Announcements

Assignment 1 on the web page:
www.cs.cmu.edu/~jkh/anim_class.html

Due in 2.5 weeks, worth 10% of grade

Orientation Representation and Interpolation

Parent:
Chapter 2.2, 3.3

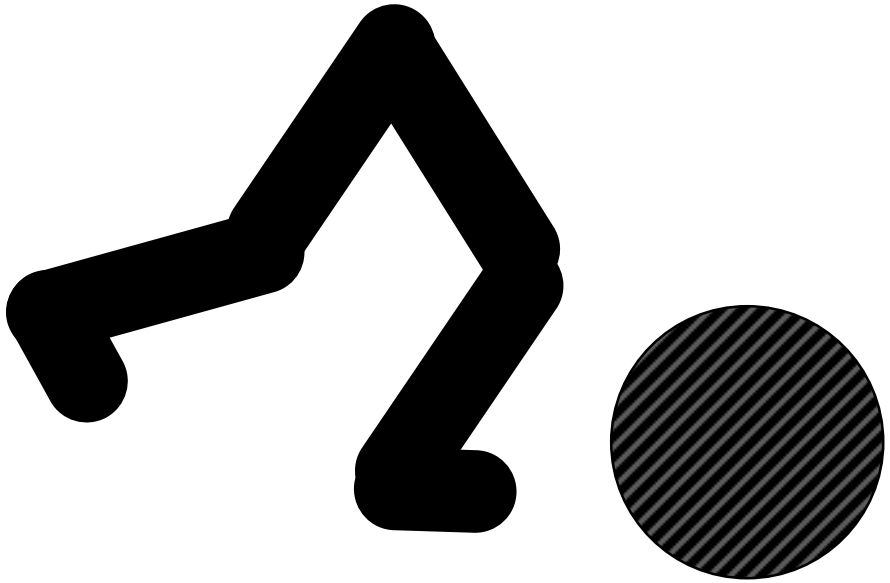
COMPUTER ANIMATION

15-497/15-861

02/22/02

Keyframing

- Last Class: how to interpolate positions/translations
- But we also need to orient things in 3D



Transformations (Review)

- Translation, scaling, and rotation:

$$P' = T + P \qquad \text{Translation}$$

$$P' = SP \qquad \text{Scaling}$$

$$P' = RP \qquad \text{Rotation}$$

- treat all transformations the same so that they can be easily combined (streamline software and hardware)
- P is a point of the model
- Transformation is for animation, viewing, modeling
- P' is where it should be drawn

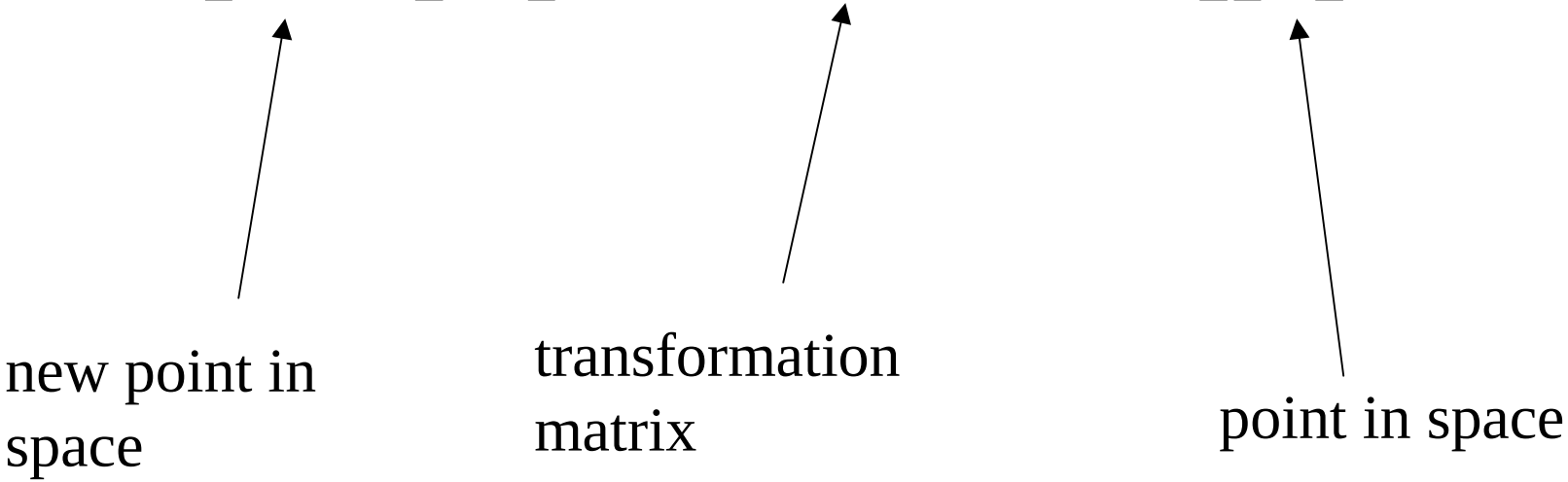
Homogenous Coordinates

- In graphics, we use homogenous coordinates for transformations
- 4x4 matrix can be used to represent translation, rotation, scaling, and other transformations
- We're dealing with 3-space, so the 4th coordinate is typically 1

$$\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, w\right) = [x, y, z, w] \qquad (x, y, z) = [x, y, z, 1]$$

Translation

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



Scaling

$$\begin{bmatrix} xS_x \\ yS_y \\ zS_z \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation

In the upper left 3x3 submatrix

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

X axis

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

Y axis

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

Z axis

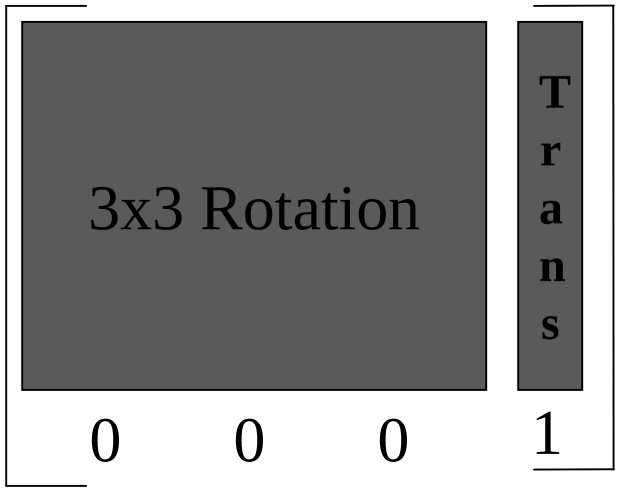
Composite Transformations

- We can now treat transformations as a series of matrix multiplications

$$P' = M_1 M_2 M_3 M_4 M_5 M_6 P$$

$$M = M_1 M_2 M_3 M_4 M_5 M_6$$

$$P' = MP$$



Back to Keyframing…

- In order to “move things” we need both translations and rotations
- Interpolating the translations was easy but what about rotations?

Interpolating Rotations

The upper left 3x3 submatrix of a transformation matrix is the rotation matrix

Maybe we can just interpolate the entries of that matrix to get the inbetween rotations?

Problem:

- Rows and columns are orthonormal (unit length and perpendicular to each other)
- Linear interpolation doesn't maintain this property, leading to nonsense for the inbetween rotations

Interpolating Rotation

Example:

- interpolate linearly from a positive 90 degree rotation about y to a negative 90 degree rotation about y

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

Linearly interpolate each component and halfway between, you get this…

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

No longer a rotation matrix---not orthonormal
Makes no sense!

Orientation Representations

Direct interpolation of transformation matrices is not acceptable…

Where does that leave us?

How best do we represent orientations of an object and interpolate orientation to produce motion over time?

- Rotation Matrices
- Fixed Angle
- Euler Angle
- Axis Angle
- Quaternions

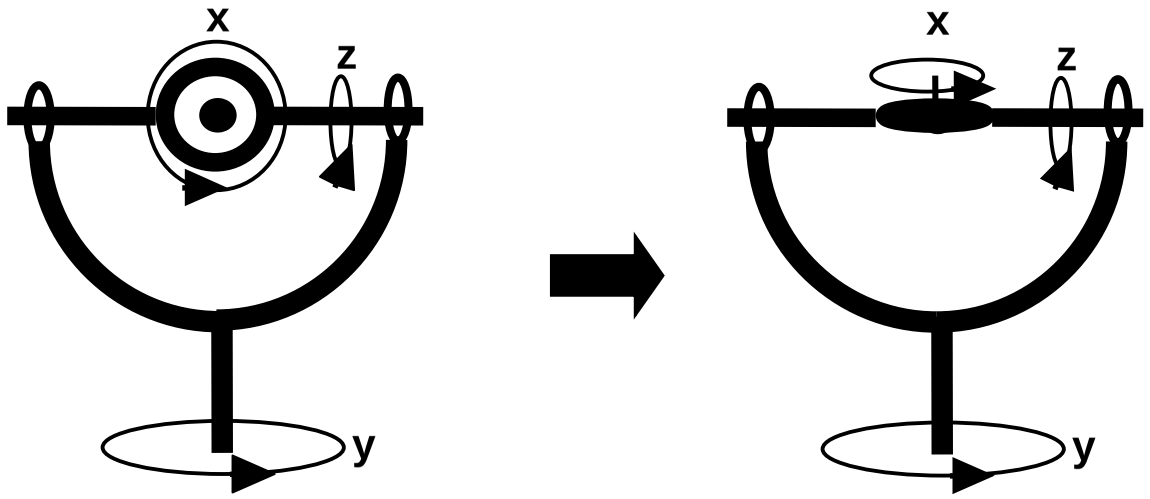
Fixed Angle Representation

- Angles used to rotate about fixed axes
- Orientations are specified by a set of 3 ordered parameters that represent 3 ordered rotations about fixed axes, i.e. first about x, then y, then z
- Many possible orderings, don’t have to use all 3 axes, but can’t do the same axis back to back

Fixed Angle

- A rotation of 10,45, 90 would be written as
  - Rz(90) Ry(45), Rx(10) since we want to first rotate about x, y, z. It would be applied then to the point P…. RzRyRx P
- Problem occurs when two of the axes of rotation line up. Gimbal Lock

Gimbal Lock



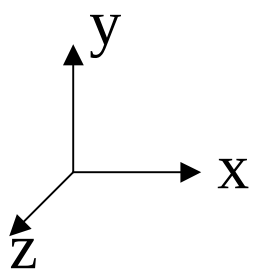
Gimbal **Locked Gimbal**

A *Gimbal* is a hardware implementation of Euler angles (used for mounting gyroscopes, expensive globes)

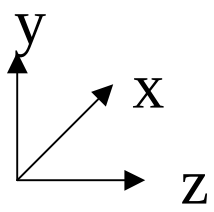
Gimbal lock is a basic problem with representing 3-D rotations using Euler angles or fixed angles

Gimbal Lock—Shown another way

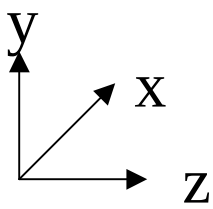
- A 90 degree rotation about the y axis aligns the first axis of rotation with the third.



Rotx(0)



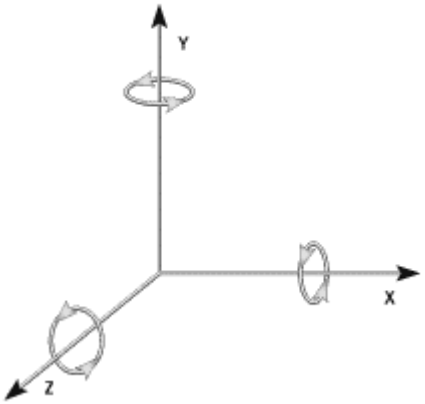
Roty(90)



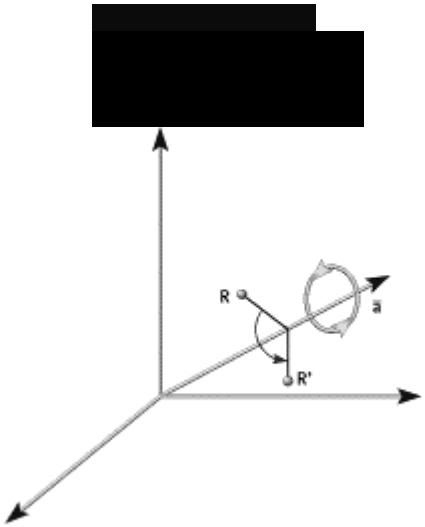
Rotz(0)

- Incremental changes in x,z produce the same results – lost a degree of freedom

Interpolating Rotations



Euler angles



Axis-angle

Q: What kind of compound rotation do you get by successively turning about each of the 3 axes at a constant rate?

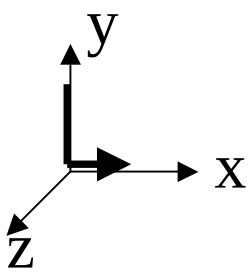
A: Not the one you want

Example

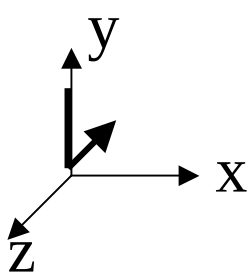
- Especially a problem if interpolating say…

(0,90,0) ⟶ (90, 45, 90)

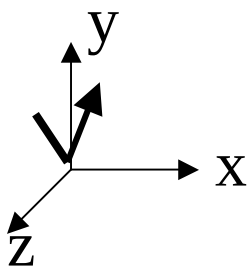
Just a 45 degree rotation from one orientation to the next, so we expect 90, 22.5, 90, but get 45, 67.5, 45



Initial Orientation (object space)



(0,90,0)



(90, 45, 90)

Euler Angles

- Same as fixed axis, except now, the axes move with the object
- roll, pitch, yaw of an aircraft
- Euler Angle rotations about moving axes written in reverse order are the same as the fixed axis rotations.

$$\underbrace{R_x(\alpha)R_y(\beta)R_z(\gamma)P}_{\text{Euler}} = \underbrace{R_z(\gamma)R_y(\beta)R_x(\alpha)P}_{\text{Fixed}}$$

Same problem with Gimbal Lock

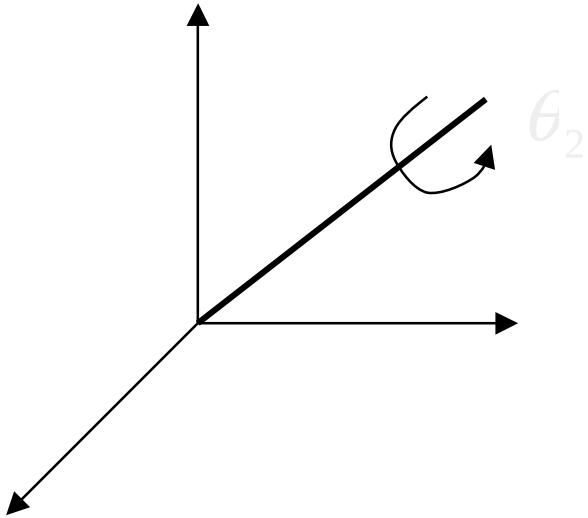
Axis Angle

Euler's Rotation Theorem:

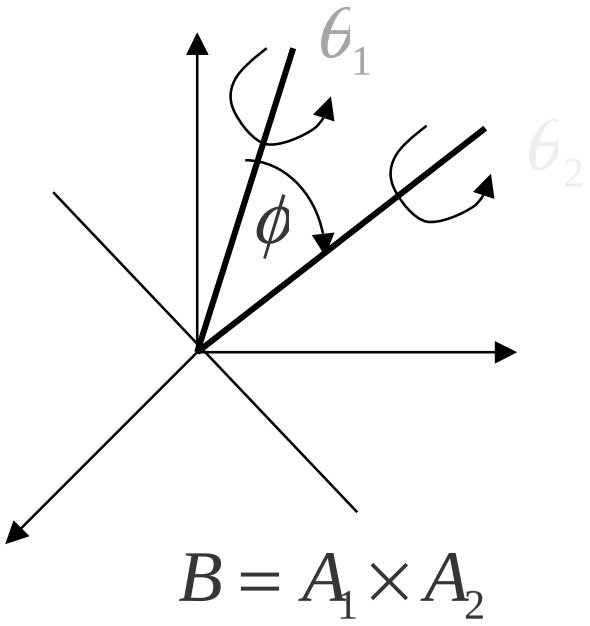
Any orientation can be represented by a 4-tuple

- angle, vector(x,y,z) where the angle is the amount to rotate by and the vector is the axis to rotate about

- Can interpolate the angle and axis separately



Axis Angle Interpolation



$$B = A_1 \times A_2$$

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

$$A_k = R_B(k\phi)A_1$$

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

Axis Angle

- Can interpolate the angle and axis separately
- No gimbal lock
- But, can’t efficiently compose rotations…must convert to matrices first

Quaternions

- Good interpolation
- Can be multiplied (composed)
- No gimbal lock

Quaternions

- 4-tuple of real numbers
  - s,x,y,z or [s,v]
  - s is a scalar
  - v is a vector
- Same information as axis/angle but in a different form

$$q = Rot_{\theta(x,y,z)} = \left[\cos(\theta/2), \sin(\theta/2) \bullet (x, y, z)\right]$$

Quaternion Math

Addition:

$$[s_1, v_1] + [s_{2,} v_2] = [s_1 + s_2, v_1 + v_2]$$

Multiplication:

$$[s_1, v_1] \cdot [s_{2,} v_2] = [s_1 \cdot s_2 - v_1 \bullet v_{2,} s_1 \cdot v_2 + s_2 \cdot v_1 \times v_{2]}]$$

Multiplication is not commutative but is associative
(just like transformation matrices, as you would expect)

$$q_1 q_2 \neq q_2 q_1$$

$$(q_1 q_2) q_3 = q_1 (q_2 q_3)$$

Quaternion Math

A point in space is represented as $[0, v]$

$[1, (0,0,0)]$ multiplicative identity

$$q^{-1} = (1/\|q\|)^2 \cdot [s, -v]$$

$$\textit{where} \quad \|q\| = \sqrt{s^2 + x^2 + y^2 + z^2}$$

$q \cdot q^{-1} = [1, (0,0,0)]$ the unit length quaternion
(and multiplicative identity)

Quaternion Rotation

To rotate a vector, v using quaternions

- represent the vector as [0,v]
- represent the rotation as a quaternion, q

$$q = Rot_{\theta,(x,y,z)} = [\cos(\theta/2), \sin(\theta/2) \cdot (x, y, z)]$$

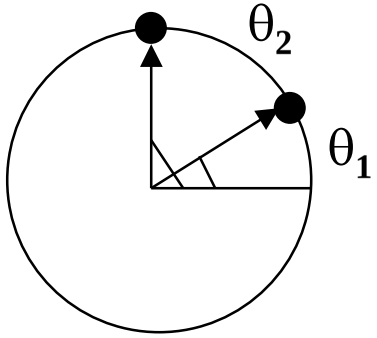
$$v' = Rot_q(v) = q \cdot v \cdot q^{-1}$$

Can compose rotations as well

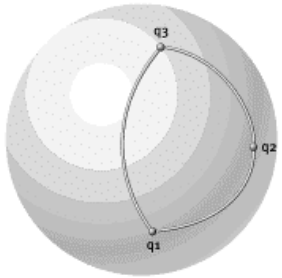
Looks good so far…we can easily specify and compose rotations!

Quaternion Interpolation

- We can think of rotations as lying on an n-D unit sphere



1-angle (θ) rotation
(unit circle)



2-angle (θ-ϕ) rotation
(unit sphere)

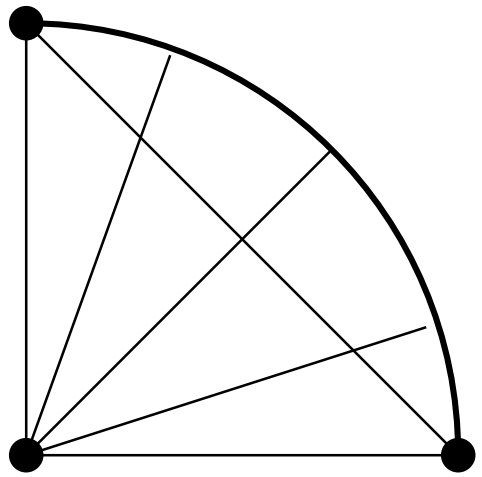
- Interpolating rotations means moving on n-D sphere
  - Can encode position on sphere by unit vector
  - How about 3-angle rotations?

Quaternion Interpolation

- Interpolating quaternions produces better results than Euler angles
- A quaternion is a point on the 4-D unit sphere
  - interpolating rotations requires a unit quaternion at each step - another point on the 4-D sphere
  - move with constant angular velocity along the great circle between the two points
  - Spherical Linear intERPolation (SLERPing)
- Any rotation is given by 2 quaternions, so pick the shortest SLERP
- To interpolate more than two points:
  - solve a non-linear variational constrained optimization (numerically)
- Further information: Ken Shoemake in the Siggraph '85 proceedings (*Computer Graphics*, V. 19, No. 3, P.245)

Quaternion Interpolation

- Direct linear interpolation does not work
  - Linearly interpolated intermediate points are not uniformly spaced when projected onto the circle



- Use a special interpolation technique
  - Spherical linear interpolation
  - viewed as interpolating over the surface of a sphere

$$slerp(q1, q2, u)$$

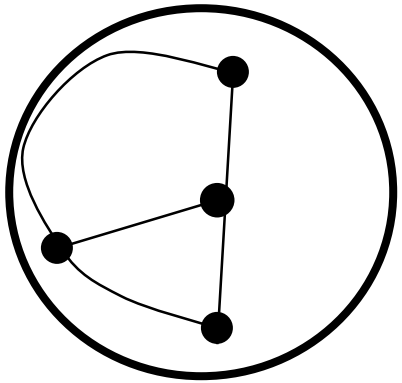
$$= ((\sin((1-u)\cdot\theta))/(\sin\theta))\cdot q_1 + (\sin(u\cdot\theta))/(\sin\theta)\cdot q_2$$

- Normalize to regain unit quaternion

Two Representations of a Rotation

A quaternion and its negation [-s,-v] represent the same rotation:

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



Have to go the short way around!

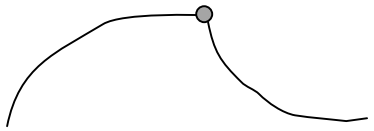
$$\cos(\theta) = q_1 \bullet q_2 = s_1 s_2 + v_1 \bullet v_2$$

$$\text{if } \cos(\theta) > 0 \Rightarrow q_1 \rightarrow q_2 \text{ shorter}$$

$$\text{else } q_1 \rightarrow -q_2 \text{ shorter}$$

Quaternion Interpolation

- As in linear interpolation in Euclidean space, we can have first order discontinuity



Solution is to formulate a cubic curve interpolation—see book for details

Quaternion Rotation

The rotation matrix corresponding to a quaternion,q, is

$$
\begin{aligned}
q &= Rot_{\theta,(x,y,z)} \\
&= [\cos(\theta/2), \sin(\theta/2)\cdot(x,y,z)] \\
&= [s,a,b,c]
\end{aligned}
$$

$$
\begin{bmatrix}
1-2b^2-2c & 2ab+2sc & 2ac-2sb \\
2ab-2sc & 1-2a^2-2c^2 & 2bc+2sc \\
2ac+2sb & 2bc-2sa & 1-2a^2-2b^2
\end{bmatrix}
$$

Rotations in Reality

- We can convert to/from any of these representations
  - but the mapping is not one-to-one
- Choose the best representation for the task
  - input: Euler angles
  - interpolation: quaternions
  - composing rotations: quaternions, orientation matrix
  - drawing: orientation matrix

Problems with Interpolation

- Splines don’t always do the right thing
- Classic problems
  - Important constraints may break between keyframes
    - » feet sink through the floor
    - » hands pass through walls
  - 3D rotations
    - » Euler angles don’t always interpolate in a natural way
- Solutions:
  - More keyframes!
  - Quaternions help fix rotation problems

Summary of Keyframing

- We know how to move points in 3D – translation and rotation
- So we can set keyframes – position, orientation
- We can describe interpolation methods – linear, cubic polynomial
- We can control interpolation speed with speed curves and arclength reparameterization

What's Next?
Kinematics & Inverse Kinematics

- We need help in positioning the object to be animated
- Kinematics
  - gives motions in terms of joint angles, velocities, and positions
  - But many things are hard to specify this way
- *Inverse* kinematics
  - determine joint angles from positions
  - e.g. "calculate the shoulder, elbow, and wrist rotation parameters in order to put the hand here"
  - better for interaction
  - sometimes underdetermined (i.e. many combinations of joint angles to achieve a given end result)
  - used in robotics