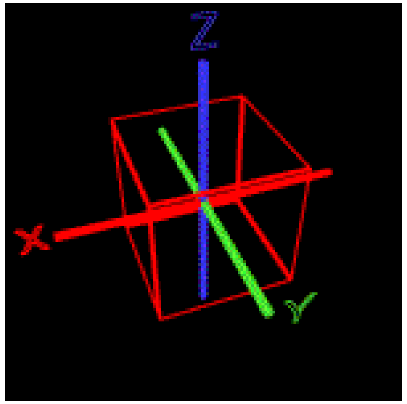
变换和旋转表示



金小刚

Email: jin@cad.zju.edu.cn

浙江大学CAD&CG国家重点实验室

紫金港校区蒙民伟楼512

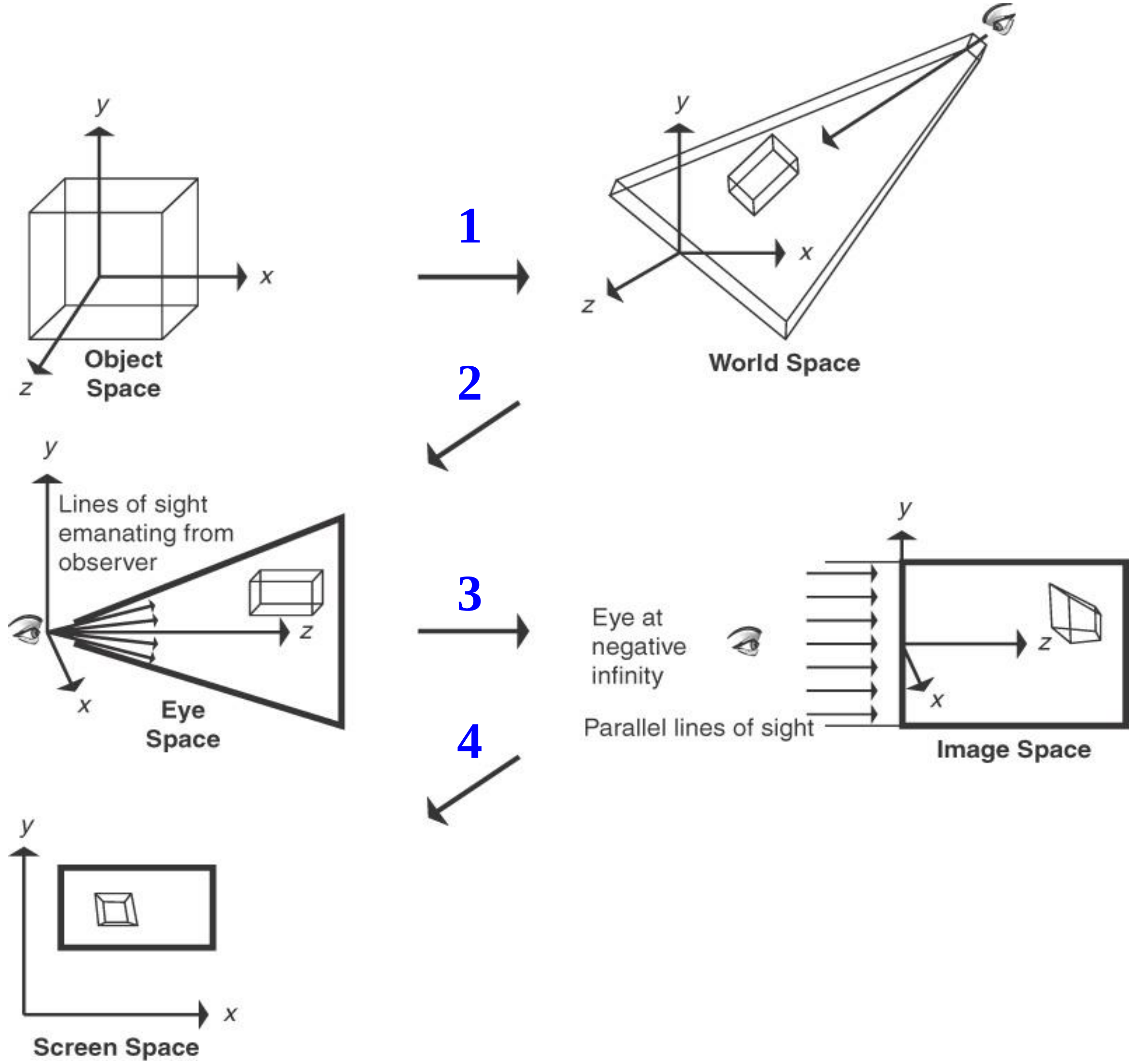
Motion Specification

- **Low level** techniques (techniques that aid the animator in precisely specifying motion)
- **High level** techniques (techniques used to describe general motion behavior)
- 平移变换、比例变换和旋转变换的运动指定大都属于Low Level
- 变换可以用来改变物体的位置、形状；对物体、**光源和摄像机**设置动画等

Technical Background

- Spaces and transformations
  - Coordinate Space: left-handed, right-handed, local coordinate system, global coordinate system
  - Viewing pipeline
    - Homogeneous coordinate, Transformation matrix, Matrix Concatenation
- Orientation representation
  - Rotation matrix
  - Fixed angle
  - Euler angle
  - Angle and Axis
  - **Quaternion**

Space Transformation in Display Pipeline



3-D Transformations

- Translate, scale, or rotate a point p to p'
  - p'=p+T
  - p'=Sp
  - pP'=Rp
- How to treat these transformations in a unified way?
  - p' = **M**p
  - All in the homogeneous coordinate
- **M** can be used for **animation**, viewing, or modeling

Homogeneous Coordinate

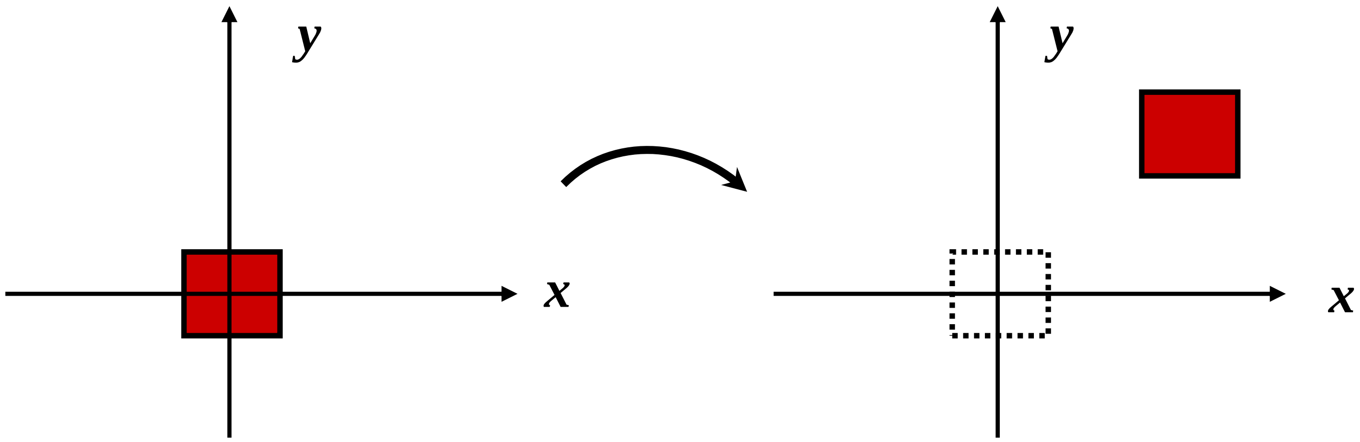
- In graphics, we use homogeneous coordinate for transformation
- 4x4 matrix can represent translation, scaling, and rotation and other transformations

$$\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right) \Leftarrow [x, y, z, w]$$

- Typically, when transforming a point in 3D space, we set $w = 1$

$$(x, y, z) \Leftarrow [x, y, z, 1]$$

Translation



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

New point in 3D space

Point in 3D space

Transformation matrix

Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation

- X axis

$$R_x(\theta) \quad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Y axis

$$R_y(\theta) \quad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Z axis

$$R_z(\theta) \quad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation

- **性质1：** 迹与旋转轴无关，都为

$$tr(R) = 1 + \cos 2\,\theta$$

- **性质2：** 所有旋转矩阵为正交阵，多个旋转矩阵相乘仍为正交阵，且

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

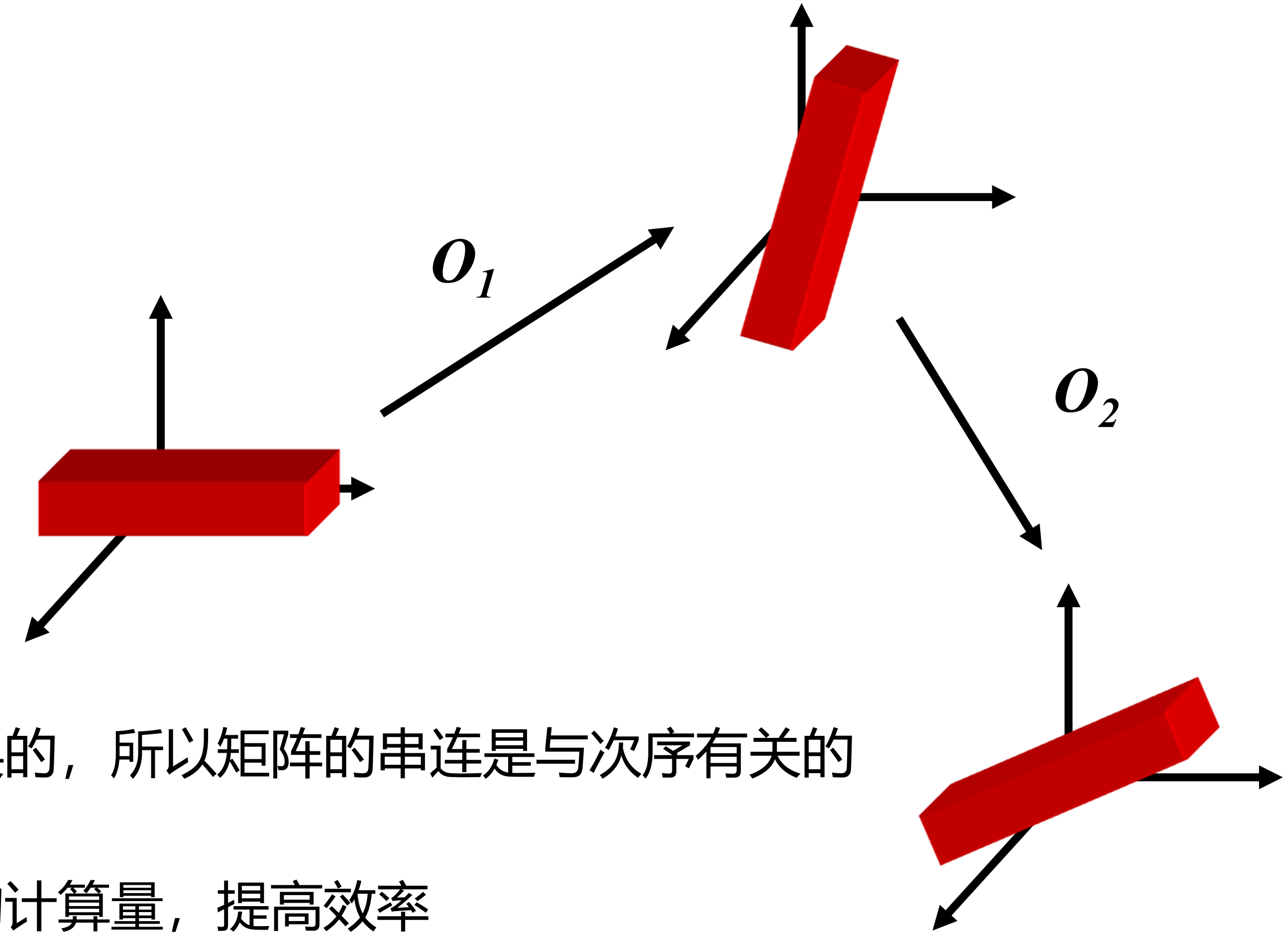
Transformations Concatenation(矩阵的串连)

- Transformations can be treated as a series of matrix multiplications

$$\mathbf{P}' = \mathbf{M}_1\mathbf{M}_2\mathbf{M}_3\cdots\mathbf{M}_n\mathbf{P}$$
$$\mathbf{M} = \mathbf{M}_1\mathbf{M}_2\mathbf{M}_3\cdots\mathbf{M}_n$$
$$\mathbf{P}' = \mathbf{M}\mathbf{P}$$

$$\mathbf{P}'^T = (\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3\cdots\mathbf{M}_n\mathbf{P})^T$$
$$\mathbf{M}^T = \mathbf{M}_n^T\mathbf{M}_{n-1}^T\cdots\mathbf{M}_2^T\mathbf{M}_1^T$$
$$\mathbf{P}'^T = \mathbf{P}^T\mathbf{M}^T$$

Transformation Concatenation



- 因矩阵相乘是不可交换的，所以矩阵的串连是与次序有关的
- 矩阵串连的好处：节约计算量，提高效率

Compound Transformation(复合变换)

rotation, scaling

translation

$$\begin{bmatrix} s_x \cos\theta & -\sin\theta & 0 & t_x \\ \sin\theta & s_y \cos\theta & 0 & t_y \\ 0 & 0 & s_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

刚体变换

- 只有物体的位置（平移变换）和朝向（旋转变换）发生改变，而形状不变，得到的变换称为**刚体变换**
- 特点：保持长度和角度

$$\mathbf{X} = \mathbf{T}(\mathbf{t})\mathbf{R} = \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

其逆矩阵的计算：

$$\mathbf{X}^{-1} = (\mathbf{T}(\mathbf{t})\mathbf{R})^{-1} = \mathbf{R}^{-1}\mathbf{T}(\mathbf{t})^{-1} = \mathbf{R}^T\mathbf{T}(-\mathbf{t})$$

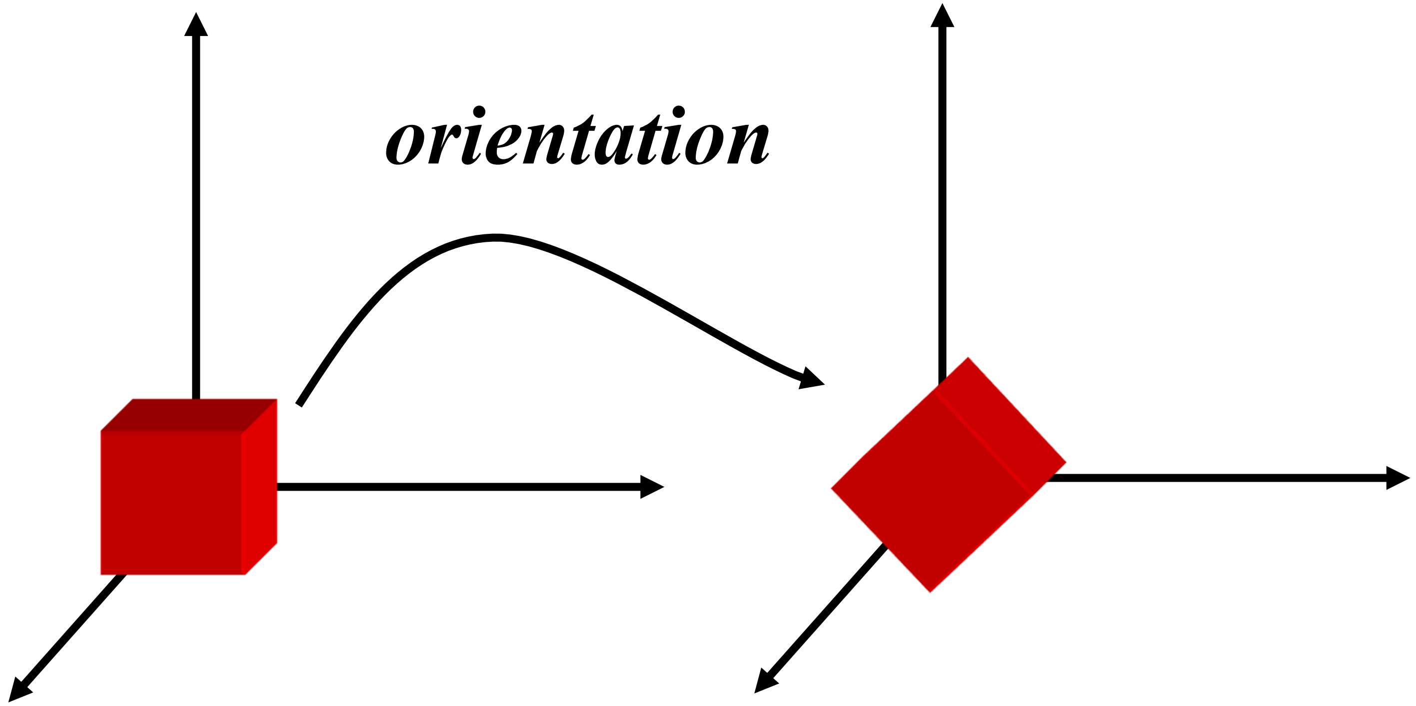
逆矩阵的计算

- 如果矩阵由一个或多个简单变换复合而成，而且已知参数，则逆矩阵可通过“逆参数”和矩阵相乘次序来得到。

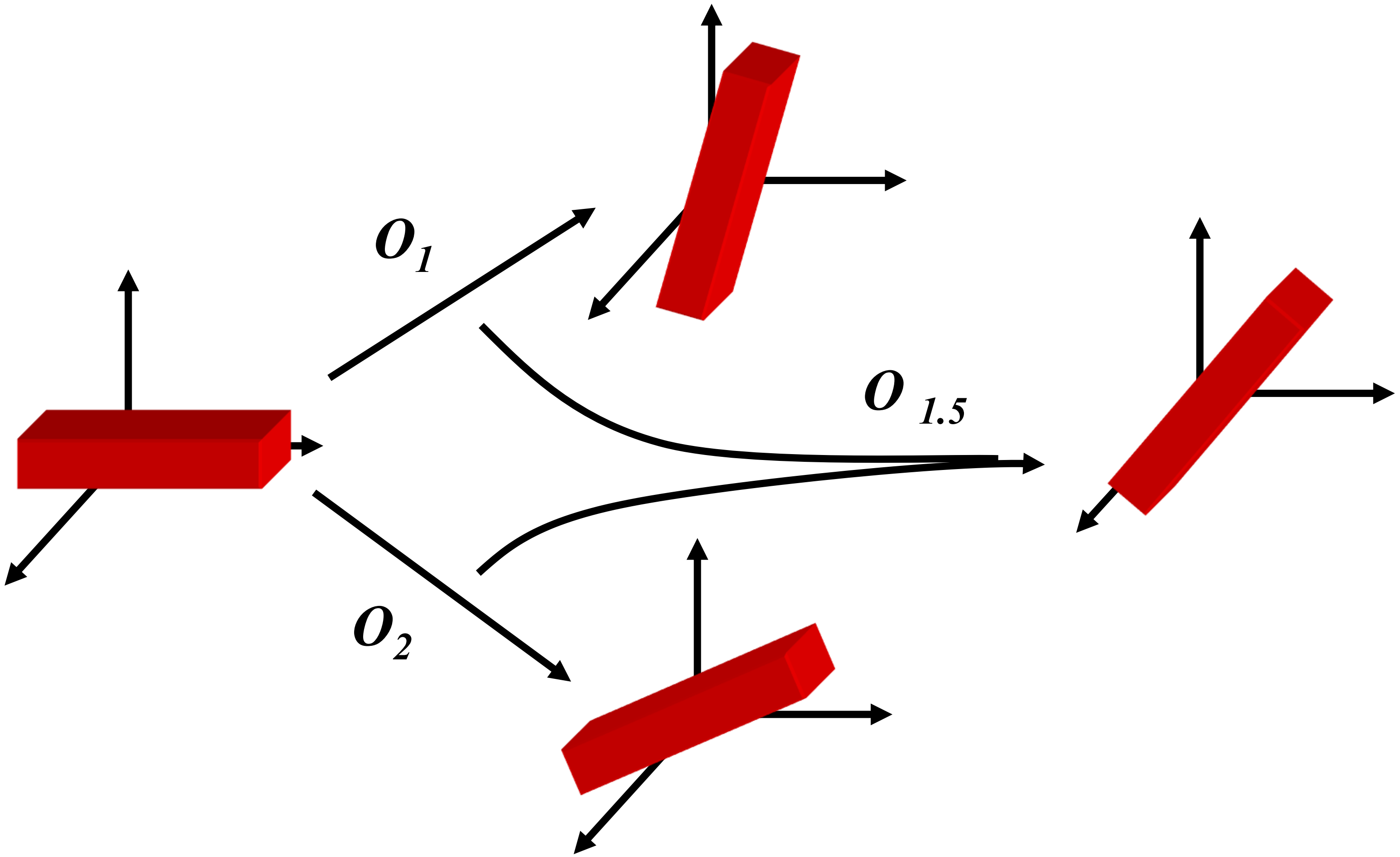
  例子：$\mathbf{M}=\mathbf{T}(t)\mathbf{R}(\theta)$，则$\mathbf{M}^{-1}=\mathbf{R}(-\theta)\mathbf{T}(-t)$
- 如果矩阵已知是正交的，则$\mathbf{M}^{-1}=\mathbf{M}^{T}$
- 如果未知任何信息：伴随矩阵法、Cramer法、LU分解法、Gauss消去法
- Cramer法和伴随矩阵法具有较少的“if”分叉，应优先选用。

  在现代的体系结构中，“if”测试最好避免

Orientation Representation



Interpolation



Rotation Matrix(用旋转矩阵表示旋转)

- Rows/columns of matrix must be orthonormal
  - Unit length and orthogonal（单位长度和正交）
- **Numerical errors** cause a nonorthonormal matrix when a series of rotations apply
- How to interpolate between matrices?
  - Interpolating the components of two matrices doesn’t maintain the orthonormality
  - The generated matrix is not a rotation matrix

Interpolating Rotation Matrices?

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

90^o z-axis **-90^o z-axis**

- The halfway matrix you get by linearly interpolating each entry is

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  - Not a rotation matrix any more!

Other 3D rotation representations

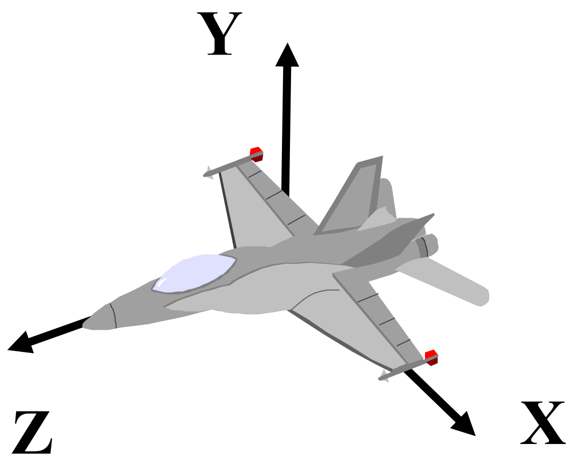
- **Rotation Matrix（旋转矩阵）**
- Fixed Angle（定角）
- Euler Angle（欧拉角）
- Axis angle （轴线角）
- Quaternion（四元数）

Fixed Angle Representation

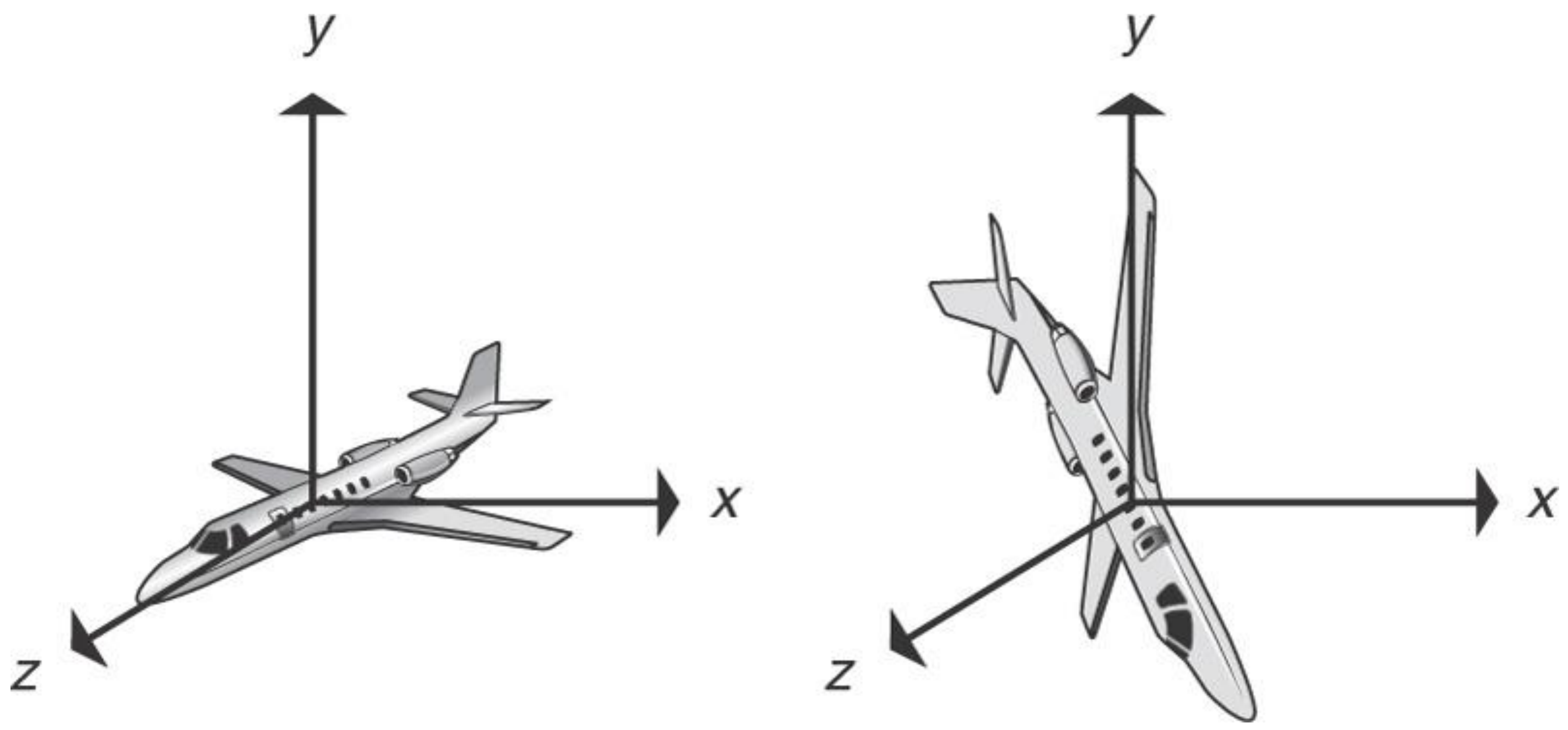
- Ordered triple of rotations about **fixed axes**
- Any triple can be used that doesn't repeat an axis immediately, e.g., *x*-*y*-*z* is fine, so is *x*-*y*-*x*. But *x*-*x*-*z* is not.

e.g., x-y-z order $(\theta_x, \theta_y, \theta_z)$

$$P' = R_z(\theta_z)R_y(\theta_y)R_x(\theta_x)P$$



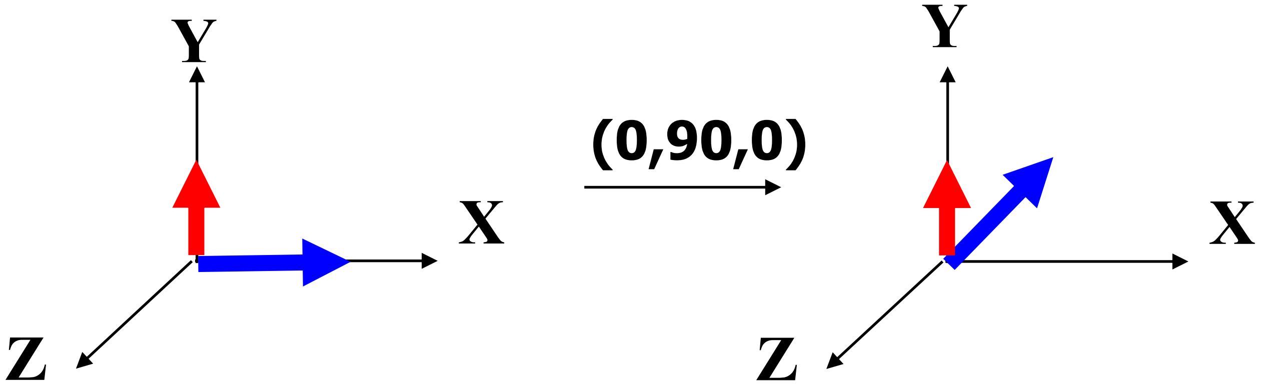
Fixed Angle Representation



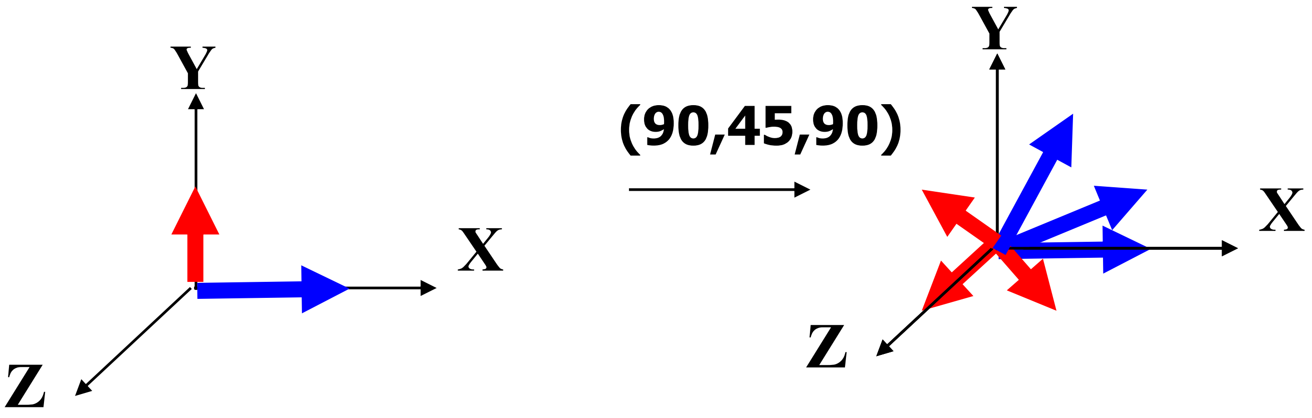
Orientation (10, 45, 90)

Fixed Angle Representation

- (0,90,0) in *x*-*y*-*z* order

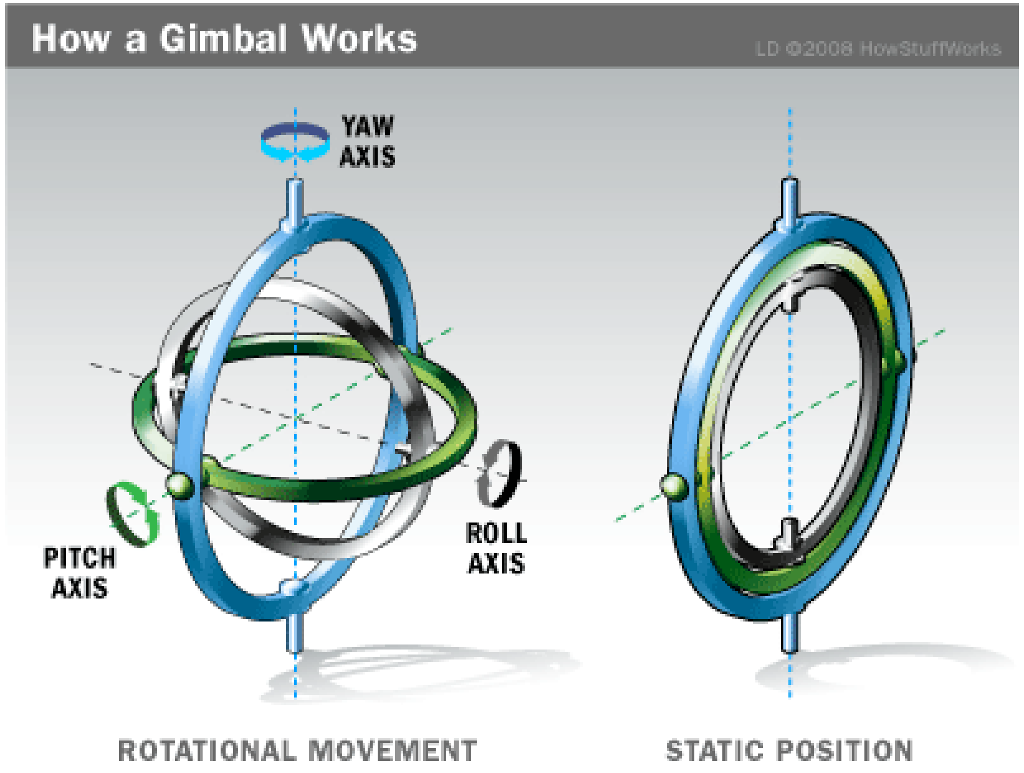


- (90,45,90) in *x*-*y*-*z* order



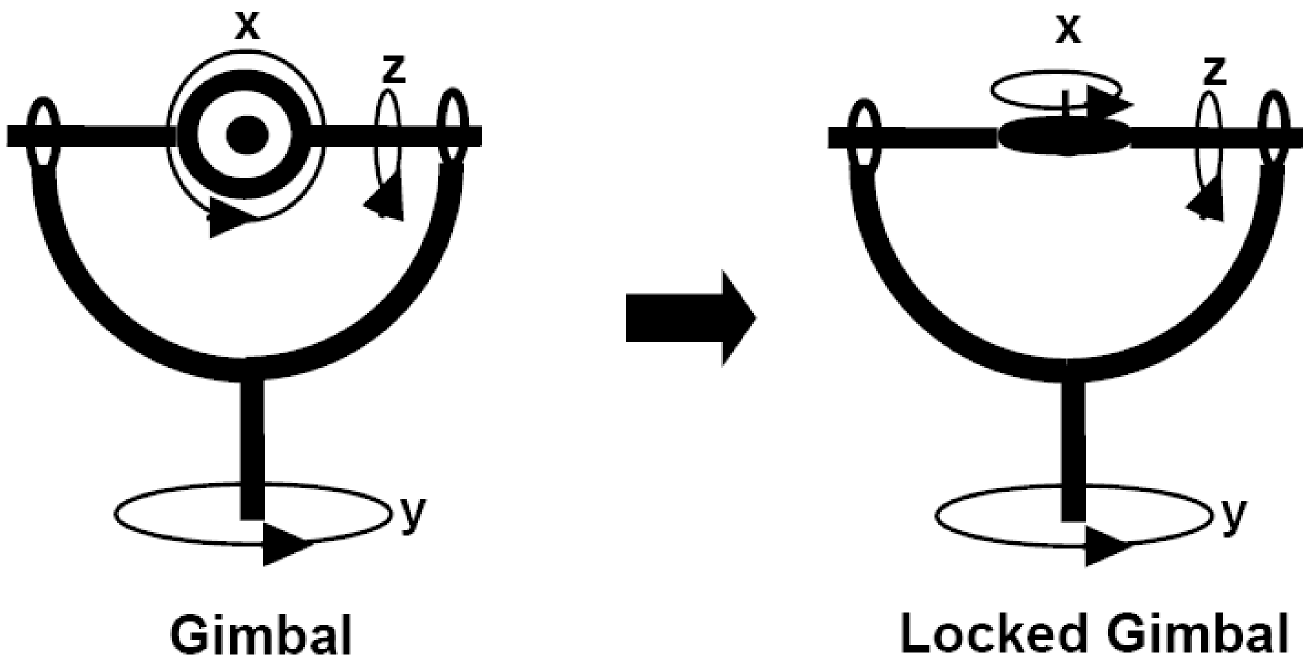
Gimbal(万向节)

- A gimbal is a mechanical device allowing the rotation of an object in multiple dimensions



Gimbal Lock(万向节死锁)

- Gimbal lock occurs when two of the rotation axes align; e.g., *x* and *y* axes in the figure
  - Lost a degree of freedom; cannot rotate about *x*-axis (*x*和*y*等效)



Gimbal Lock(万向节死锁)

- For an orientation (0, 90, 0),
  - A slight change in the first value (+/-ε, 90, 0)
  - A slight change in the 3rd value (0, 90, +/-ε)
  - 90-degree y-axis rotation essentially makes *x*-axis align with *z*-axis → gimbal lock
    - From (0, 90, 0), the object can no longer be rotated about *x*-axis by a small change since orientation actually performed is (90, 90+ ε, 90)

Gimbal Lock(万向节死锁)

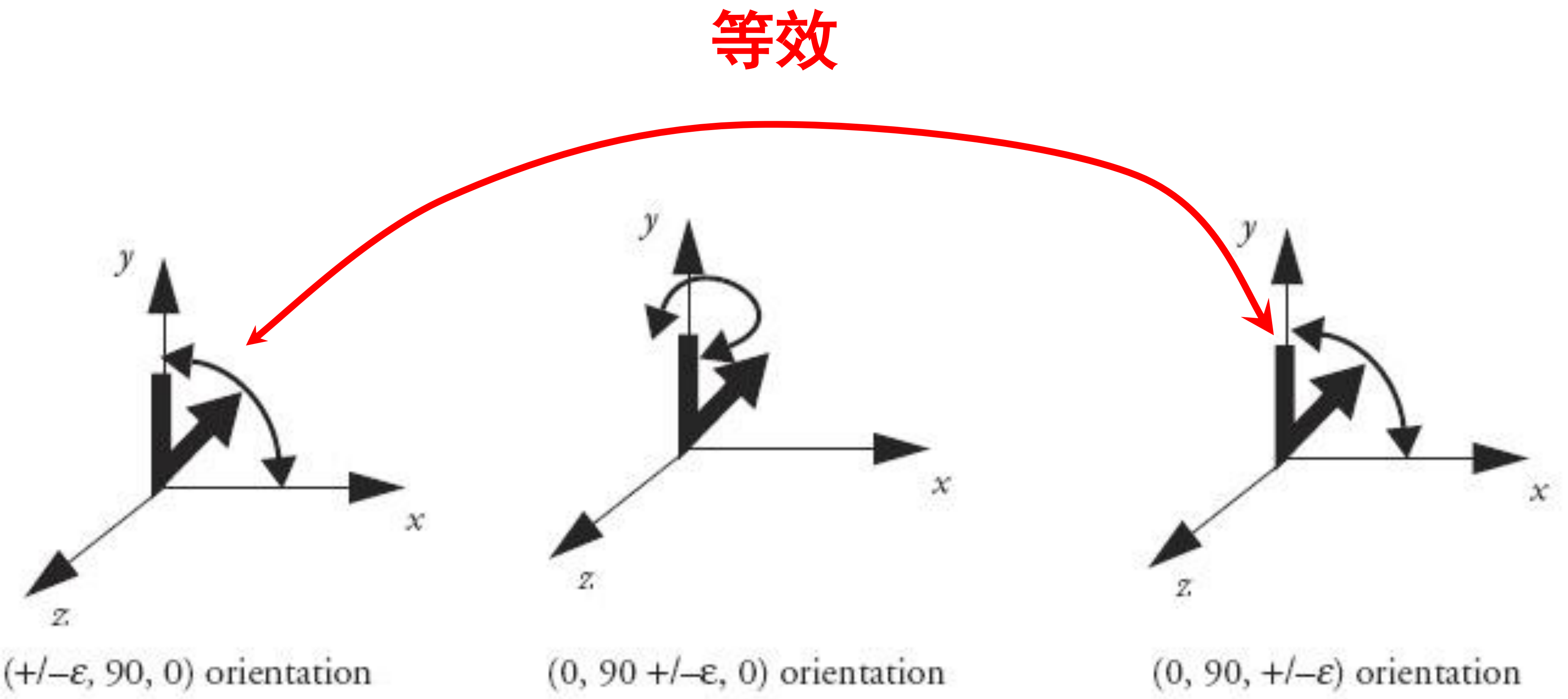
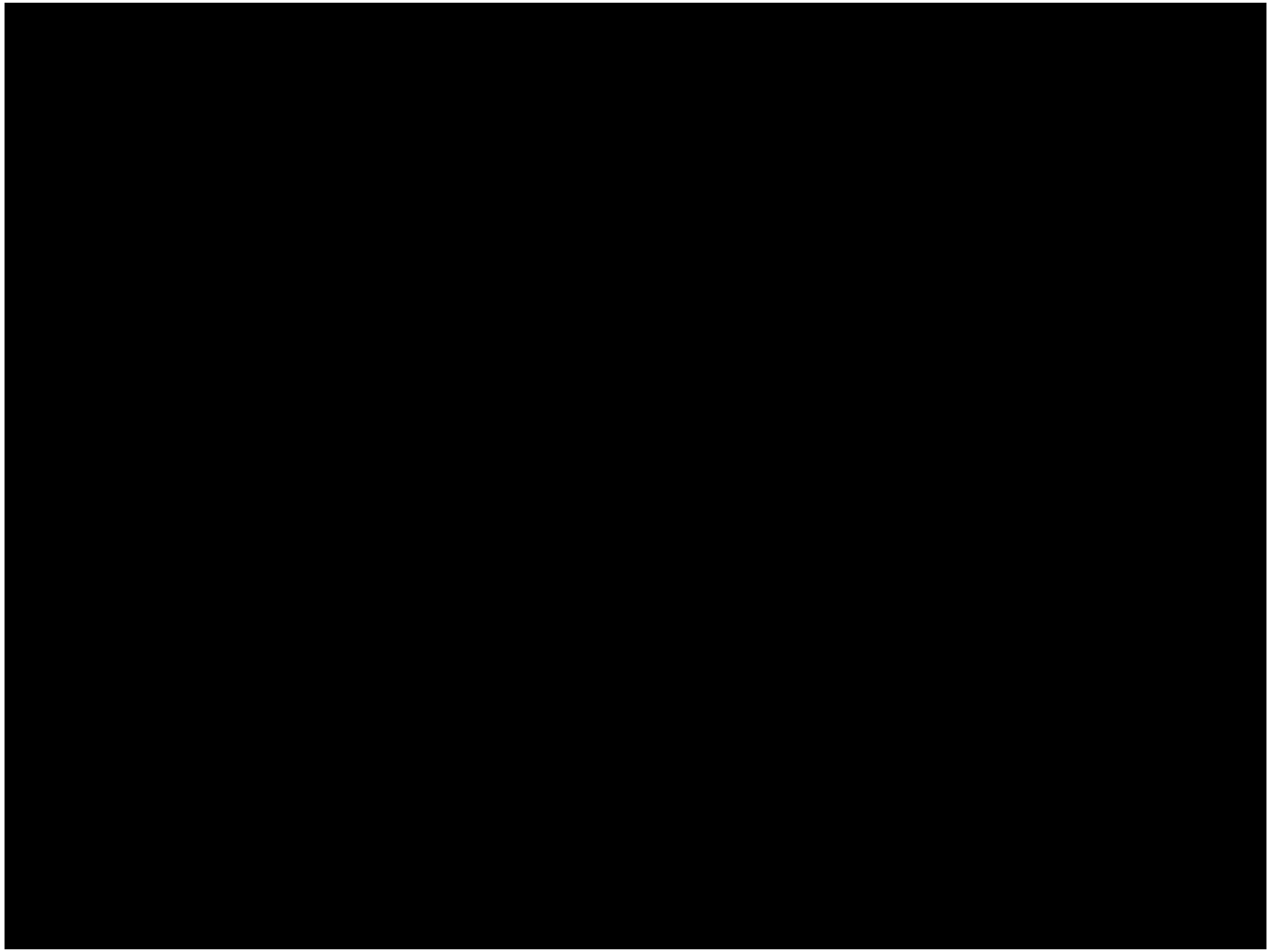


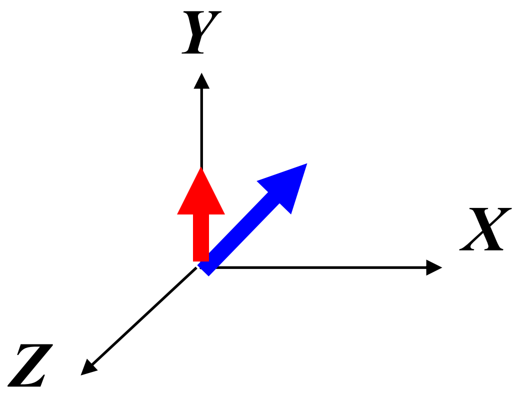
Figure 2.17 Effect of slightly altering values of fixed angle representation (0, 90, 0)

Gimbal lock Video

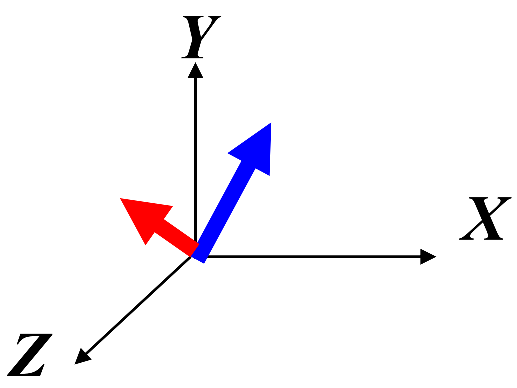


Interpolation Problem in Fixed Angle

- The rotation from (0,90,0) to (90,45,90) is a 45-degree x-axis rotation
  - Impossible because the first 90-degree y-axis rotation
- Directly interpolating between (0,90,0) and (90,45,90) produces a halfway orientation (45, 67.5, 45)
  - Desired halfway orientation is (90, 22.5, 90)



(0,90,0)



(90,45,90)

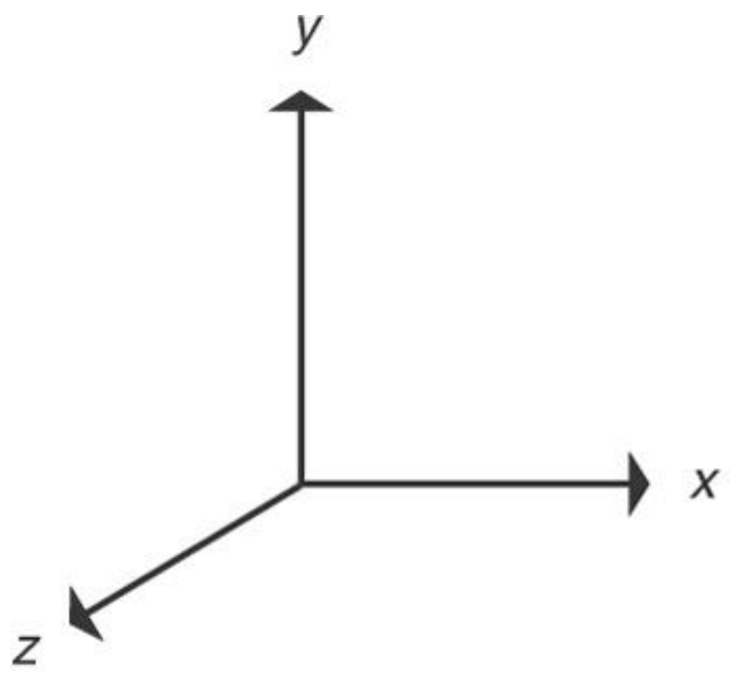
Fixed Angle Representation

- Compact
- Fairly intuitive
- Easy to work
- But not the most desirable representation to use because of gimbal lock problem.

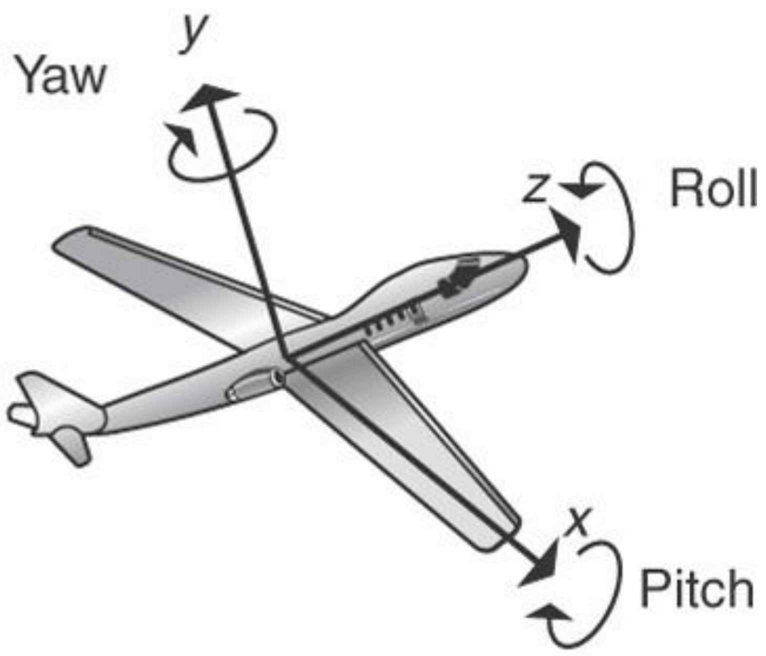
Euler Angle(欧拉角)

- **Euler**变换是一种直观的使一个物体（或摄像机）朝向一指定方向的有效方法。其来源：瑞士大数学家**Leonard Euler**
- Ordered triple of rotations about **local axes**
- As with fixed angles, any triple can be used that doesn't immediately repeat an axis, e.g., x-y-z, is fine, so is x-y-x. But x-x-z is not.
- Euler angle ordering is equivalent to reverse ordering in fixed angles
  - Why?
  - The Euler angle representation has exactly the same advantages and disadvantages as those of the fixed angle representation.

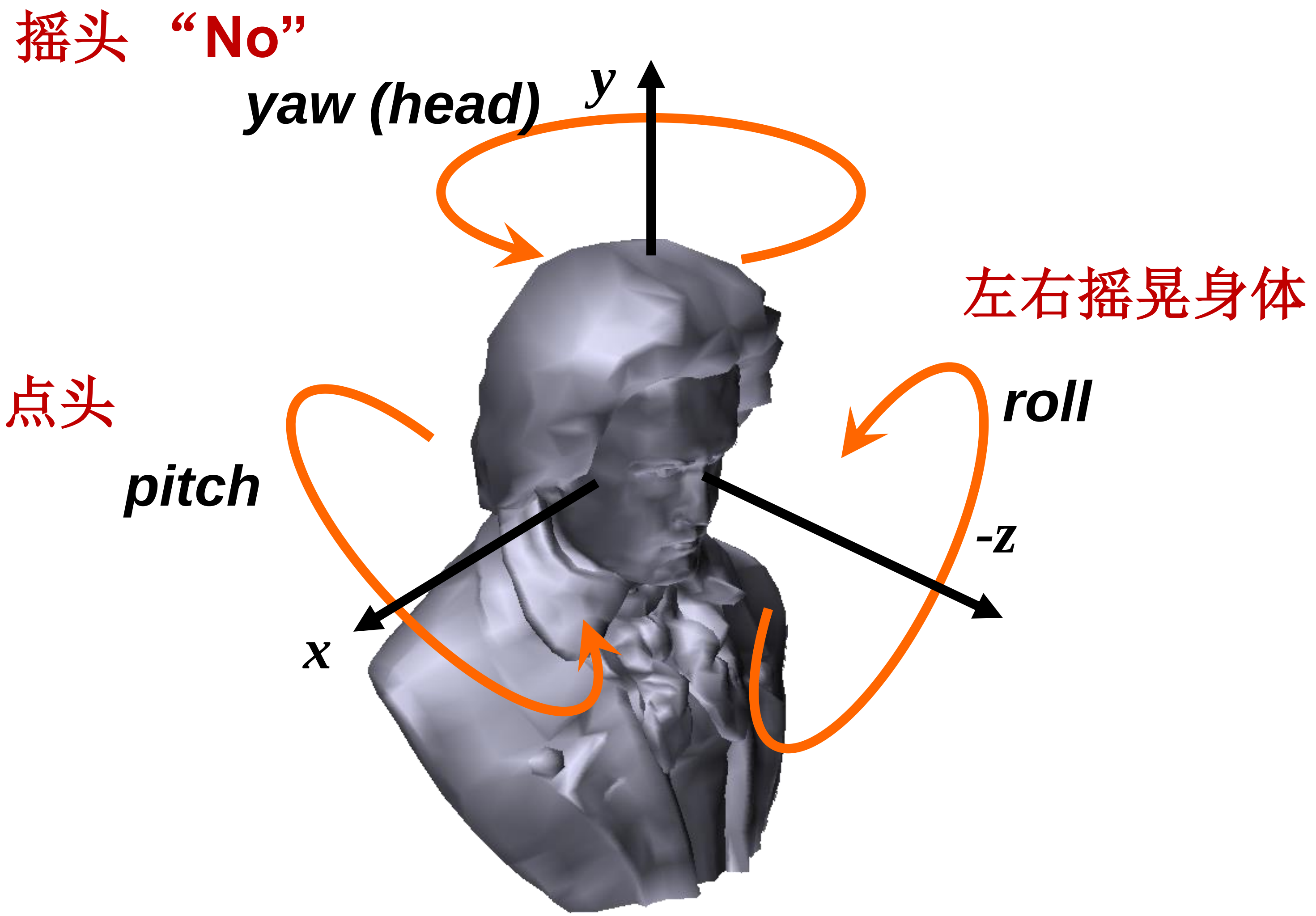
Euler Angle(欧拉角)



Global coordinate system



Local coordinate system
attached to object



Euler变换：Head，Pitch and Roll

其它命名方式：**x-roll, y-roll, z-roll**。在飞行仿真中，采用**yaw**而非**head**

Euler Angle

- Euler angle ordering is equivalent to **reverse ordering** in fixed angles.
- For a Euler angle representation in x-y-z ordering
  - Y-axis rotation: around the y-axis of the local, rotated coordinate system

Fixed Angle **Euler Angle**

$$R'_y(\beta)R_x(\alpha) = \overbrace{R_x(\alpha)R_y(\beta)R_x(-\alpha)}R_x(\alpha) = R_x(\alpha)R_y(\beta)$$

  - Z-axis rotation: around the twice-rotated frame

$$R''_z(\gamma)R'_y(\beta)R_x(\alpha)$$

Fixed Angle

$$= \overbrace{R_x(\alpha)R_y(\beta)R_z(\gamma)R_y(-\beta)R_x(-\alpha)}R_x(\alpha)R_y(\beta)$$

$$= R_x(\alpha)R_y(\beta)R_z(\gamma)$$

Euler Angle

欧拉角中的Gimbal lock

- **Gimbal lock现象**：当一个自由度丧失时。
- 当$p=\pi/2$ 时，矩阵只依赖一个角(*r*+*h*)

$$
\mathbf{E}\left(h,\frac{\pi}{2},r\right)=\begin{pmatrix}\cos r\cos h-\sin r\sin h & 0 & \cos r\sin h+\sin r\cos h\\ \sin r\cos h+\cos r\sin h & 0 & \sin r\sin h-\cos r\cos h\\ 0 & 1 & 0\end{pmatrix}
$$

$$
=\begin{pmatrix}\cos(r+h) & 0 & \sin(r+h)\\ \sin(r+h) & 0 & -\cos(r+h)\\ 0 & 1 & 0\end{pmatrix}
$$

从Euler变换获取参数

- 从一正交矩阵反求Euler参数

$$\mathbf{F} = \begin{pmatrix} f_{00} & f_{01} & f_{02} \\ f_{10} & f_{11} & f_{12} \\ f_{20} & f_{21} & f_{22} \end{pmatrix} = \mathbf{R}_z(r)\mathbf{R}_x(p)\mathbf{R}_y(h) = \mathbf{E}(h, p, r)$$

- 把上式展开，得到

$$\mathbf{F}=\begin{pmatrix}\cos r\cos h-\sin r\sin p\sin h & -\sin r\cos p & \cos r\sin h+\sin r\sin p\cos h\\ \sin r\cos h+\cos r\sin p\sin h & \cos r\cos p & \sin r\sin h-\cos r\sin p\cos h\\ -\cos p\sin h & \sin p & \cos p\cos h\end{pmatrix}$$

- 由于sin p= f_{21}， f_{01}/f_{11}= - tan r, f_{20}/f_{22}= -tan h

故三个欧拉参数的值为

h = atan2 (-f_{20}, f_{22})

p = arcsin(f_{21})

r = atan2(-f_{01},f_{11})

特殊情况处理

- 当$\cos p = 0$时，$f_{01}=f_{11}=0$，此时$r = \mathrm{atan2}(-f_{01}, f_{11})$无解。因$\cos p = 0$时，$\sin p = \pm 1$,故

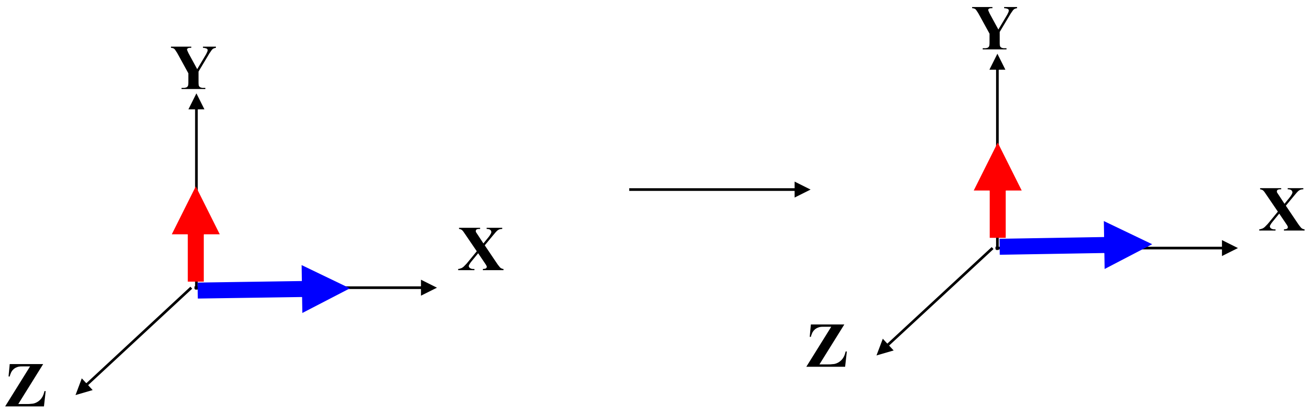
$$\mathbf{F} = \begin{pmatrix} \cos(r \pm h) & 0 & \sin(r \pm h) \\ \sin(r \pm h) & 0 & -\cos(r \pm h) \\ 0 & \pm 1 & 0 \end{pmatrix}$$

可任意设定 $h=0$, 再得到 $r = \mathrm{atan2}(f_{10}, f_{00})$

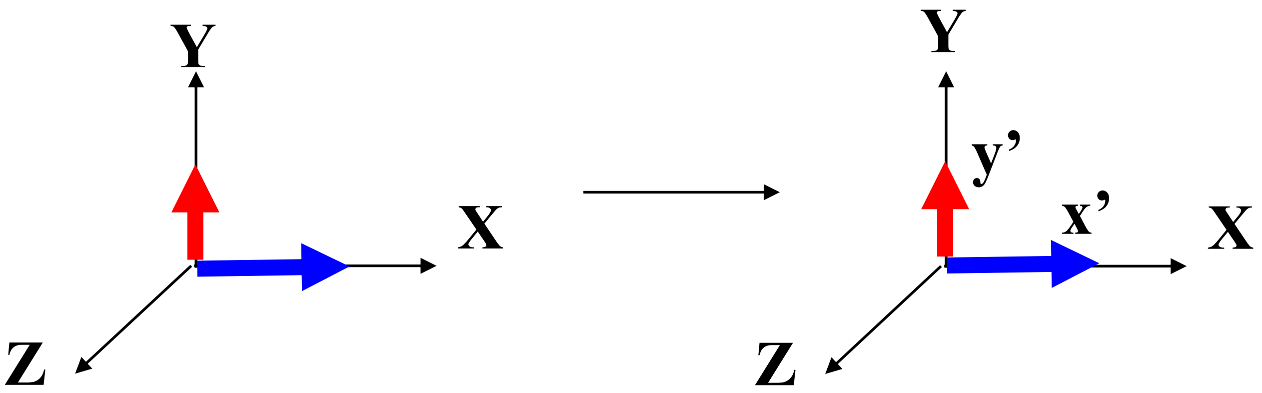
- 由于p的值域为$[-90^0, 90^0]$，如果p的值不在这个范围，原始参数无法求得。故求得的h、p、r不是唯一的。

Fixed Angle vs. Euler Angle

- Fixed angle: (90,45,90) in x-y-z order

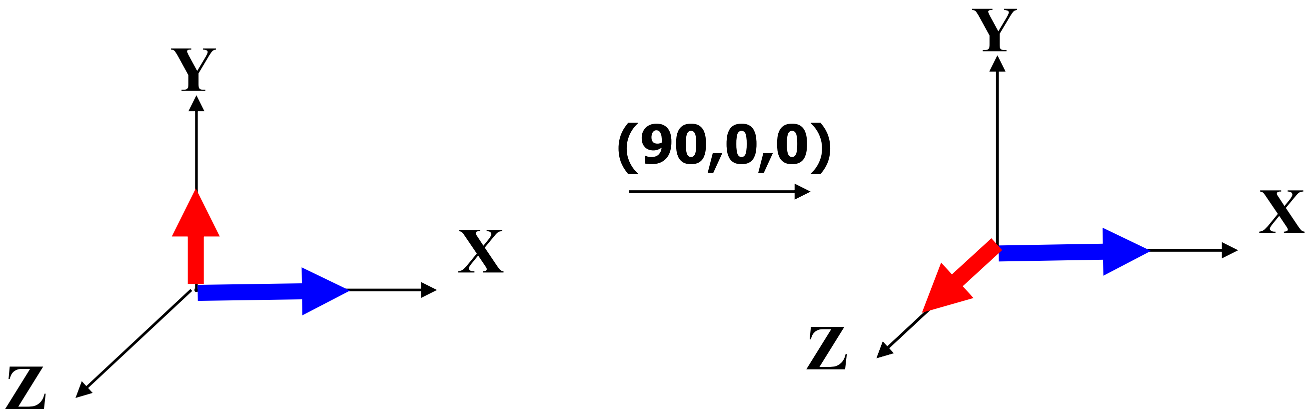


- Euler angle: (90,45,90) in z'-y'-x' order

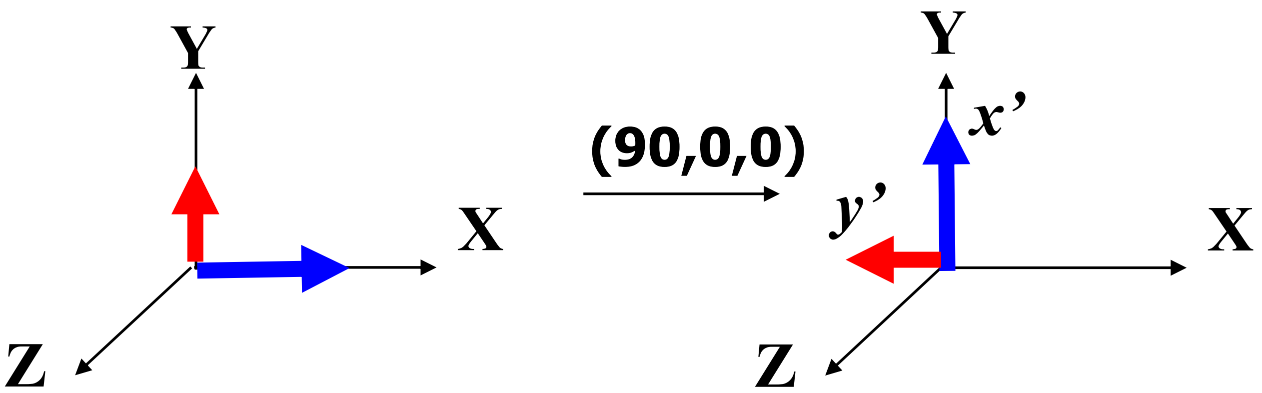


Fixed Angle vs. Euler Angle

- Fixed angle: (90,45,90) in x-y-z order

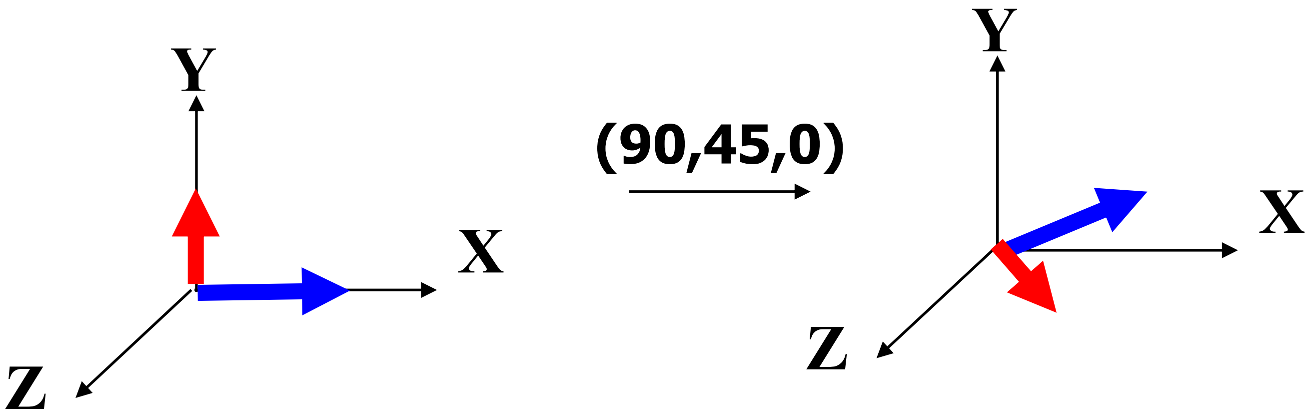


- Euler angle: (90,45,90) in z’-y’-x’ order

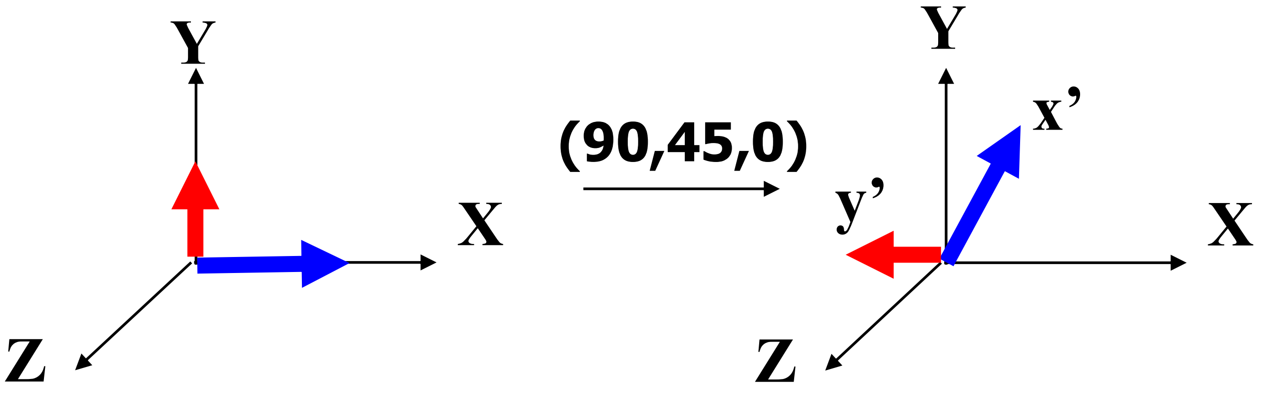


Fixed Angle vs. Euler Angle

- Fixed angle: (90,45,90) in x-y-z order

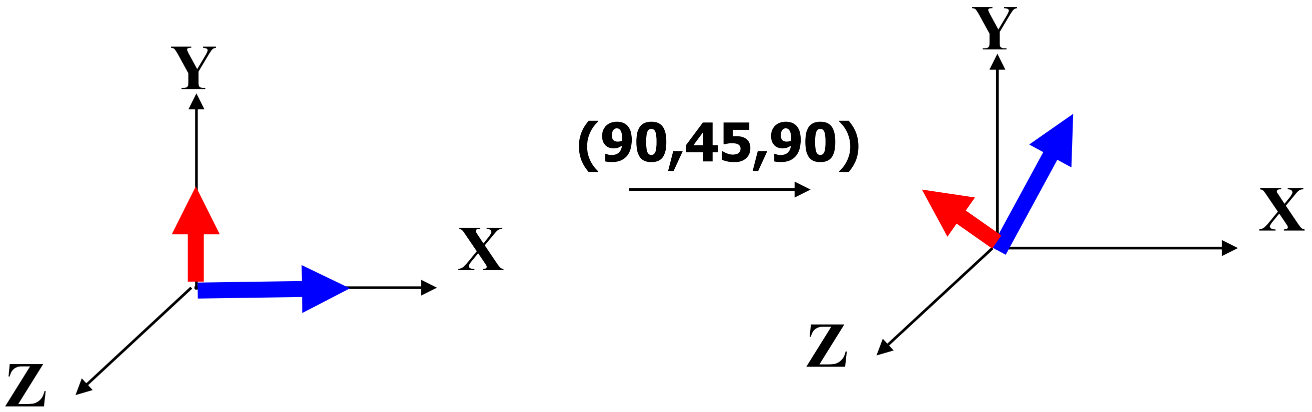


- Euler angle: (90,45,90) in z'-y'-x' order

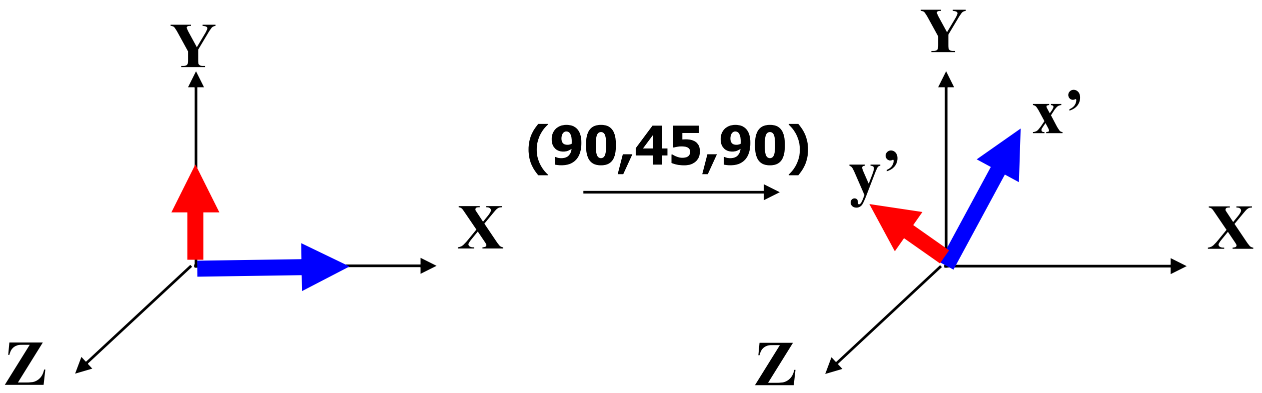


Fixed Angle vs. Euler Angle

- Fixed angle: (90,45,90) in x-y-z order



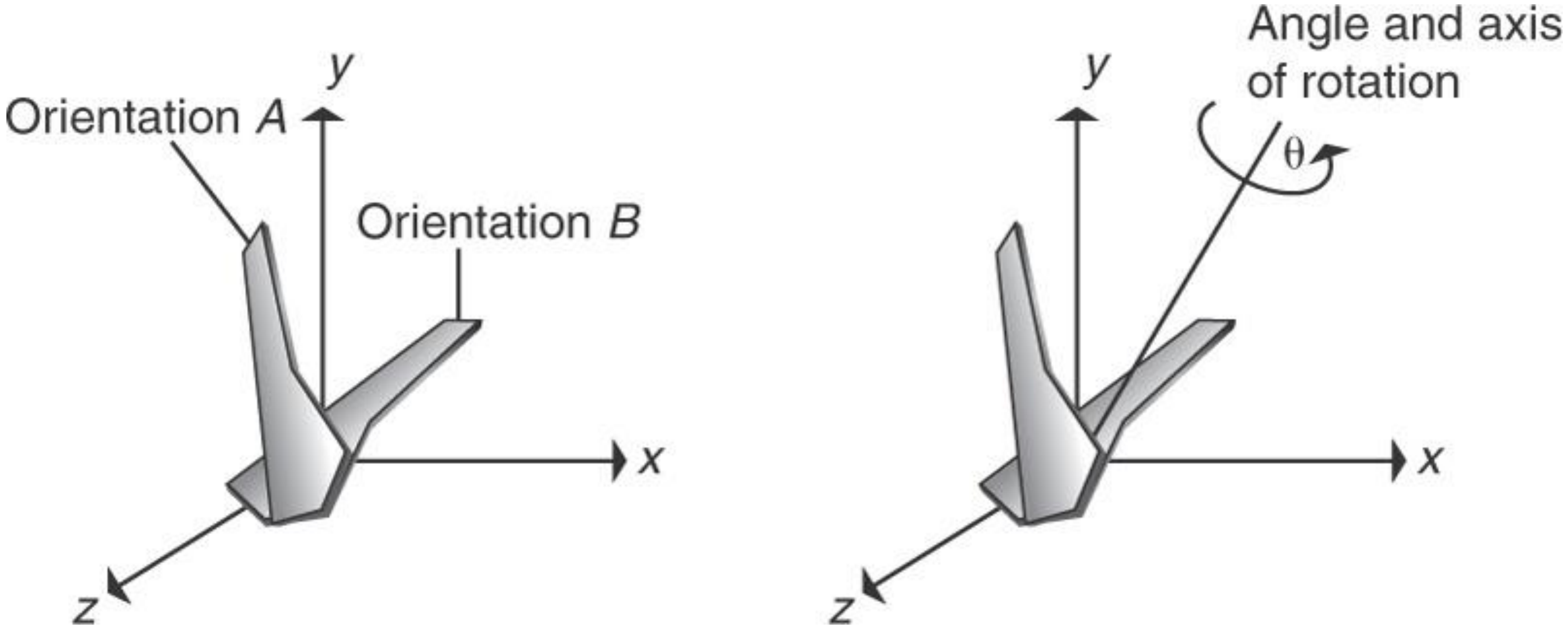
- Euler angle: (90,45,90) in z'-y'-x' order



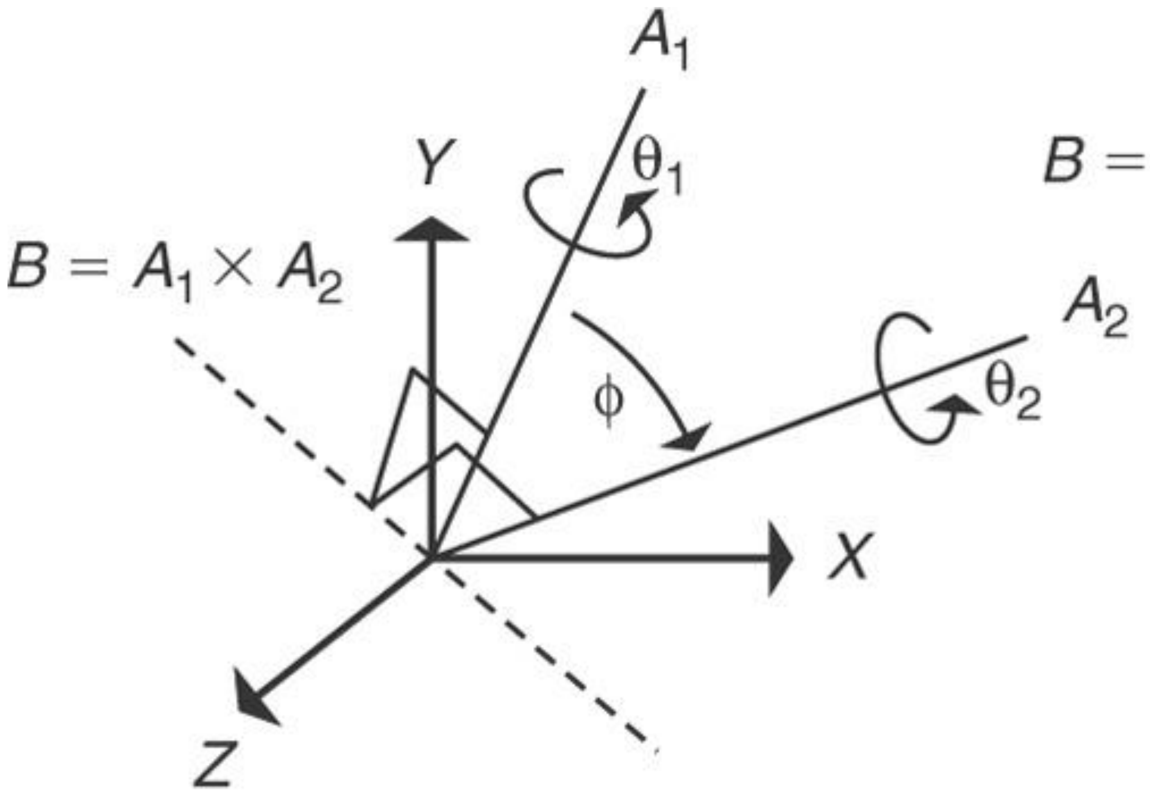
Angle and Axis Representation（角位移）

- **Euler's rotation theorem**
  - One orientation can be derived from another by a single rotation about an axis
  - So, we can use an axis and a single angle to represent an orientation (with respect to the object's initial orientation)
- Interpolation can be implemented by interpolating axes of rotation and angles separately; but the transformation concatenation cannot be done easily.

Angle and Axis Representation



Angle and Axis Representation



$$B = A_1 \times A_2$$

$$\phi = \cos^{-1}\left(\frac{A_1 \cdot A_2}{|A_1|\,|A_2|}\right)$$

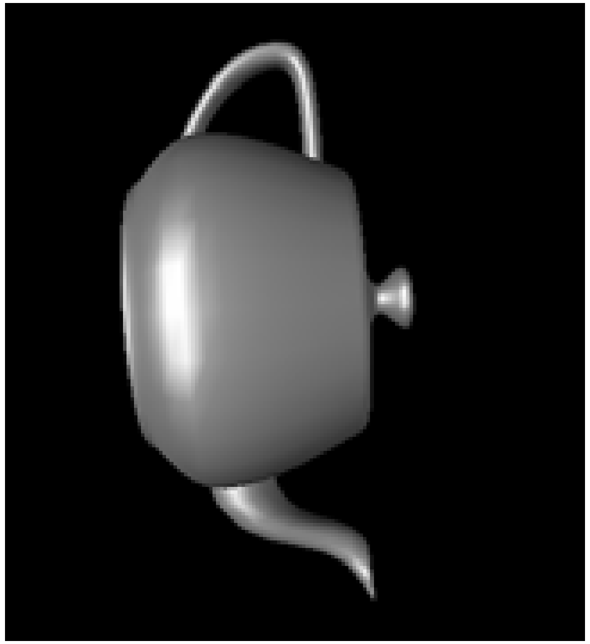
$$A_k = Rotate(B, k\phi, A_1)$$

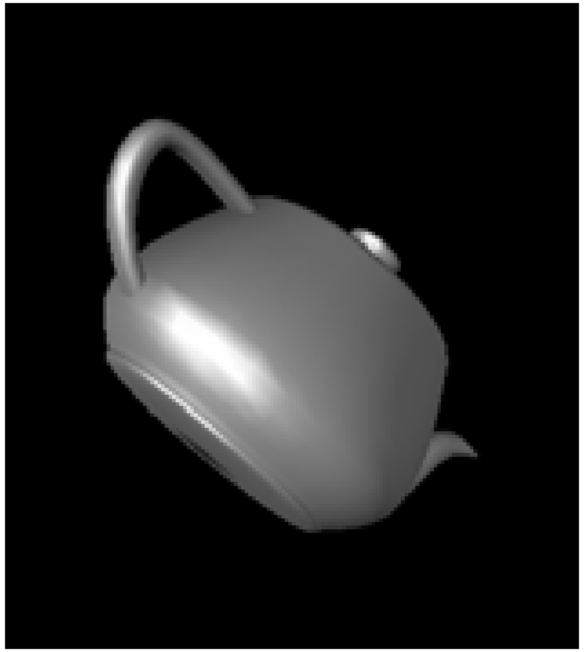
$$\theta_k = (1 - k) \cdot \theta_1 + k \cdot \theta_2$$

Interpolating axis-angle representations from $(\mathbf{A}_1, \theta_1)$ to $(\mathbf{A}_2, \theta_2)$

Rotate(x, y, z): rotate z around x by y degree

Angle and Axis Representation





3D Rotation Representations (review)

- Rotation Matrix
  - orthonormal columns/rows
  - bad for interpolation
- Fixed Angle
  - rotate about global axes
  - bad for interpolation, has gimbal lock
- Euler Angle
  - rotate about local axes
  - same problem as fixed angle (also has gimbal lock)

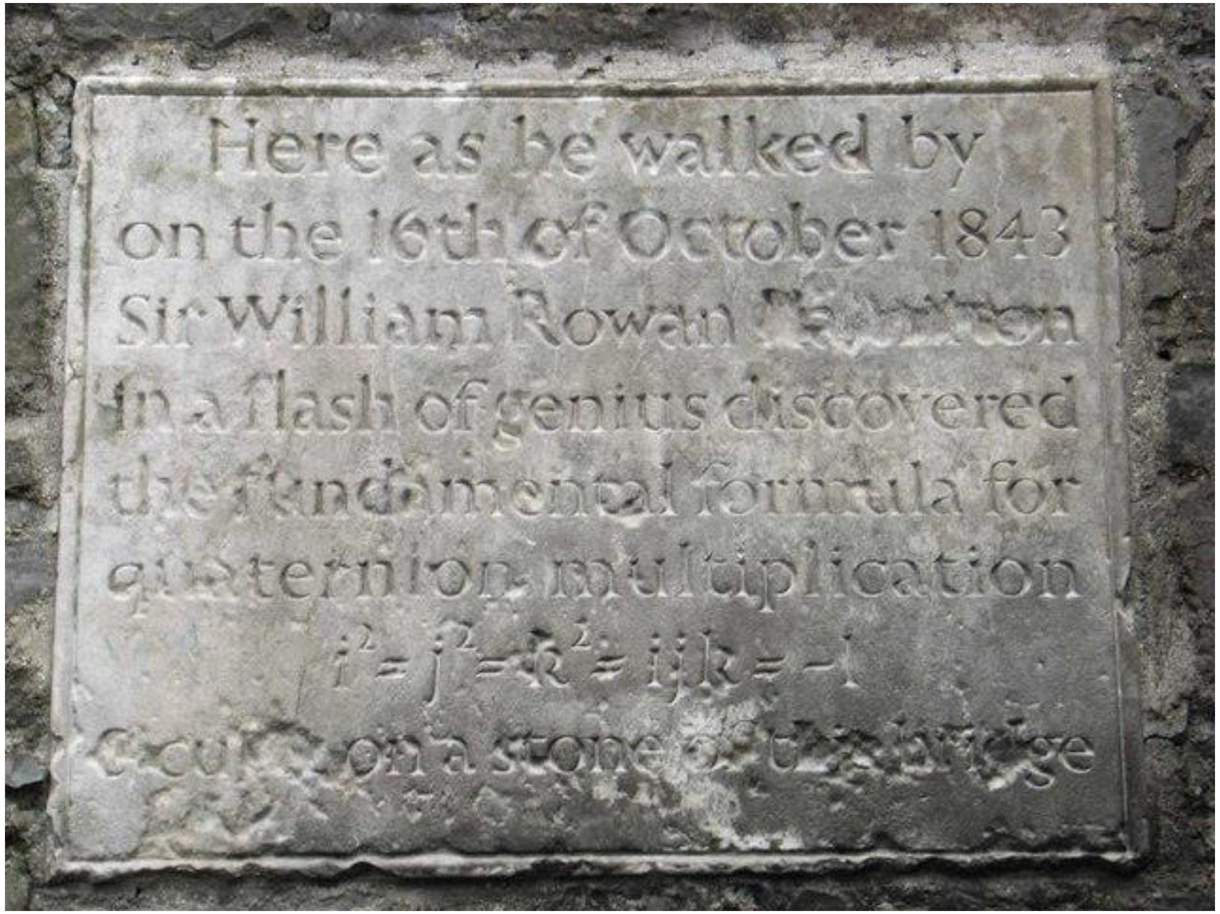
3D Rotation Representations (review)

- Axis angle
  - rotate about A by θ, (θ, A_x, A_y, A_z)
  - good interpolation, no gimbal lock
  - bad for compounding rotations
- **Quaternion**
  - similar to axis angle but in different form
  - $q=[s, \mathbf{v}]$
  - good for interpolation and compounding rotations

Quaternions(四元数)

- 最早由Sir William Rowan Hamilton于1843年提出，从复数推广到四维空间
- 1985年，Shoemake把四元数引入计算机图形学
- 在表示旋转和朝向方面，优于Euler角。具有表示紧凑，朝向插值稳定的优点

Sir William Rowan Hamilton



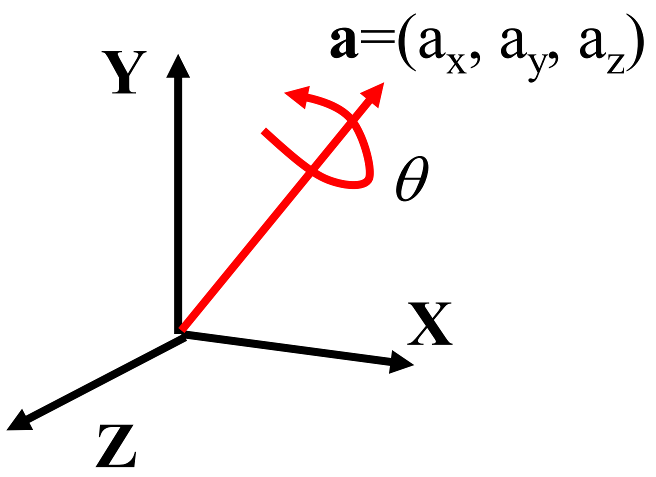
石桥上的纪念碑

(Brougham Bridge，现称为金雀花桥 Broom Bridge)

Quaternions (四元数)

- Similar to axis-angle representations
  - 4-tuple of real numbers
    - $q=(s, x, y, z)$ or $[s, \mathbf{v}]$, s is a scalar; $\mathbf{v}$ is a vector
- The quaternion for rotating an angle about an axis (an axis-angle rotation):

$$q = \mathbf{Rot}(\theta, (a_x, a_y, a_z)) = \left[\cos(\frac{\theta}{2}), \sin(\frac{\theta}{2}) \cdot (a_x, a_y, a_z)\right]$$



If a is unit length, then q will be also

Quaternions（四元数）

If a is unit length, then q will be also

$$|\mathbf{q}| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

$$= \sqrt{\cos^2\frac{\theta}{2} + a_x^2 \sin^2\frac{\theta}{2} + a_y^2 \sin^2\frac{\theta}{2} + a_z^2 \sin^2\frac{\theta}{2}}$$

$$= \sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}\left(a_x^2 + a_y^2 + a_z^2\right)}$$

$$= \sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}|\mathbf{a}|^2} = \sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2}}$$

$$= \sqrt{1} = 1$$

Quaternions (四元数)

- Rotating some angle around an axis is the same as rotating the negative angle around the negated axis

$$\mathbf{Rot}_{-\theta,-(x,y,z)} = \left[-\cos(\frac{\theta}{2}), -\sin(\frac{\theta}{2})(x,y,z)\right] = -q$$

$$\begin{aligned}
-q &= Rot_{-\theta,-(x,y,z)} \\
&= [\cos(-\theta/2), \sin((-\theta)/2) \bullet (-(x,y,z))] \\
&= [\cos(\theta/2), -\sin(\theta/2) \bullet (-(x,y,z))] \\
&= [\cos(\theta/2), \sin(\theta/2) \bullet (x,y,z)] \\
&= Rot_{\theta,(x,y,z)} \\
&= q
\end{aligned}$$

Quaternion Math

- Addition

$$[s_1, \mathbf{v}_1] + [s_2, \mathbf{v}_2] = [s_1 + s_2, \mathbf{v}_1 + \mathbf{v}_2]$$

- Multiplication

$$[s_1, \mathbf{v}_1]\,[s_2, \mathbf{v}_2] = [s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2]$$

- Multiplication is associative but not commutative（满足结合律，不满足交换律)

$$q_1(q_2 q_3) = (q_1 q_2) q_3 \qquad\qquad q_1 q_2 \neq q_2 q_1$$

Quaternion Math (cont.)

- Multiplicative identity(乘法单位): [1, 0,0,0]

$$q\,[1,0,0,0]=q$$

- Inverse: $q^{-1}=\dfrac{[s,\ -v]}{\|q\|^2}$ $\qquad qq^{-1}=[1,0,0,0]$

- Normalization for unit quaternion

$$\hat{q}=\frac{q}{\|q\|} \qquad \|q\|=\sqrt{s^2+x^2+y^2+z^2}$$

Quaternion Math (cont.)

- 对于单位复数，有 $\cos\theta + i\sin\theta = e^{i\theta}$
  对于单位四元数有：$\mathbf{q} = \sin\theta\mathbf{u}_q + \cos\theta = e^{\theta\mathbf{u}_q}$

- 对数运算：$\log(\mathbf{q}) = \log(e^{\theta\mathbf{u}_q}) = \theta\mathbf{u}_q$

- 指数运算：$\mathbf{q}^t = (\sin\theta\mathbf{u}_q, \cos\theta)^t = e^{\theta t\mathbf{u}_q} = \sin(\theta t)\mathbf{u}_q + \cos(\theta t)$

四元数到旋转矩阵的转换

- 对于单位四元数 $q = (q_w, q_x, q_y, q_z)$，把它转化为对应的旋转矩阵，可得到：

$$\mathbf{M}^q = \begin{pmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y-q_wq_z) & 2(q_xq_z+q_wq_y) & 0 \\ 2(q_xq_y+q_wq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z-q_wq_x) & 0 \\ 2(q_xq_z-q_wq_y) & 2(q_yq_z+q_wq_x) & 1-2(q_x^2+q_y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

旋转矩阵到四元数的转换

- 由$\mathbf{M}^q$可得到:

$$m_{21}^q - m_{12}^q = 4q_w q_x$$
$$m_{02}^q - m_{20}^q = 4q_w q_y$$
$$m_{10}^q - m_{01}^q = 4q_w q_z$$

故若得到q_w，则q_x、q_y、q_z便可得。因为：

$$\begin{aligned} tr(\mathbf{M}^q) &= 4 - 2s(q_x^2 + q_y^2 + q_z^2) \\ &= 4\left(1 - \frac{q_x^2 + q_y^2 + q_z^2}{q_x^2 + q_y^2 + q_z^2 + q_w^2}\right) \\ &= \frac{4q_w^2}{q_x^2 + q_y^2 + q_z^2 + q_w^2} = \frac{4q_w^2}{n(\mathbf{q})} \end{aligned}$$

- 故单位四元数为

$$q_w = \frac{1}{2}\sqrt{\mathrm{tr}(\mathbf{M}^q)},$$

$$q_x = \frac{m_{21}^q - m_{12}^q}{4q_w},$$

$$q_y = \frac{m_{02}^q - m_{20}^q}{4q_w},$$

$$q_z = \frac{m_{10}^q - m_{01}^q}{4q_w}.$$

Rotating Vectors Using Quaternion

- A point in space, **v**, is represented as [0, **v**]
- To rotate a vector **v** using quaternion *q*
  - Represent the vector as *v* = [0, **v**]
  - Represent the rotation as a quaternion *q*
  - Using quaternion multiplication

$$\mathbf{v}' = Rot_q(\mathbf{v}) = q\mathbf{v}q^{-1}$$

    - The proof isn't that hard
  - Note that the result **v**' always has zero scalar value

Compose Rotations

- Rotating a vector $\mathbf{v}$ by first quaternion p followed by a quaternion q is like rotation using qp

$$
\begin{aligned}
Rot_q(Rot_p(\mathbf{v})) &= Rot_q(p\mathbf{v}p^{-1}) \\
&= qp\mathbf{v}p^{-1}q^{-1} \\
&= (qp)\mathbf{v}(qp)^{-1} \\
&= Rot_{qp}(\mathbf{v})
\end{aligned}
$$

Prove by yourself that: $p^{-1}q^{-1} = (qp)^{-1}$

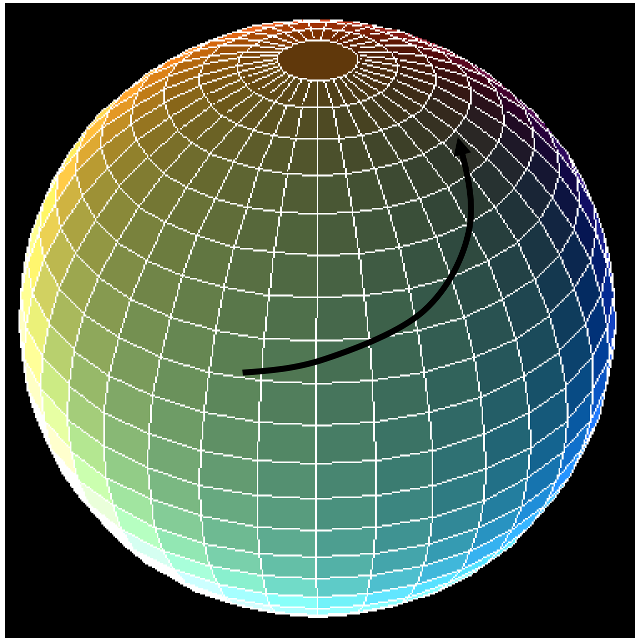
Compose Rotations

- To rotate a vector **v** by quaternion q followed by its inverse quaternion q^{-1}

$$
\begin{aligned}
Rot_{q^{-1}}(Rot_q(\mathbf{v})) &= Rot_{q^{-1}}(q\mathbf{v}q^{-1}) \\
&= q^{-1}q\mathbf{v}q^{-1}q \\
&= \mathbf{v}
\end{aligned}
$$

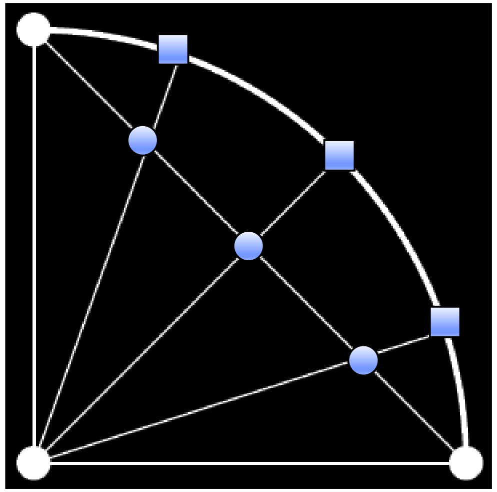
Quaternion Interpolation

- A quaternion is a point on a 4D unit sphere
  - Unit quaternion: $q=(s,x,y,z)$, $||q|| = 1$
    - Form a subspace: a 4D sphere
- Interpolating quaternion means moving between two points on the 4D unit sphere
  - A unit quaternion at each step – another point on the 4D unit sphere
  - Move with constant angular velocity along the greatest circle between the two points on the 4D unit sphere



Linear Interpolation

- Linear interpolation generates unequal spacing of points after projecting to circle



Spherical Linear Interpolation (Slerp)

- Want equal increment along arc connecting two quaternion on the spherical surface
  - Spherical linear interpolation (slerp)

$$slerp(q_1, q_2, u) = \frac{\sin((1-u)\theta)}{\sin\theta} q_1 + \frac{\sin(u\theta)}{\sin\theta} q_2$$

  - Normalize to regain unit quaternion

Spherical Linear Interpolation (Slerp)

Let $q = \alpha q_1 + \beta q_2$

We can solve for given:

$||q|| = 1,$

$q_1 \cdot q_2 = \theta,$

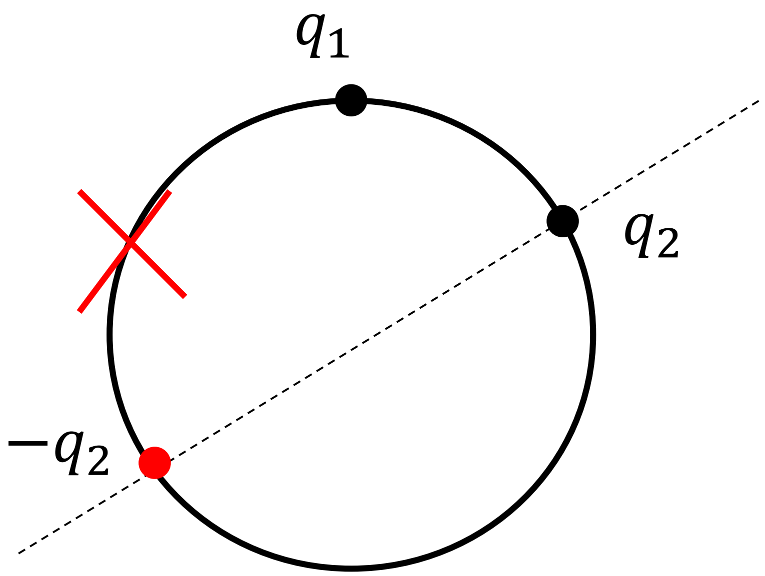
$q_1 \cdot q = u\theta$

to give

$$slerp(q_1, q_2, u) = \frac{\sin((1-u)\theta)}{\sin\theta} q_1 + \frac{\sin(u\theta)}{\sin\theta} q_2$$

Spherical Linear Interpolation (Slerp)

$$q_1 \cdot q_2 > 0$$



Spherical Linear Interpolation (Slerp)

- Recall that *q* and –*q* represent same rotation
- What is the difference between:

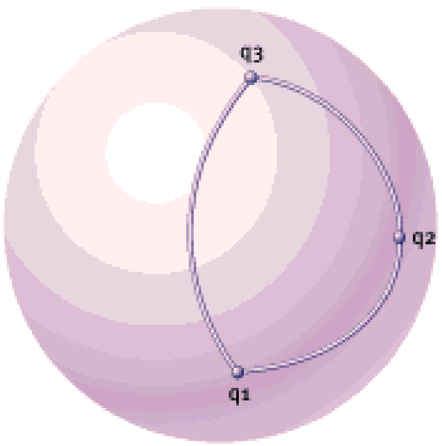
  Slerp(u, $\mathbf{q}_1$, $\mathbf{q}_2$) and Slerp(u, $\mathbf{q}_1$, -$\mathbf{q}_2$) ?

  - One of these will travel less than 90 degrees while the other will travel more than 90 degrees across the sphere
  - This corresponds to rotating the ‘short way’ or the ‘long way’
- Usually, we want to take the short way, so we negate one of them if their dot product is < 0

Spherical Linear Interpolation (Slerp)

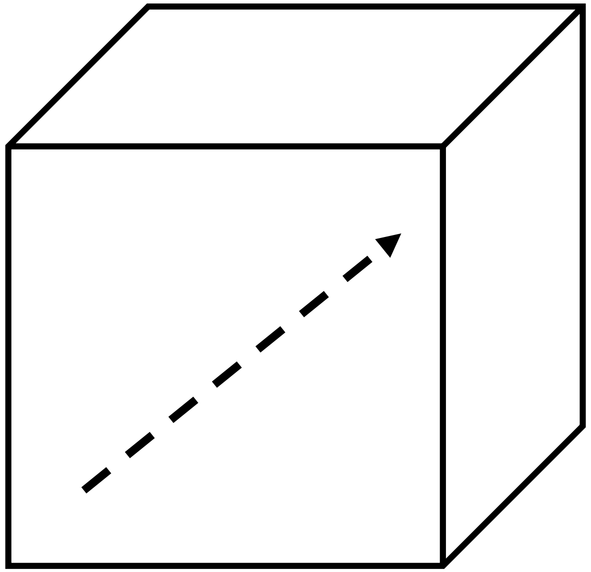
- If we have an intermediate position q_2, the interpolation from q_1-->q_2-->q_3 will not necessarily follow the same path as the interpolation from q_1 to q_3.

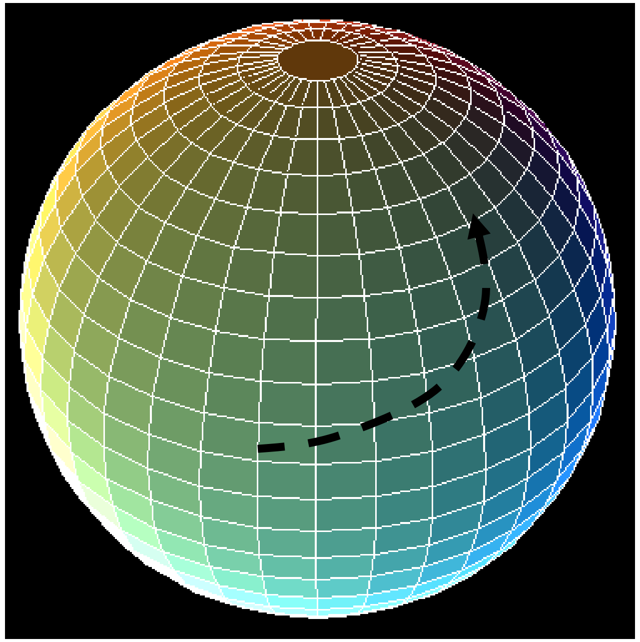
GD,9801,Fig3



Useful Analogies

Euclidean Space	⇔	4D Spherical Space
Position		Orientation
Linear interpolation		Spherical linear interpolation (Slerp)



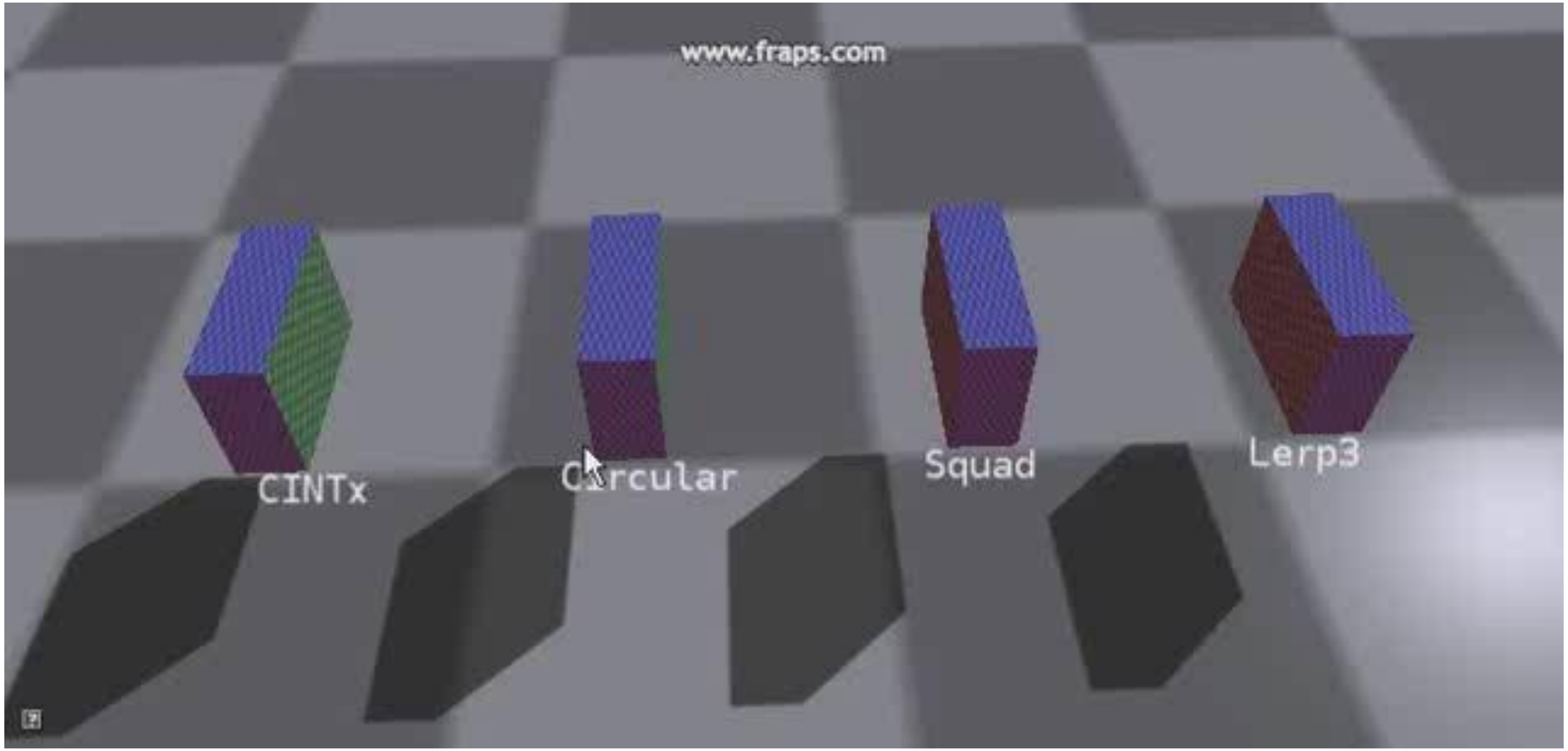


Advantages of Quaternion

- Good and smooth interpolation
- No gimbal lock
- Can be composed much more efficiently (requiring 8 multiplications and 4 divides)

- But
  - Impossible to visualize
  - Unintuitive
- **Good for internal representation of rotation!**

四元数插值DEMO



Cocos2d-x中的Quaternion类

- Quaternion () 构造一个四元数，初始化为(0,0,0,1)。
- Quaternion (float xx, float yy, float zz, float ww) 构造一个四元数。
- Quaternion getConjugated () const 得到共轭四元数。
- Quaternion getInversed () const 得到逆四元数。
- void multiply (const Quaternion &q) 右乘四元数q，并且将结果存储在this中。
- const Quaternion operator* (const Quaternion &q) const 计算该四元数和另一个四元数q的右乘乘积。
- … …

Cocos2d-x中的Quaternion类

- static void squad (const Quaternion & *q1*, const Quaternion & *q2*, const Quaternion & *s1*, const Quaternion & *s2*, float *t*, Quaternion * *dst*)
- 在一系列四元数中，使用球面样条插值。
- 球面样条插值能在不同的旋转姿态中进行平滑过渡，通常用于物体和摄像机的3D动画
- 注意: 输入必须是单位四元数。该方法不会自动归一化输入的四元数, 所以在计算之前必需自行归一化。
- 参数q1第一个四元数， q2第二个四元数， s1第一个控制点， s2第二个控制点。 t插值参数， dst存储插值结果。

有关四元数的参考文献

- Shoemake K. Animating rotation with quaternion curves. Computer Graphics, 1985, 19(3):245~254
- Pletincks D. Quaternion calculus as a basic tool in computer graphics. The Visual Computer, 1989,5(1):2~13
- Kim M J, Kim M S, Shin S Y. A general construction scheme for unit quaternion curves with simple high order derivatives. Computer Graphics, 1995, 29(3):369~376
- Kim M J, Kim M S, Shin S Y. A compact differential formula for the first derivative of a unit quaternion curve. The Journal of Visualization and Computer Animation, 1996,7(2):43~57

The End