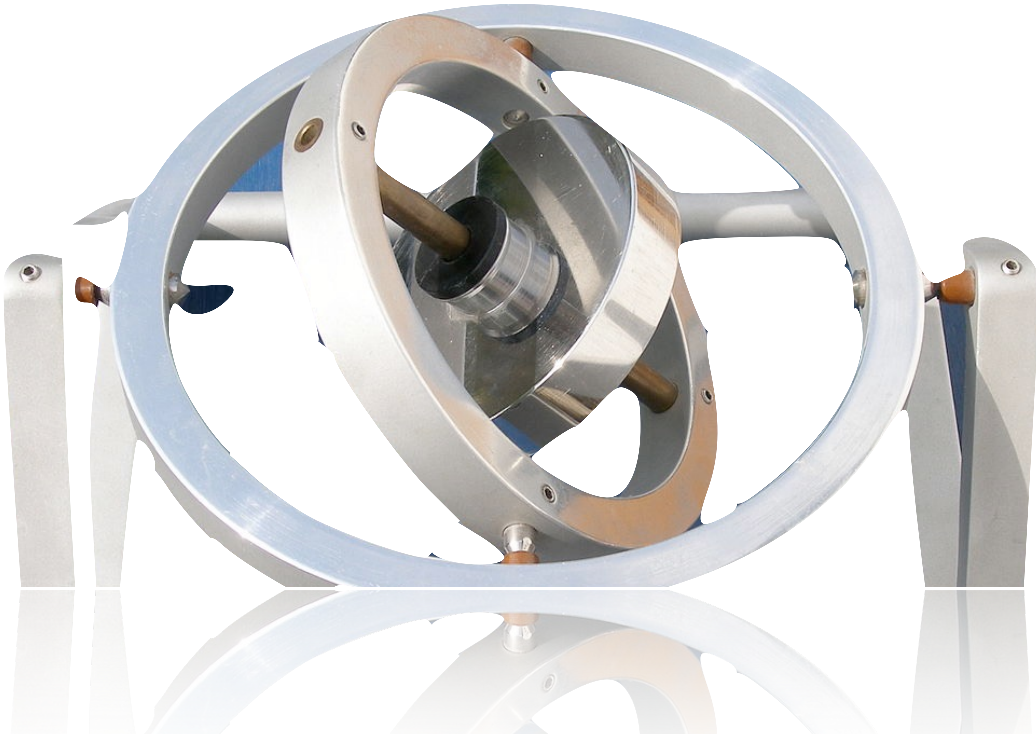
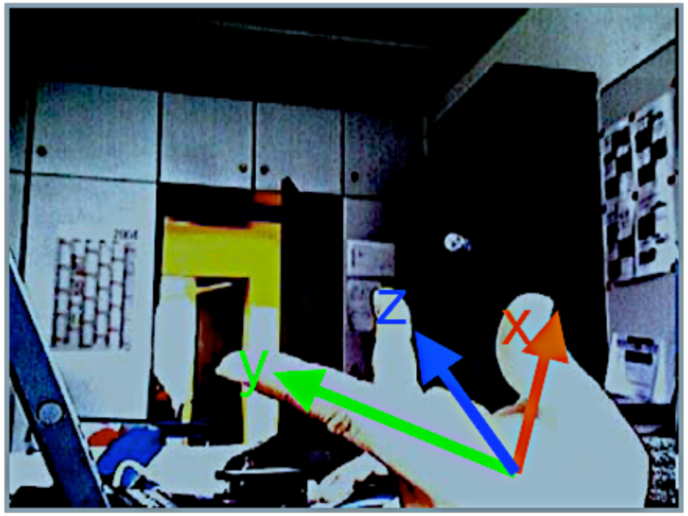
Orientation

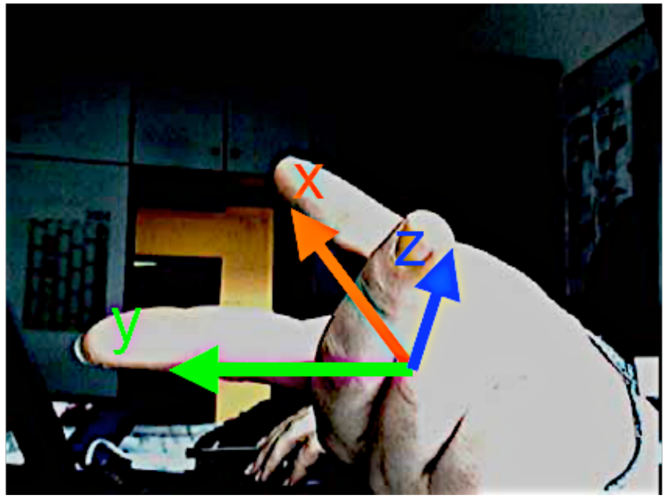
Thuerey / Game Physics



Coordinates in 3D

- Cartesian space
- Orthogonal system
  - –Unit vectors
  - –Left-handed
  - –Right-handed





Re-cap Affine Transformations

- Parallel lines stay parallel
- Angles, length, and handed-ness may change
- Examples
  - Translation
  - Rotation
  - Scaling
  - Mirroring
  - Shearing

Rotate, mirror, etc.

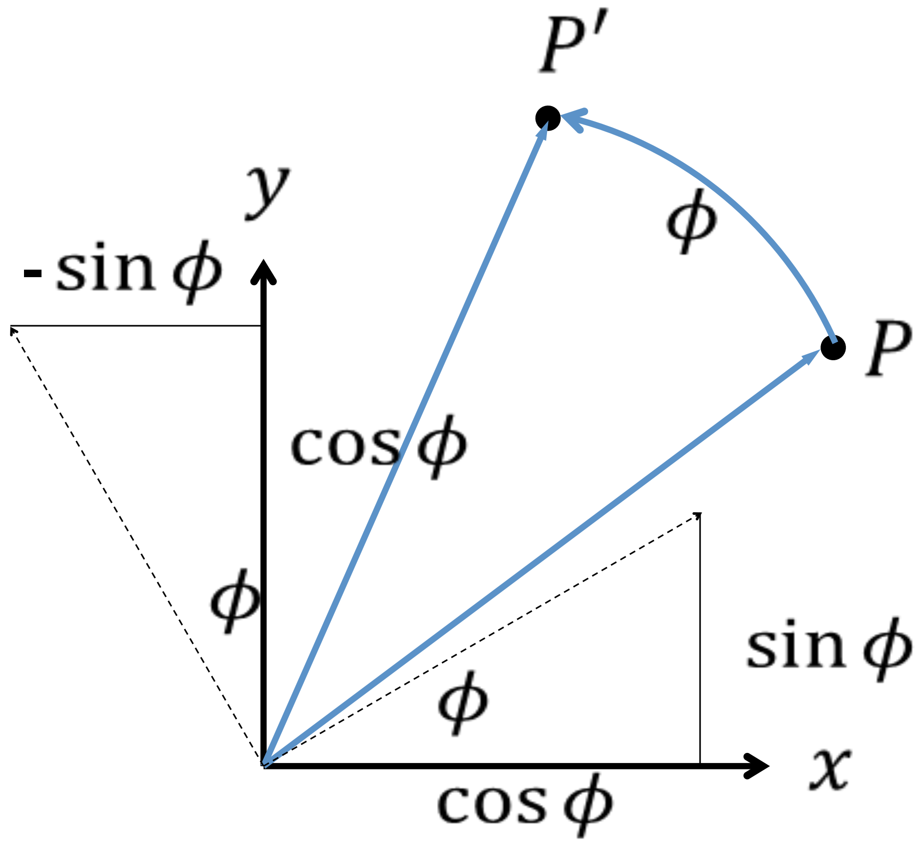
$$\left(\begin{array}{ccc|c} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

Translate

Homogenous part

2D Rotation

$$\mathbf{p}' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\phi) & \text{-}\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = R_\phi \mathbf{p}$$



3D Rotation around Cartesian Axis

$$R_{x,\phi} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \text{-}\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix}$$

$$R_{y,\phi} = \begin{pmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ \text{-}\sin(\phi) & 0 & \cos(\phi) \end{pmatrix}$$

$$R_{z,\phi} = \begin{pmatrix} \cos(\phi) & \text{-}\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation around Arbitrary Axis

- Rotate around axis ***a***:

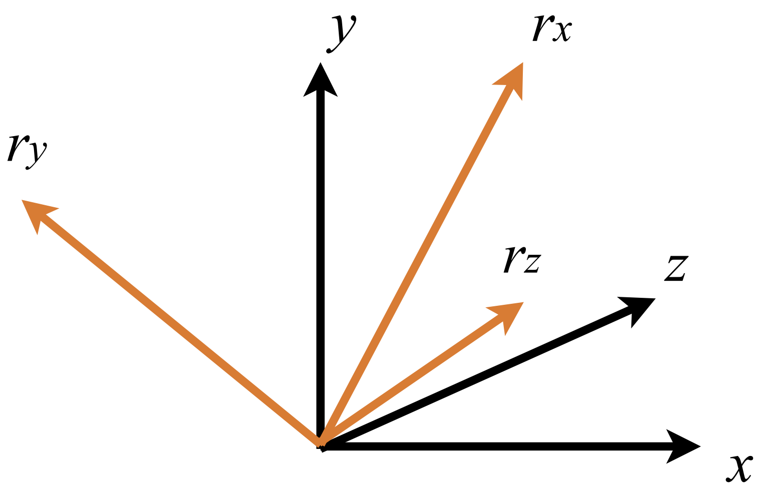
$$R_{a,\phi} = \begin{pmatrix} c + (1-c)a_x^2 & (1-c)a_x a_y - s a_z & (1-c)a_x a_z - s a_y \\ (1-c)a_x a_y - s a_z & c + (1-c)a_y^2 & (1-c)a_y a_z - s a_x \\ (1-c)a_x a_z - s a_y & (1-c)a_y a_z - s a_x & c + (1-c)a_z^2 \end{pmatrix}$$

$$\text{with } c = \cos(\phi), \text{ and } s = \sin(\phi),$$

- Note - no range check necessary for angle; sine and cosine take care of that

Rotation Matrix

$$\mathbf{p}' = R_\phi \mathbf{p} = \begin{pmatrix} r_{x_x} & r_{y_x} & r_{z_x} \\ r_{x_y} & r_{y_y} & r_{z_y} \\ r_{x_z} & r_{y_z} & r_{z_z} \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} \mathbf{r}_x & \mathbf{r}_y & \mathbf{r}_z \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

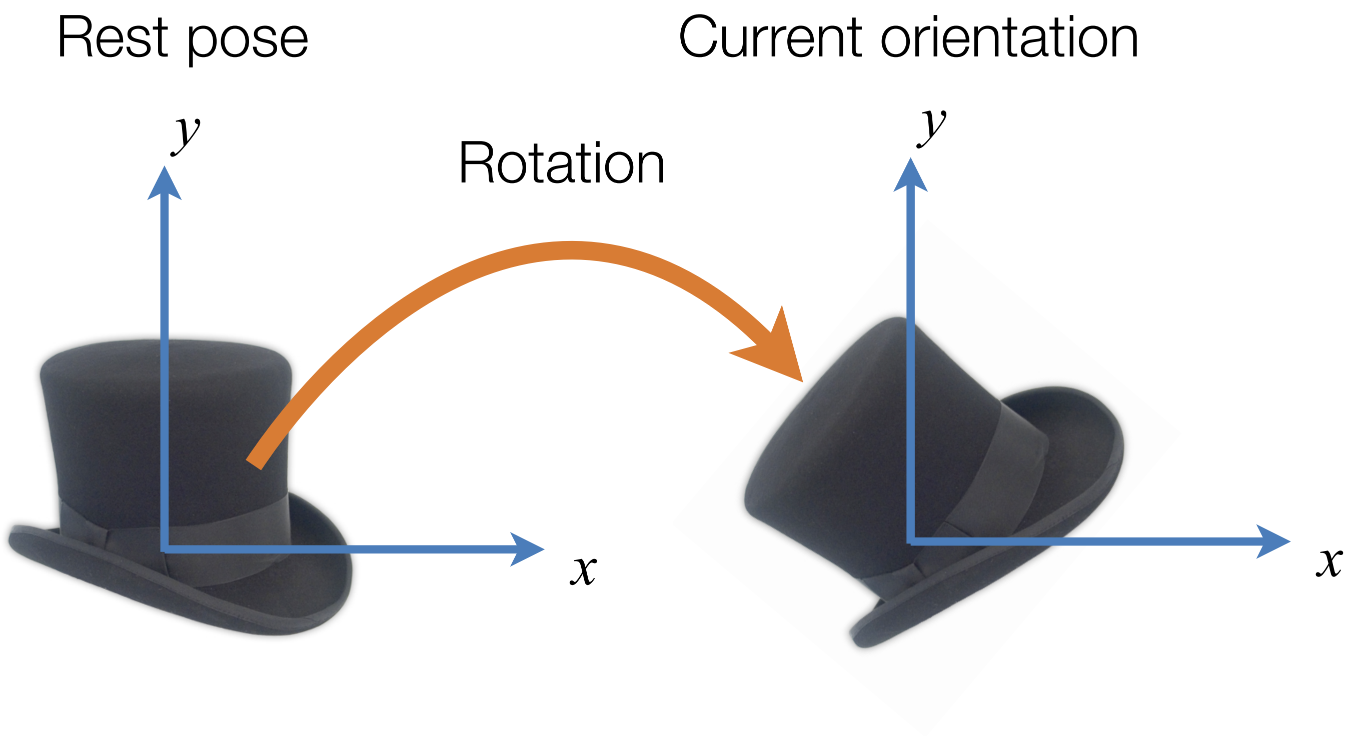


Rotation Matrix

- Columns are unit vectors, and mutually orthogonal
- Columns are formed by images of coordinate axes
- Inverse == transpose
- Preserves length, angles, handed-ness
- Points on rotation axis won’t change

Orientation

- Different semantics



Orientation

- Different representations possible:
  - Fixed Angles
  - Euler Angles
  - Quaternions
  - [Rotation Matrices]

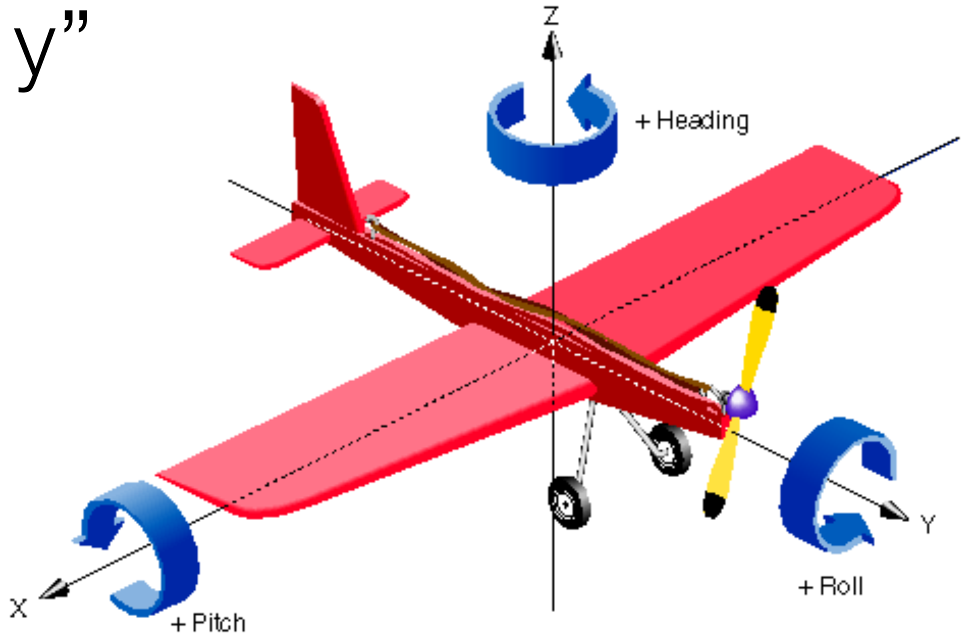
Orientation - Fixed Angles

- General rotation from three consecutive rotations about fixed axes
- E.g.: *x,y,z* or *y,z,x*
- Not: *x,x,y* !

$$R_{x,y,z} = R_z R_y R_x$$

Orientation - Euler Angles

- General rotation from three consecutive rotations about axes fixed to the object
- E.g. from aviation: z, x’, y”
  - –Heading / Yaw = rotation z
  - –Pitch = rotation x
  - –Roll = rotation y

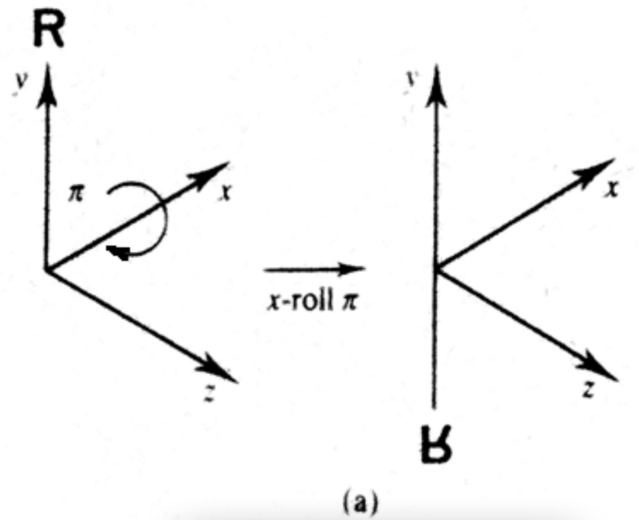


$$R_{z,x',y''} = R_z R_x R_y$$

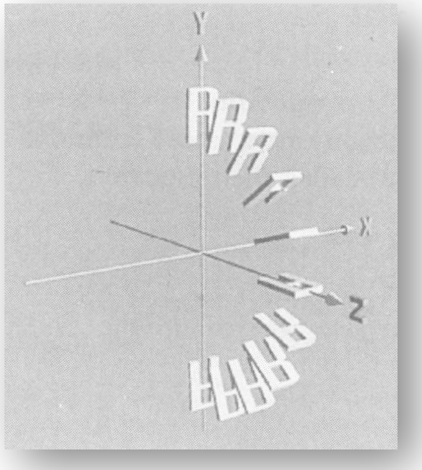
Inverse order, compared to fixed angles!

Problem - Both not unique...

Example: Fixed angles, x-y-z

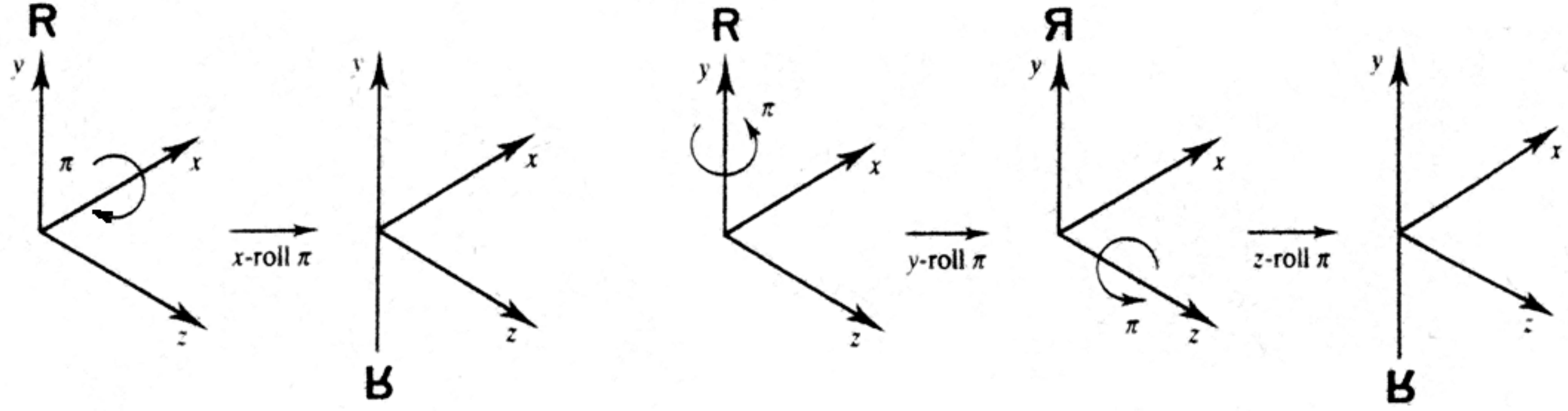


Componentwise linear interpolation $(0°, 0°, 0°) \rightarrow (180°, 0°, 0°)$



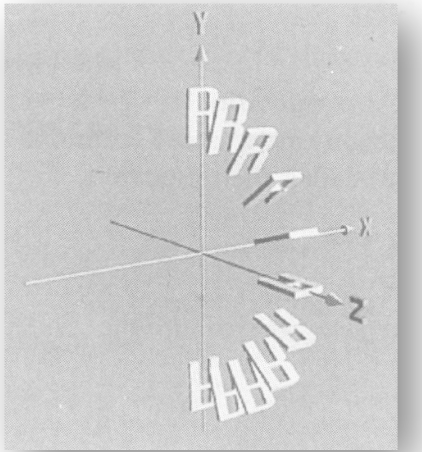
Problem1 - Both not unique...

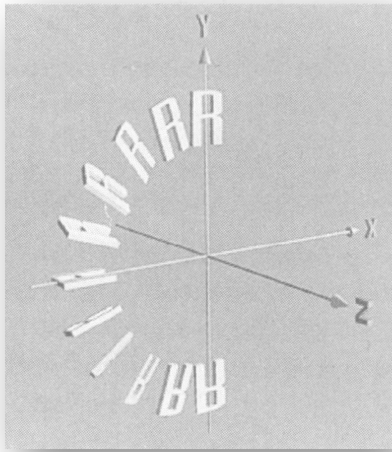
Example: Fixed angles, x-y-z



(a) (b)

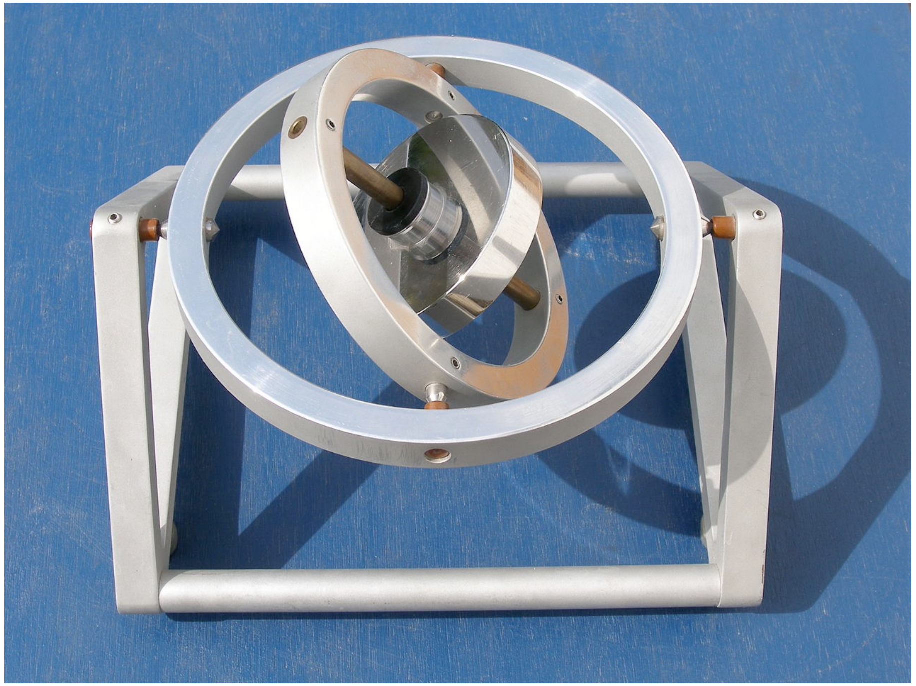
Componentwise linear interpolation $(0°, 0°, 0°) \rightarrow (180°, 0°, 0°)$



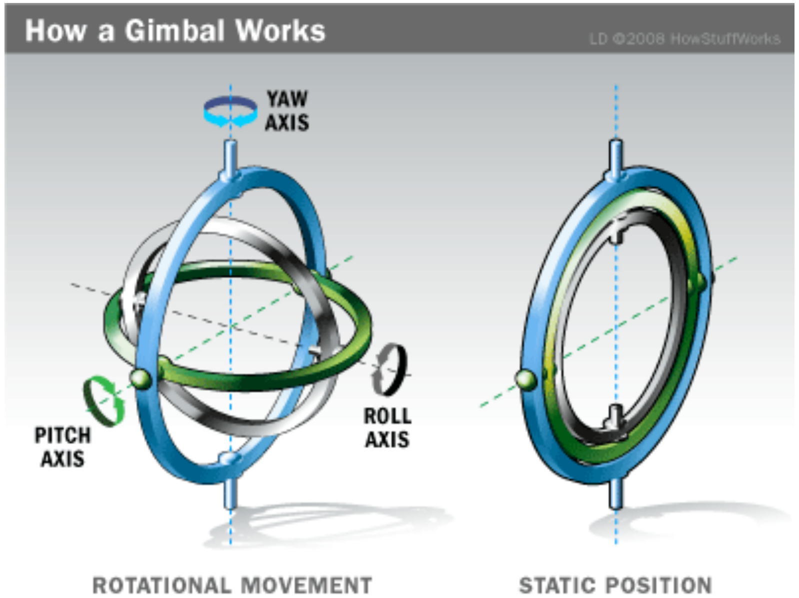


Componentwise linear interpolation $(0°, 0°, 0°) \rightarrow (0°, 180°, 180°)$

Problem2 - Gimbal Lock



“Kardanische Aufhängung”



Rotation Matrices for Orientations

- Over time, numerical errors can add up (e.g. concatenating many rotations)
  - Not a “real” rotation anymore
  - Can deform shape
- Many unnecessary degrees of freedom (9 for 3 unknowns)
- Re-orthonormalization necessary (Gram-Schmidt)

Quaternions

- Hamilton, *Elements of Quaternions*; 1866
- [Shoemake, *Animating rotation with quaternion curves*; SIGGRAPH 1985]
- Re-cap - imaginary numbers: $a+bi$, with $i*i=-1$
- Quaternions are an extension: three imaginary numbers

$$\mathbf{q} = (s, \mathbf{v}) = (s, x, y, z) = s1 + xi + yj + zk$$

(“real” ,“imaginary”)

$$s, x, y, z \in \mathbb{R}, v \in \mathbb{R}^3$$

Quaternion Multiplication

- Examples:
  - $i*i = -1$
  - Also $j*j = k*k = -1$
  - $ij = k$
  - $ji = -k$ ...

	1	*i*	*j*	*k*
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

·	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

- Quaternion multiplication is not commutative!

Quaternions for Rotation

- Unit-length quaternion of the form

$$\mathbf{q} = \big(\cos(\phi/2), \mathbf{n}\,\sin(\phi/2)\big) = \begin{pmatrix} \cos(\phi/2) \\ n_x\,\sin(\phi/2)i \\ n_y\,\sin(\phi/2)j \\ n_z\,\sin(\phi/2)k \end{pmatrix}$$

$$\text{with angle } \phi,\ \text{ axis } \mathbf{n} = (x, y, z)^T, |\mathbf{n}| = 1$$

- Rotate a 3D point *p* with:

$$\mathbf{q}(0, \mathbf{p})\tilde{\mathbf{q}} = \mathbf{p}'$$

- Using conjugate $\tilde{\mathbf{q}} = \widetilde{(s, \mathbf{v})} = (s, -\mathbf{v})$

Quaternions for Rotations

- Consecutive rotations:
  - Euler theorem: two consecutive rotations can be expressed as a single one
  - Product of two unit quaternions also again a unit quaternion (group property)

$$\mathbf{q}_2(\mathbf{q}_1(0,\mathbf{p})\mathbf{q}_1^{-1})\mathbf{q}_2^{-1} = (\mathbf{q}_2\mathbf{q}_1)(0,\mathbf{p})(\mathbf{q}_2\mathbf{q}_1)^{-1}$$

Discussion

EULER MATRIX QUAT

Euler / Fixed

– Advantages

- 3 degrees of freedom - 3 variables in representation
- Simple...

– Disadvantages

- Not a unique representation: infinitely many!
- Gimbal lock
- Singularities in general: it can be shown that 3 free variables are not enough! We need more...

Discussion

Matrices

– Advantages

- Efficient concatenation
- Fit into graphics frameworks (matrix code usually exists)
- No gimbal lock

– Disadvantages

- 9 variables need to be stored!
- Re-orthonormalization necessary
- Also not unique...

Discussion

Quaternions

– Advantages

- "Almost" unique representation of orientation (only ***q*** and -***q*** describe same orientation)
- No gimbal lock
- Efficient concatenation of multiple rotations

– Disadvantages

- Re-normalization necessary
- Not very intuitive...

Working with Quaternions

- Unusable for manually specifying orientations...
- Thus:
  - Manually specify (e.g. Euler or Matrices)
  - Convert rotation matrix to quaternion
  - Perform work (concatenate/integrate orientations)
  - Convert quaternion back to rotation matrix, e.g., apply to a set of points

Quaternions supported in DirectXMath

- Standard XMVECTOR data type
- Warning, order is different: (x,y,z,w) instead of commonly used (w,x,y,z)
- Set of tool functions, e.g.:
  - XMVECTOR XMQuaternionNormalize(XMVECTOR Q);
- Conversion functions, e.g.:
  - XMMatrixRotationQuaternion (quaternion to matrix)
  - XMQuaternionRotationMatrix (vice versa)

Addendum - Quaternion Operations

(Note - don't learn by heart)

- Addition:

$$q_1 + q_2 = (s_1, v_1) + (s_2, v_2) = (s_1 + s_2, v_1 + v_2)$$

- Multiplication:

$$q_1 q_2 = (s_1, v_1)(s_2, v_2) = (s_1 s_2 - v_1 \cdot v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2)$$

- Conjugate:

$$\bar{q} = \overline{(s, v)} = (s, -v)$$

- Scalar product:

$$q_1 \cdot q_2 = (s_1, v_1) \cdot (s_2, v_2) = s_1 s_2 + v_1 \cdot v_2$$
$$= s_1 s_2 + x_1 x_2 + y_1 y_2 + z_1 z_2$$

Addendum - Quaternion Operations

(Note - don't learn by heart)

- Norm:

$$|q| = \sqrt{q \cdot q} = \sqrt{s^2 + x^2 + y^2 + z^2}$$

- Norm under multiplication:

$$|q_1 q_2| = |q_1||q_2|$$

- Inverse of a quaternion:

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

- Unit quaternion q ($|q| = 1$) can always be written as

$$q = (s, v) = \left(\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right) n \right) \quad \text{with } |n| = 1$$

Conversions

(Note - don't learn by heart)

- Unit quaternion to rotation matrix:

$$R = \begin{pmatrix} 1-2y^2-2z^2 & 2xy-2sz & 2xz+2sy \\ 2xy+2sz & 1-2x^2-2z^2 & 2yz-2sx \\ 2xz-2sy & 2yz+2sx & 1-2x^2-2y^2 \end{pmatrix}$$

- Vice versa: $R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}$

$$s = 1/2\sqrt{1+R_{11}+R_{22}+R_{33}}\ ,\ x = \frac{R_{32}-R_{23}}{4s}\ ,\ y = \frac{R_{13}-R_{31}}{4s}\ ,\ z = \frac{R_{21}-R_{12}}{4s}$$