Animation

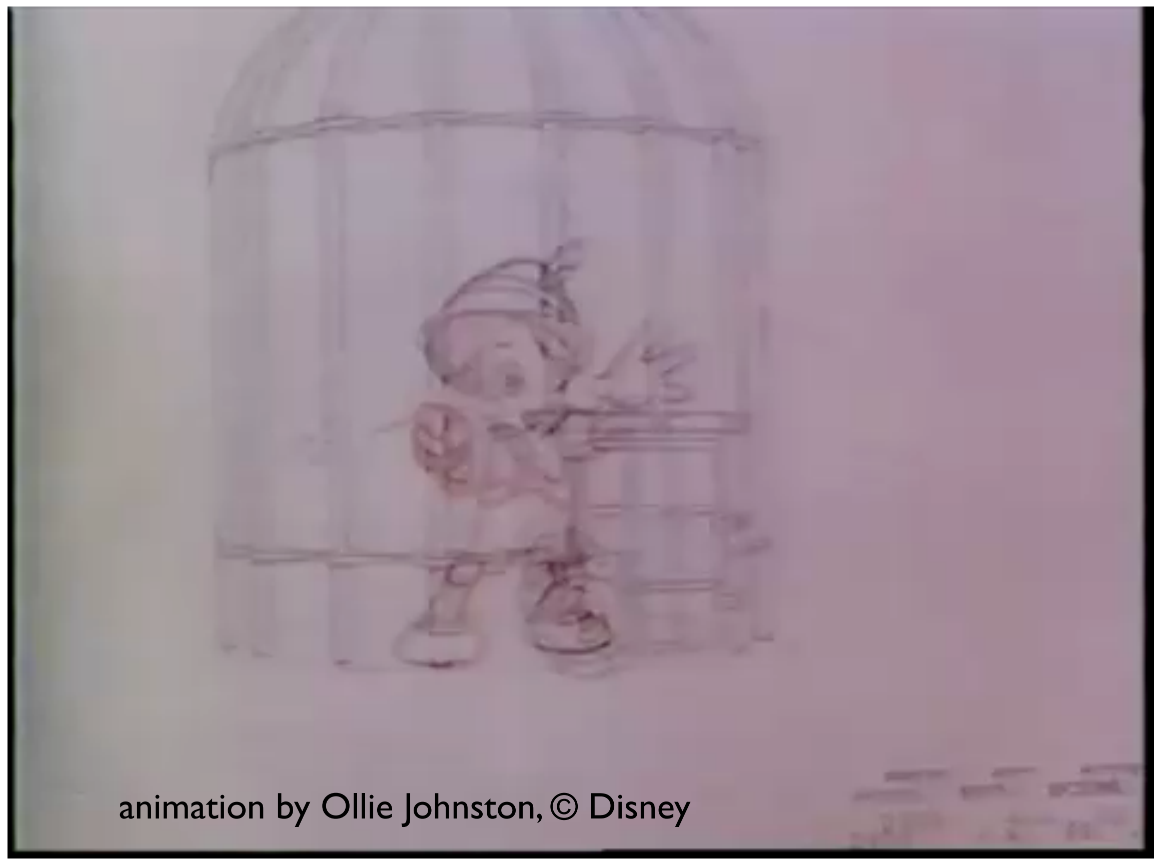
CS 4620 Lecture 32

What is animation?

- Modeling = specifying shape
    - using all the tools we've seen: hierarchies, meshes, curved surfaces…
- Animation = specifying shape as a function of time
    - just modeling done once per frame?
    - yes, but need smooth, concerted movement

Keyframes in hand-drawn animation

- End goal: a drawing per frame, with nice smooth motion
- “Straight ahead” is drawing frames in order
  - But it is hard to get a character to land at a particular pose at a particular time
- Instead use *key frames* to plan out the action
  - draw important poses first, then fill in the *in-betweens*



animation by Ollie Johnston, © Disney

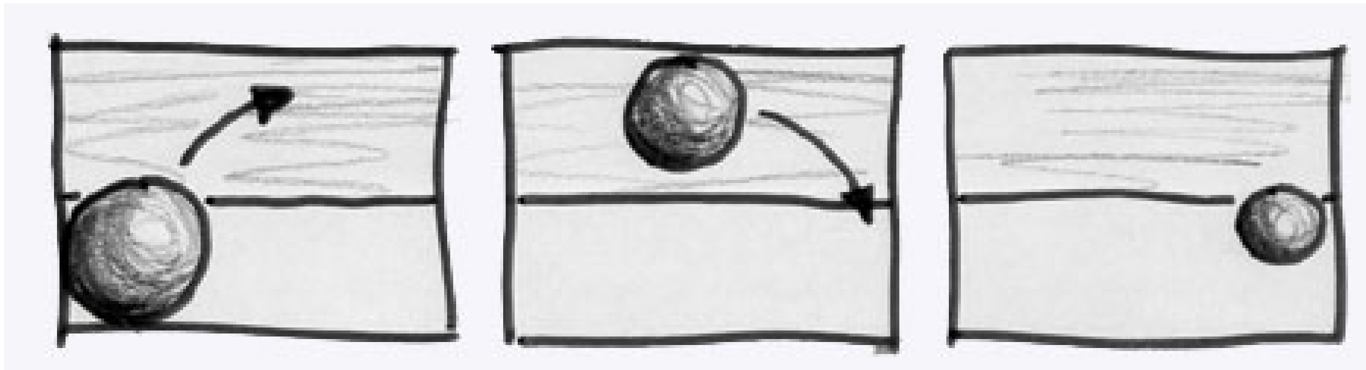
Keyframes in computer animation

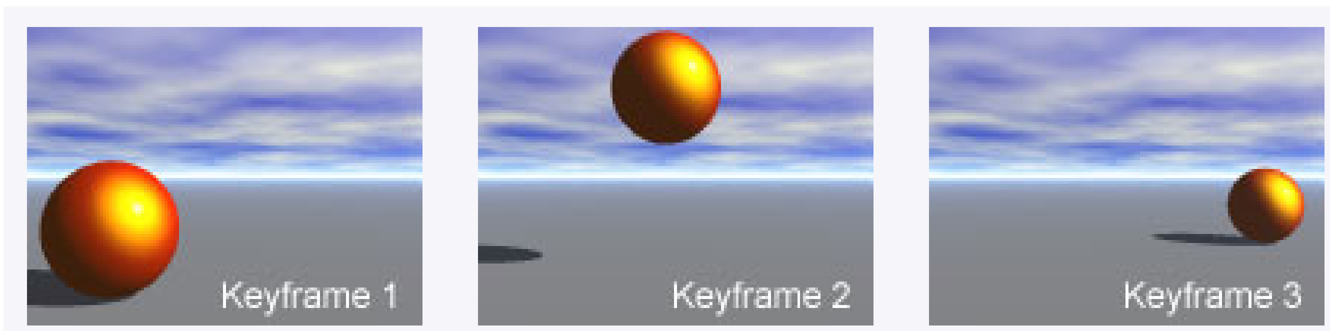
- Just as with hand-drawn animation, adjusting the model from scratch for every frame would be tedious and difficult
- Same solution: animator establishes the keyframes, software fills in the in-betweens
- Two key ideas of computer animation:
  - create *high-level* controls for adjusting geometry
  - interpolate these controls over time between keyframes

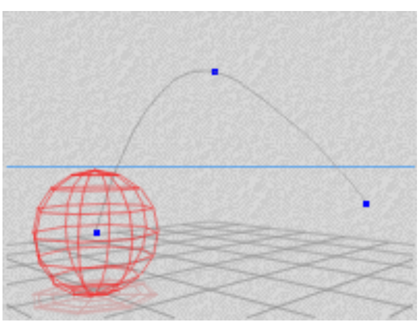
The most basic animation control

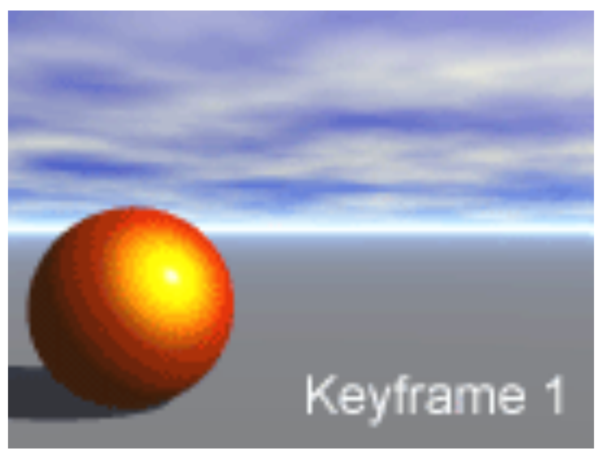
- Affine transformations position things in modeling
- Time-varying affine transformations move things around in animation
- A hierarchy of time-varying transformations is the main workhorse of animation
    - and the basic framework within which all the more sophisticated techniques are built

Keyframe animation





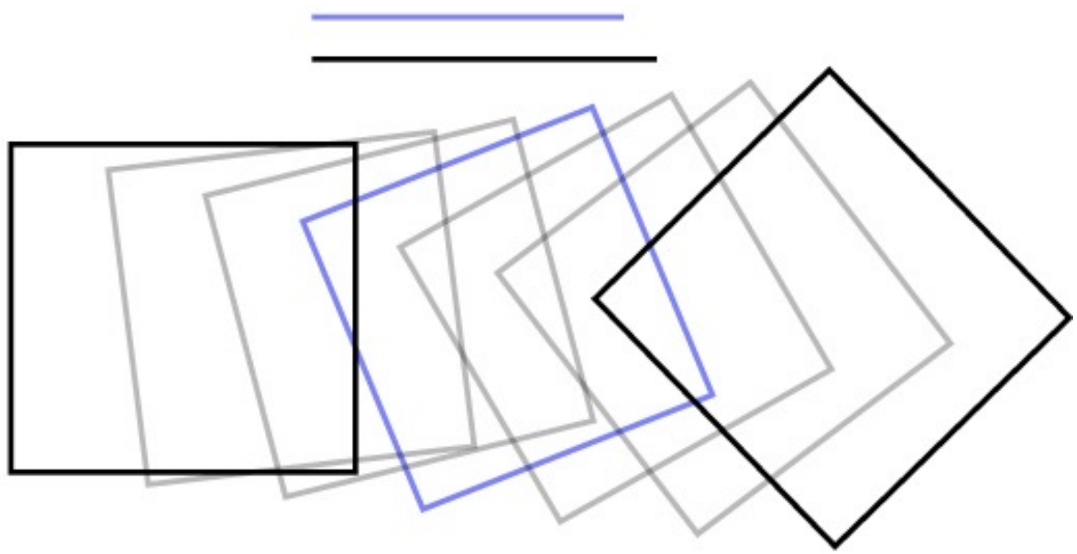




[Bryce Tutorial http://www.cadtutor.net/dd/bryce/anim/anim.html]

Interpolating transformations

- Move a set of points by applying an affine transformation
- How to animate the transformation over time?
  - Interpolate the matrix entries from keyframe to keyframe?
    - This is fine for translations but bad for rotations



Animation

- Industry production process leading up to animation
- What animation is
- How animation works (very generally)
- Artistic process of animation
- Further topics in how it works

Approaches to animation

- Straight ahead

  Draw/animate one frame at a time

  Can lead to spontaneity, but is hard to get exactly what you want

- Pose-to-pose

  Top-down process:

  - Plan shots using storyboards
  - Plan key poses first
  - Finally fill in the in-between frames

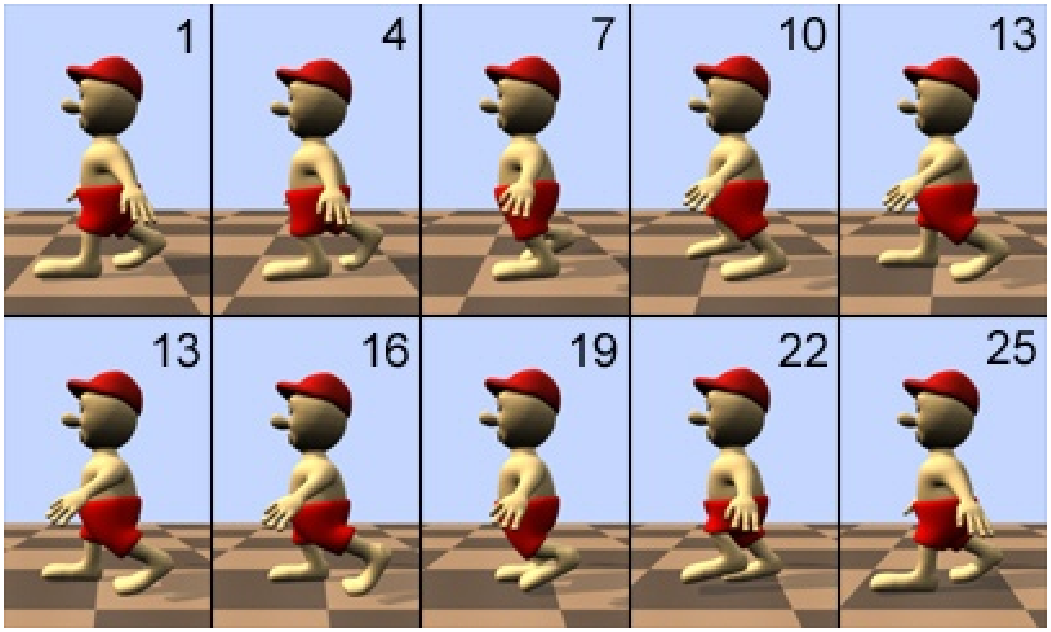
Pose-to-pose animation planning

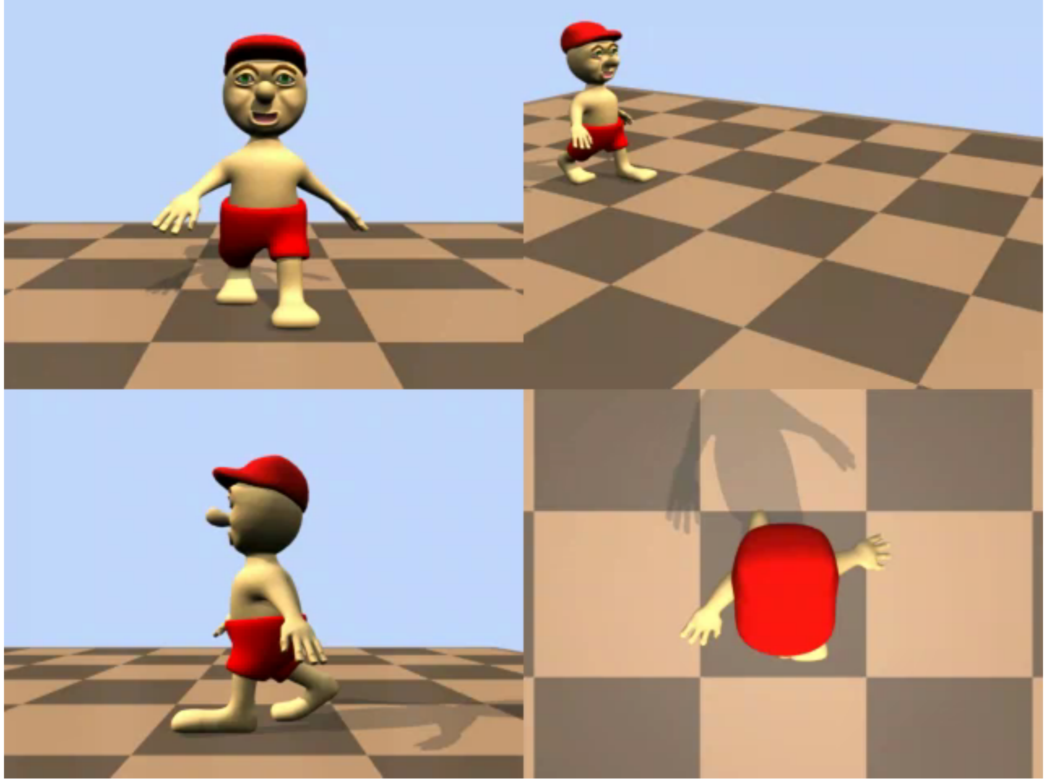
– First work out poses that are key to the story
– Next fill in animation in between

Keyframe animation

- Keyframing is the technique used for pose-to-pose animation
  - Head animator draws key poses—just enough to indicate what the motion is supposed to be
  - Assistants do "in-betweening" and draws the rest of the frames
  - In computer animation substitute "user" and "animation software"
- *Interpolation* is the main operation

Walk cycle



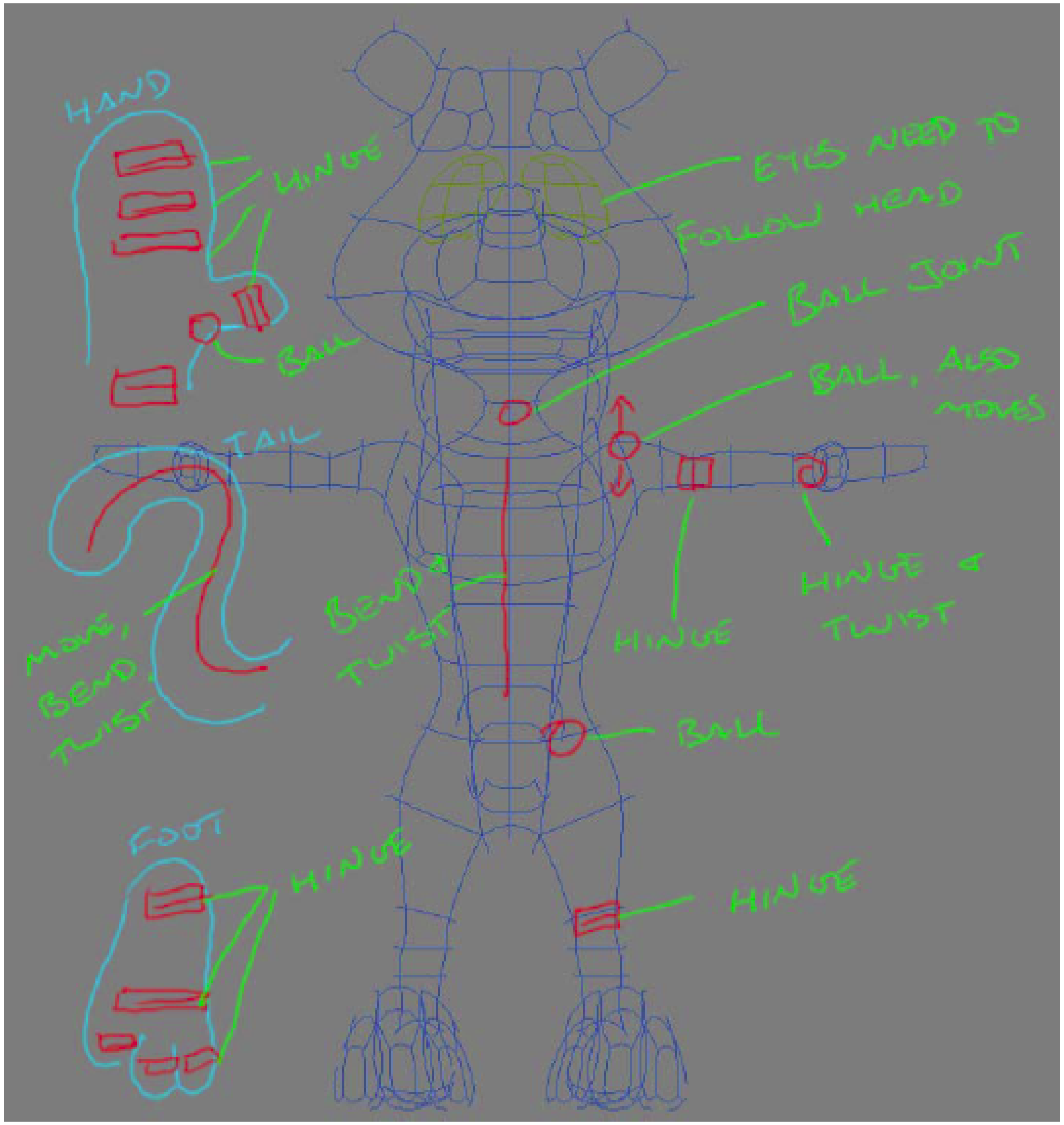


[Christopher Lutz http://www.animationsnippets.com]

Controlling geometry conveniently

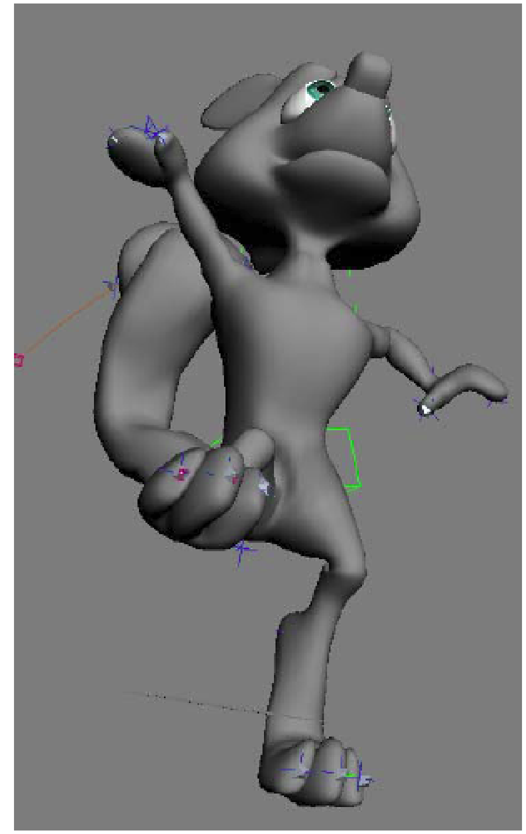
- Could animate by moving every control point at every keyframe
  - This would be labor intensive
  - It would also be hard to get smooth, consistent motion
- Better way: animate using smaller set of meaningful *degrees of freedom* (DOFs)
  - Modeling DOFs are inappropriate for animation
    - E.g. "move one square inch of left forearm"
  - Animation DOFs need to be higher level
    - E.g. "bend the elbow"

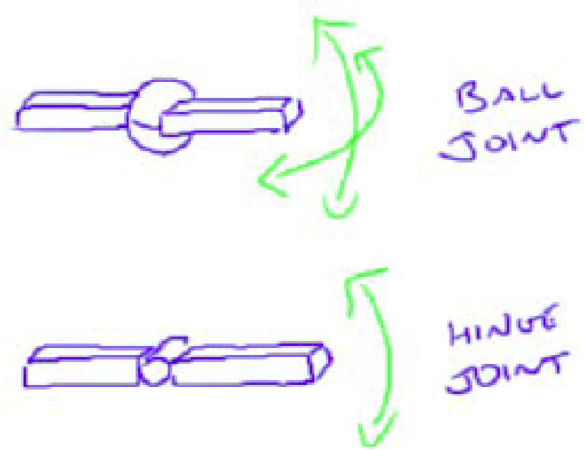
Character with DOFs



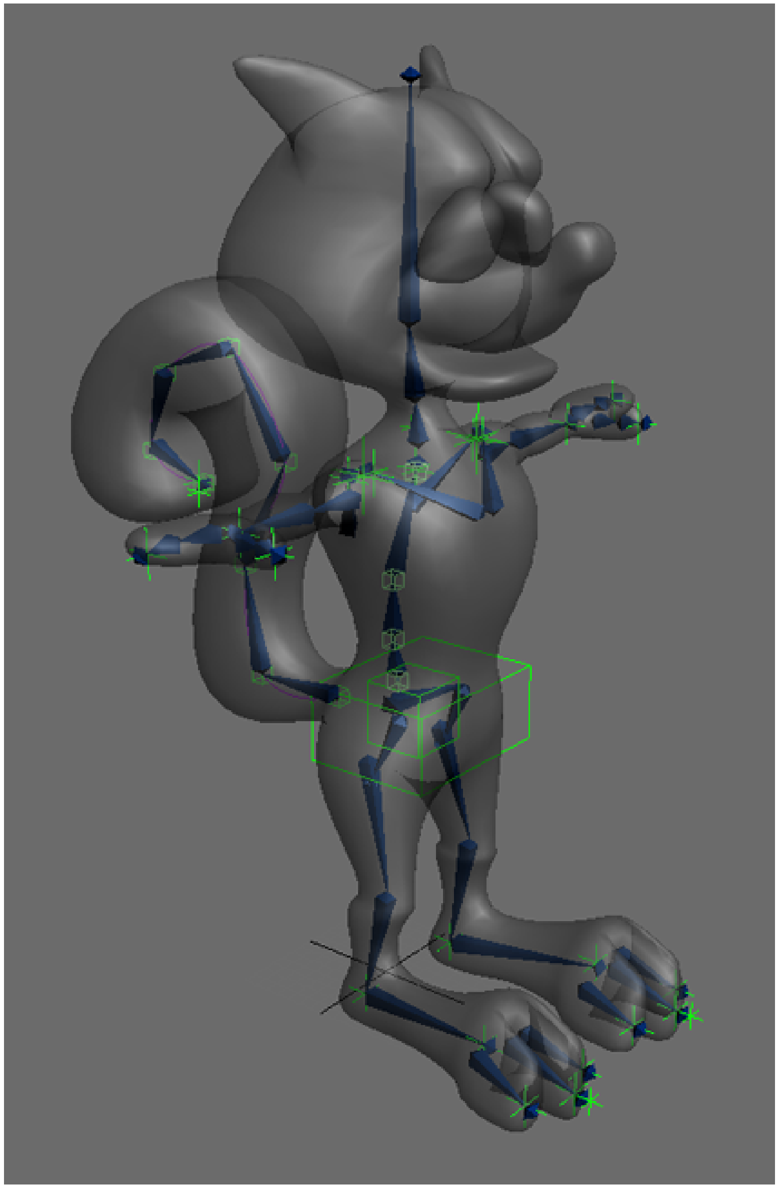
A visual description of the possible movements for the squirrel

[Greenberg/Pellacini | CIS 565]





Rigged character



[CIS 565 staff]

- Surface is deformed by a set of *bones*
- Bones are in turn controlled by a smaller set of *controls*
- The controls are useful, intuitive DOFs for an animator to use

Keyframe animation

- Keyframing is the technique used for pose-to-pose animation
  - –User creates key poses—just enough to indicate what the motion is supposed to be
  - –Interpolate between the poses

Rigid motion: the simplest deformation

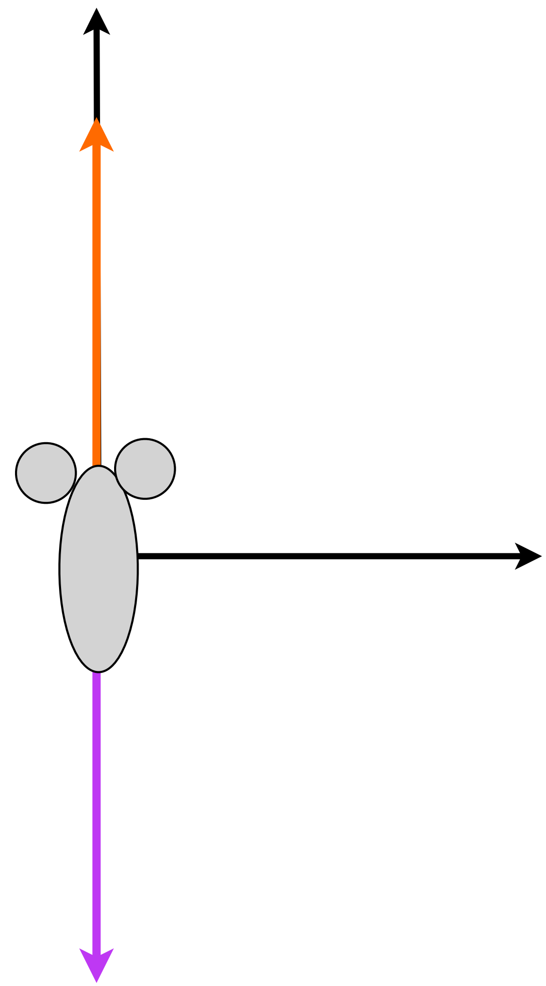
- Move a set of points by applying an affine transformation
- How to animate the transformation over time?
  - –Interpolate the matrix entries from keyframe to keyframe?
    - Translation: ok
      - – start location, end location, interpolate
    - Rotation: not so good

Rigid motion: the simplest deformation

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

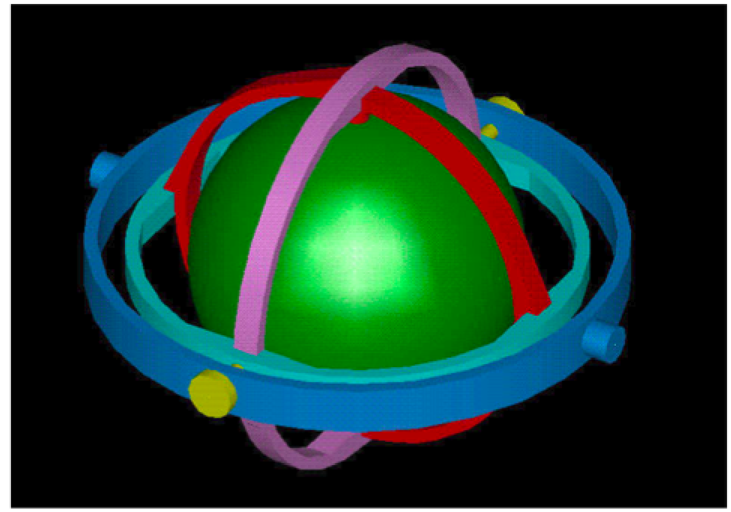
start

end



Parameterizing rotations

- Euler angles
  - –Rotate around x, then y, then z
  - –Problem: gimbal lock
    - If two axes coincide, you lose one DOF



- Unit quaternions
  - –A 4D representation
  - –Good choice for interpolating rotations

Quaternions

- Remember that
  - –Orientations can be expressed as rotation
    - Why?
      - –Start in a default position (say aligned with z axis)
      - –New orientation is rotation from default position
  - –Rotations can be expressed as (axis, angle)

- Quaternions let you express (axis, angle)

Quaternions for Rotation

- A quaternion is an extension of complex numbers

$$q = (s, v) = (s, v_1, v_2, v_3)$$

- Review of complex numbers

$$z = a + bi$$
$$z' = a - bi$$
$$||z|| = \sqrt{z.z'} = \sqrt{a^2 + b^2}$$

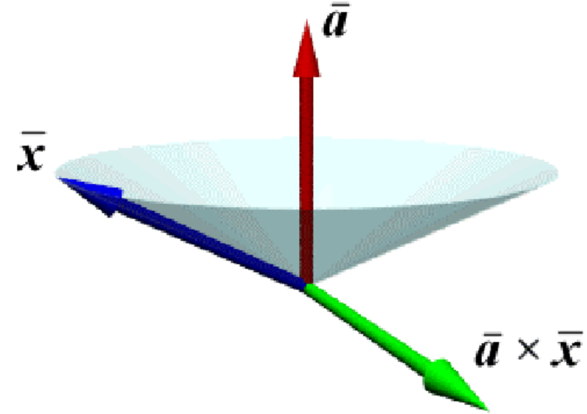
Quaternion for Rotation

- Rotate about axis a by angle θ

$$q = (s, v) = (s, v_1, v_2, v_3)$$

$$s = cos\left(\frac{\theta}{2}\right)$$

$$v = sin\left(\frac{\theta}{2}\right)\hat{a}$$



Quaternion Properties

- Linear combination of 1, *i, j, k*

$$q = w + xi + yj + zk = (s, v)$$

$$s = w, v = [x, y, z]$$

- Each of i, j and k are three square roots of –1

$$i^2 = j^2 = k^2 = ijk = -1$$

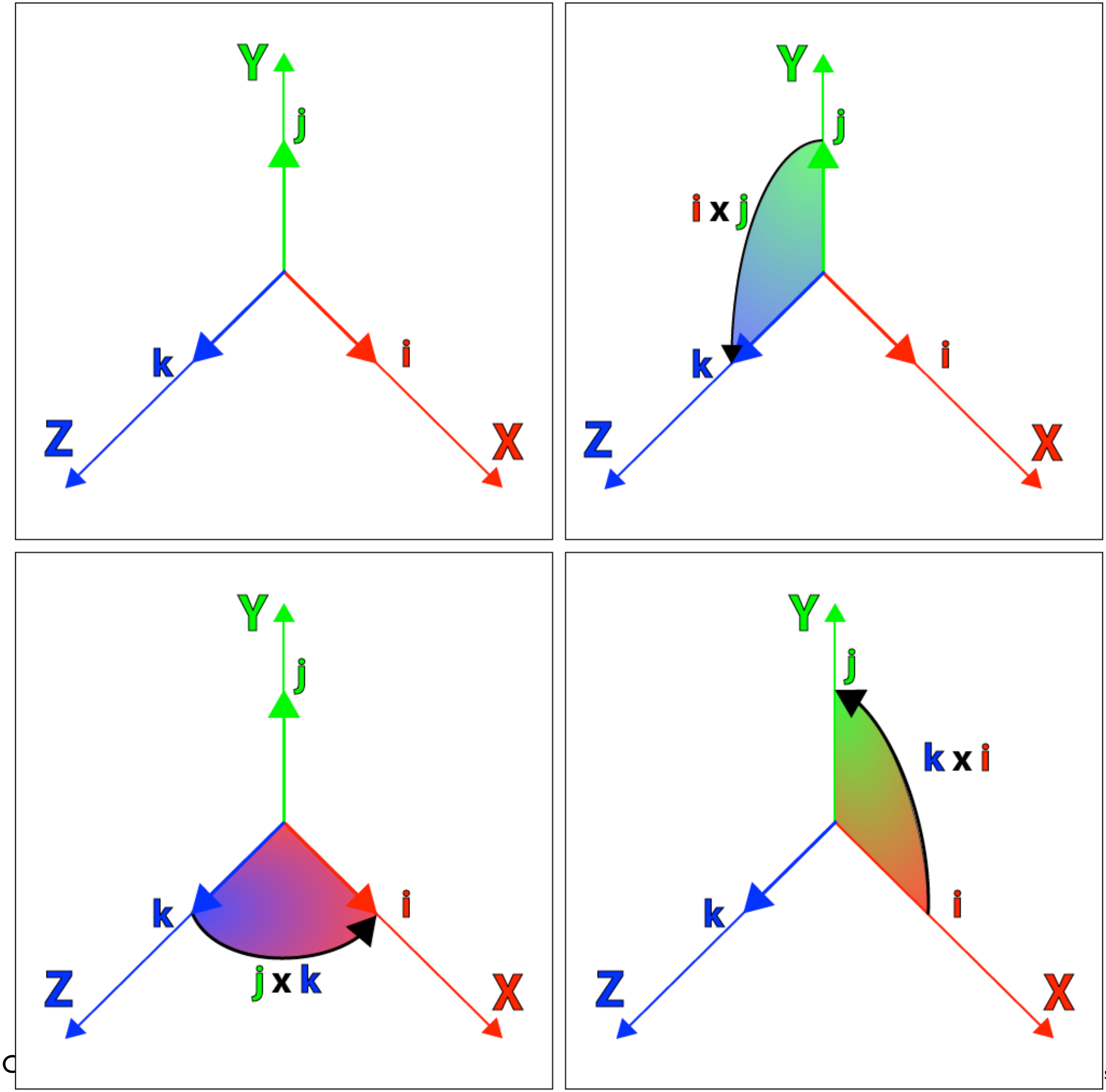
Review complex numbers

- Cross-multiplication is like cross product

$$ij = -ji = k$$

$$jk = -kj = i$$

$$ki = -ik = -j$$


Y
j
k
i
Z
X
Y
j
i x j
k
i
Z
X
Y
j
k
i
j x k
Z
X
Y
j
k x i
k
i
Z
X

ONB in quaternions

- Quaternion is extension of complex number in 4D space

$$q = w + xi + yj + zk$$

$$q' = w - xi - yj - zk$$

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

- Multiplication

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

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

Quaternion Properties

- Associative

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

- Not commutative

$$q_1 * q_2 \neq q_2 * q_1$$

- Unit quaternion

$$||q|| = 1$$
$$q^{-1} = q'$$

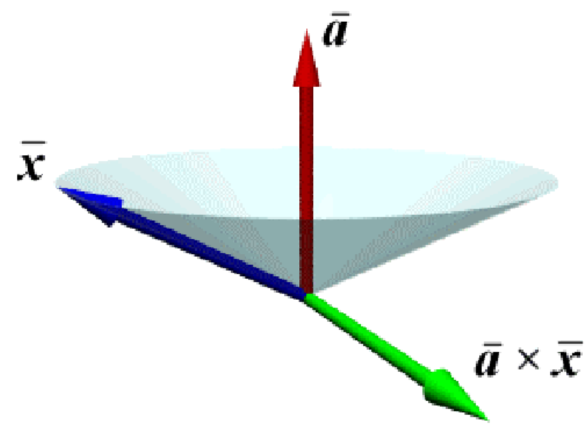
Quaternion for Rotation

- Rotate about axis a by angle θ

$$q = (s, v) = (s, v_1, v_2, v_3)$$

$$s = cos\left(\frac{\theta}{2}\right)$$

$$v = sin\left(\frac{\theta}{2}\right)\hat{a}$$



- Note: unit quaternion

Rotation Using Quaternion

- A point in space is a quaternion with 0 scalar

$$X = (0, \vec{x})$$

Rotation Using Quaternion

- A point in space is a quaternion with 0 scalar

$$X = (0, \vec{x})$$

- Rotation is computed as follows

$$x_{rotated} = qXq^{-1} = qXq'$$

- See Buss 3D CG: A mathematical introduction with OpenGL, Chapter 7

Matrix for quaternion

$$\begin{bmatrix} (w^2+x^2-y^2-z^2) & 2(xy-wz) & 2(xz+wy) & 0 \\ 2(xy+wz) & w^2-x^2+y^2-z^2 & 2(yz-wx) & 0 \\ 2(xz-wy) & 2(yz+wx) & w^2-x^2-y^2+z^2 & 0 \\ 0 & 0 & 0 & w^2+x^2+y^2+z^2 \end{bmatrix}$$

Rotation Using Quaternion

- Composing rotations
  - q1 and q2 are two rotations
  - First, q1 then q2

$$x_{rot} = q_2(q_1 X q_1^{-1})q_2^{-1}$$

$$x_{rot} = (q_2 q_1) X (q_1^{-1} q_2^{-1})$$

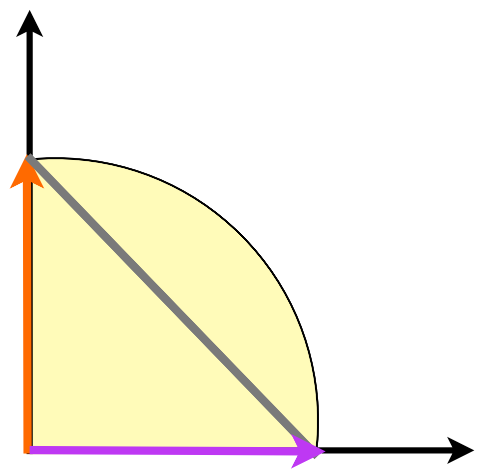
$$x_{rot} = (q_2 q_1) X (q_2 q_1)^{-1}$$

Why Quaternions?

- Fast, few operations, not redundant
- Numerically stable for incremental changes
- Composes rotations nicely
- Convert to matrices at the end
- Biggest reason: spherical interpolation

Interpolating between quaternions

- Why not linear interpolation?
  - Need to be normalized
  - Does not have a constant rate of rotation

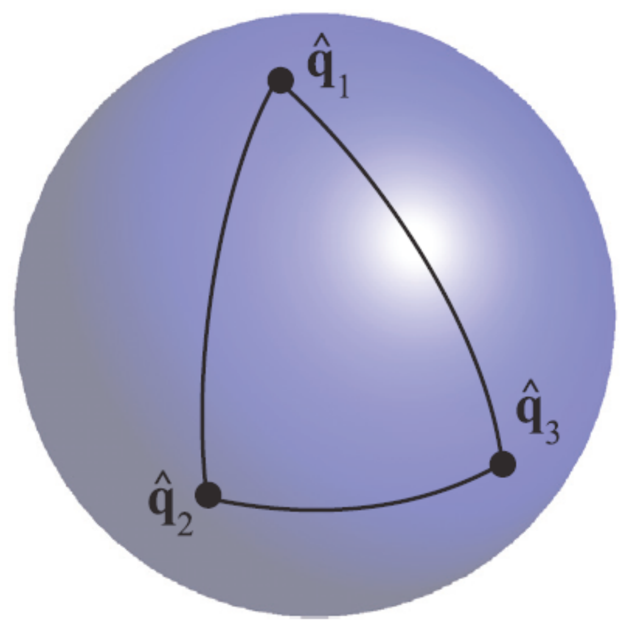


$$\frac{(1-\alpha)x + \alpha y}{||(1-\alpha)x + \alpha y||}$$

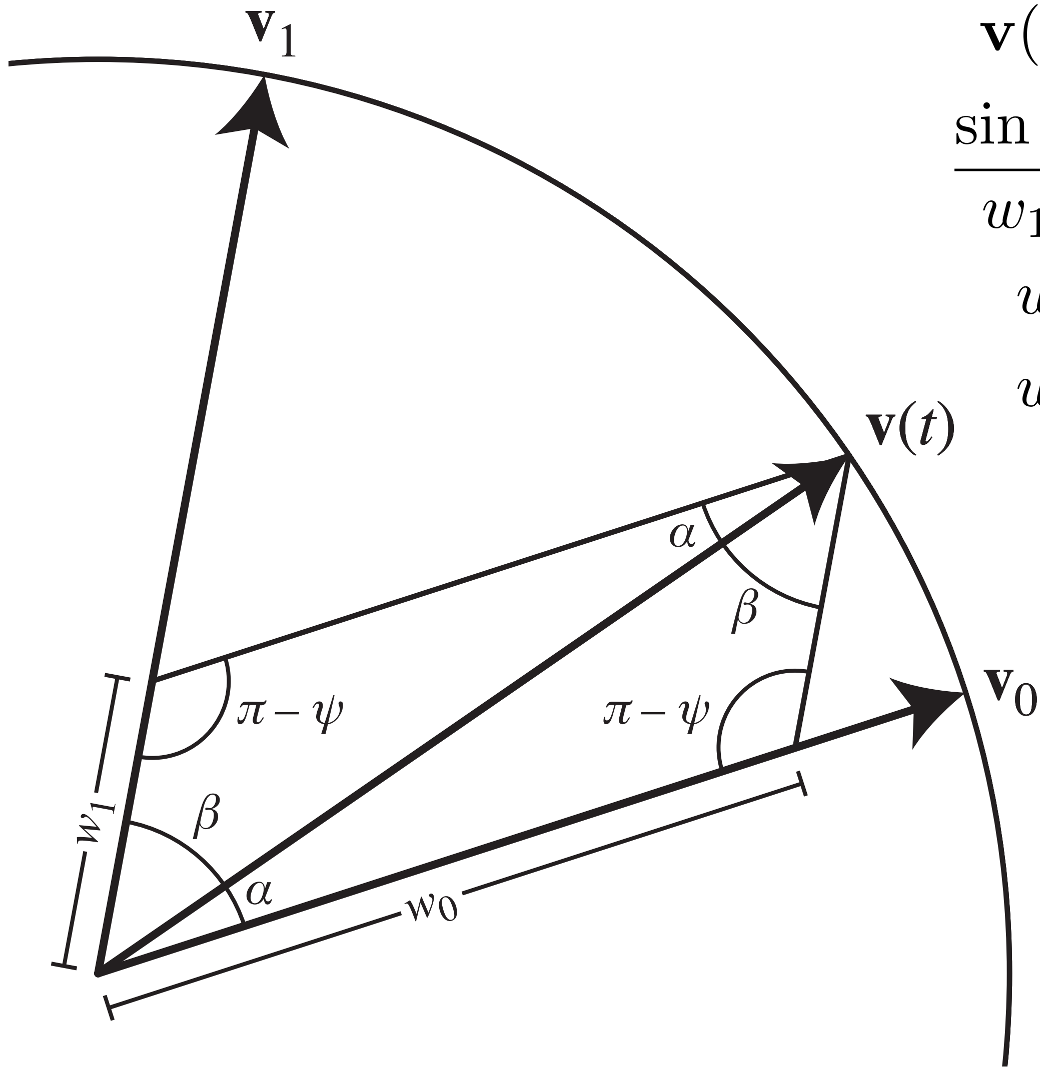
Spherical Linear Interpolation

- Intuitive interpolation between different orientations
  - Nicely represented through quaternions
  - Useful for animation
  - Given two quaternions, interpolate between them

  - Shortest path between two points on sphere
    - Geodesic, on Great Circle



Spherical linear interpolation ("slerp")



$$\alpha + \beta = \psi$$

$$\mathbf{v}(t) = w_0 \mathbf{v}_0 + w_1 \mathbf{v}_1$$

$$\frac{\sin \alpha}{w_1} = \frac{\sin \beta}{w_0} = \frac{\sin(\pi - \psi)}{1} = \sin \psi$$

$$w_0 = \sin \beta / \sin \psi$$

$$w_1 = \sin \alpha / \sin \psi$$

$$\psi = \cos^{-1}(\mathbf{v}_0 \cdot \mathbf{v}_1)$$

Quaternion Interpolation

- Spherical linear interpolation naturally works in any dimension
- Traverses a great arc on the sphere of unit quaternions
  - Uniform angular rotation velocity about a fixed axis

$$\psi = \cos^{-1}(q_0 \cdot q_1)$$

$$q(t) = \frac{q_0 \sin(1-t)\psi + q_1 \sin t\psi}{\sin \psi}$$

Practical issues

- When angle gets close to zero, use small angle approximation
  - degenerate to linear interpolation
- When angle close to 180, there is no shortest geodesic, but can pick one
- q is same rotation as -q
  - if q1 and q2 angle < 90, slerp between them
  - else, slerp between q1 and -q2

