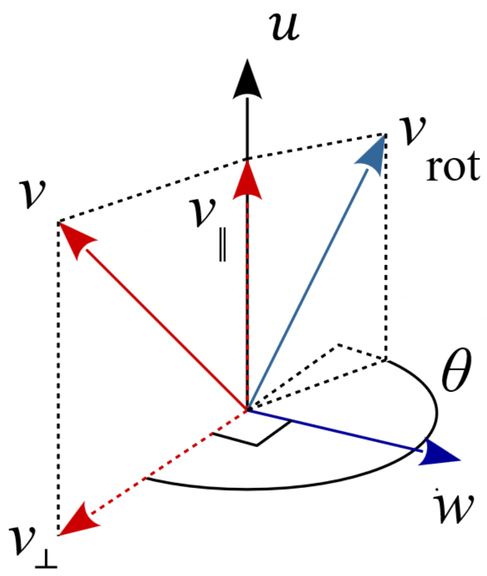
q_1



总结

- 欧拉角：操作直观，注意规范，有万向锁，不好插值
- 轴角：旋转定义，需要矩阵运算，相对旋转插值
- 四元数：计算效率高，方便组合，方便插值，方便采样，但是理解不直观

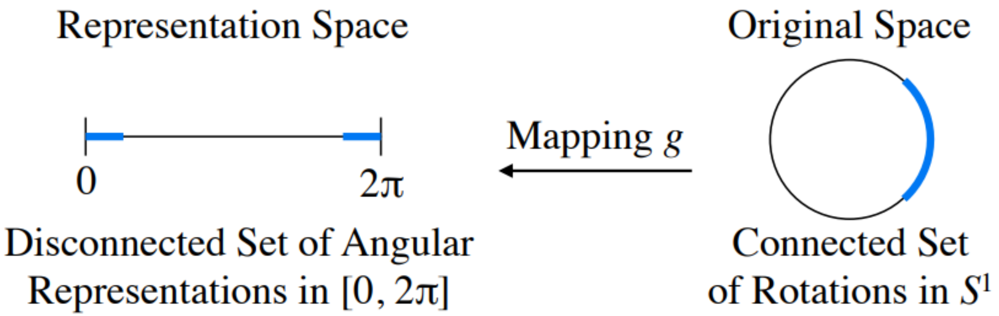


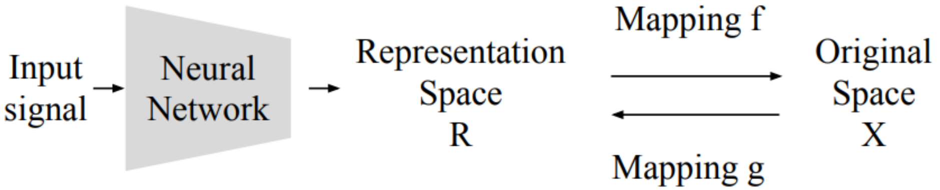


×	1	i	j	k
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1

适合神经网络的旋转表示

- 多义性
  - 欧拉角：相差2π的角度表示的是相同的旋转
  - 轴角：(v,θ)表示的旋转与$(-v,\pi-\theta)$相同
  - 四元数：q与$-q$结果相同
- 不连续
  - 假设我们规定角度范围在$[0,2\pi)$，那么在$\theta=0$ 附近的相似旋转被映射到了角度范围的两个端点附近



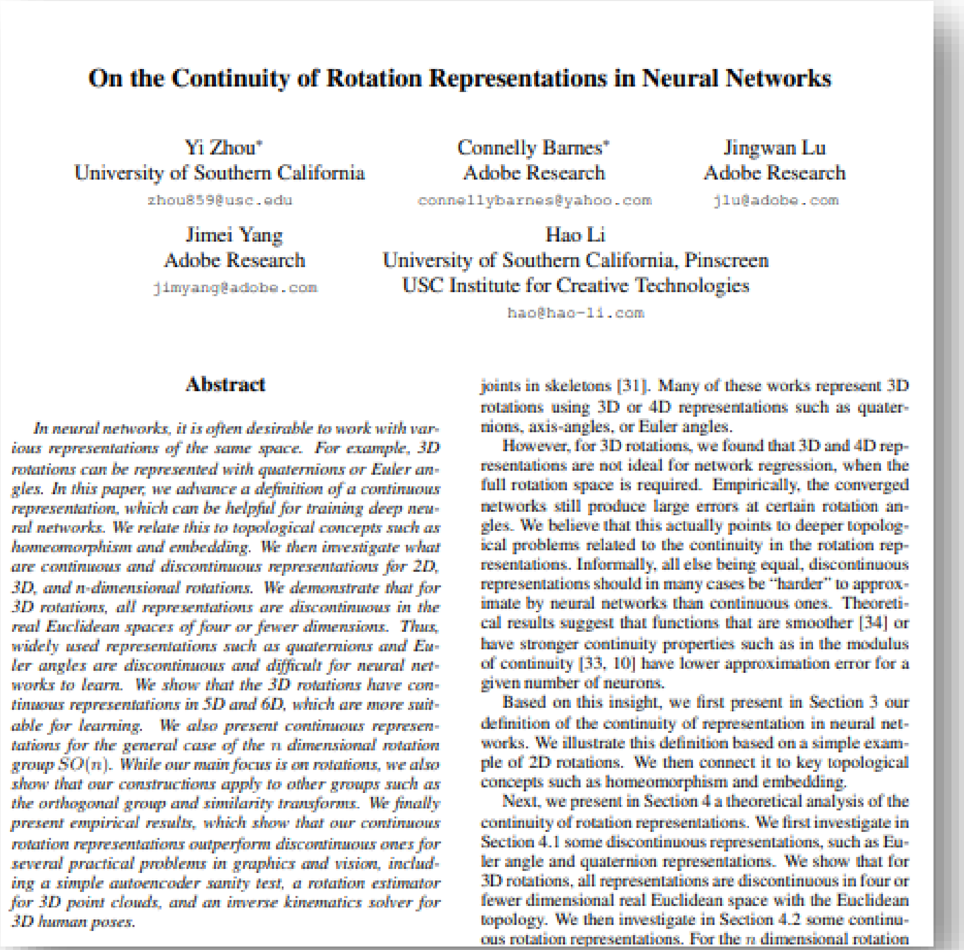


6D旋转表达

- 网络输出两个3维向量a_1, a_2
- 对a_1, a_2进行Gram-Schmidt正交化：

$$
\begin{cases}
b_1 = N(a_1) \\
b_2 = N(a_2 - (b_1 \cdot a_1) b_1) \\
b_3 = b_1 \times b_2
\end{cases}
$$

- 旋转矩阵 $R = (b_1, b_2, b_3)$



On the Continuity of Rotation Representations in Neural Networks

Yi Zhou*
University of Southern California
zhou859@usc.edu

Connelly Barnes*
Adobe Research
connellybarnes@yahoo.com

Jingwan Lu
Adobe Research
jlu@adobe.com

Jimei Yang
Adobe Research
jimyang@adobe.com

Hao Li
University of Southern California, Pinscreen
USC Institute for Creative Technologies
hao@hao-li.com

Abstract

In neural networks, it is often desirable to work with various representations of the same space. For example, 3D rotations can be represented with quaternions or Euler angles. In this paper, we advance a definition of a continuous representation, which can be helpful for training deep neural networks. We relate this to topological concepts such as homeomorphism and embedding. We then investigate what are continuous and discontinuous representations for 2D, 3D, and n-dimensional rotations. We demonstrate that for 3D rotations, all representations are discontinuous in the real Euclidean spaces of four or fewer dimensions. Thus, widely used representations such as quaternions and Euler angles are discontinuous and difficult for neural networks to learn. We show that the 3D rotations have continuous representations in 5D and 6D, which are more suitable for learning. We also present continuous representations for the general case of the n dimensional rotation group $SO(n)$. While our main focus is on rotations, we also show that our constructions apply to other groups such as the orthogonal group and similarity transforms. We finally present empirical results, which show that our continuous rotation representations outperform discontinuous ones for several practical problems in graphics and vision, including a simple autoencoder sanity test, a rotation estimator for 3D point clouds, and an inverse kinematics solver for 3D human poses.

joints in skeletons [31]. Many of these works represent 3D rotations using 3D or 4D representations such as quaternions, axis-angles, or Euler angles.

However, for 3D rotations, we found that 3D and 4D representations are not ideal for network regression, when the full rotation space is required. Empirically, the converged networks still produce large errors at certain rotation angles. We believe that this actually points to deeper topological problems related to the continuity in the rotation representations. Informally, all else being equal, discontinuous representations should in many cases be "harder" to approximate by neural networks than continuous ones. Theoretical results suggest that functions that are smoother [34] or have stronger continuity properties such as in the modulus of continuity [33, 10] have lower approximation error for a given number of neurons.

Based on this insight, we first present in Section 3 our definition of the continuity of representation in neural networks. We illustrate this definition based on a simple example of 2D rotations. We then connect it to key topological concepts such as homeomorphism and embedding.

Next, we present in Section 4 a theoretical analysis of the continuity of rotation representations. We first investigate in Section 4.1 some discontinuous representations, such as Euler angle and quaternion representations. We show that for 3D rotations, all representations are discontinuous in four or fewer dimensional real Euclidean space with the Euclidean topology. We then investigate in Section 4.2 some continuous rotation representations. For the n dimensional rotation

根据具体情况选择

3D Gaussian Splatting for Real-Time Radiance Field Rendering

BERNHARD KERBL*, Inria, Université Côte d'Azur, France
GEORGIOS KOPANAS*, Inria, Université Côte d'Azur, France
THOMAS LEIMKÜHLER, Max-Planck-Institut für Informatik, Germany
GEORGE DRETTAKIS, Inria, Université Côte d'Azur, France

Fig. 1. Our method achieves real-time rendering of radiance fields with quality that equals the previous method with the best quality [Barron et al. 2022], while only requiring optimization times competitive with the fastest previous methods [Fridovich-Keil and Yu et al. 2022; Müller et al. 2022]. Key to this performance is a novel 3D Gaussian scene representation coupled with a real-time differentiable renderer, which offers significant speedup to both scene optimization and novel view synthesis. Note that for comparable training times to InstantNGP [Müller et al. 2022], we achieve similar quality to theirs; while this is the maximum quality they reach, by training for 51min we achieve state-of-the-art quality, even slightly better than Mip-NeRF360 [Barron et al. 2022].

Radiance Field methods have recently revolutionized novel-view synthesis of scenes captured with multiple photos or videos. However, achieving high visual quality still requires neural networks that are costly to train and render, while recent faster methods inevitably trade off speed for quality. For unbounded and complete scenes (rather than isolated objects) and 1080p resolution rendering, no current method can achieve real-time display rates. We introduce three key elements that allow us to achieve state-of-the-art visual quality while maintaining competitive training times and importantly allow high-quality real-time ($\geq$ 30 fps) novel-view synthesis at 1080p resolution. First, starting from sparse points produced during camera calibration, we represent the scene with 3D Gaussians that preserve desirable properties of continuous volumetric radiance fields for scene optimization while avoiding unnecessary computation in empty space; Second, we perform interleaved optimization/density control of the 3D Gaussians, notably optimizing anisotropic covariance to achieve an accurate representation of the scene; Third, we develop a fast visibility-aware rendering algorithm that supports anisotropic splatting and both accelerates training and allows real-time rendering. We demonstrate state-of-the-art visual quality and real-time rendering on several established datasets.

CCS Concepts: • **Computing methodologies → Rendering**; **Point-based models**; **Rasterization**; **Machine learning approaches**.

*Both authors contributed equally to the paper.

Authors' addresses: Bernhard Kerbl, bernhard.kerbl@inria.fr, Inria, Université Côte d'Azur, France; Georgios Kopanas, georgios.kopanas@inria.fr, Inria, Université Côte d'Azur, France; Thomas Leimkühler, thomas.leimkuehler@mpi-inf.mpg.de, Max-Planck-Institut für Informatik, Germany; George Drettakis, george.drettakis@inria.fr, Inria, Université Côte d'Azur, France.

0730-0301/2018/0-ART0 $15.00
https://doi.org/XXXXXXX.XXXXXXX

Additional Key Words and Phrases: novel view synthesis, radiance fields, 3D gaussians, real-time rendering

ACM Reference Format:
Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2018. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 0, 0, Article 0 (2018), 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Meshes and points are the most common 3D scene representations because they are explicit and are a good fit for fast GPU/CUDA-based rasterization. In contrast, recent Neural Radiance Field (NeRF) methods build on continuous scene representations, typically optimizing a Multi-Layer Perceptron (MLP) using volumetric ray-marching for novel-view synthesis of captured scenes. Similarly, the most efficient radiance field solutions to date build on continuous representations by interpolating values stored in, e.g., voxel [Fridovich-Keil and Yu et al. 2022] or hash [Müller et al. 2022] grids or points [Xu et al. 2022]. While the continuous nature of these methods helps optimization, the stochastic sampling required for rendering is costly and can result in noise. We introduce a new approach that combines the best of both worlds: our 3D Gaussian representation allows optimization with state-of-the-art (SOTA) visual quality and competitive training times, while our tile-based splatting solution ensures real-time rendering at SOTA quality for 1080p resolution on several previously published datasets [Barron et al. 2022; Hedman et al. 2018; Knapitsch et al. 2017] (see Fig. 1).

Our goal is to allow real-time rendering for scenes captured with multiple photos, and create the representations with optimization times as fast as the most efficient previous methods for typical real scenes. Recent methods achieve fast training [Fridovich-Keil

ACM Trans. Graph., Vol. 42, No. 4, Article . Publication date: August 2023.

四元数

An Analysis of SVD for Deep Rotation Estimation

Jake Levinson[1] Carlos Esteves[2] Kefan Chen[3] Noah Snavely[3]
Angjoo Kanazawa[3] Afshin Rostamizadeh[3] Ameesh Makadia[3]
[1]University of Washington [2]University of Pennsylvania [3]Google Research

Abstract

Symmetric orthogonalization via SVD, and closely related procedures, are well-known techniques for projecting matrices onto $O(n)$ or $SO(n)$. These tools have long been used for applications in computer vision, for example optimal 3D alignment problems solved by orthogonal Procrustes, rotation averaging, or Essential matrix decomposition. Despite its utility in different settings, SVD orthogonalization as a procedure for producing rotation matrices is typically overlooked in deep learning models, where the preferences tend toward classic representations like unit quaternions, Euler angles, and axis-angle, or more recently-introduced methods. Despite the importance of 3D rotations in computer vision and robotics, a single universally effective representation is still missing. Here, we explore the viability of SVD orthogonalization for 3D rotations in neural networks. We present a theoretical analysis that shows SVD is the natural choice for projecting onto the rotation group. Our extensive quantitative analysis shows simply replacing existing representations with the SVD orthogonalization procedure obtains state of the art performance in many deep learning applications covering both supervised and unsupervised training.

1 Introduction

There are many ways to represent a 3D rotation matrix. But what is the ideal representation to predict 3D rotations in a deep learning framework? The goal of this paper is to explore this seemingly low-level but practically impactful question, as currently the answer appears to be ambiguous.

In this paper we present a systematic study on estimating rotations in neural networks. We identify that the classic technique of SVD orthogonalization, widely used in other contexts but rarely in the estimation of 3D rotations in deep networks, is ideally suited for this task with strong empirical and theoretical support.

3D rotations are important quantities appearing in countless applications across different fields of study, and are now especially ubiquitous in learning problems in 3D computer vision and robotics. The task of predicting 3D rotations is common to estimating object pose [47, 24, 29, 38, 43, 21, 39], relative camera pose [27, 32, 5], ego-motion and depth from video [48, 26], and human pose [49, 19].

A design choice common to all of these models is selecting a representation for 3D rotations. The most frequent choices are classic representations including unit quaternion, Euler angles, and axis-angle. Despite being such a well-studied problem, there is no universally effective representation or regression architecture due to performance variations across different applications.

A natural alternative to these classic representations is symmetric orthogonalization, a long-known technique which projects matrices onto the orthogonal group $O(3)$ [23, 35]. Simple variations can restrict the projections onto the special orthogonal (rotation) group $SO(3)$ [13, 18, 44]. This procedure, when executed by Singular Value Decomposition (SVD [9]), has found many applications in computer vision, for example at the core of the Procrustes problem [2, 35] for point set alignment, as

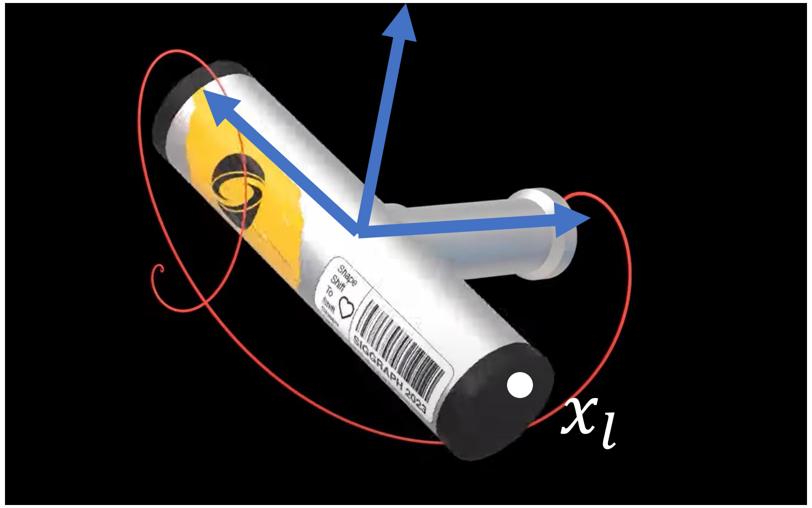
Preprint. Under review.

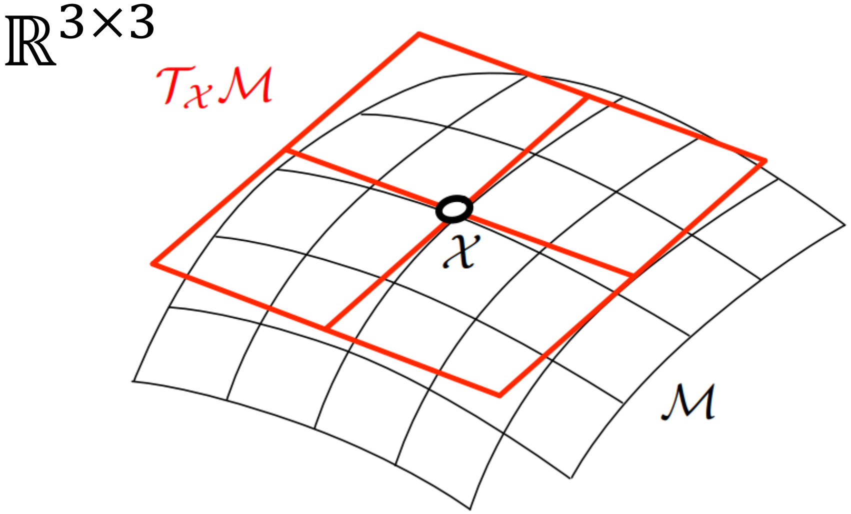
9D表达

角速度与旋转

角速度

$$[u] = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}$$





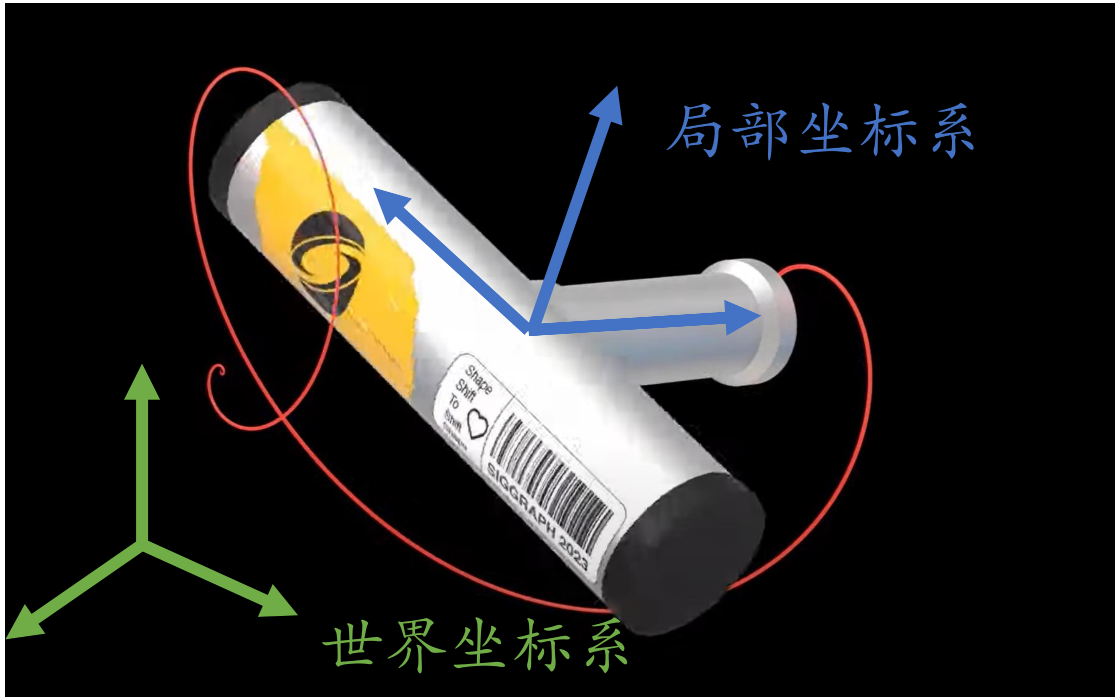
$$x_w = \mathrm{R}x_l$$
$$\dot{x_w} = \omega \times x_w = [\omega]\mathrm{R}x_l$$
$$\dot{x_w} = \dot{\mathrm{R}}x_l$$
$$\dot{\mathrm{R}} = [\omega]\mathrm{R}$$

局部坐标x_l是固定值

$$\mathrm{R}^\mathrm{T}\mathrm{R} = \mathrm{I}$$
$$\mathrm{R}^\mathrm{T}\dot{\mathrm{R}} + \dot{\mathrm{R}}\mathrm{R}^\mathrm{T} = \left(\dot{\mathrm{R}}\mathrm{R}^\mathrm{T}\right)^\mathrm{T} + \dot{\mathrm{R}}\mathrm{R}^\mathrm{T} = 0$$
$$[\omega] = \dot{\mathrm{R}}\mathrm{R}^\mathrm{T}$$
$$\dot{\mathrm{R}} = [\omega]\mathrm{R}$$

反对称矩阵都可以写为$[u]$的形式，此式可以看成角速度的定义

世界坐标角速度与局部坐标角速度



$$[\boldsymbol{\omega}]\mathbf{R}u = \left(\mathrm{R}\mathrm{R}^{\mathrm{T}}\omega\right) \times (\mathrm{R}u) = \mathrm{R}\left((\mathrm{R}^{\mathrm{T}}\omega) \times u\right) = \mathbf{R}[\mathbf{R}^{\mathbf{T}}\boldsymbol{\omega}]u$$

$$\dot{\mathrm{R}} = [\omega]\mathrm{R} = \mathrm{R}[\mathrm{R}^{\mathrm{T}}\omega] = \mathrm{R}[\omega']$$

世界坐标角速度　　　　局部坐标角速度

四元数与角速度

$$\dot{\mathrm{R}} = [\boldsymbol{\omega}]\mathrm{R} = \mathrm{R}[\mathrm{R}^{\mathrm{T}}\boldsymbol{\omega}] = \mathrm{R}[\boldsymbol{\omega}']$$

$$x_w = q x_l q^*$$

$$\dot{x}_w = \dot{q} x_l q^* + q x_l \dot{q}^*$$

$$= \dot{q} x_l q^* + (\dot{q} x_l^* q^*)^*$$

$$= \dot{q} x_l q^* - (\dot{q} x_l q^*)^*$$

$$= 2\dot{q} x_l q^*$$

$(pq)^* = q^* p^*$

纯虚四元数$t^* = -t$

$$\dot{x}_w = \omega x_w = \omega q x_l q^* = q \omega' x_l q^*$$

$\omega x_w = (0, \boldsymbol{\omega})(0, \boldsymbol{x}_w) = (0, \boldsymbol{\omega} \times \boldsymbol{x}_w)$

$$\dot{q} = \frac{1}{2}\omega q = \frac{1}{2} q \omega'$$

参考资料

- Keenan Crane: https://www.bilibili.com/video/BV1Pf4y1E7GJ
- 3Blue1Brown: https://www.bilibili.com/video/BV1SW411y7W1
- Krasjet: https://krasjet.github.io/quaternion/quaternion.pdf
- Libin Liu: https://www.bilibili.com/video/BV1GG4y1p7fF
- Shinjiro Sueda: https://github.com/sueda/redmax
- Wikipedia

谢谢