Animation

CS 4620 Lecture 21

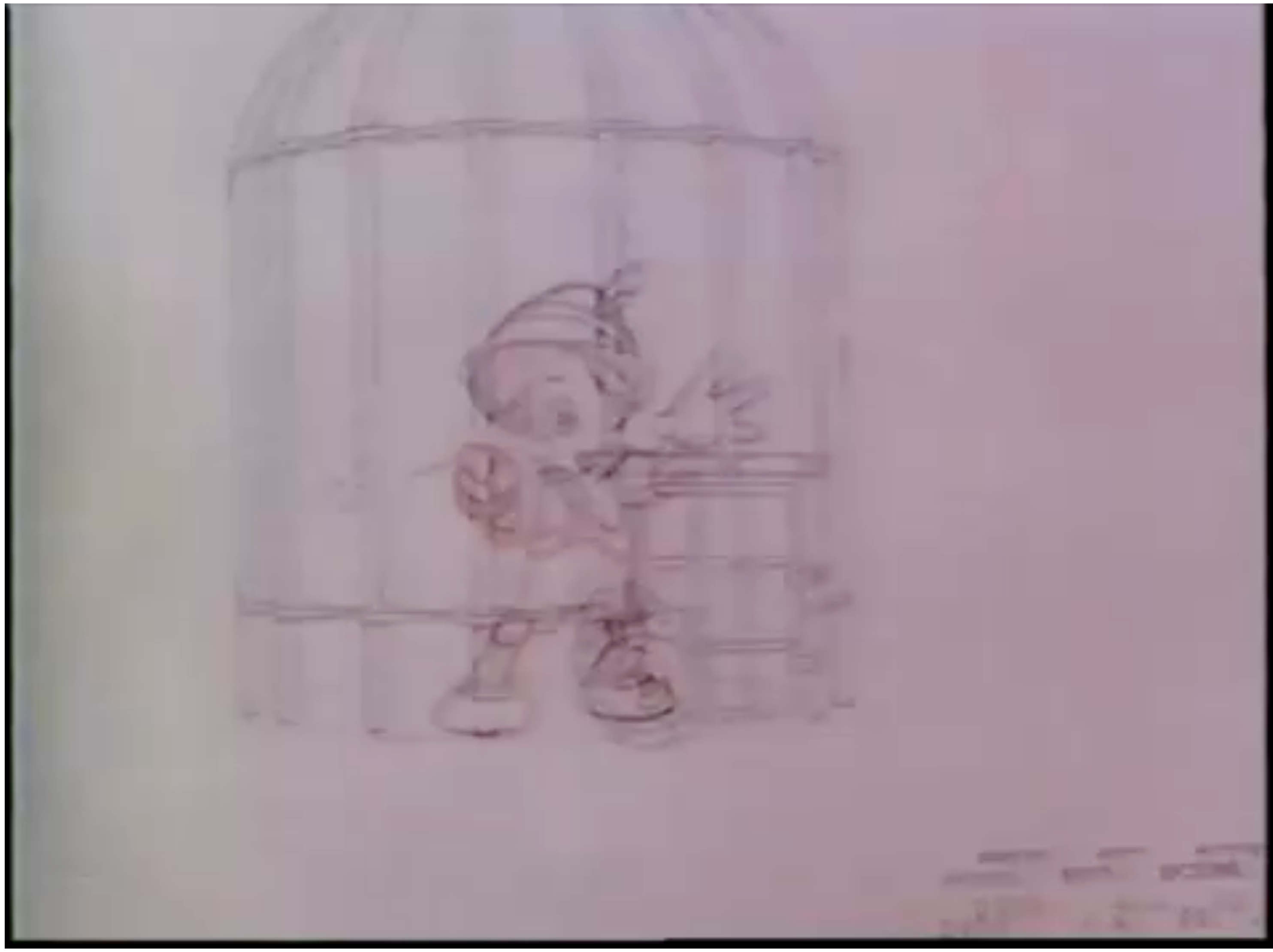
(with previous instructors James/Bala)

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 (using a lightbox to see the previous frame or frames)
  - 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

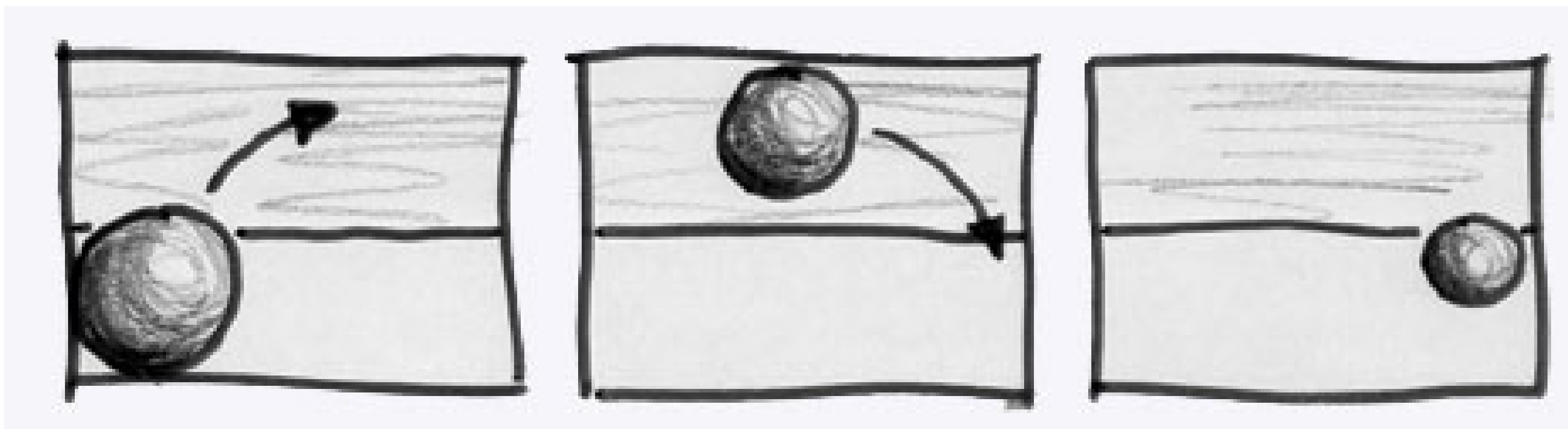
Keyframes in computer animation

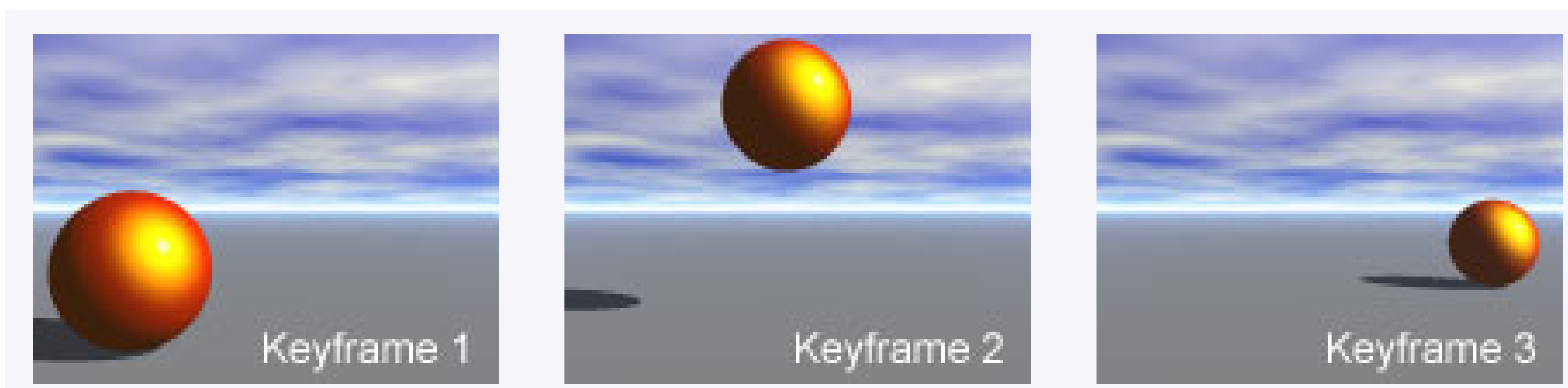
- Just as with hand-drawn animation, adjusting the model from scratch for every frame would be tedious and difficult
- Same solution: animator establishes the keyframes, software fills in the in-betweens
- Two key ideas of computer animation:
  - create *high-level* controls for adjusting geometry
 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

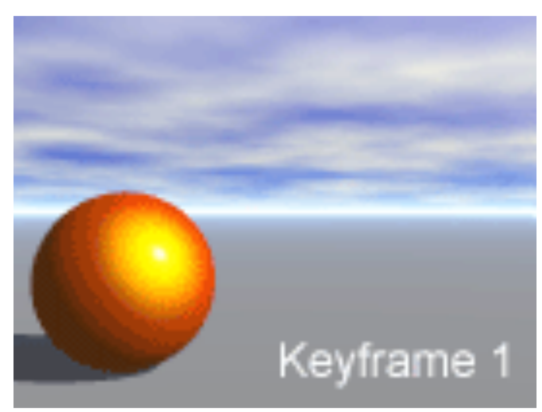
The most basic animation control

- 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





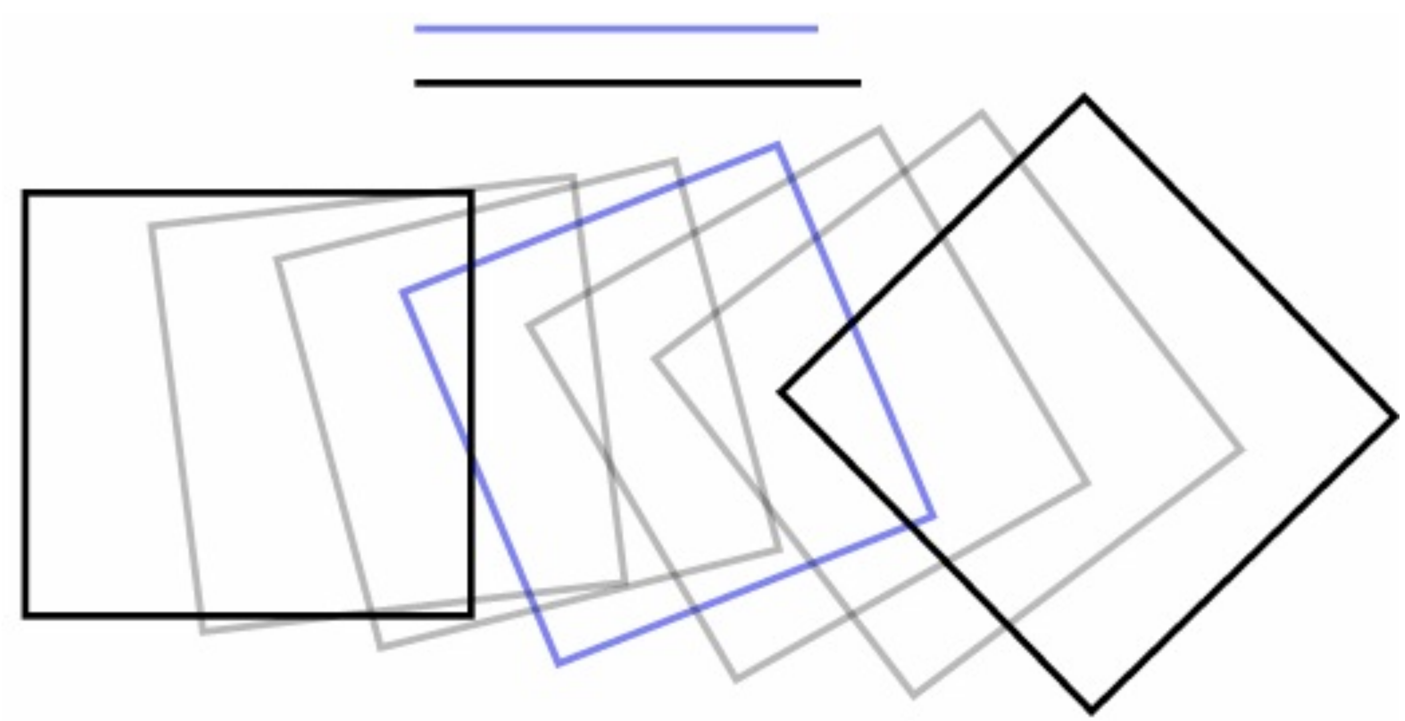


[Bryce Tutorial http://www.cadtutor.net/dd/bryce/anim/anim.html]

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?

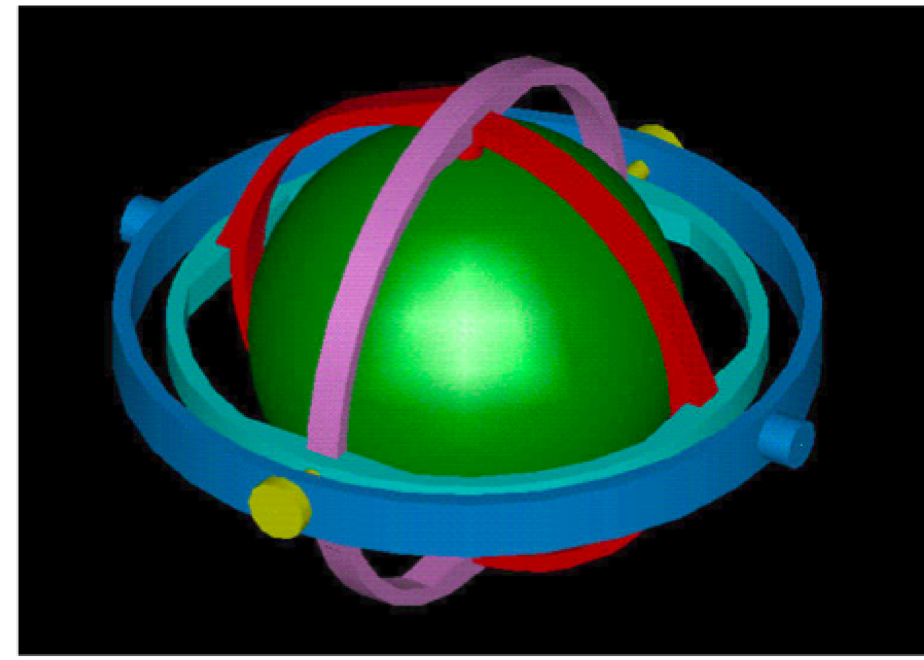
    This is fine for translations but bad for rotations



Parameterizing rotations

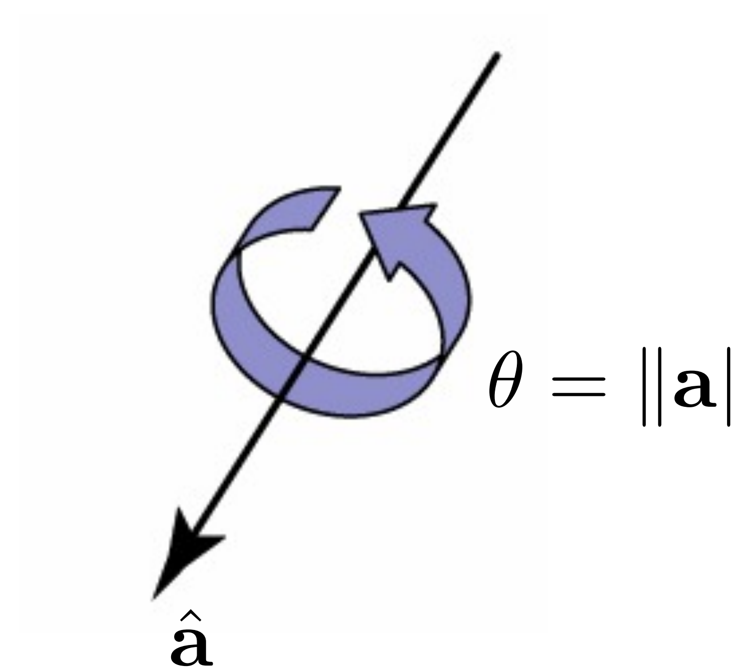
- Euler angles
  - rotate around x, then y, then z
  - nice and simple

$$f(\alpha, \beta, \gamma) = R_{\mathbf{z}}(\gamma) R_{\mathbf{y}}(\beta) R_{\mathbf{x}}(\alpha)$$



- Axis/angle
  - specify axis to rotate around, then angle by which to rotate
  - multiply axis and angle to get a more compact form

$$f(\mathbf{a}) = R_{\hat{\mathbf{a}}}(\|\mathbf{a}\|)$$



Problems

- Euler angles
  - gimbal lock (saw this before)
  - some rotations have many representations
- Axis/angle
  - multiple representations for identity rotation
  - even with combined rotation angle, making small changes near 180 degree rotations requires larger changes to parameters
- These resemble the problems with polar coordinates on the sphere
  - as with choosing poles, choosing the reference orientation for an object changes how the representation works

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

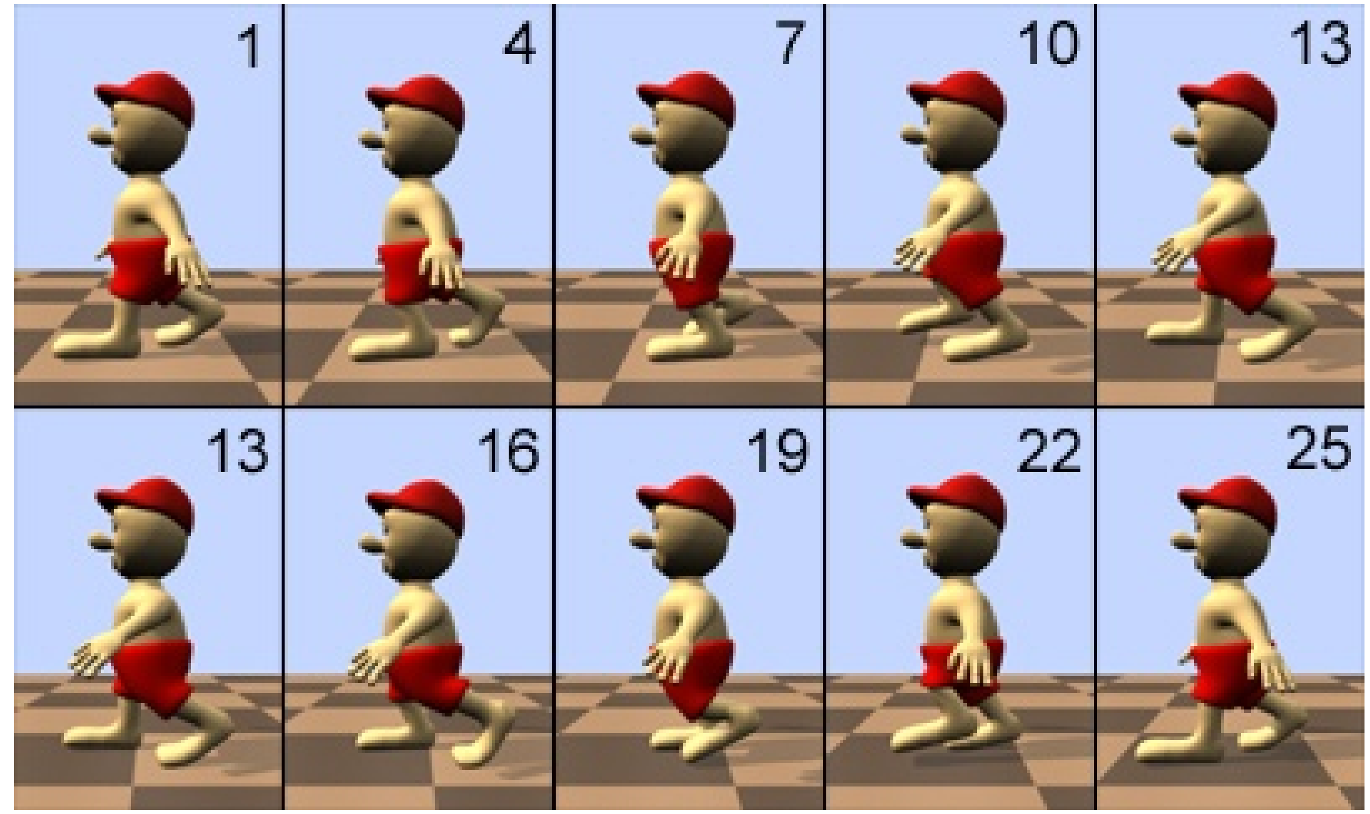
Pose-to-pose animation planning

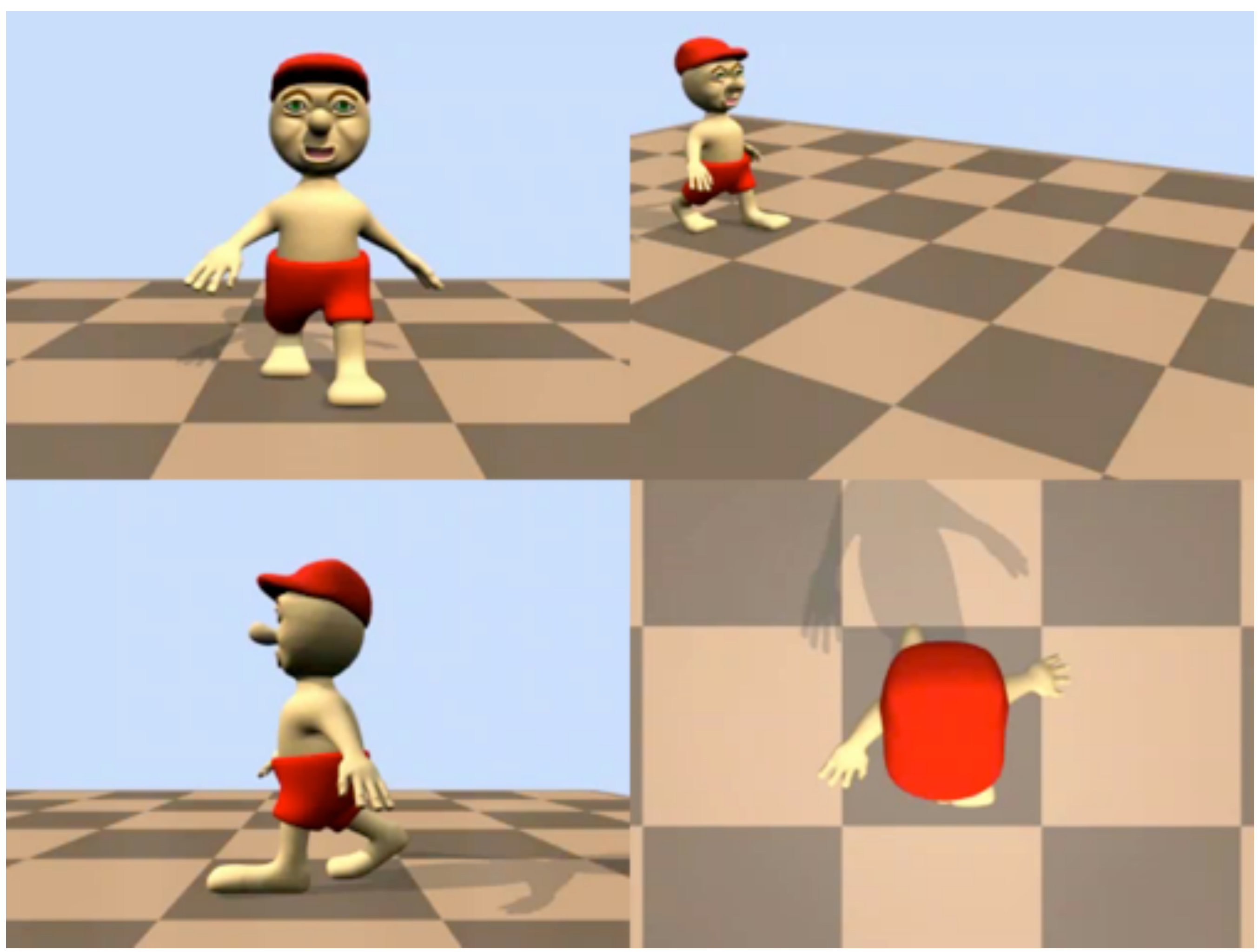
– First work out poses that are key to the story
– Next fill in animation in between

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 principal operation

Walk cycle





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

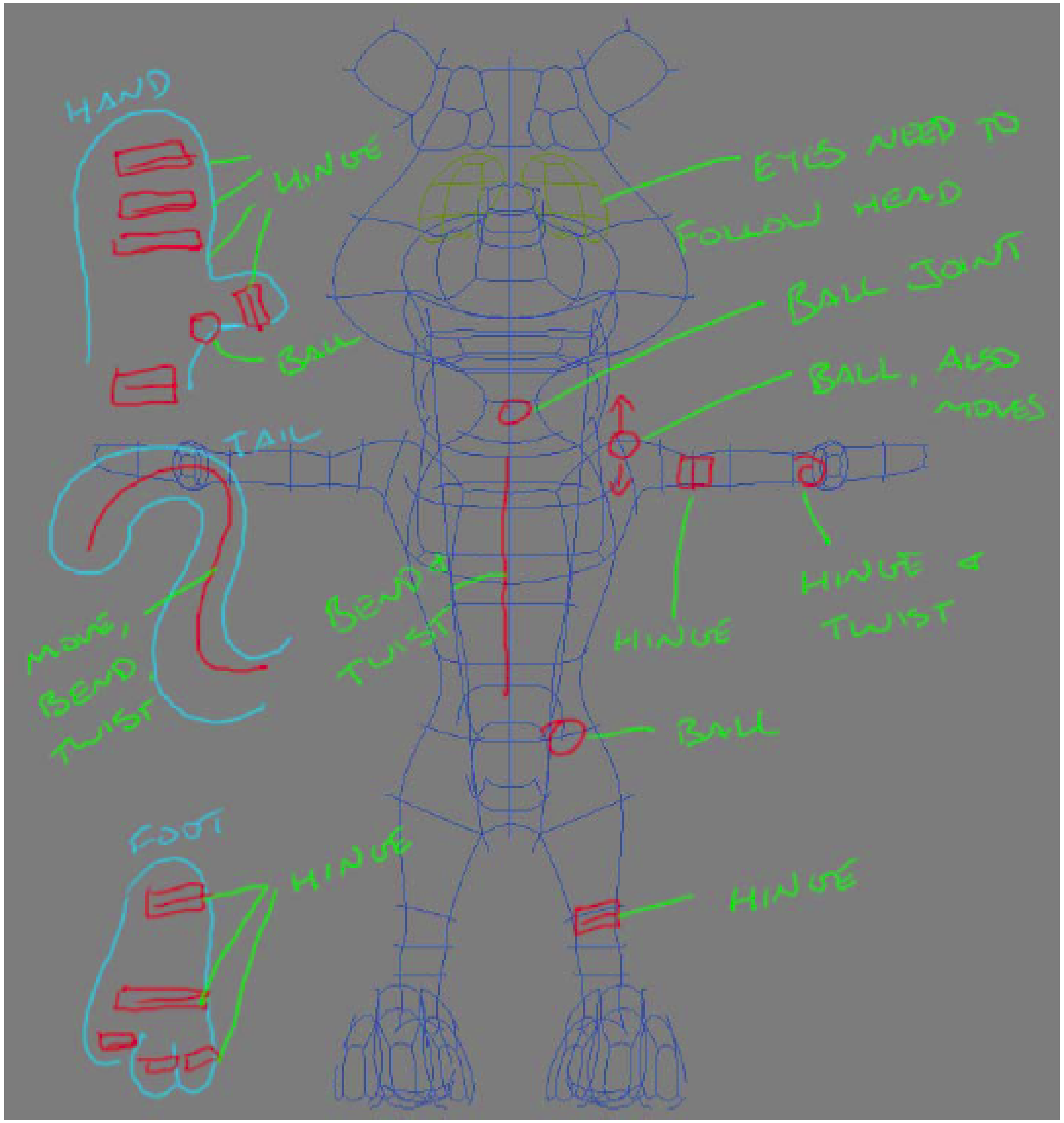
Controlling geometry conveniently

- 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

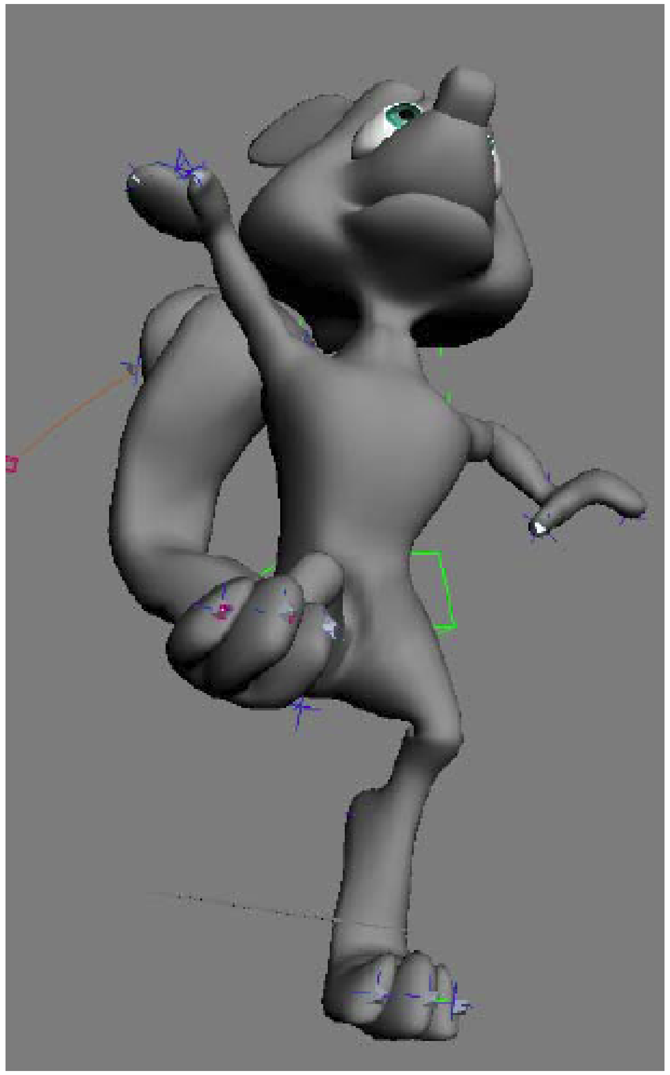
    E.g. "bend the elbow"

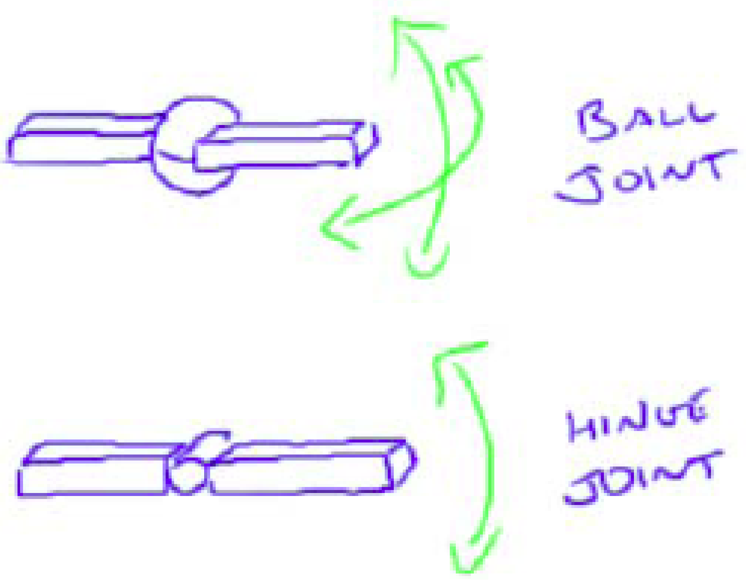
Character with DOFs



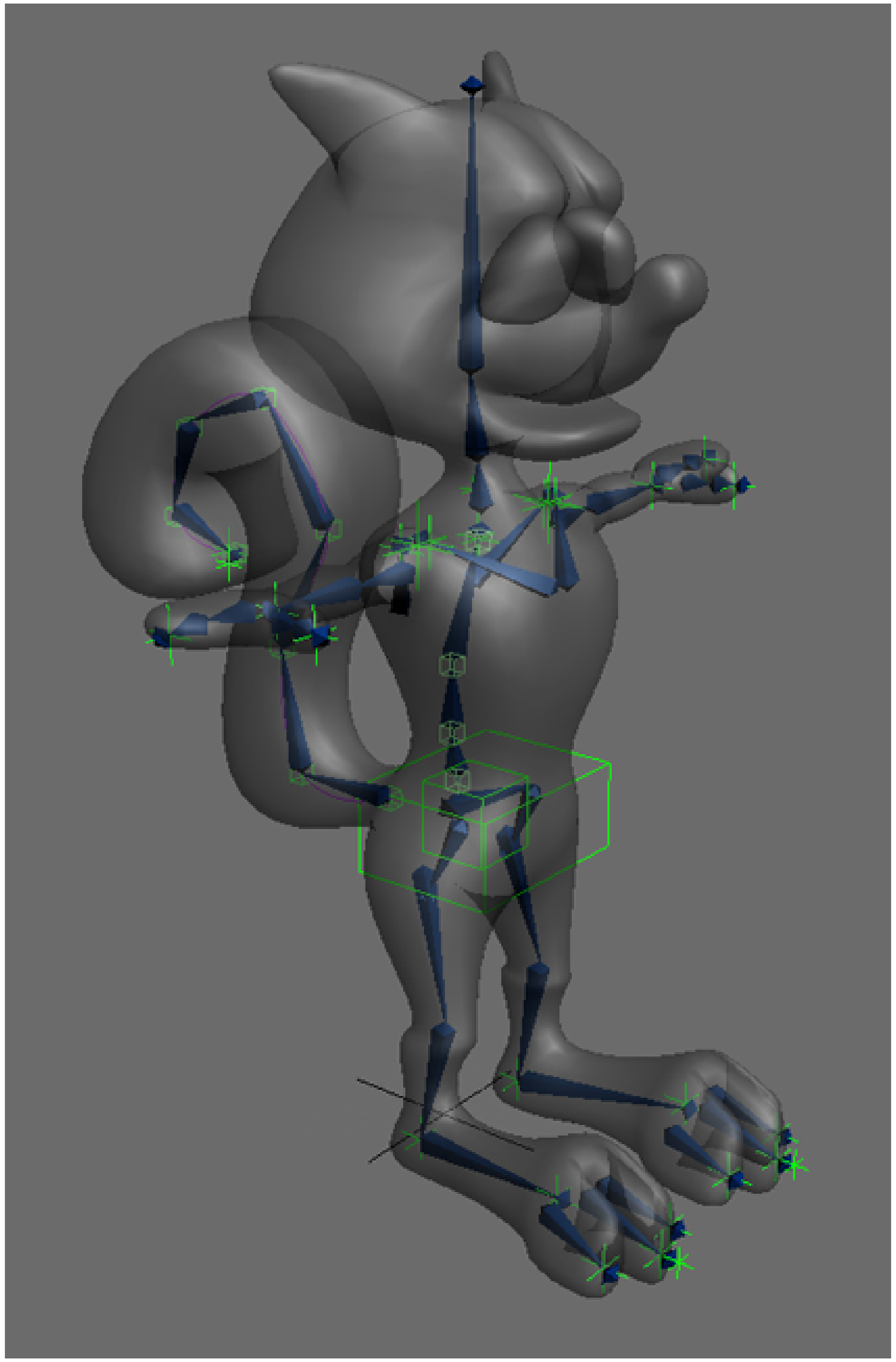
A visual description of the possible movements for the squirrel

[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

The artistic process of animation

- What are animators trying to do?
  - Important to understand in thinking about what tools they need
- Basic principles are universal across media
  - 2D hand-drawn animation
  - 2D and computer animation
  - 3D computer animation
- Widely cited set of principles laid out by Frank Thomas and Ollie Johnston in *The Illusion of Life* (1981)
- The following slides follow Michael Comet's examples:

  www.comet-cartoons.com

Animation principles: timing

- Speed of an action is crucial to the impression it makes
  - examples with same keyframes, different times:

[Michael B. Comet]

60 fr: looking around　　30 fr: "no"　　5 fr: just been hit

Animation principles: timing

- Speed of an action is crucial to the impression it makes
  - examples with same keyframes, different times:

[Michael B. Comet]

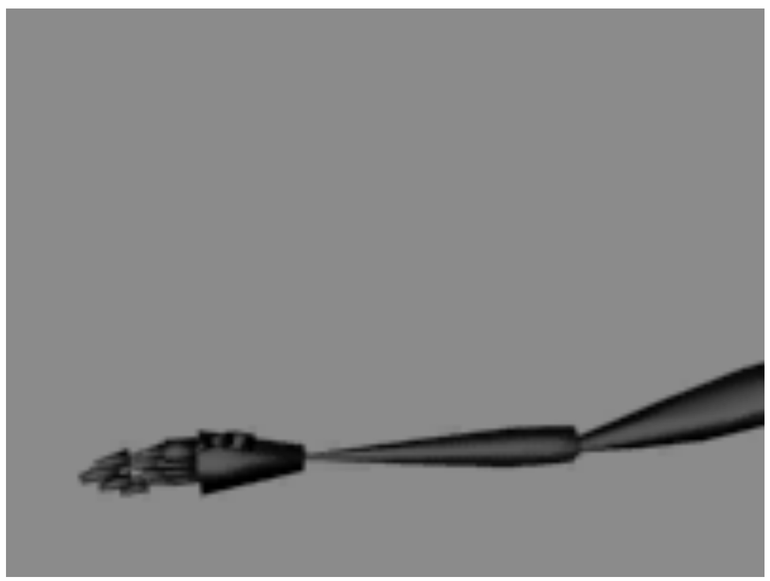
60 fr: looking around

30 fr: "no"

5 fr: just been hit

Animation principles: ease in/out

- Real objects do not start and stop suddenly
  - animation parameters shouldn't either



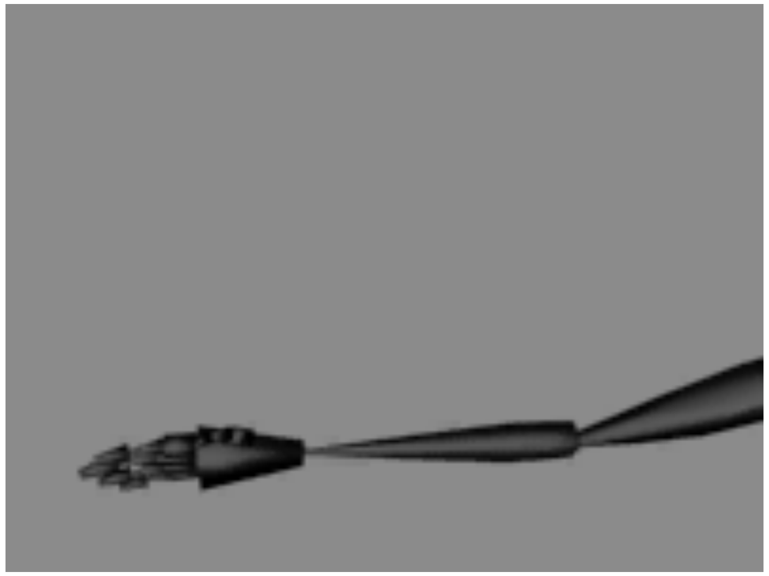
straight linear interp. ease in/out

  - a little goes a long way (just a few frames acceleration or deceleration for "snappy" motions)

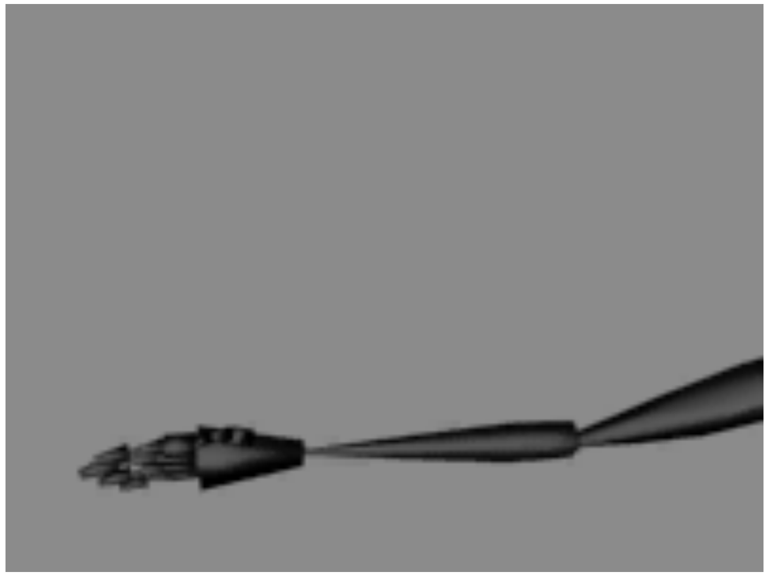
[Michael B. Comet]

Animation principles: ease in/out

- Real objects do not start and stop suddenly
  - animation parameters shouldn't either



straight linear interp.



ease in/out

[Michael B. Comet]

  - a little goes a long way (just a few frames acceleration or deceleration for "snappy" motions)

Animation principles: moving in arcs

- Real objects also don't move in straight lines
  - generally curves are more graceful and realistic

[Michael B. Comet]

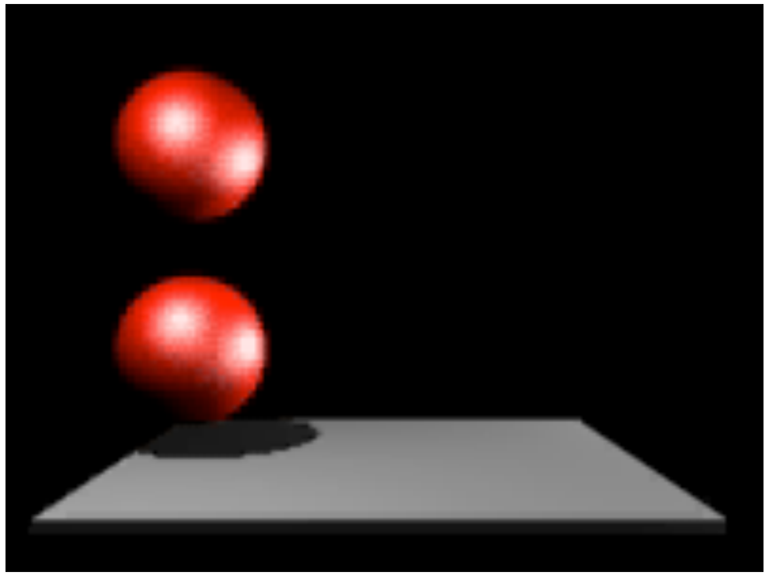
Animation principles: moving in arcs

- Real objects also don't move in straight lines
  - generally curves are more graceful and realistic

[Michael B. Comet]

Animation principles: anticipation

- Most actions are preceded by some kind of “wind-up”

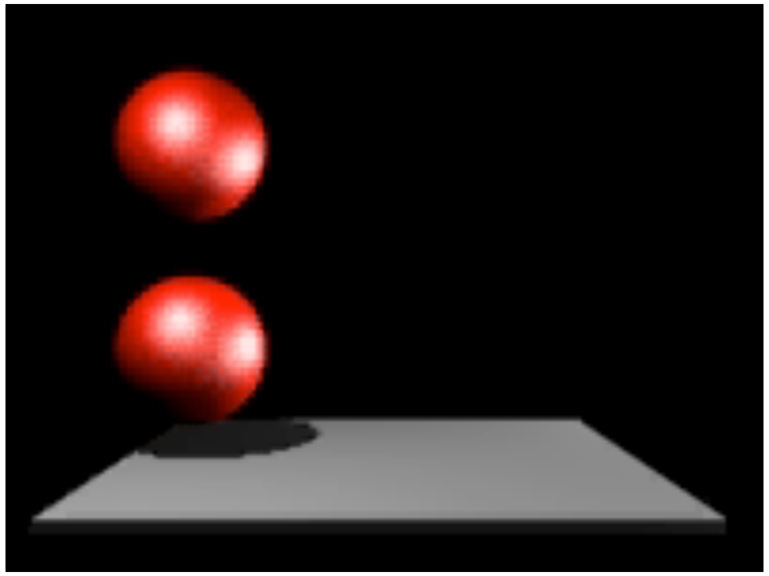


[Michael B. Comet]

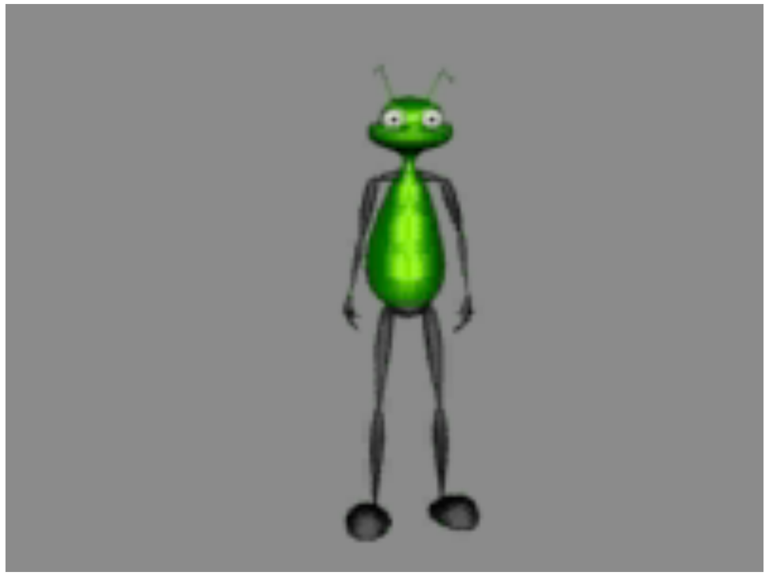
[Michael B. Comet]

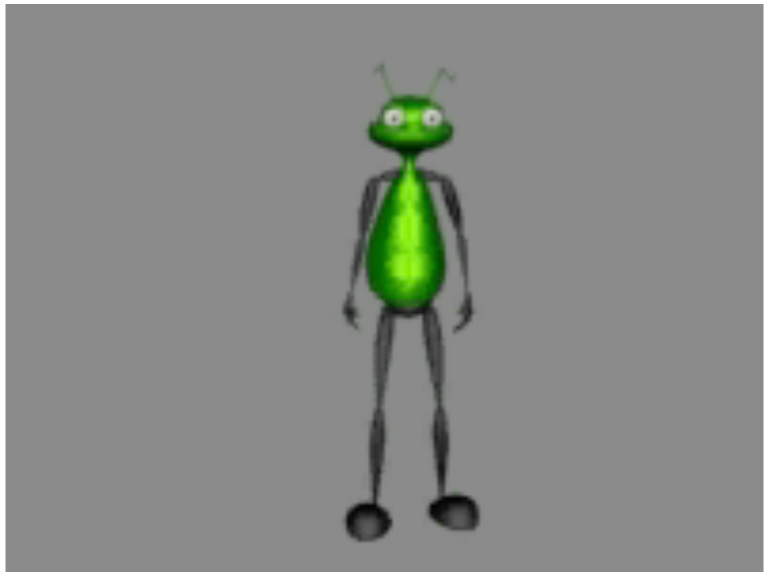
Animation principles: anticipation

- Most actions are preceded by some kind of "wind-up"

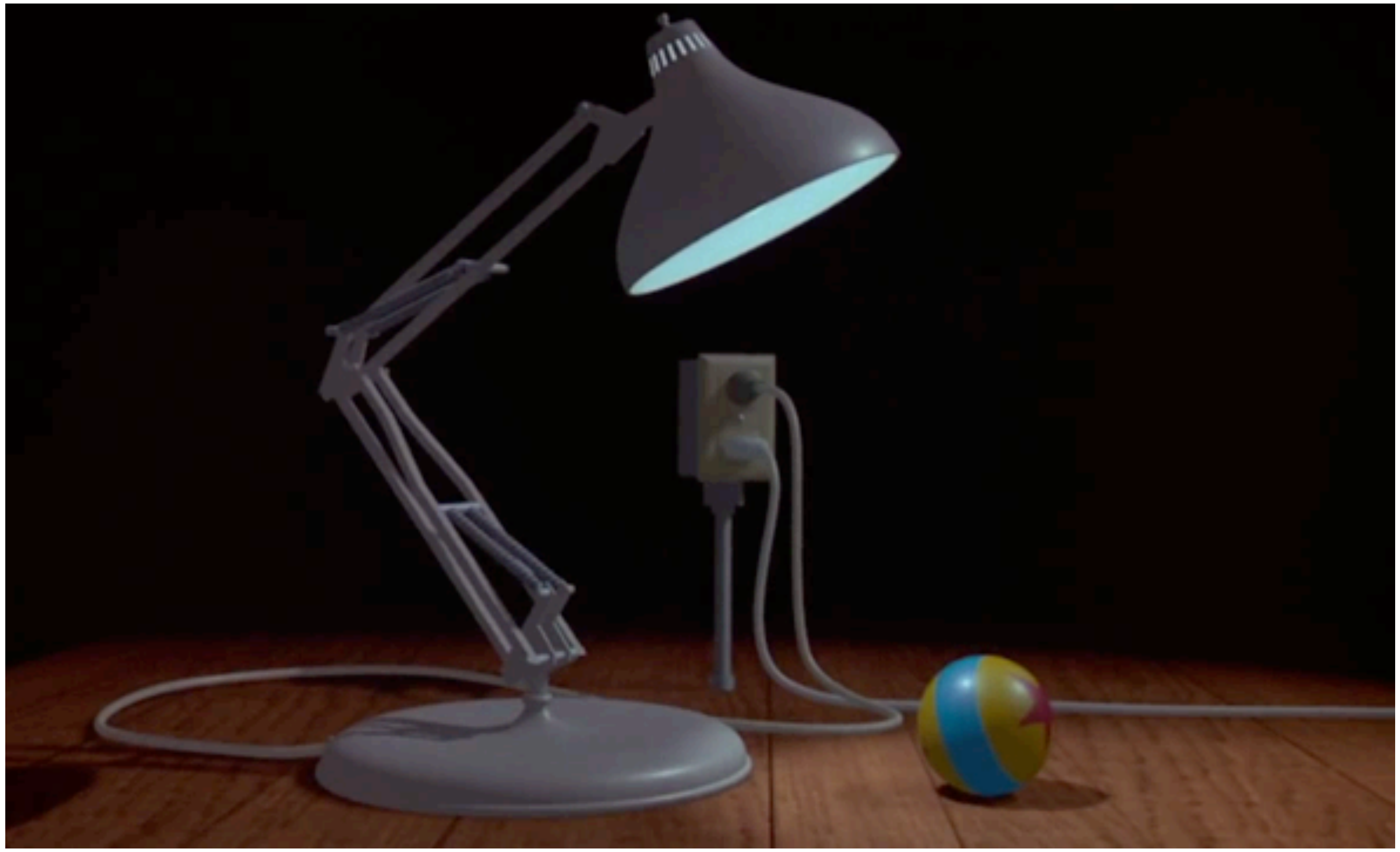


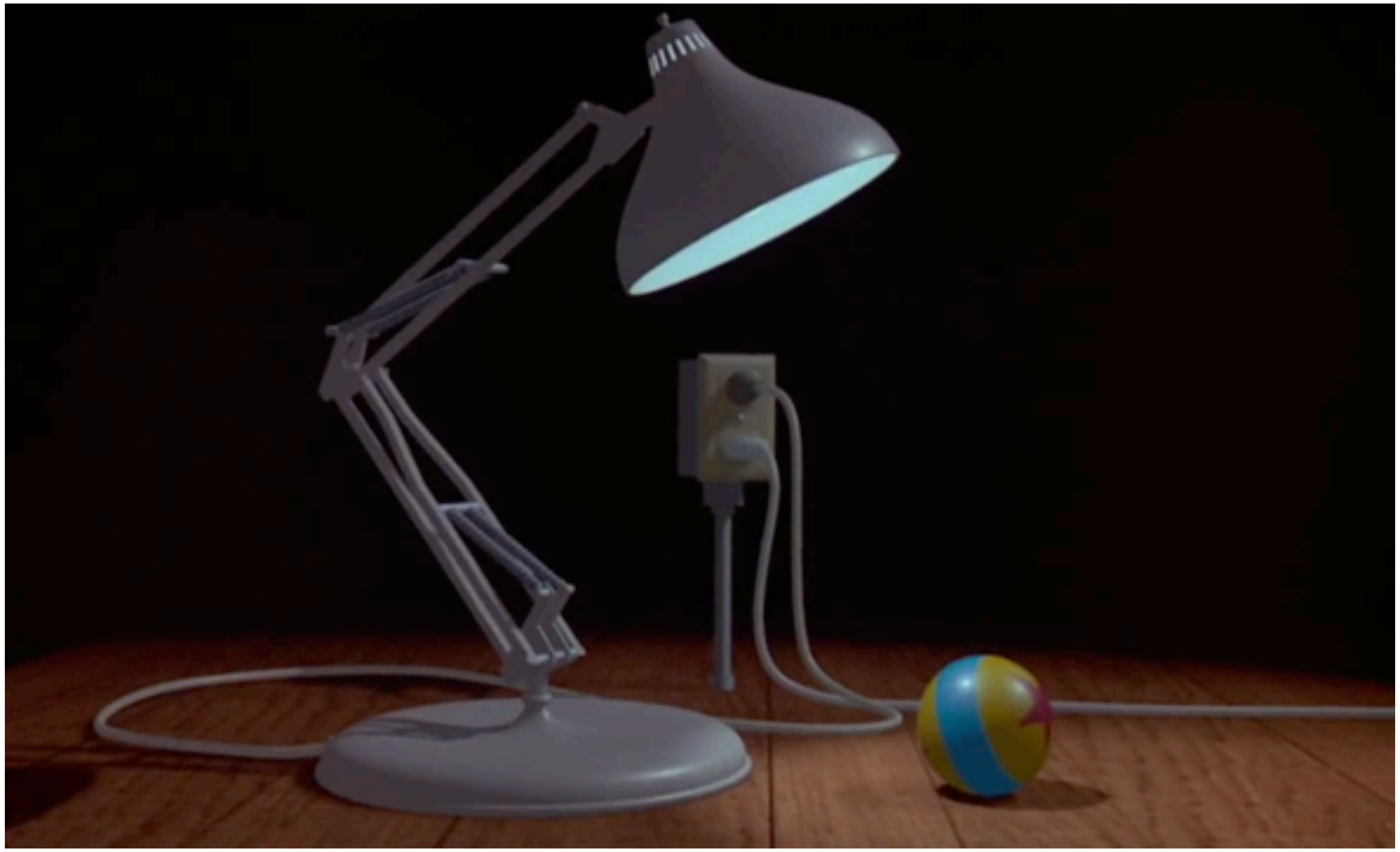
[Michael B. Comet]

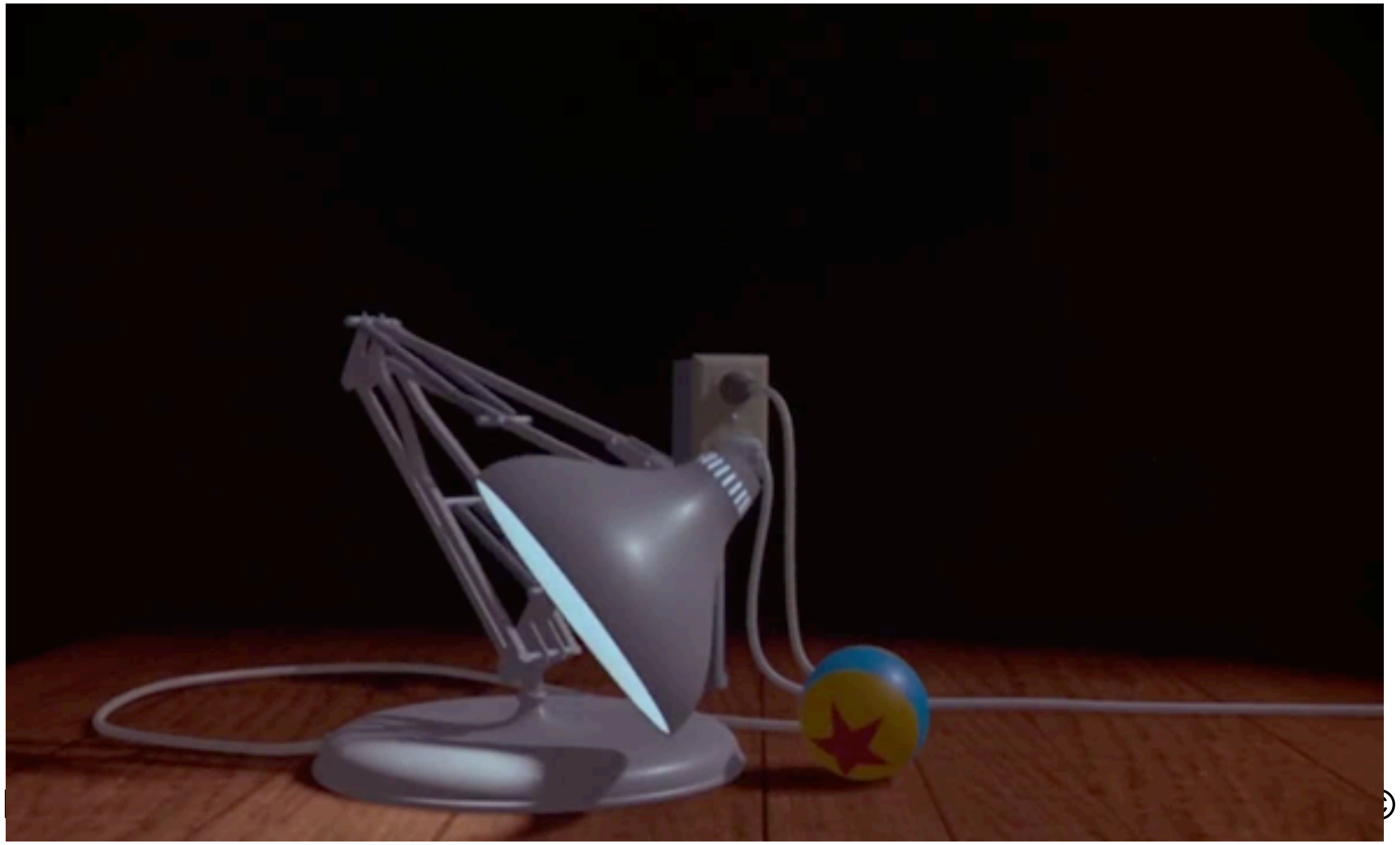




[Michael B. Comet]







Animation principles: exaggeration

- Animation is not about exactly modeling reality
- Exaggeration is very often used for emphasis

[Michael B. Comet]

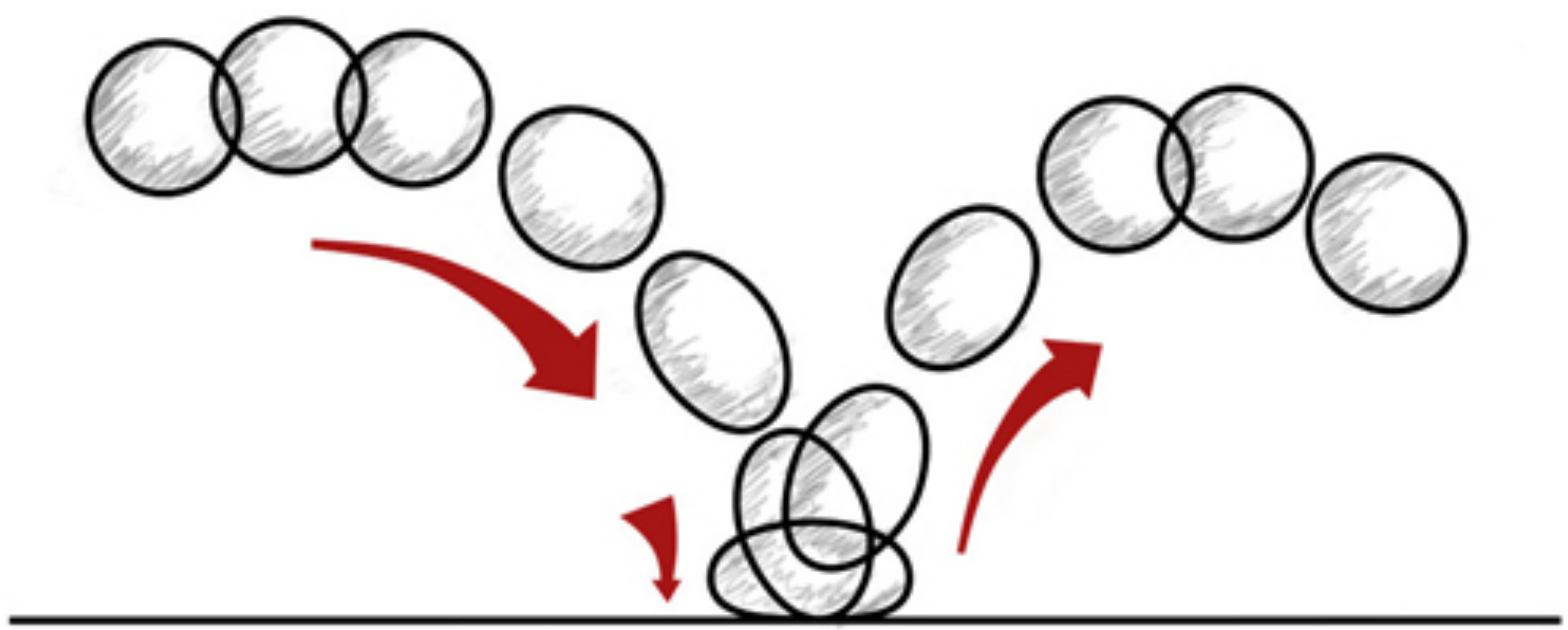
Animation principles: exaggeration

- Animation is not about exactly modeling reality
- Exaggeration is very often used for emphasis

[Michael B. Comet]

Animation principles: squash & stretch

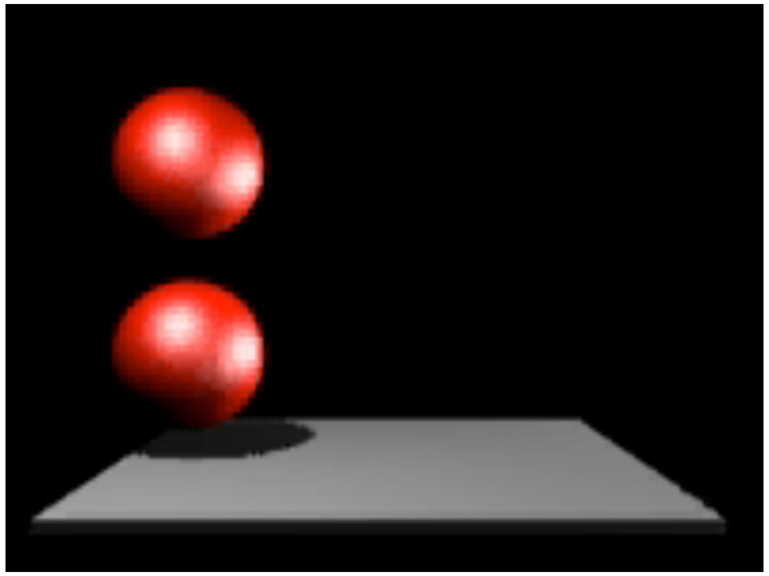
- Objects do not remain perfectly rigid as they move
- Adding stretch with motion and squash with impact:
  - models deformation of soft objects
  - indicates motion by simulating exaggerated "motion blur"



[www.animdesk.com]

Animation principles: follow through

- We've seen that objects don't start suddenly
- They also don't stop on a dime



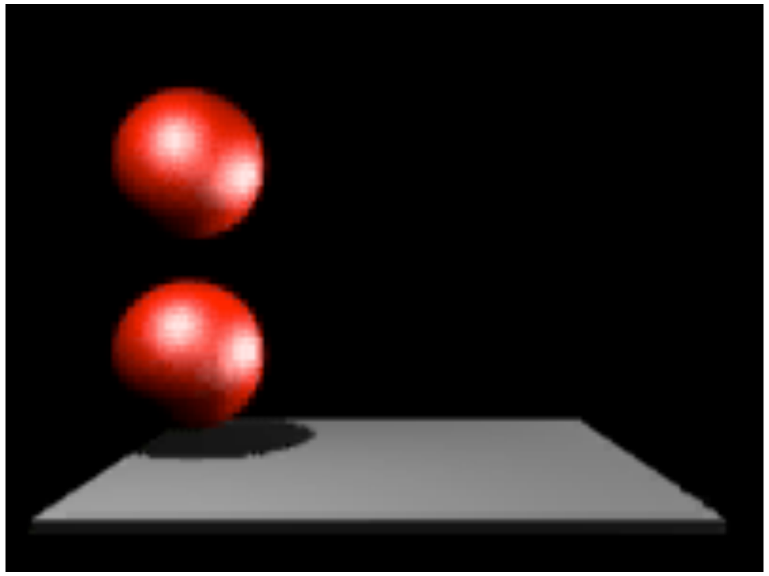
[Michael B. Comet]

[Michael B. Comet]

Animation principles: follow through

- We’ve seen that objects don’t start suddenly
- They also don’t stop on a dime

[Michael B. Comet]



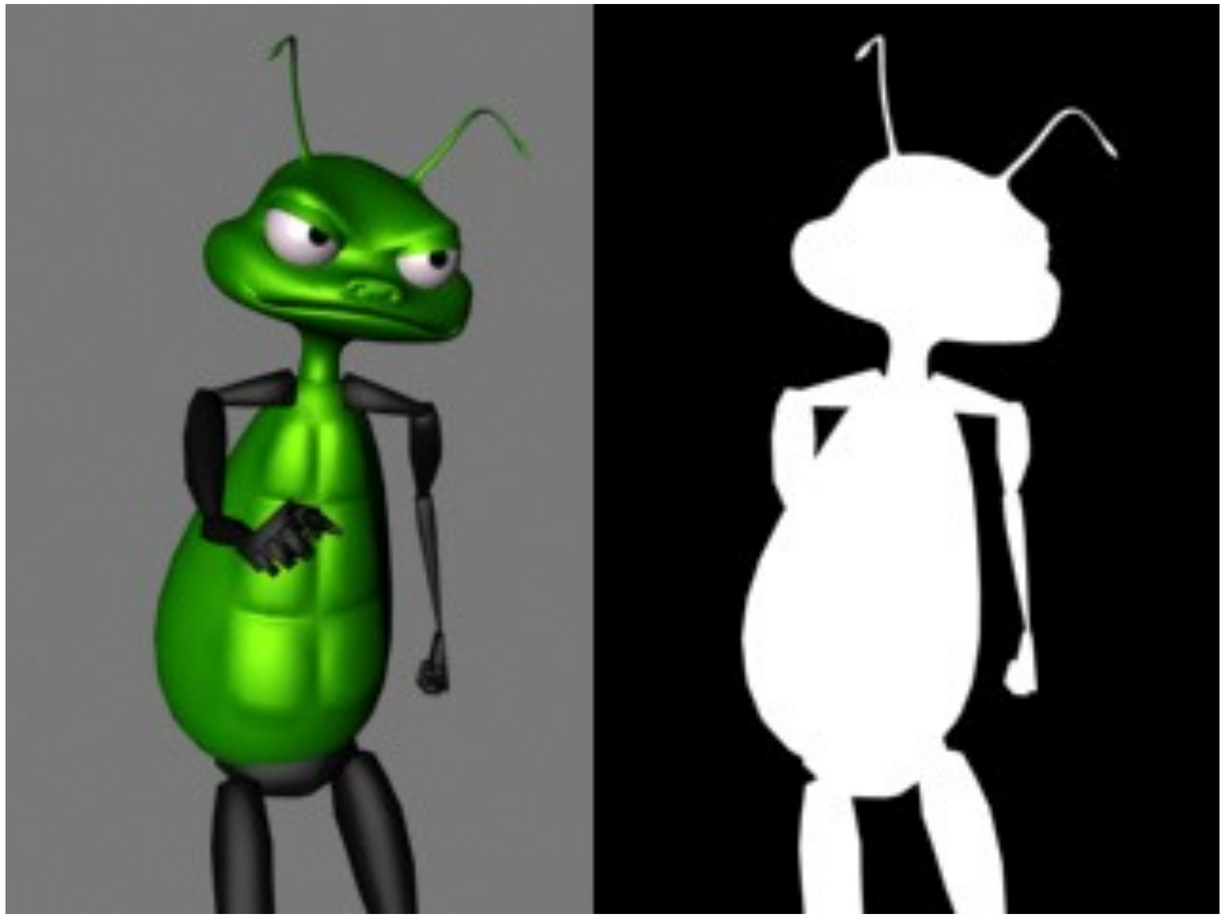
[Michael B. Comet]

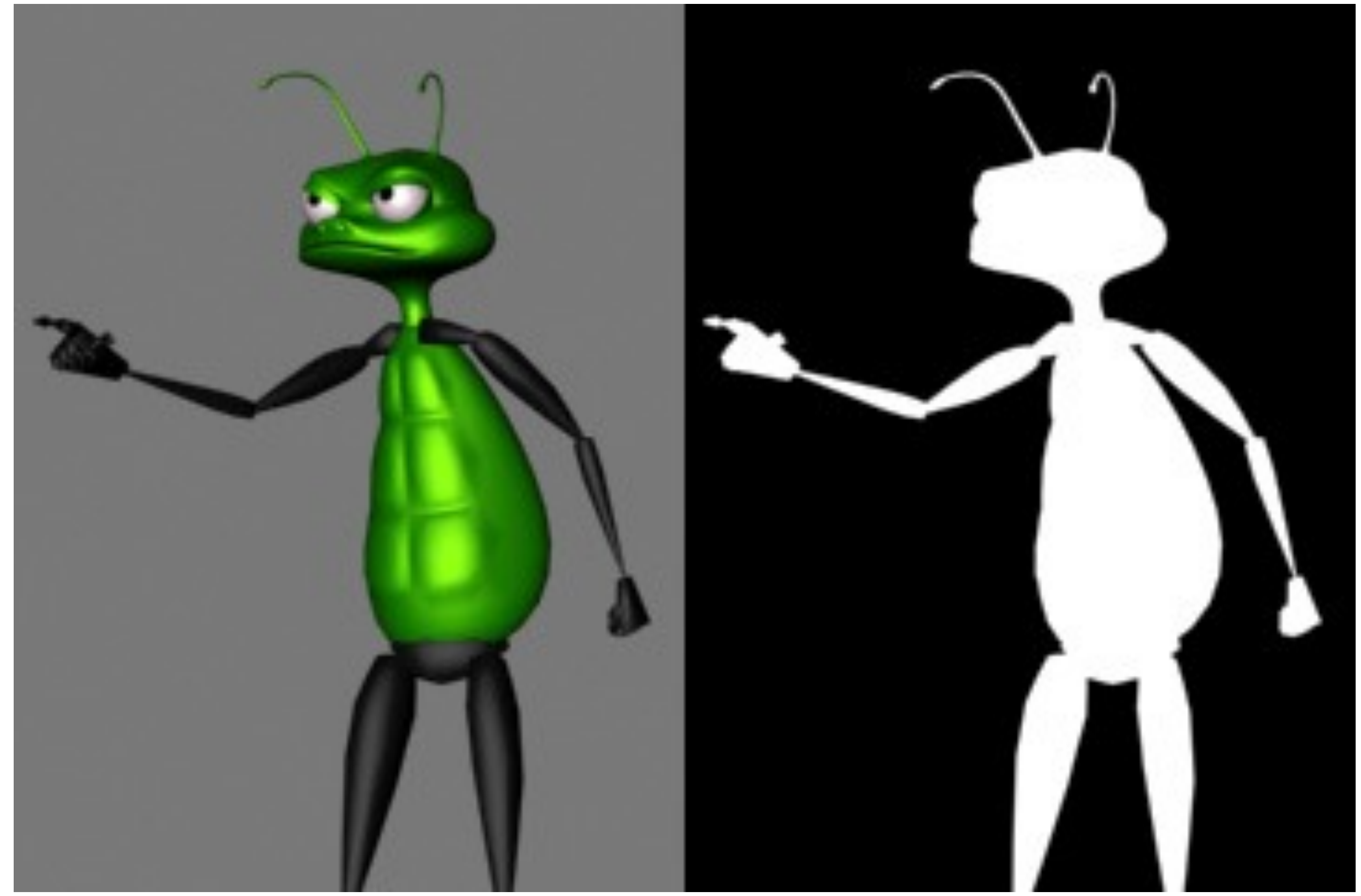
Anim. principles: overlapping action

- Usually many actions are happening at once

[Michael B. Comet]

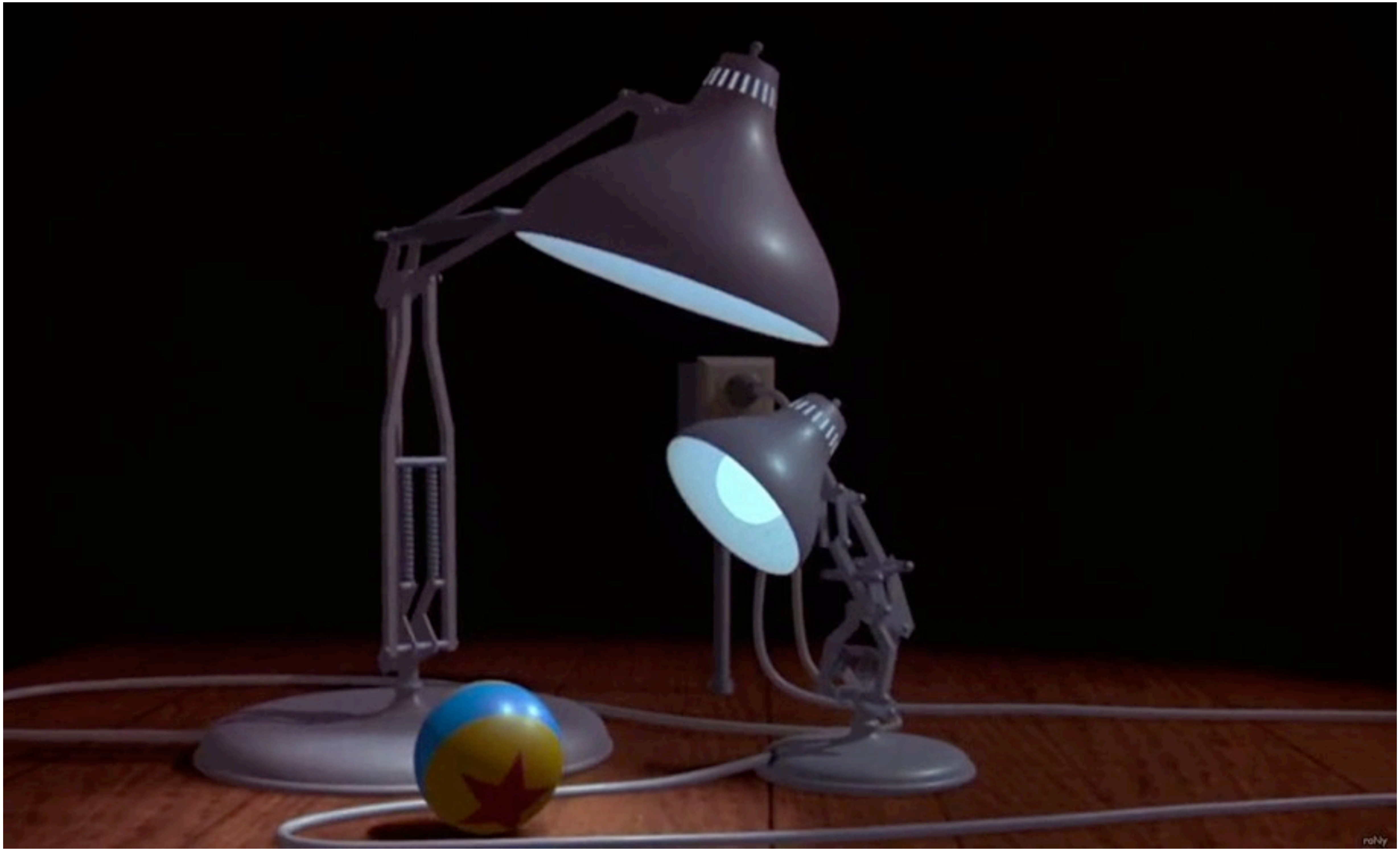
Animation principles: staging





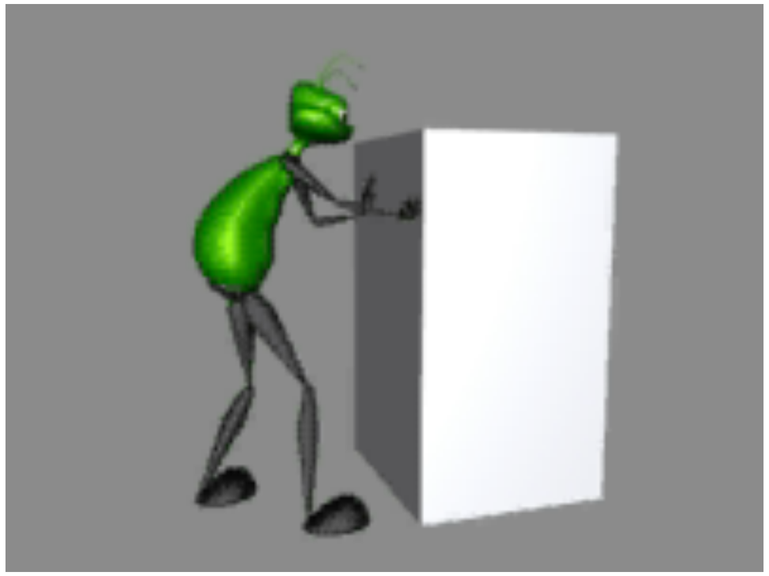
[Michael B. Comet]

- Want to produce clear, good-looking 2D images
  - need good camera angles, set design, and character positions



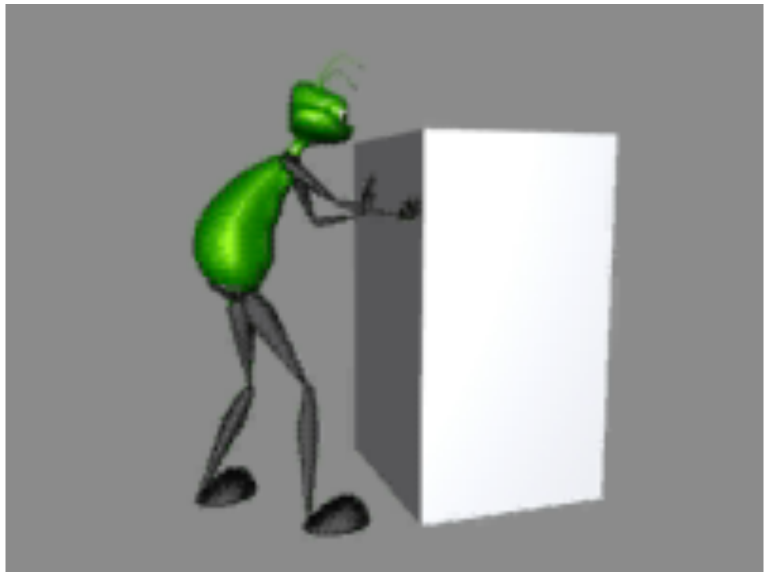
Principles at work: weight

[Michael B. Comet]



Principles at work: weight

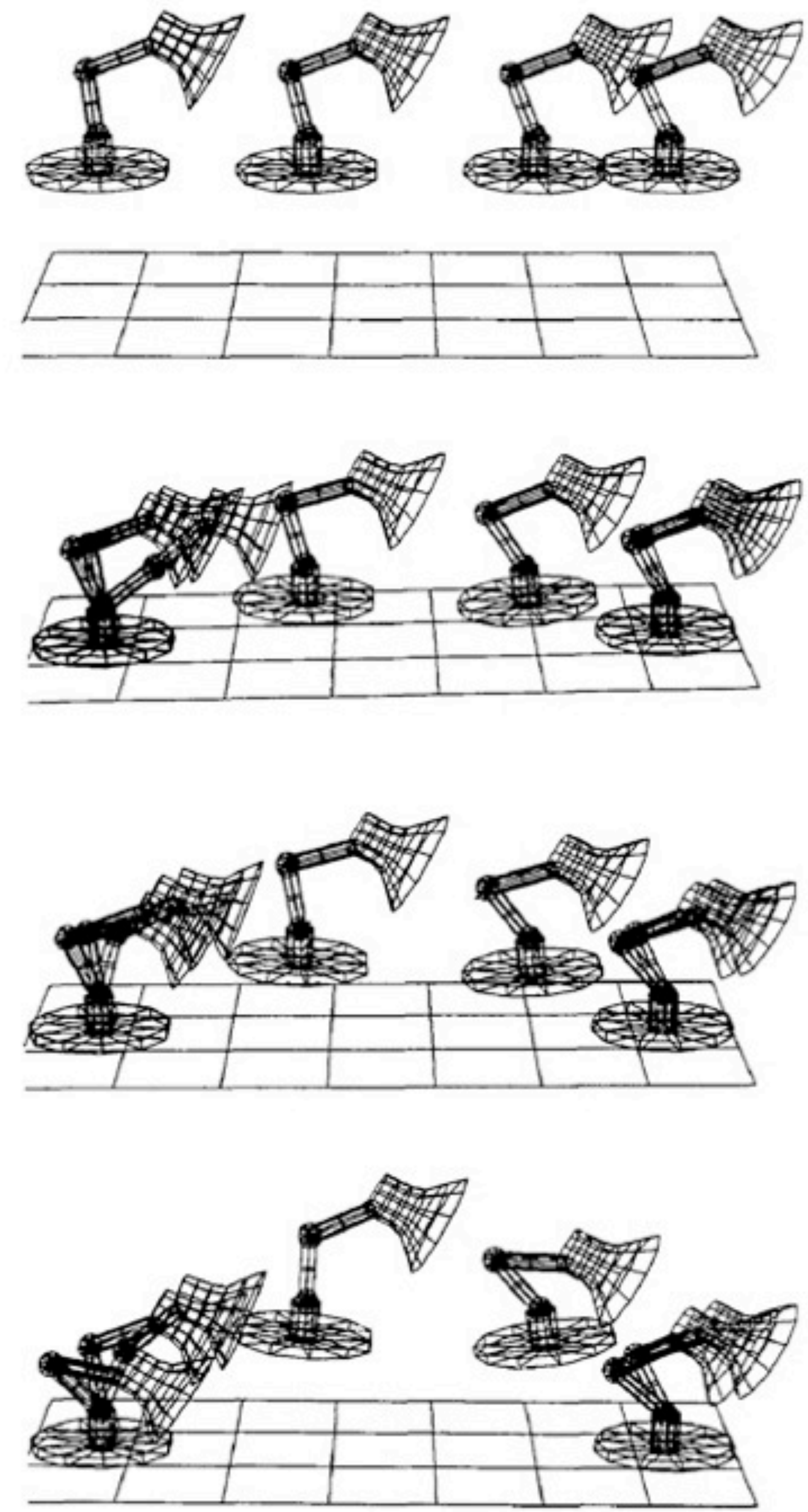
[Michael B. Comet]



Extended example: Luxo, Jr.

Computer-generated motion

- Interesting aside: many principles of character animation follow indirectly from physics
- Anticipation, follow-through, and many other effects can be produced by simply minimizing physical energy
- Seminal paper: "Spacetime Constraints" by Witkin and Kass in SIGGRAPH 1988



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

Interpolating transformations

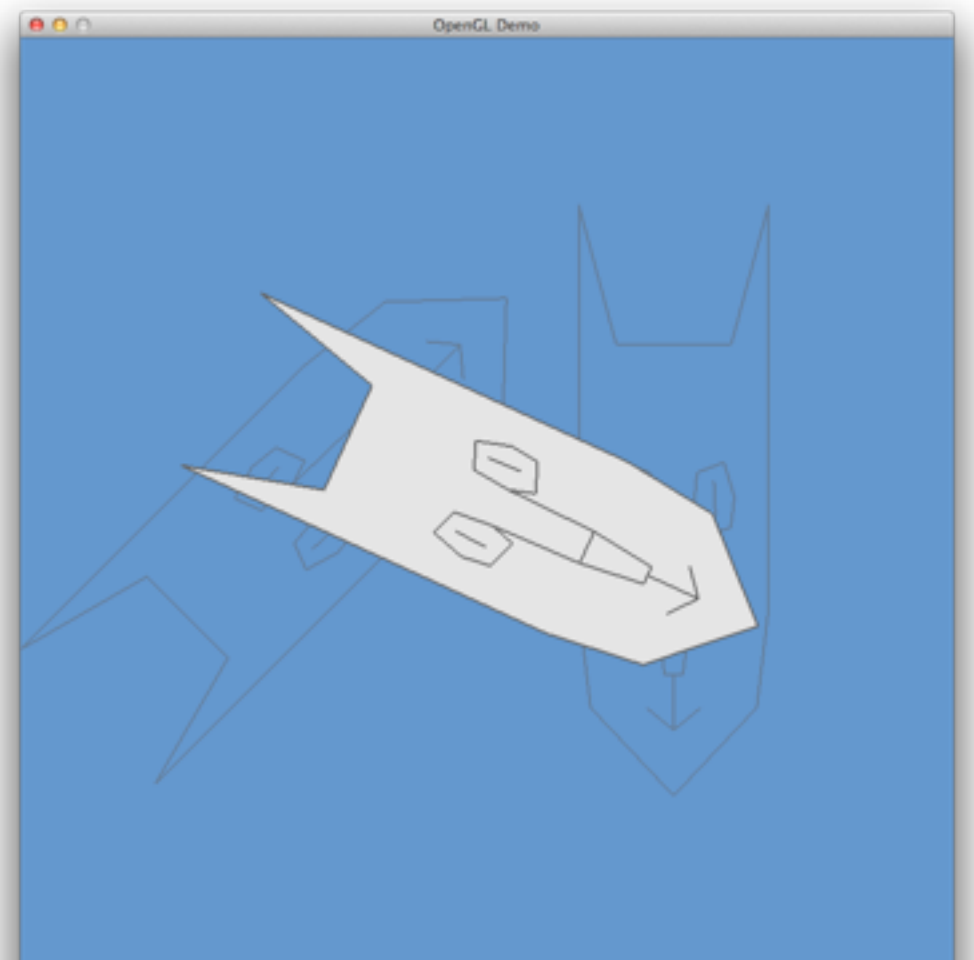
- Linear interpolation of matrices is not effective
  - leads to shrinkage when interpolating rotations
- One approach: always keep transformations in a canonical form (e.g. translate-rotate-scale)
  - then the pieces can be interpolated separately
  - rotations stay rotations, scales stay scales, all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

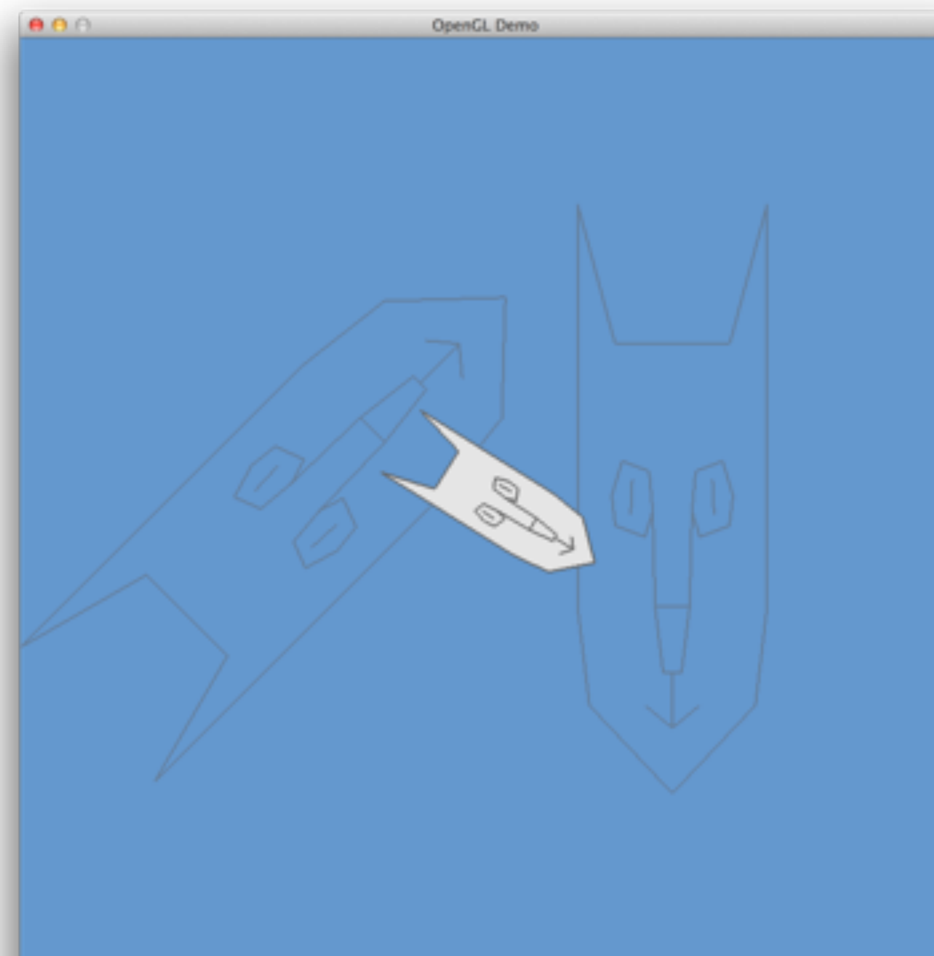
Decomposing transformations

- A product M = TRS is not hard to take apart
  - translation sits in the top right
  - RS still has three orthogonal columns (prove?)
  - scale factors are the lengths of the columns
- But that doesn't cover everything
  - count DOFs: $3 + 3 + 3 < 12$
- If we allow S to be a scale along arbitrary axes then it does work—M = TRS where
  - T is a translation
  - R is a rotation
  - S is a symmetric matrix (positive definite if no reflection)
  - linear algebra name: polar decomposition (at least the A = RS part)

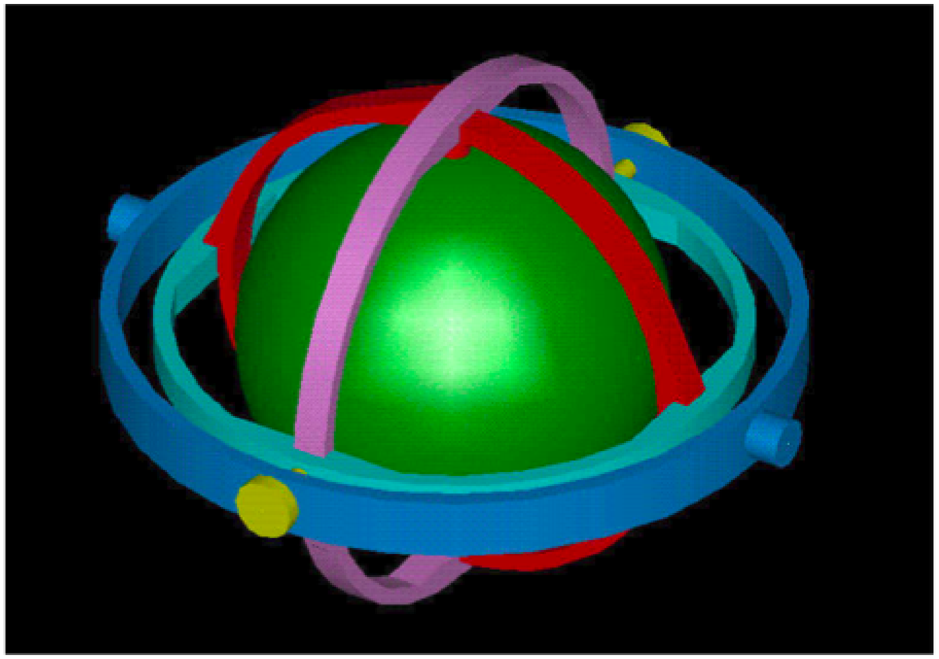
Demo: polar interpolation in 2D

- To interpolate:
  - decompose into T (2-vector), R (just an angle), S (2x2 symmetric)
  - interpolate everything linearly
  - recompose matrix from new translation, angle, and symmetric matrix





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 (like 3D unit vectors for 2D sphere)
  - Good choice for interpolating rotations
- These are first examples of motion control
  - Matrix = deformation
  - Angles/quaternion = animation controls

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

Calculating with rotations

- Representing rotations with numbers requires a function

$$f : \mathbb{R}^n \to SO(3)$$

- The situation is analogous to representing directions in 3-space
  - there we are dealing with the set S^2, the two-dimensional sphere (I mean the sphere is a 2D surface)
  - like SO(3) it is very symmetric; no directions are specially distinguished

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: 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: 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

$$\hat{\mathbf{v}}(t) = \mathrm{normalize}((1-t)\mathbf{v}_0 + t\mathbf{v}_1)$$

  - 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)

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

Warm-up: rays vs. lines

- The set of directions (unit vectors) describes the set of rays leaving a point
- The set of lines through a point is a bit different
  - no notion of "forward" vs. "backward"
- Would probably still represent using unit vectors
  - but every line has exactly two representations, **v** and –**v**

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

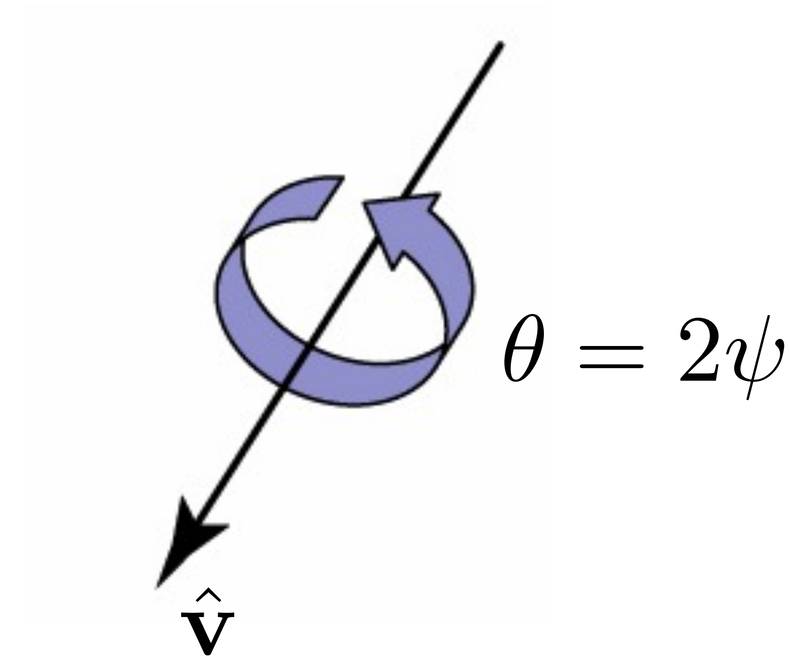
There is a natural association between the unit quaternion

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

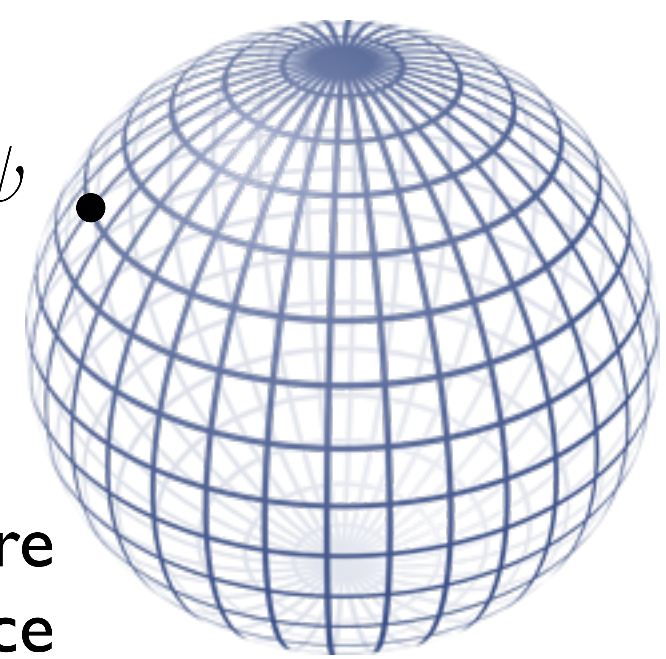
and the 3D axis-angle rotation

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

where $\theta = 2\psi$.



$\cos\psi + \hat{\mathbf{v}}\sin\psi$



unit 3-sphere
in 4D space

[Wikimedia Commons user Geek3]

Unit quaternions and axis/angle

- We can write down a parameterization of 3D rotations using unit quaternions (points on the 3-sphere)

$$
\begin{aligned}
f &: S^3 \subset \mathbf{H} \to SO(3) \\
&: \cos\psi + \hat{\mathbf{v}} \sin\psi \mapsto R_{\hat{\mathbf{v}}}(2\psi) \\
&: (w, x, y, z) \mapsto \begin{bmatrix} w^2 + x^2 - y^2 - z^2 & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & w^2 - x^2 + y^2 - z^2 & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & w^2 - x^2 - y^2 + z^2 \end{bmatrix}
\end{aligned}
$$

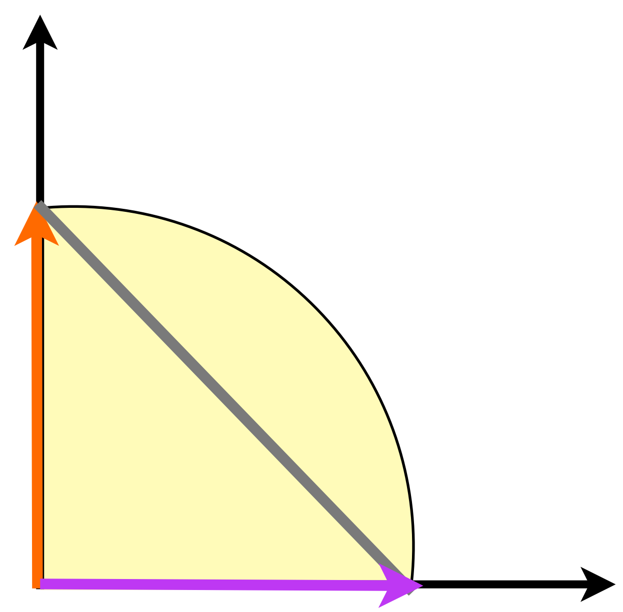
- This mapping is wonderfully uniform:
  - is exactly 2-to-1 everywhere
  - has constant speed in all directions
  - has constant Jacobian (does not distort "volume")
  - maps shortest paths to shortest paths
  - *and*… it comes with a multiplication operation (not mentioned today)

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 constant rate of rotation



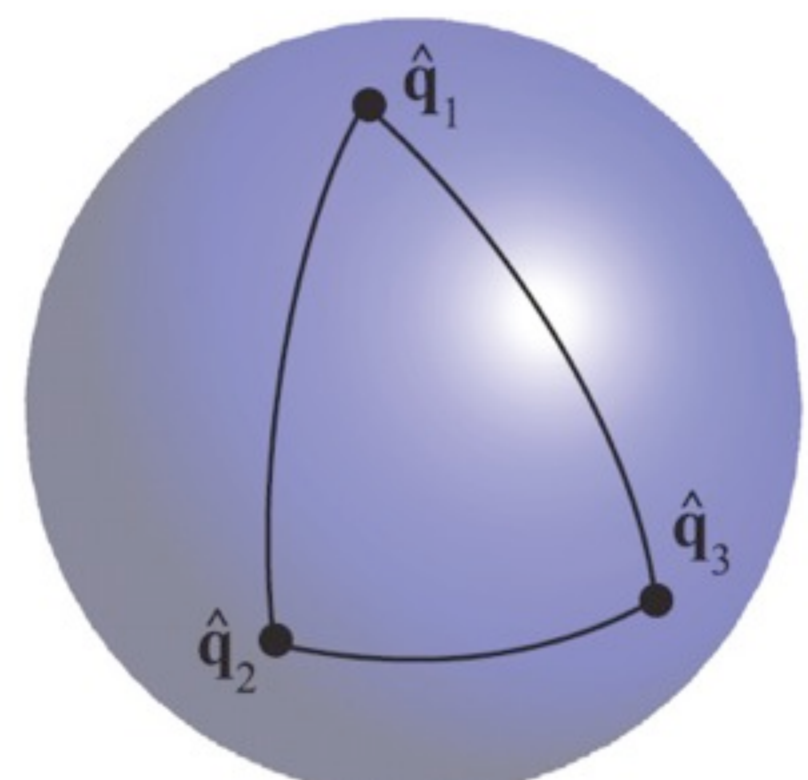
$$\frac{(1-\alpha)x + \alpha y}{||(1-\alpha)x + \alpha y||}$$

Spherical Linear Interpolation

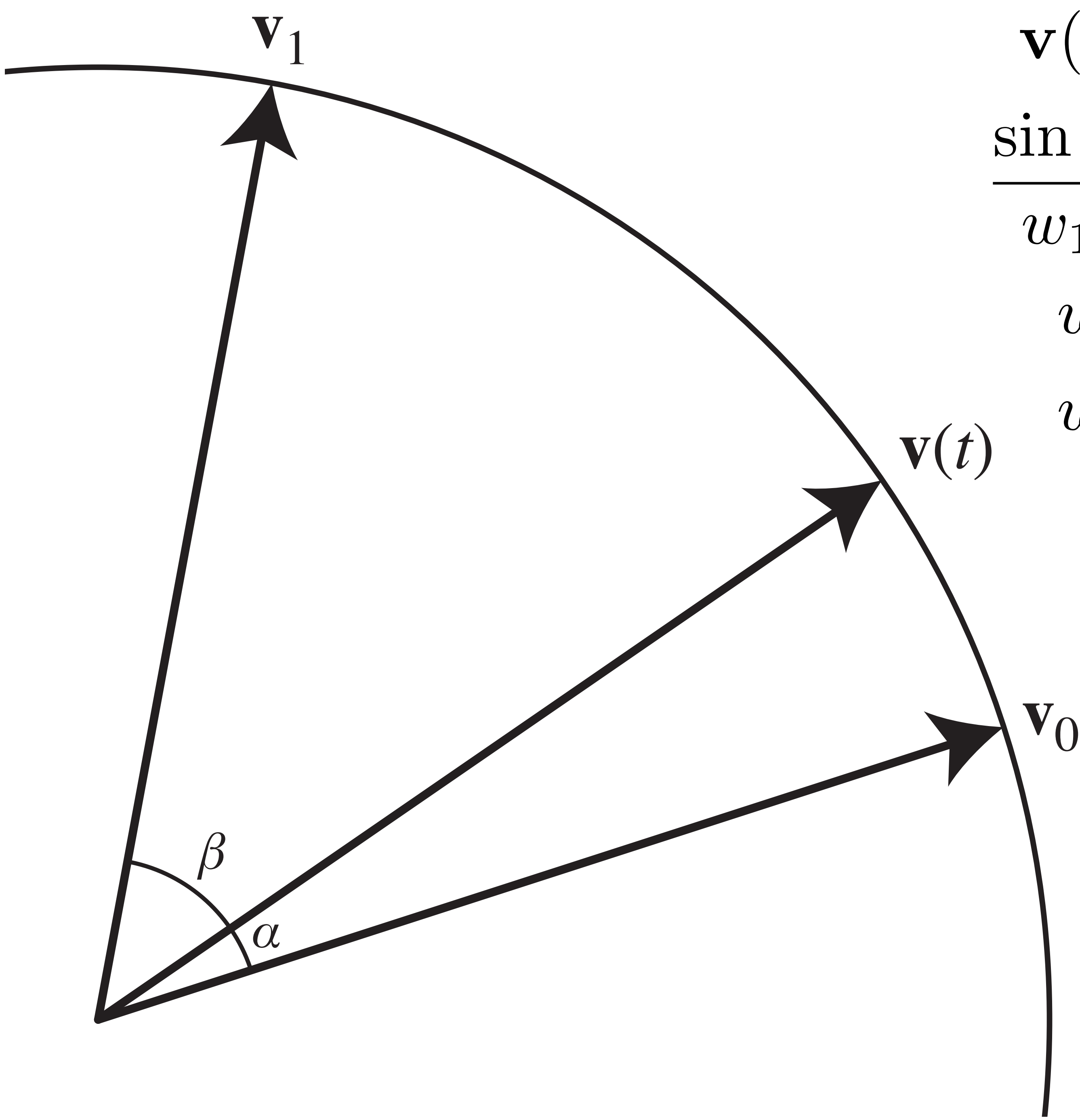
- Intuitive interpolation between different orientations
  - Nicely represented through quaternions
  - Useful for animation
  - Given two quaternions, interpolate between them

  - Shortest path between two points on sphere

    Geodesic, on Great Circle



Spherical linear interpolation ("slerp")



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

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

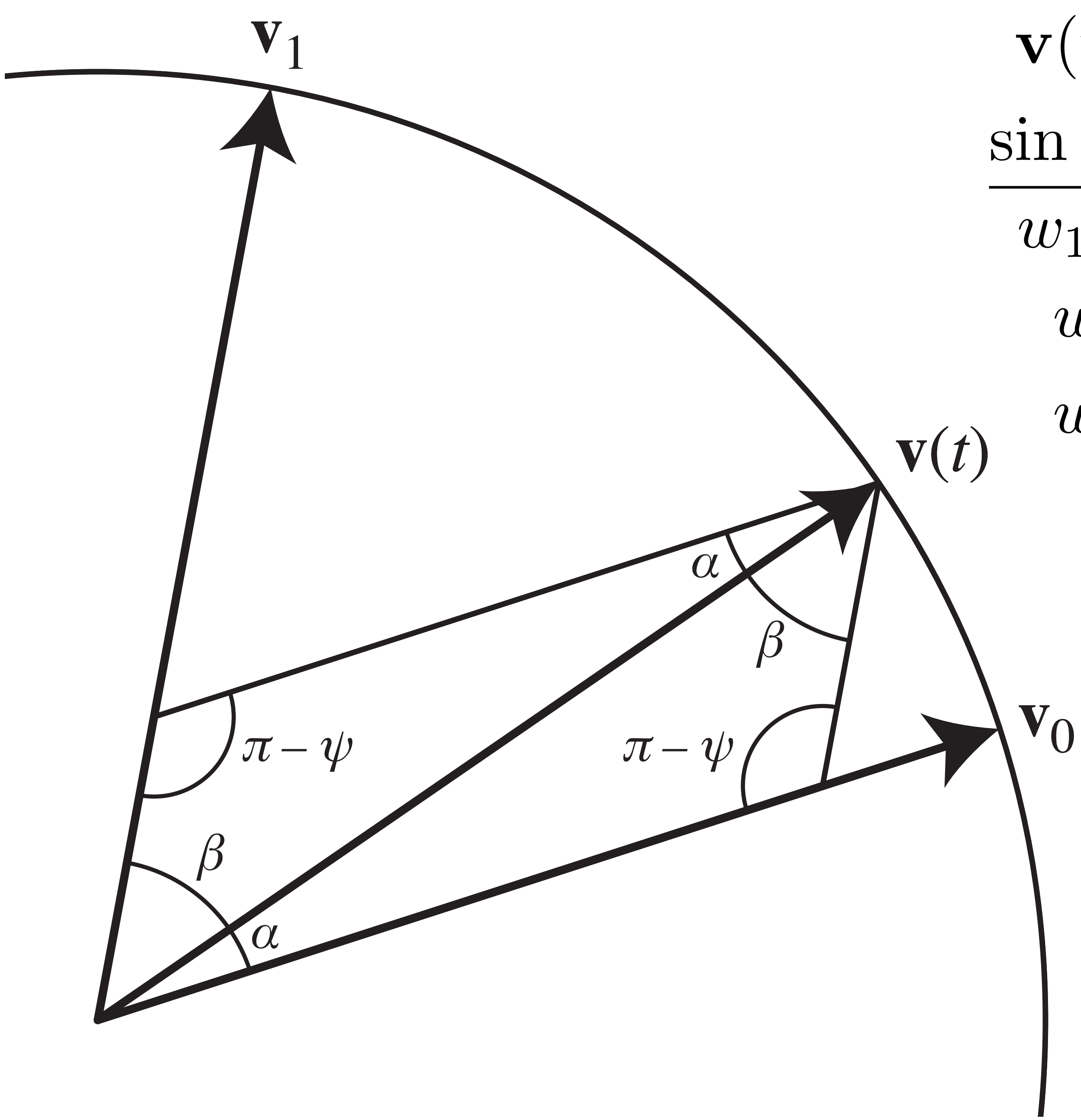
$$\frac{\sin \alpha}{w_1} = \frac{\sin \beta}{w_0} = \frac{\sin(\pi - \psi)}{1} = \sin \psi$$

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

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

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

Spherical linear interpolation ("slerp")



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

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

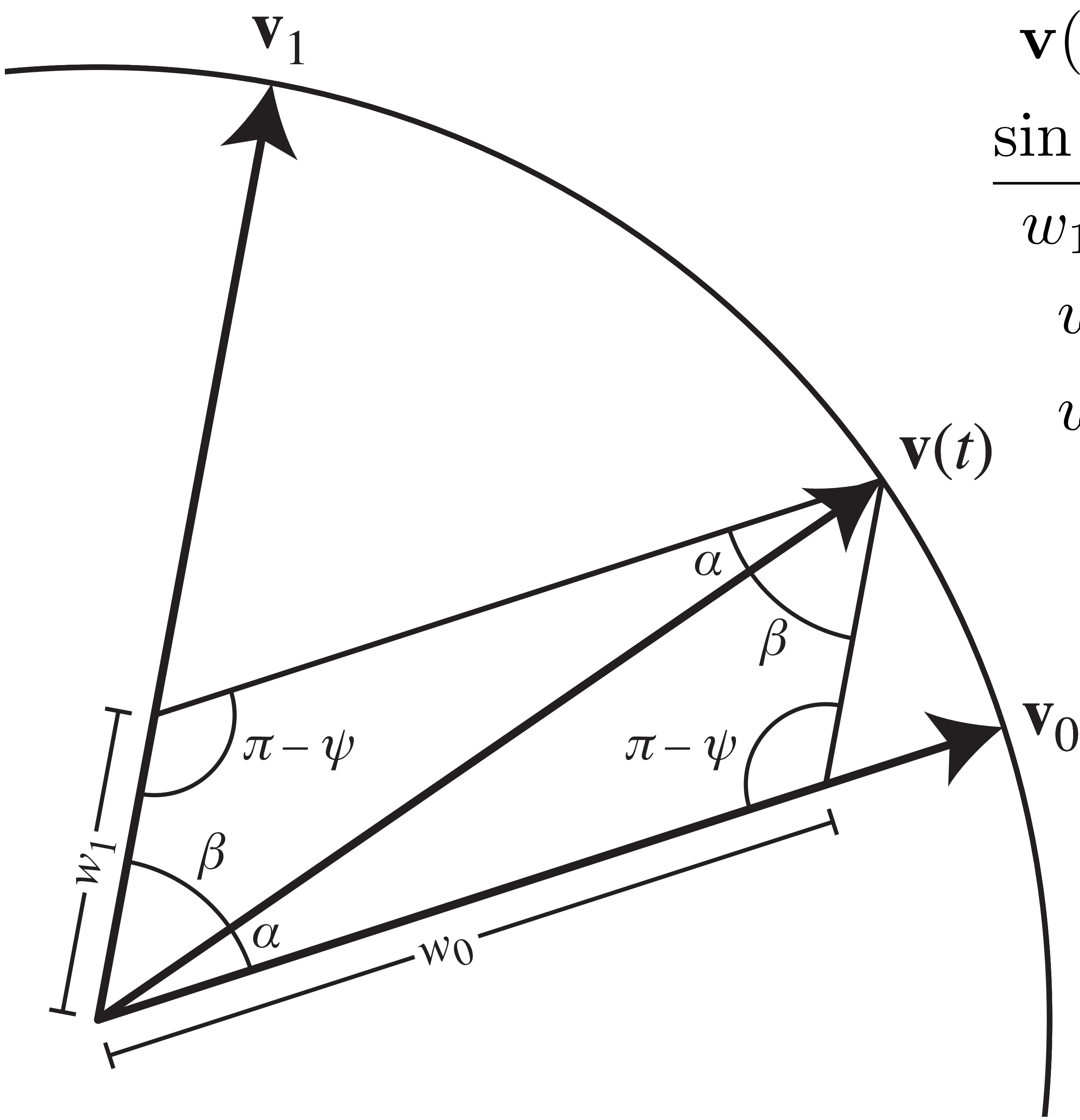
$$\frac{\sin \alpha}{w_1} = \frac{\sin \beta}{w_0} = \frac{\sin(\pi - \psi)}{1} = \sin \psi$$

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

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

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

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

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

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

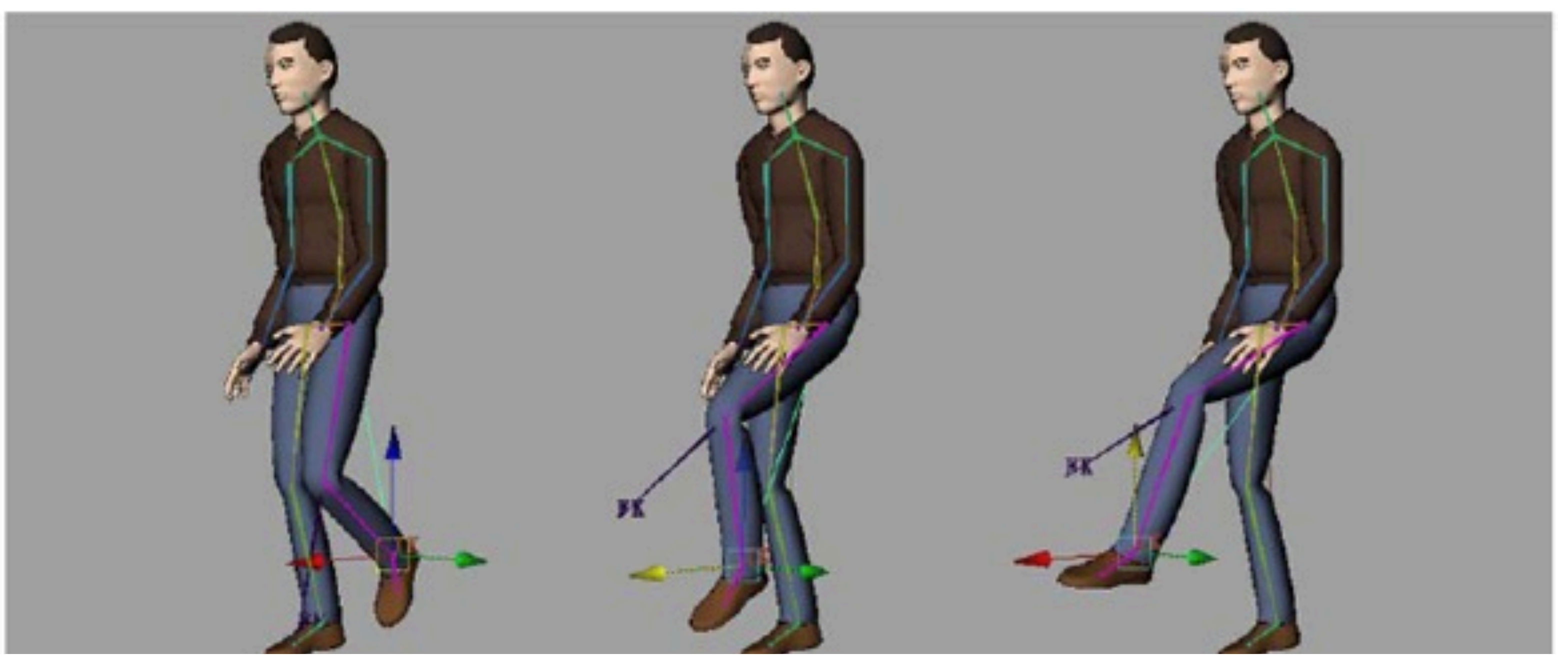
Practical issues

- When angle gets close to zero, estimation of ψ is inaccurate
  - slerp naturally approaches linear interpolation for small ψ
  - so switch to linear interpolation when $q_0 \approx q_1$.
- q is same rotation as $-q$
  - if $q_0 \cdot q_1 > 0$, slerp between them
  - else, slerp between q_0 and $-q_1$

Hierarchies and articulated figures

- Luxo as an example
  - small number of animation controls control many transformations
  - constraint: the joints hold together
- Some operations are tricky with hierarchies
  - how to ensure lampshade touches ball?
- In mechanics, the relationship between DOFs and 3D pose is *kinematics*
- Robotics as source of math. Methods
  - robots are transformation hierarchies
  - forward kinematics
  - inverse kinematics

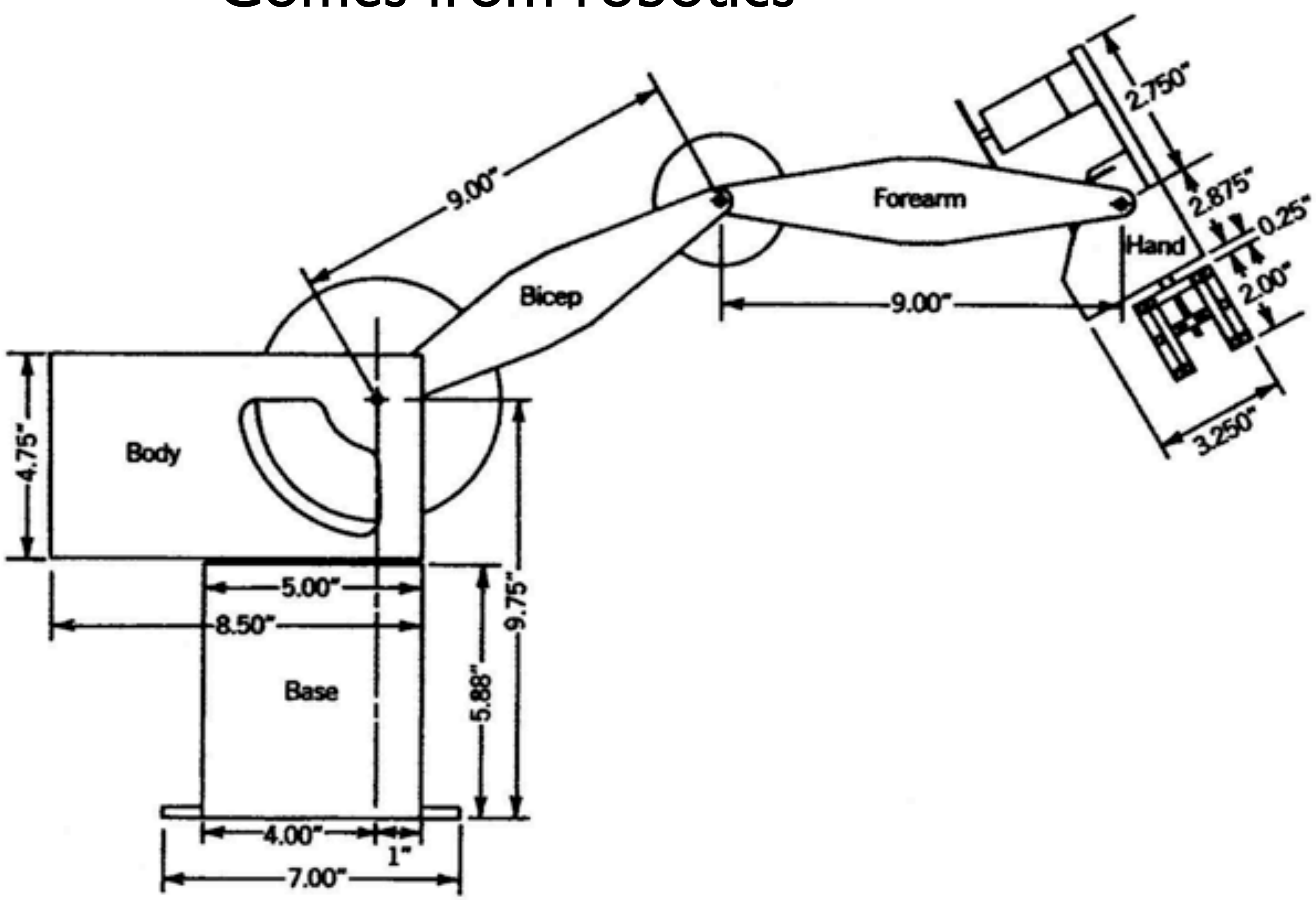
Forward Kinematics

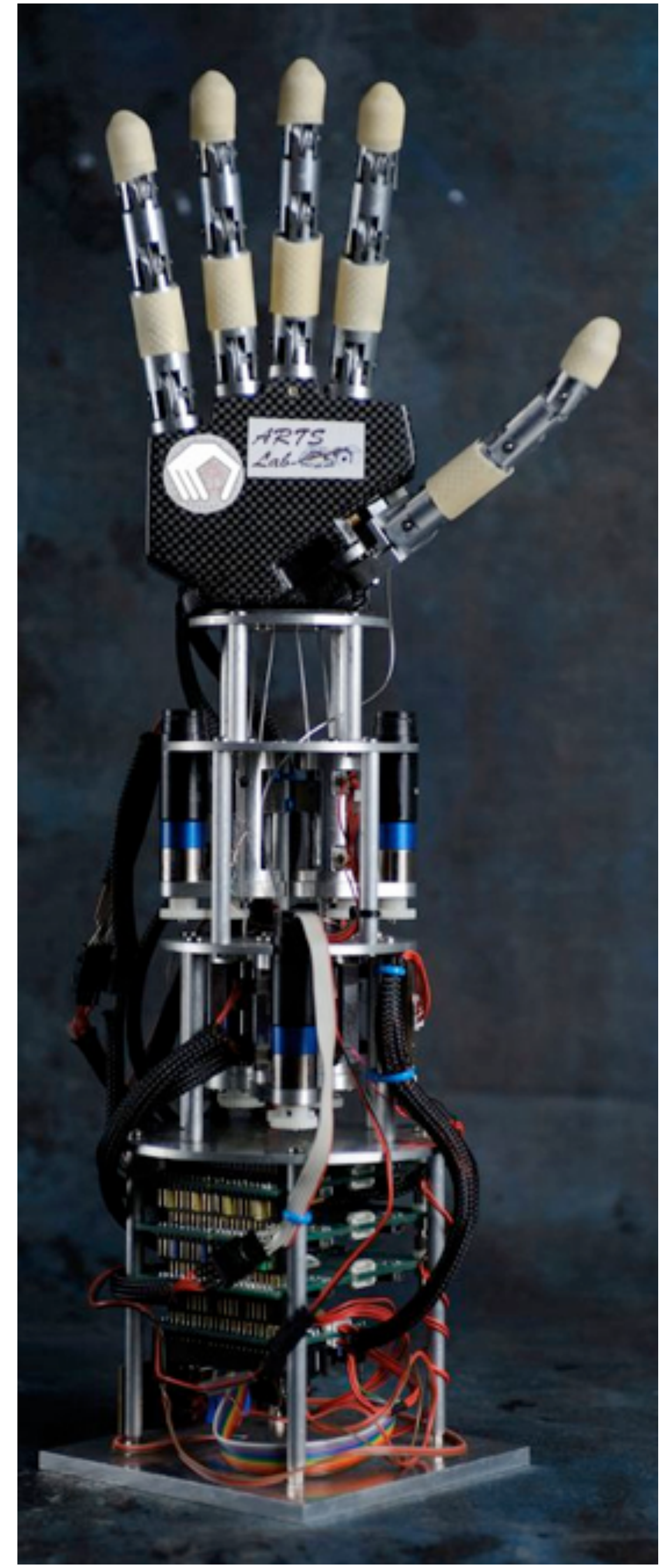


Inverse Kinematics

Forward Kinematics

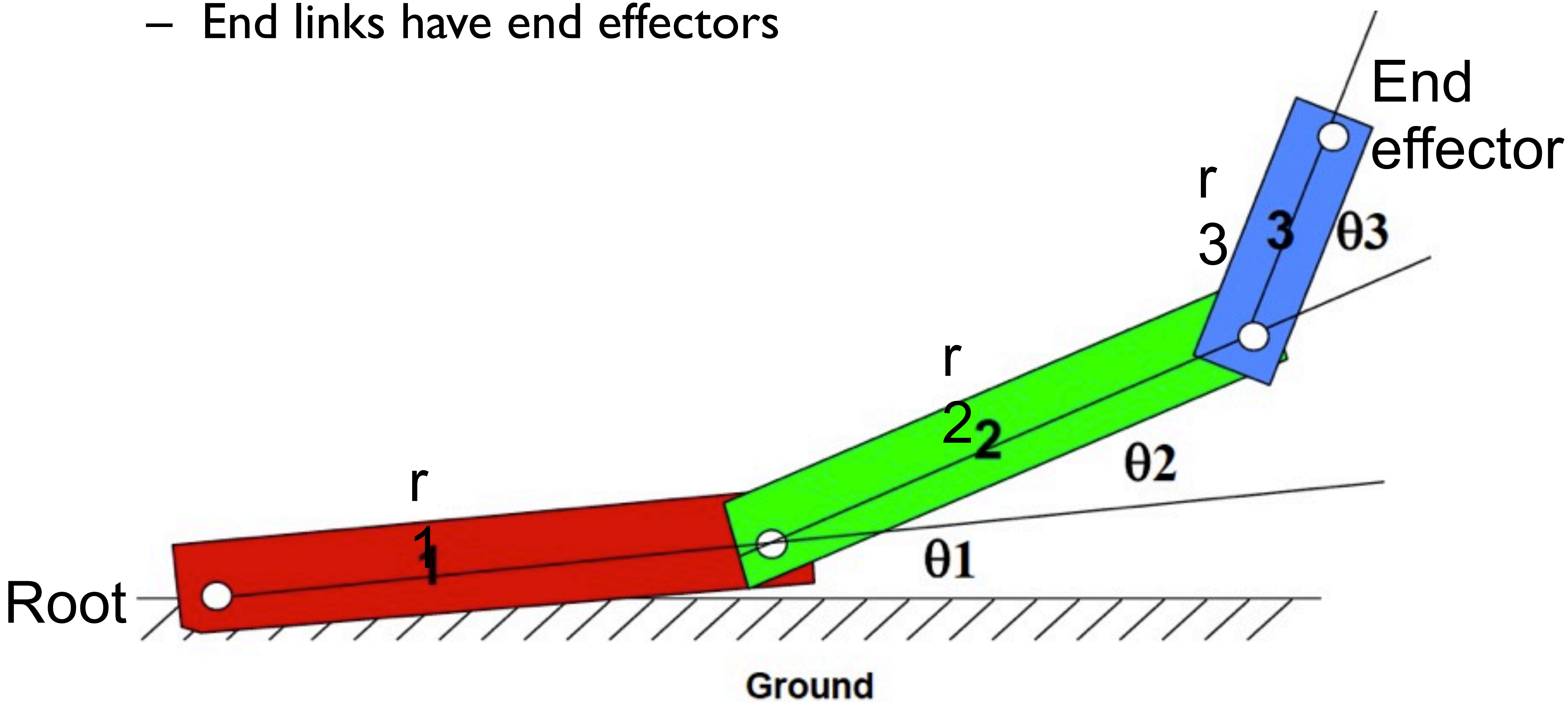
- Articulated body
  - Hierarchical transforms
  - Comes from robotics



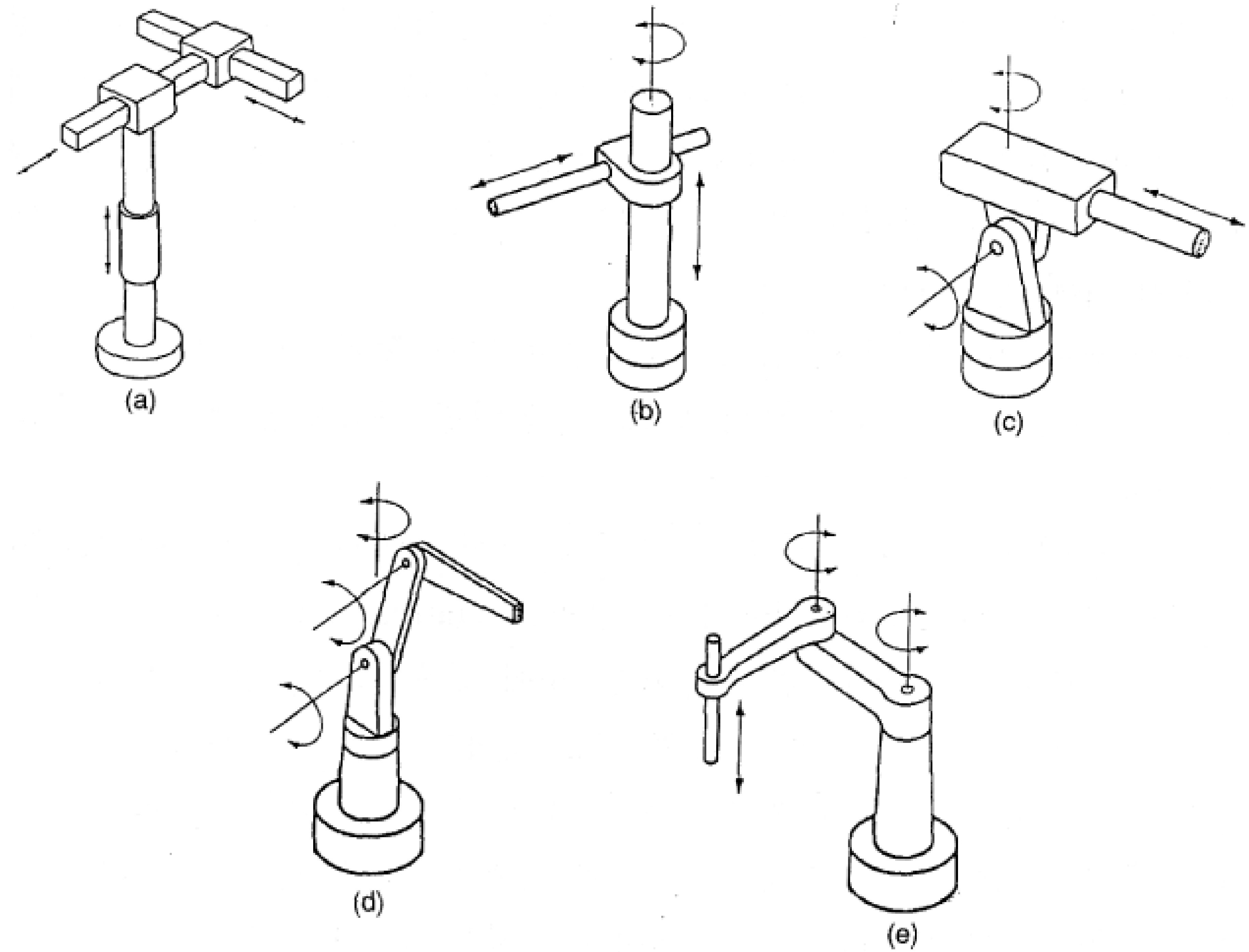


Rigid Links and Joint Structure

- Links connected by joints
  - Joints are purely rotational (single DOF)
  - Links form a tree (no loops)
  - End links have end effectors



Articulation in robotics



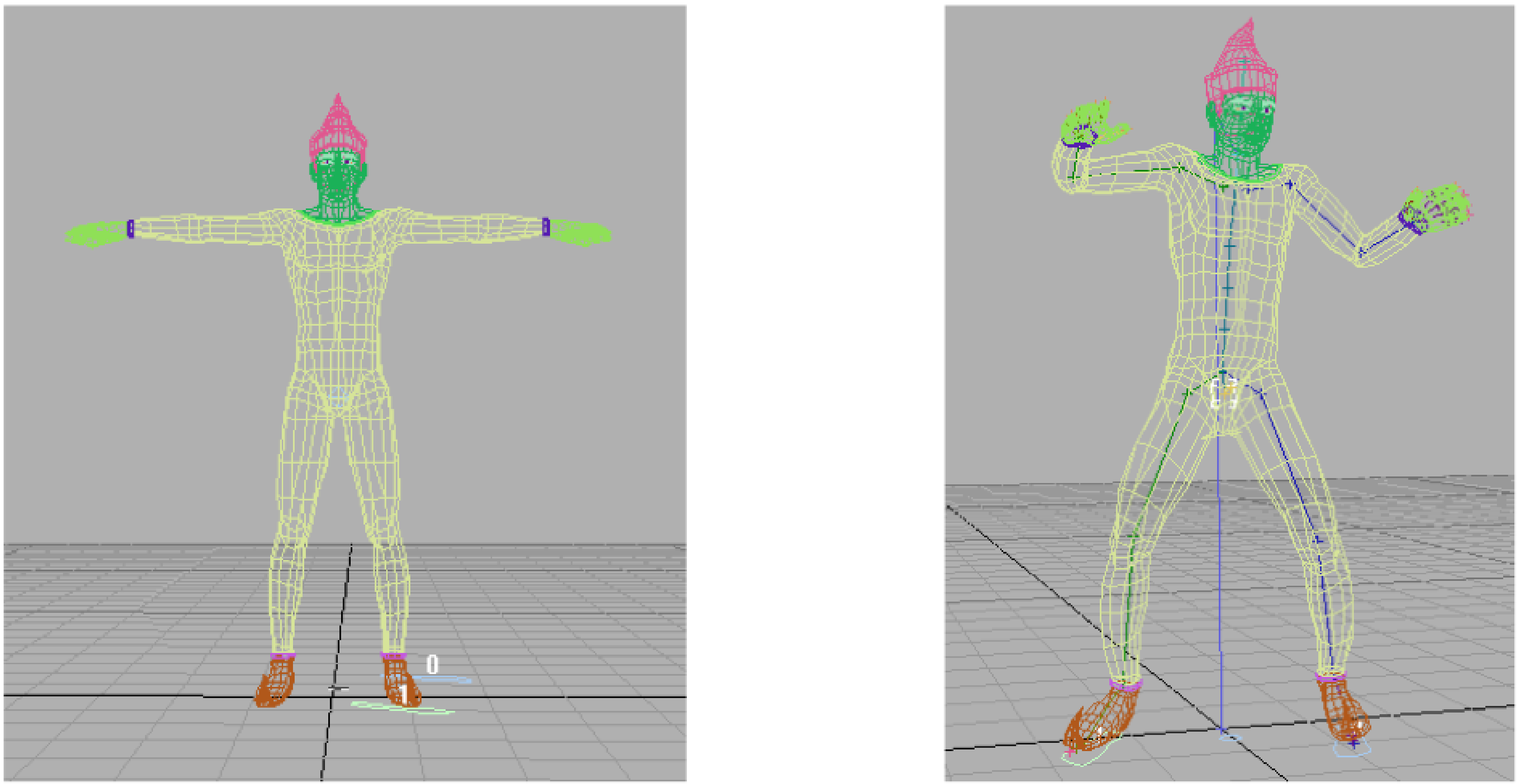
a. rectangular or cartesian
b. cylindrical or post-type
c. spherical or polar
d. joint-arm or articulated
e. SCARA (selective compliance assembly robot arm)

Basic surface deformation methods

- Mesh skinning: deform a mesh based on an underlying skeleton
- Blend shapes: make a mesh by combining several meshes
- Both use simple linear algebra
  - Easy to implement—first thing to try
  - Fast to run—used in games
- The simplest tools in the offline animation toolbox

Mesh skinning

- A simple way to deform a surface to follow a skeleton



[Sébastien Dominé | NVIDIA]

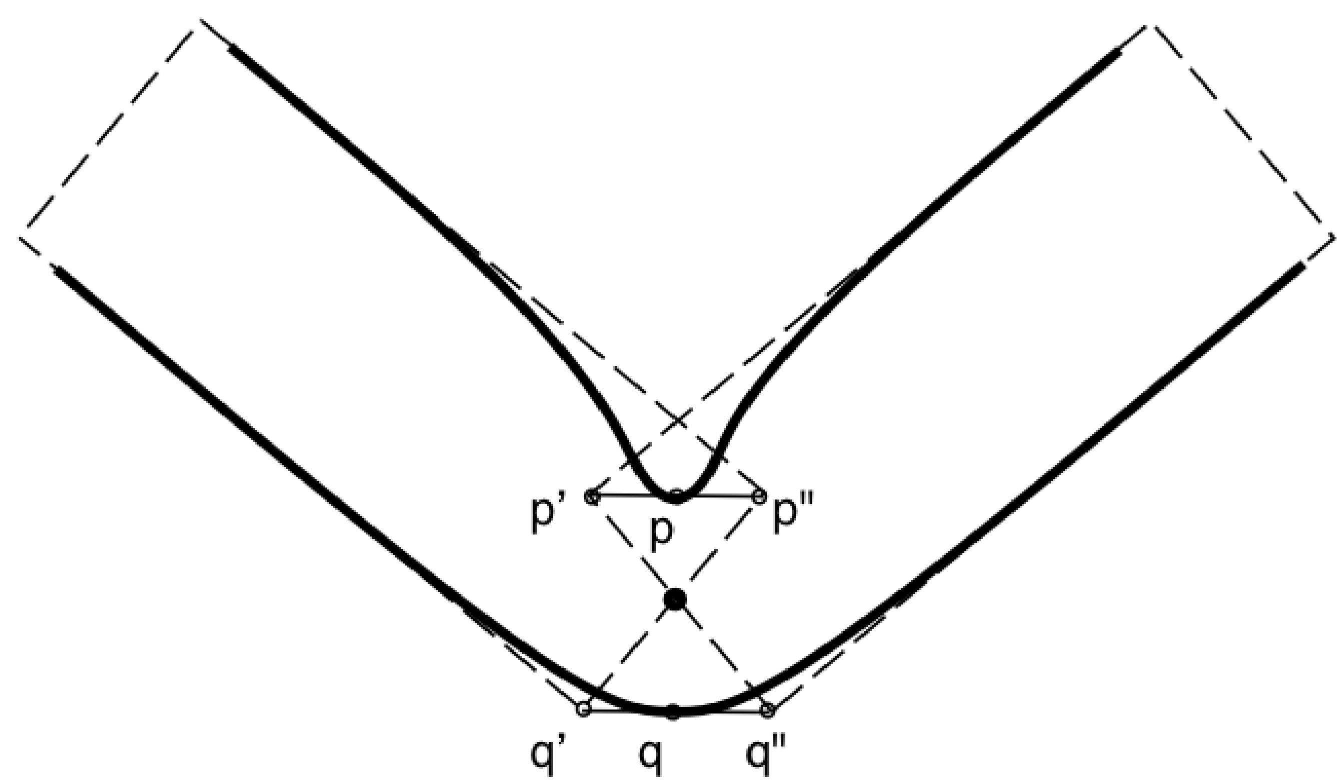
Mesh skinning math: setup

- Surface has control points $\mathbf{p}_i$
  - Triangle vertices, spline control points, subdiv base vertices
- Each bone has a transformation matrix M_j
  - Normally a rigid motion
- Every point–bone pair has a weight w_{ij}
  - In practice only nonzero for small # of nearby bones
  - The weights are provided by the user

Mesh skinning math

- Deformed position of a point is a weighted sum
  - of the positions determined by each bone's transform alone
  - weighted by that vertex's weight for that bone

$$\mathbf{p}'_i = \sum_j w_{ij} M_j \mathbf{p}_i$$



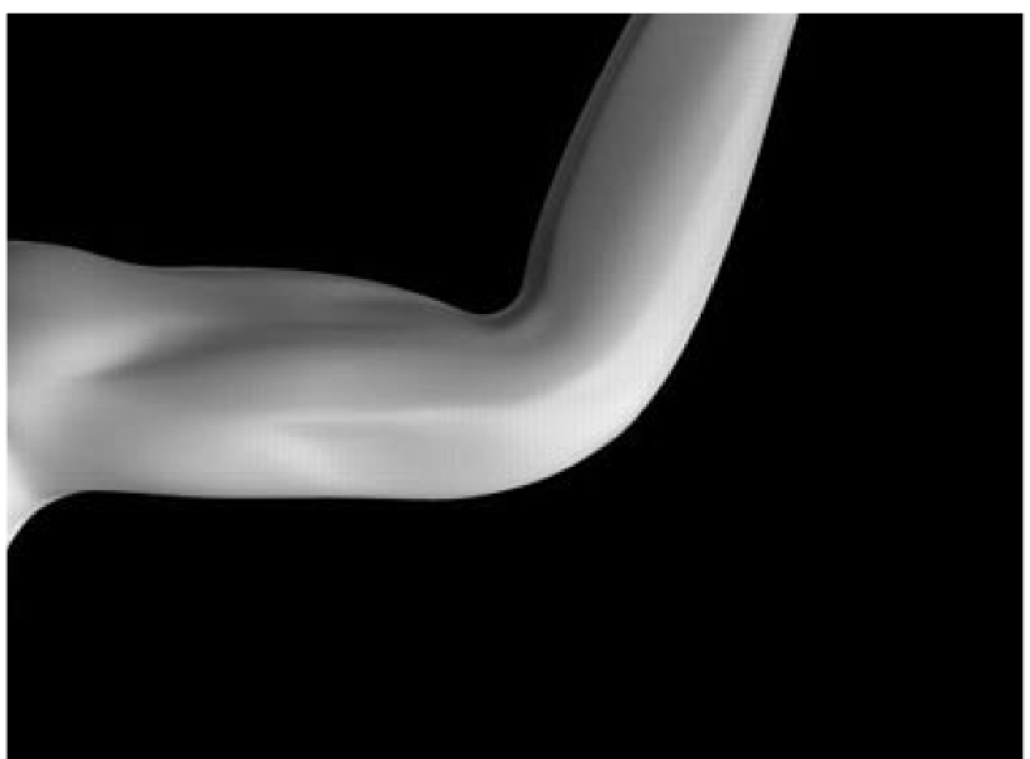
[Lewis et al. SIGGRAPH 2000]

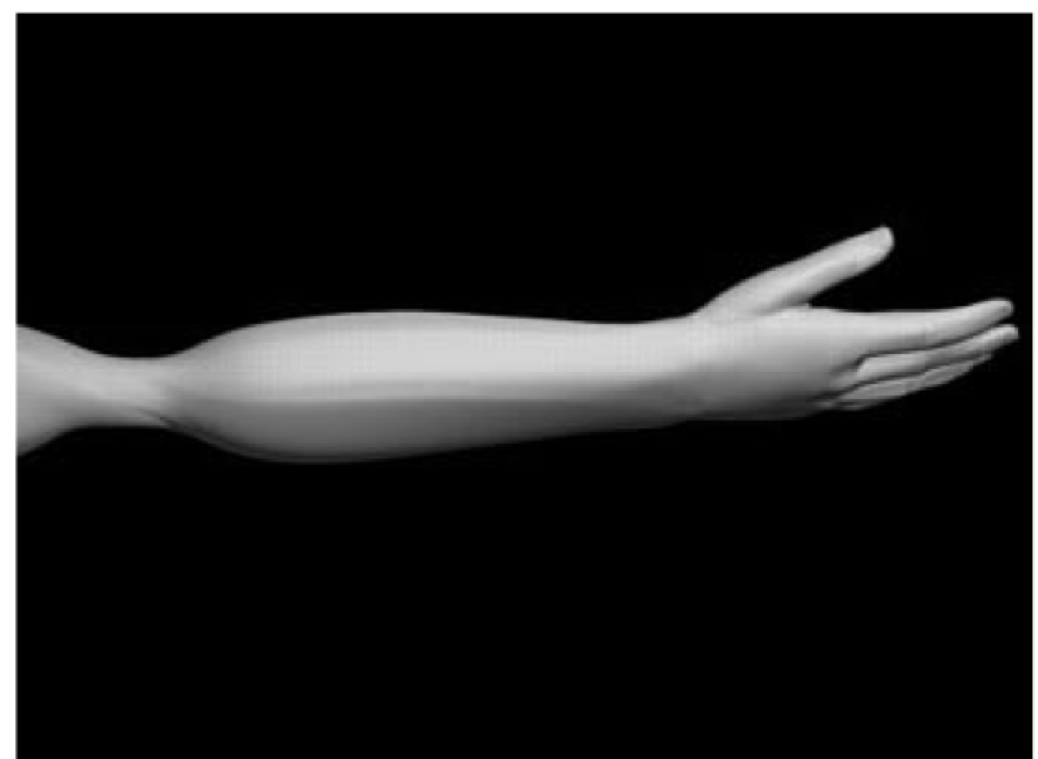
Mesh skinning

- Simple and fast to compute
  - Can even compute in the vertex stage of a graphics pipeline
- Used heavily in games
- One piece of the toolbox for offline animation
  - Many other deformers also available

Mesh skinning: classic problems

- Surface collapses on the inside of bends and in the presence of strong twists
  - Average of two rotations is not a rotation!
  - Add more bones to keep adjacent bones from being too different, or change the blending rules.

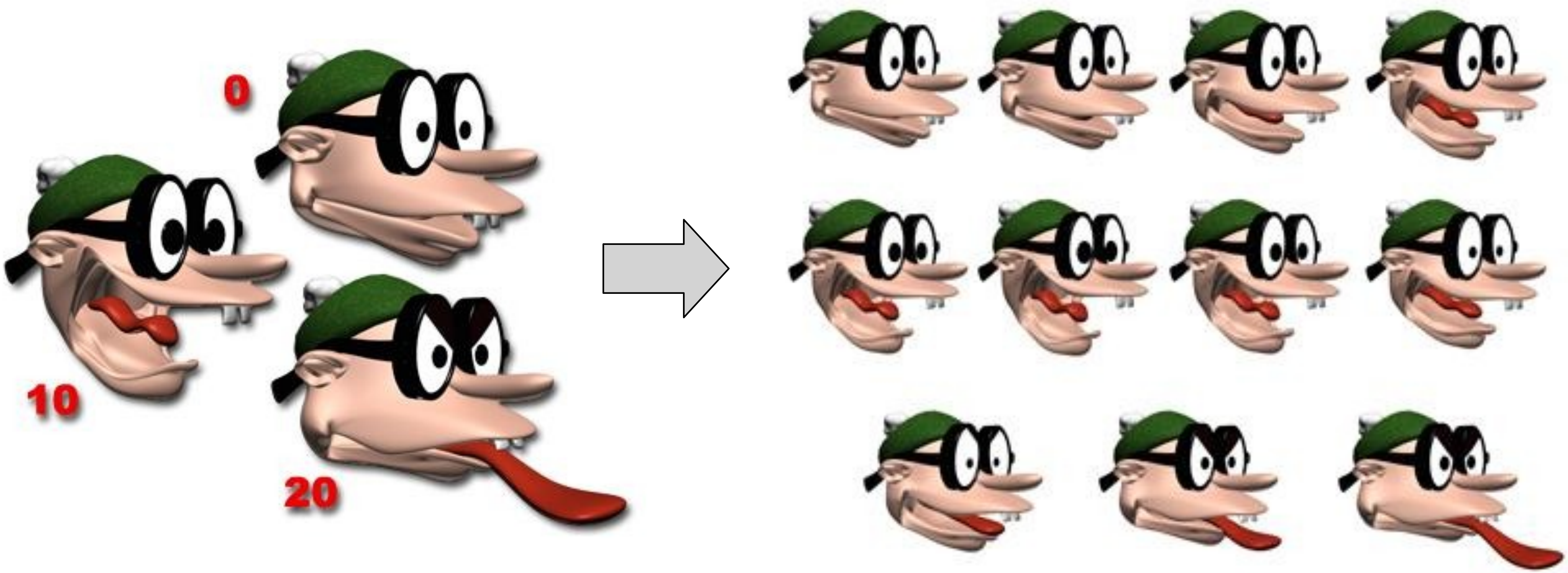




[Lewis et al. SIGGRAPH 2000]

Blend shapes

- Another very simple surface control scheme
- Based on interpolating among several key poses
  - Aka. blend shapes or morph targets



[3D Studio Max example]

Blend shapes math

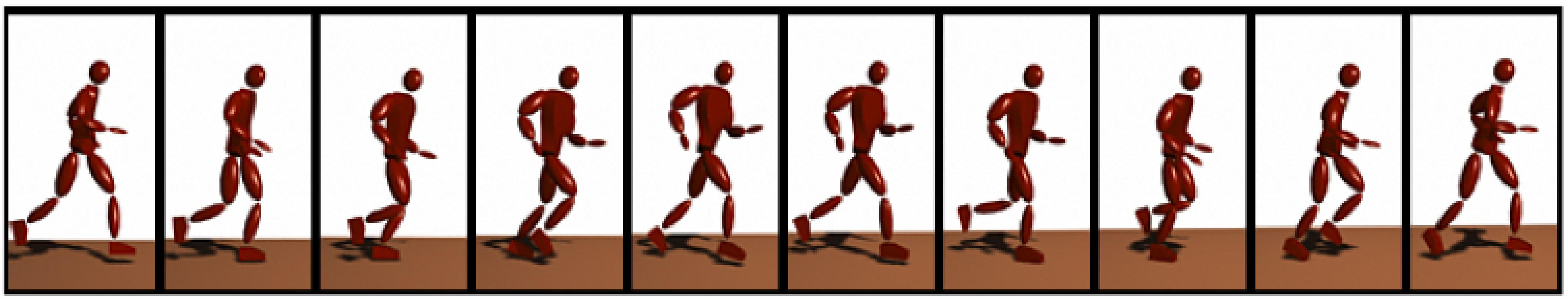
- Simple setup
  - User provides key shapes—that is, a position for every control point in every shape: $\mathbf{p}_{ij}$ for point i, shape j
  - Per frame: user provides a weight w_j for each key shape

    Must sum to 1.0
- Computation of deformed shape

$$\mathbf{p}'_i = \sum_j w_j \mathbf{p}_{ij}$$

- Works well for relatively small motions
  - Often used for for facial animation
  - Runs in real time; popular for games

Motion capture



- A method for creating complex motion quickly: measure it from the real world

[thanks to Zoran Popović for many visuals]

Motion capture in movies

[Final Fanatsy]

Motion capture in movies

[*The Two Towers* | New Line Productions]

Motion capture in games

Magnetic motion capture

- Tethered
- Nearby metal objects cause distortions
- Low freq. (60Hz)

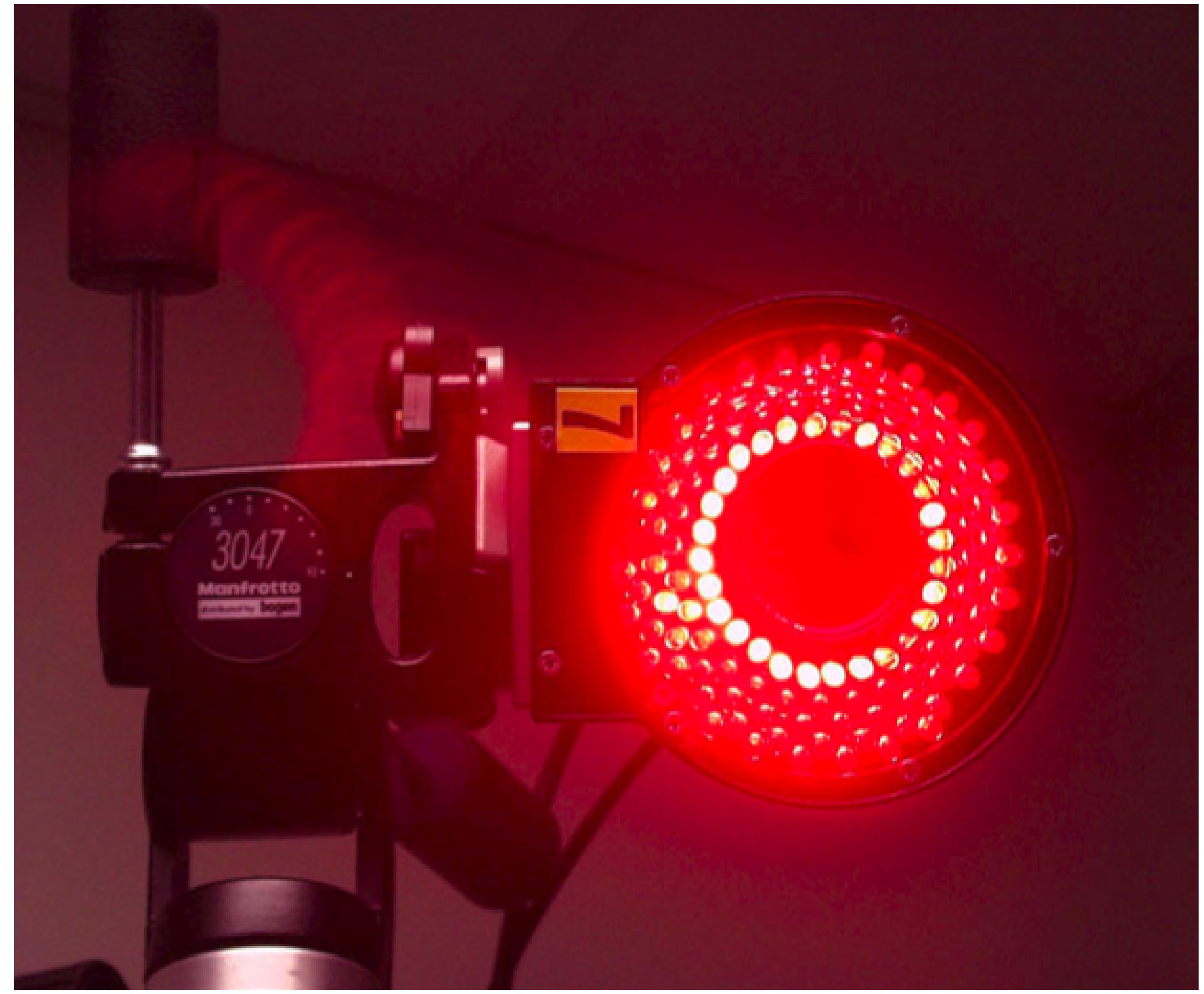
Mechanical motion capture

- Measures joint angles directly
- Works in any environment
- Restricts motion

Optical motion capture

- Passive markers on subject

Retroreflective markers



Cameras with IR illuminators

- Markers observed by cameras
  - Positions via triangulation

Optical motion capture

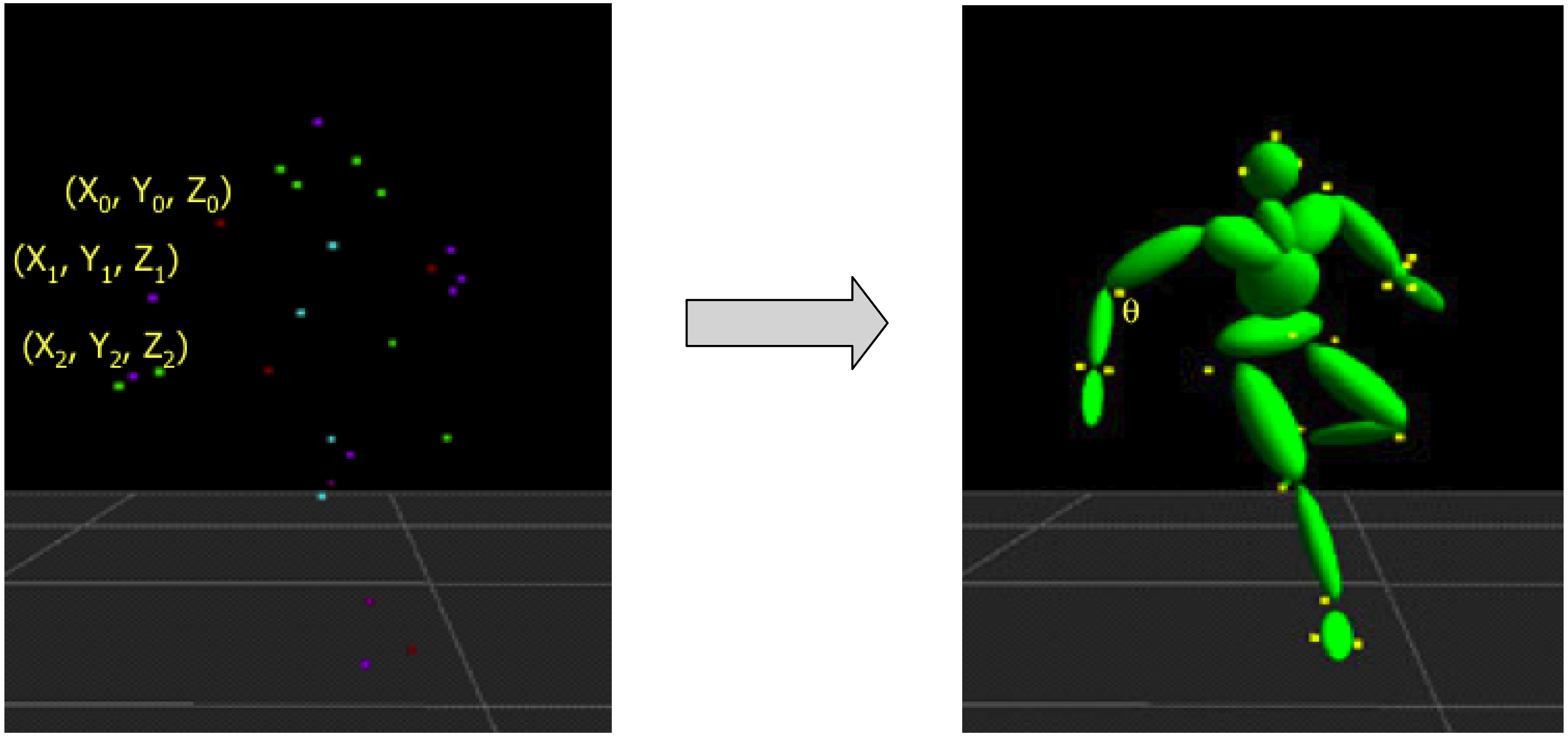
- 8 or more cameras
- Restricted volume
- High frequency (240Hz)
- Occlusions are troublesome

From marker data to usable motion

- Motion capture system gives inconvenient raw data
  - Optical is "least information" case: accurate position but:

    Which marker is which?

    Where are the markers are relative to the skeleton?



Motion capture data processing

- Marker identification: which marker is which
  - Start with standard rest pose
  - Track forward through time (but watch for markers dropping out due to occlusion!)
- Calibration: match skeleton, find offsets to markers
  - Use a short sequence that exercises all DOFs of the subject
  - A nonlinear minimization problem
- Computing joint angles: explain data using skeleton DOFs
  - A inverse kinematics problem per frame!

Motion capture in context

- Mocap data is very realistic
  - Timing matches performance exactly
  - Dimensions are exact
- But it is not enough for good character animation
  - Too few DOFs
  - Noise, errors from nonrigid marker mounting
  - Contains no exaggeration
  - Only applies to human-shaped characters
- Therefore mocap data is generally a starting point for skilled animators to create the final product