CSC 433/533 Computer Graphics

Joshua Levine
josh@email.arizona.edu

Lecture 23 Animation 1

Nov. 20, 2019

Today's Agenda

- Reminders:
  - A06 questions?
- Goals for today: introduce concepts in animation

Traditional Animation

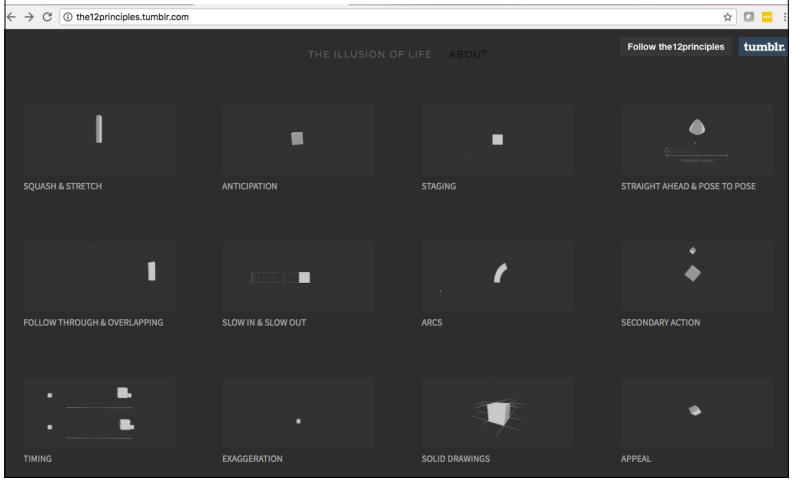
Animation vs. Modeling

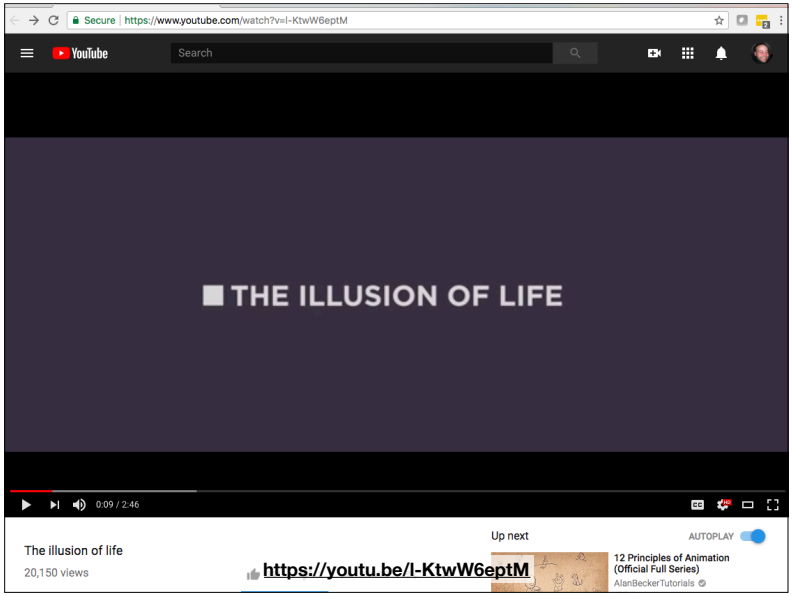
- Modeling is specifying shape in 3D
  - We've seen two forms: parametric, implicit
- Animation is specifying shape as a function of time
  - Isn't this just 4D shape? Modeling once per frame?
  - Sort of, but the constraints are different and must pay special attention to the temporal component.

Influences

- Between the 1930s and now, traditional hand-drawn animation has led to important principles that have spanned to today's computer-generated media
- Idea: while the tools have changed, the same principles still apply.
- Many of these ideas come from Disney artists, Frank Thomas and Ollie Johnston, brought to the SIGGRAPH community by John Lasseter in 1987

12 Principles from The Illusion of Life by Frank & Ollie

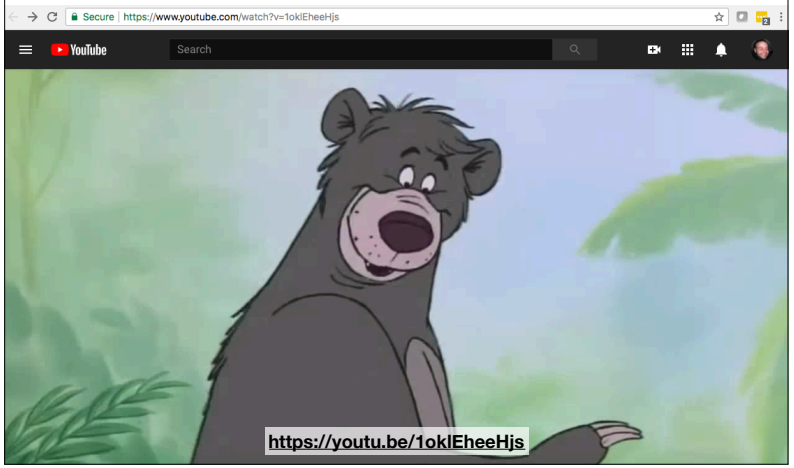




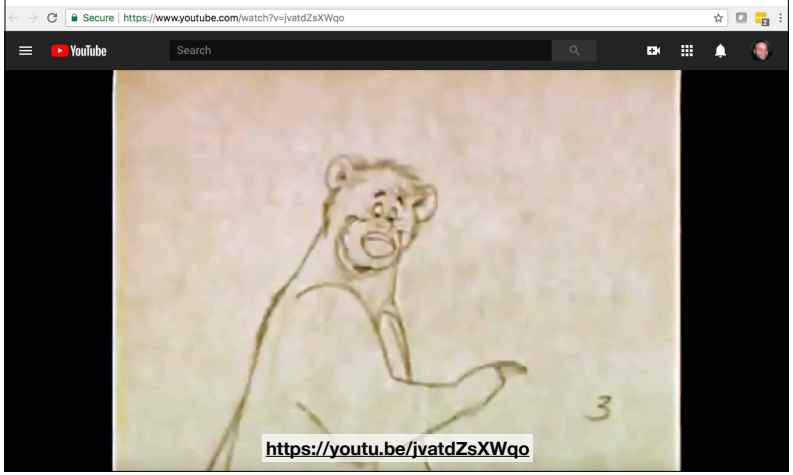
The Principles are a Framework

- They apply both to traditional methods vs. computational ones
- Also important to think about the level of control vs. amount of work. Where can computers help automate the process without sacrifice an animator's flexibility
- Also important: Physical realism vs. Artistic control

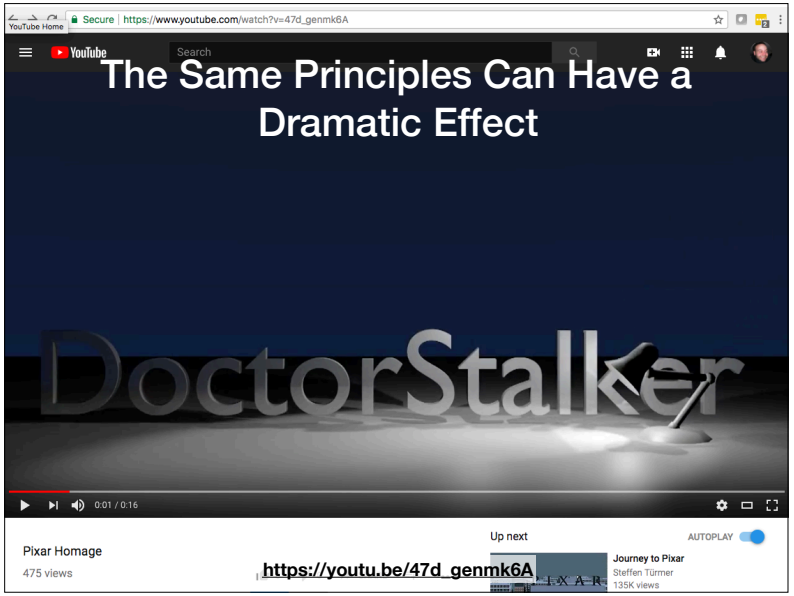
Jungle Book (1967)



Pencil Test



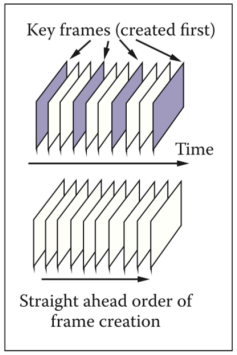
Computer Animation



Keyframing

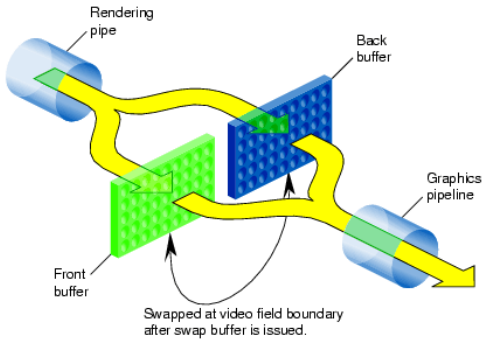
Keyframe Animation

- Idea: Draw a subset of important frames (called **key frames**) and fill in the rest with *in-betweens*
- In hand-drawn animation, the head animator would draw the poses and the assistants would do the rest
- In computer animation, the artist draws the keys and the computer does the in-betweening
  - Interpolation is used to fill in the rest!



Double Buffering

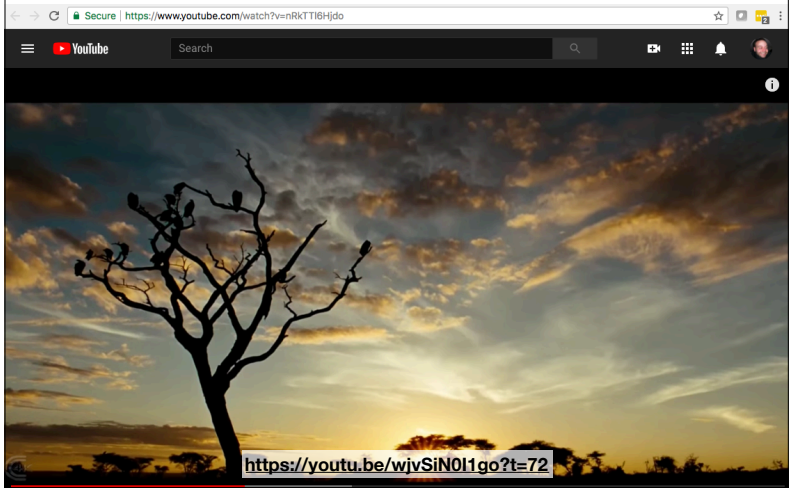
- If you draw directly to video buffer, the user will see the drawing happen
- Particularly noticeable artifacts when doing animation



http://techpubs.sgi.com/library/dynaweb_docs/linux/SGI_Developer/books/Perf_GetStarted/sgi_html/figures/double.buffering.gif

Interpolation Matters

Timing Can Also Hide a Lot



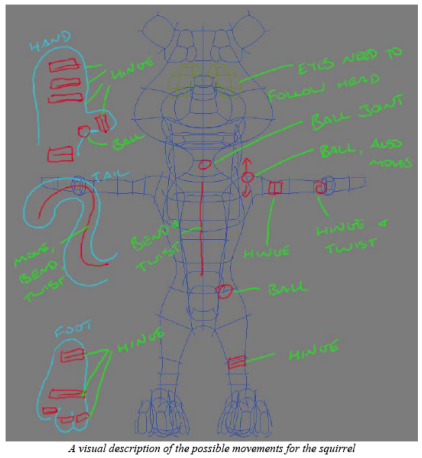
Controlling geometry conveniently

- Manually place every control point at every keyframe?
  - labor intensive
  - hard to get smooth, consistent motion
- Animate using smaller set of meaningful *degrees of freedom*
  - 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"*

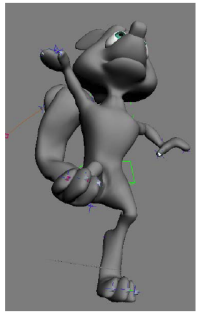
Controlling shape for animation

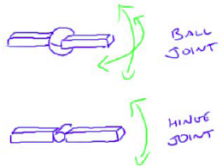
- Start with *modeling DOFs* (control points)
- *Deformations* control those DOFs at a higher level
  - Example: move first joint of second finger on left hand
- *Animation controls* control *those* DOFs at a higher level
  - Example: open/close left hand
- Both cases can be handled by the same kinds of deformers

Character with DOFs



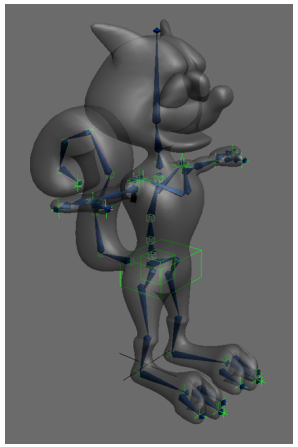
[Greenberg/Pellacini | CIS 565]





A visual description of the possible movements for the squirrel

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

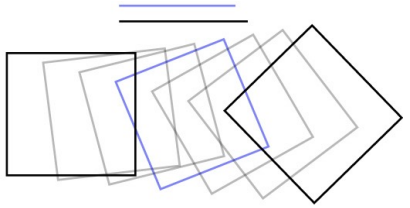
Interpolating Rotations

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

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



Interpolating Rotations

$$\frac{1}{2}\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

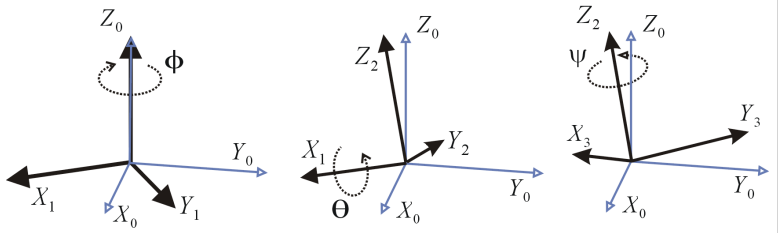
90° CW 90° CCW

Not a rotation matrix!

Interpolating transformations

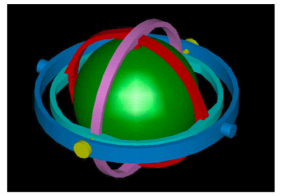
- Linear interpolation of matrices is not effective
  - leads to shrinkage when interpolating rotations
- One approach: always keep transformations in a canonical form (e.g. translate-rotate-scale)
  - then the pieces can be interpolated separately
  - rotations stay rotations, scales stay scales, all is good
- But you might be faced with just a matrix. What then?

Could Instead Decompose Rotation by Euler Angles



http://upload.wikimedia.org/wikipedia/commons/7/73/EulerG.png

Parameterizing rotations



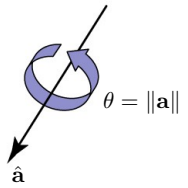
- Euler angles
  - rotate around x, then y, then z
  - nice and simple

  $R(\theta_x, \theta_y, \theta_z) = R_z(\theta_z)R_y(\theta_y)R_x(\theta_x)$
- Axis/angle
  - specify axis to rotate around, then angle by which to rotate

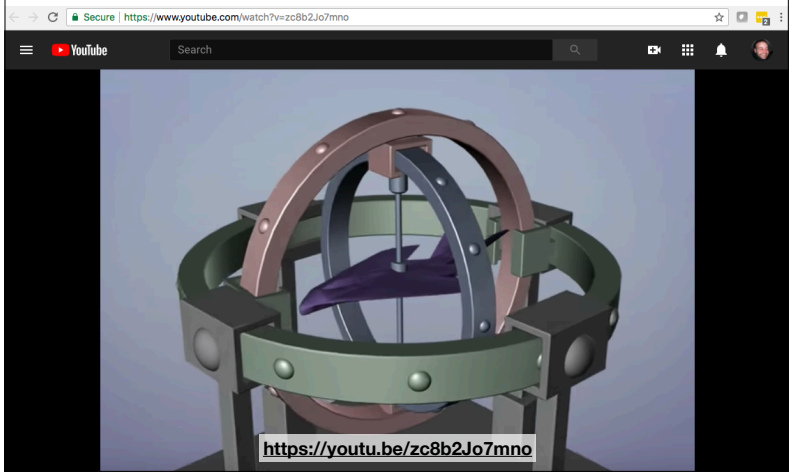
    $R(\hat{\mathbf{a}}, \theta) = F_{\hat{\mathbf{a}}} R_x(\theta) F_{\hat{\mathbf{a}}}^{-1}$

    $F_{\hat{\mathbf{a}}}$ is a frame matrix with **a** as its first column.
  - multiply axis and angle to get a more compact form

    $R(\mathbf{a}) = R(\hat{\mathbf{a}}, \|\mathbf{a}\|)$



Gimbal Lock



Quaternions

What is a rotation?

- Think of the set of possible orientations of a 3D object
  - you get from one orientation to another by rotating
  - if we agree on some starting orientation, rotations and orientations are pretty much the same thing
- It is a smoothly connected three-dimensional space
  - how can you tell? For any orientation, I can make a small rotation around any axis (pick axis = 2D, pick angle = 1D)
- This set is a subset of linear transformations called SO(3)
  - **O** for orthogonal, **S** for "special" (determinant +1), **3** for 3D

Warm-up: unit vectors

- When we want to represent directions we use unit vectors: points that are literally on the unit sphere in R^3
  - now no points are special
  - every point has a unique representation
  - equal sized changes in coordinates are equal sized changes in direction
- Down side: one too many coordinates
  - have to maintain normalization
  - but normalize() is a simple and easy operation

Warm-up: spherical coordinates

- We can use latitude and longitude to parameterize the 2-sphere (aka. directions in 3D), but with some annoyances
  - the poles are special, and are represented many times
  - if you are at the pole, going East does nothing
  - near the pole you have to change longitude a lot to get anywhere
  - traveling along straight lines in (latitude, longitude) leads to some pretty weird paths on the globe

    you are standing one mile from the pole, facing towards it; to get to the point 2 miles ahead of you the map tells you to turn right and walk 3.14 miles along a latitude line…
  - Conclusion: use unit vectors instead

Warm-up: interpolating directions

- Interpolating in the space of 3D vectors is well behaved
- Simple computation: interpolate linearly and normalize
  - this is what we do all the time, e.g. with normals for fragment shading

    $$\hat{\mathbf{v}}(t) = \text{normalize}((1-t)\mathbf{v}_0 + t\mathbf{v}_1)$$
  - but for far-apart endpoints the speed is uneven (faster towards the middle)
- For constant speed: spherical linear interpolation
  - build basis $\{\mathbf{v}_0, \mathbf{w}\}$ from $\mathbf{v}_0$ and $\mathbf{v}_1$
  - interpolate angle from 0 to θ
  - (slicker way in a few slides)

$$\begin{aligned}\mathbf{w} &= \hat{\mathbf{v}}_1 - (\hat{\mathbf{v}}_0 \cdot \hat{\mathbf{v}}_1)\hat{\mathbf{v}}_0 \\ \hat{\mathbf{w}} &= \mathbf{w}/\|\mathbf{w}\| \\ \theta &= \text{acos}(\hat{\mathbf{v}}_0 \cdot \hat{\mathbf{v}}_1) \\ \hat{\mathbf{v}}(t) &= (\cos t\theta)\,\hat{\mathbf{v}}_0 + (\sin t\theta)\,\hat{\mathbf{w}}\end{aligned}$$

Quaternions

- A quaternion is a 4-vector
  $q = (w, x, y, z) \in \mathbf{H}$
- We tend to think of it as a scalar and a 3-vector
  $q = (s, \mathbf{v}) \text{ where } s = w \text{ and } \mathbf{v} = (x, y, z)$
  - alternate notation: $q = s + \mathbf{v}$
- Unit quaternions are unit 4-vectors:
  $w^2 + x^2 + y^2 + z^2 = 1, \quad \text{or} \quad s^2 + \|\mathbf{v}\|^2 = 1$
  - think of s and ||**v**|| as sin and cos of an angle: $q = \cos\psi + \hat{\mathbf{v}}\sin\psi$
- They can represent rotations in an axis-angle-like way
  - direction of **v** tells you the axis of rotation
  - s and and ||**v**|| tell you the angle of rotation

Quaternions and rotations

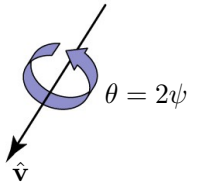
There is a natural association between the unit quaternion

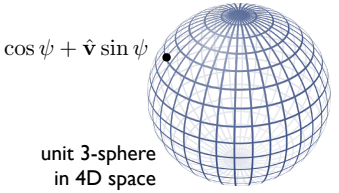
$\cos\psi + \hat{\mathbf{v}}\sin\psi \in S^3 \subset \mathbf{H}$

and the 3D axis-angle rotation

$R_{\hat{\mathbf{v}}}(\theta) \in SO(3)$

where $\theta = 2\psi$.





[Wikimedia Commons user Geek3]

Rotation with Quaternions

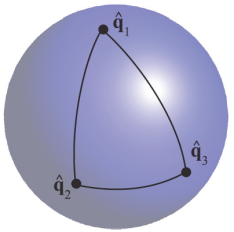
- To rotate a point ***p***, about quaternion ***q***
  - One turns it into the quaternion $\boldsymbol{q_p} = [0 ; \boldsymbol{p}]$
  - And then computes $\boldsymbol{q'_p} = \boldsymbol{q} \cdot \boldsymbol{q_p} \cdot \boldsymbol{q}^{-1}$
- Where

$$q^{-1} = (1/|q|)[s; -\mathbf{v}]$$

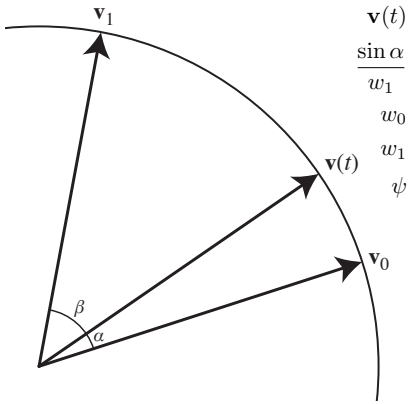
$$q_1 \cdot q_2 \equiv [s_1 s_2 - \mathbf{v}_1\mathbf{v}_2; s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2]$$

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

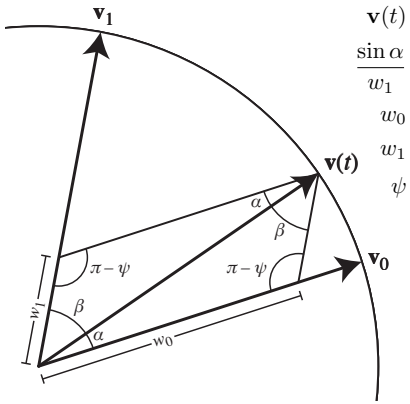


Spherical linear interpolation ("slerp")



$$\alpha + \beta = \psi$$
$$\mathbf{v}(t) = w_0\mathbf{v}_0 + w_1\mathbf{v}_1$$
$$\frac{\sin\alpha}{w_1} = \frac{\sin\beta}{w_0} = \frac{\sin(\pi-\psi)}{1} = \sin\psi$$
$$w_0 = \sin\beta/\sin\psi$$
$$w_1 = \sin\alpha/\sin\psi$$
$$\psi = \cos^{-1}(\mathbf{v}_0 \cdot \mathbf{v}_1)$$

Spherical linear interpolation ("slerp")



$$\alpha + \beta = \psi$$
$$\mathbf{v}(t) = w_0\mathbf{v}_0 + w_1\mathbf{v}_1$$
$$\frac{\sin\alpha}{w_1} = \frac{\sin\beta}{w_0} = \frac{\sin(\pi-\psi)}{1} = \sin\psi$$
$$w_0 = \sin\beta/\sin\psi$$
$$w_1 = \sin\alpha/\sin\psi$$
$$\psi = \cos^{-1}(\mathbf{v}_0 \cdot \mathbf{v}_1)$$

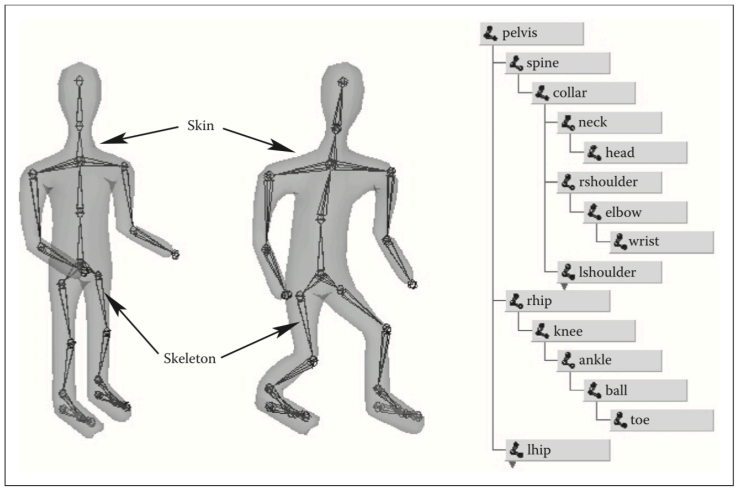
Quaternion Interpolation

- 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

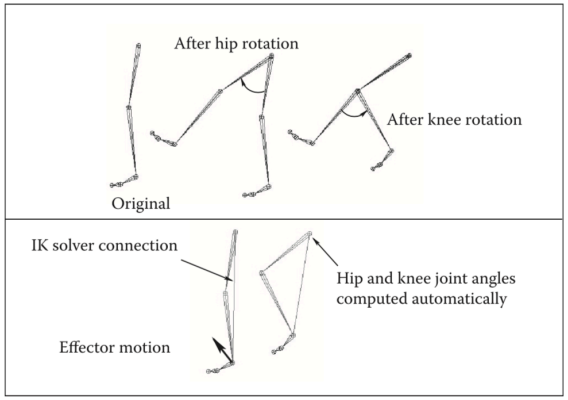
$$\psi = \cos^{-1}(q_0 \cdot q_1)$$
$$q(t) = \frac{q_0 \sin(1-t)\psi + q_1 \sin t\psi}{\sin\psi}$$

Character Animation

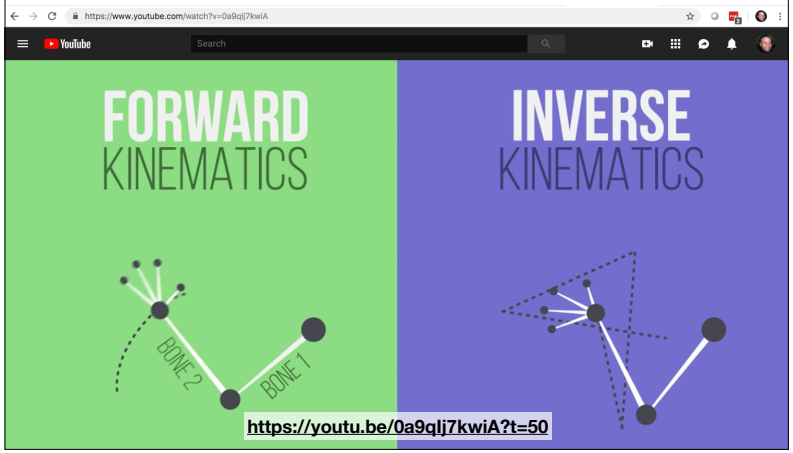
Animating w/ Skeletal Hierarchies



Forward vs. Inverse Kinematics



Inverse Kinematics Solves for all Intermediate Constraints



Lec24 Required Reading

- FOCG, Ch. 16.3, 16.5

Reminder: Assignment 06

Assigned: Wednesday, Nov. 13
Written Due: Monday, Nov. 25, 4:59:59 pm
Program Due: Wednesday, Nov. 27, 4:59:59 pm