Design of Human Interface Game Software

– Character Animations

Animations

- When to use animation
  - Feedback to player about interaction with UI and in-game action
  - Communicating environmental conditions
  - Conveying emotion and expression in player characters and NPC
  - For visual appeal and dynamic interest

2D Animation

- Borrow from traditional cel animation
  - Draw layers of the scene on translucent cels and superimpose animations

Image courtesy of George T. Henion.

3D Animation

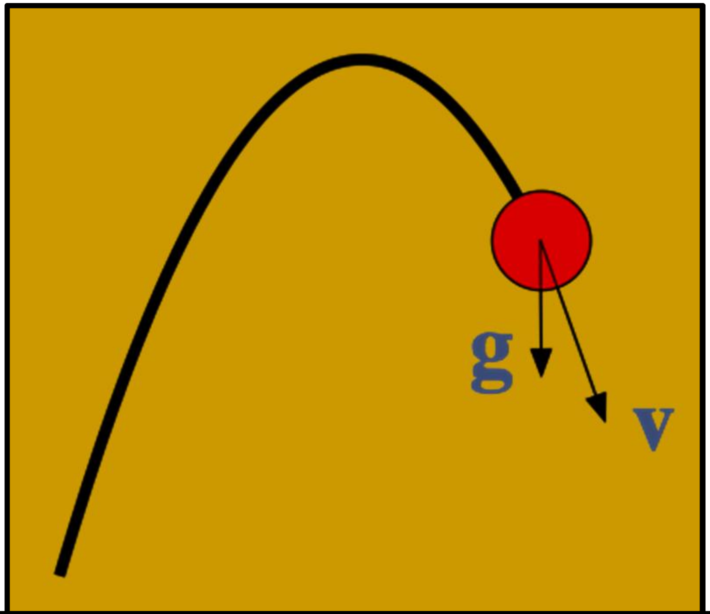
- Designing 3D motions to be viewed from more than one camera angle

Animation

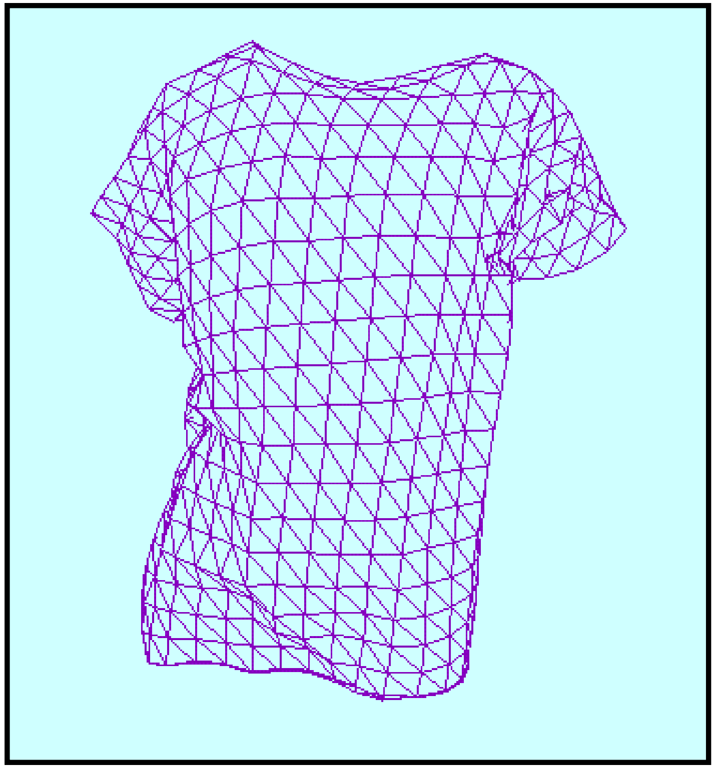
- Simulation
  - Rigid-body simulation
  - Fluid simulation
- Character animation
  - Vertex animation
  - Skeleton-based animation

Simulation

- Generate motion of objects using numerical simulation methods



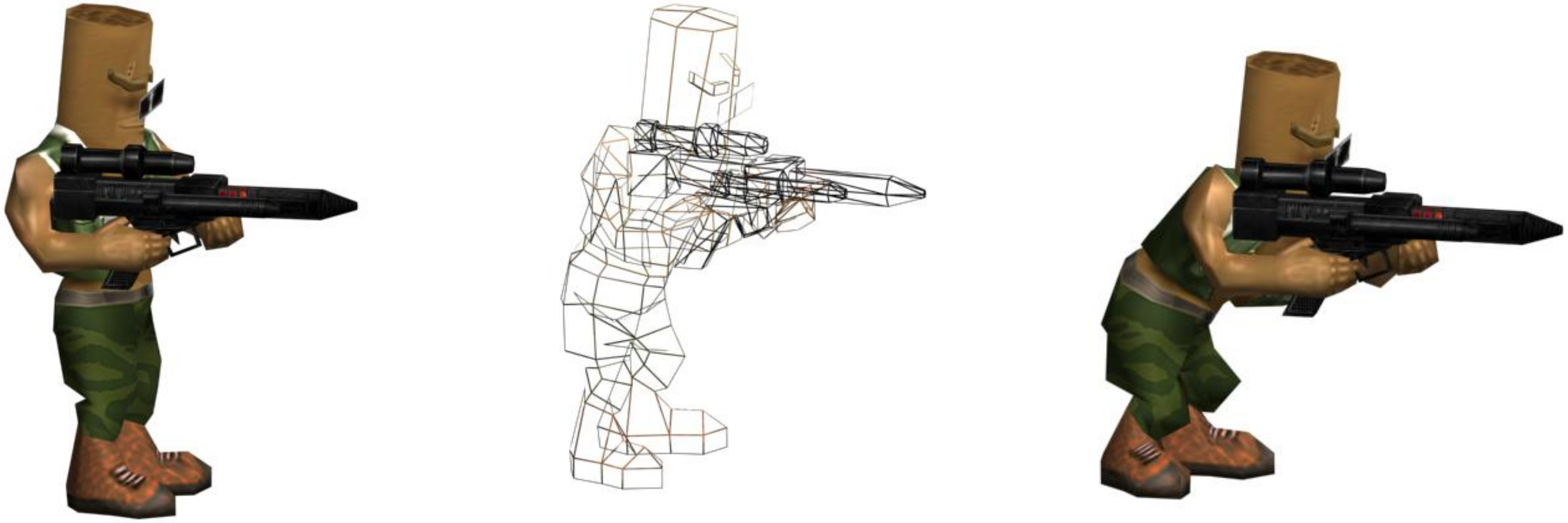
$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^{t} + \Delta t\,\mathbf{v}^{t} + \frac{1}{2}\Delta t^{2}\,\mathbf{a}^{t}$$



Character Animation: Vertex Animation

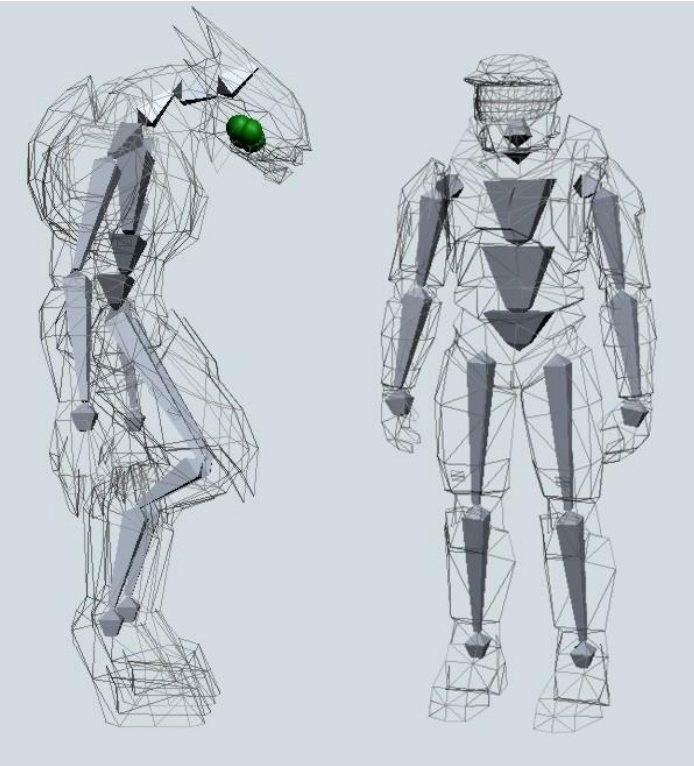
- Vertex animation (Morph animation)
  - Store a series of vertex positions in the key frames
  - Compute vertex positions in between key frames by using linear interpolation.

$$V_{world} = (1-\alpha)\times V_1 + \alpha \times V_2 \qquad 0 \le \alpha \le 1$$

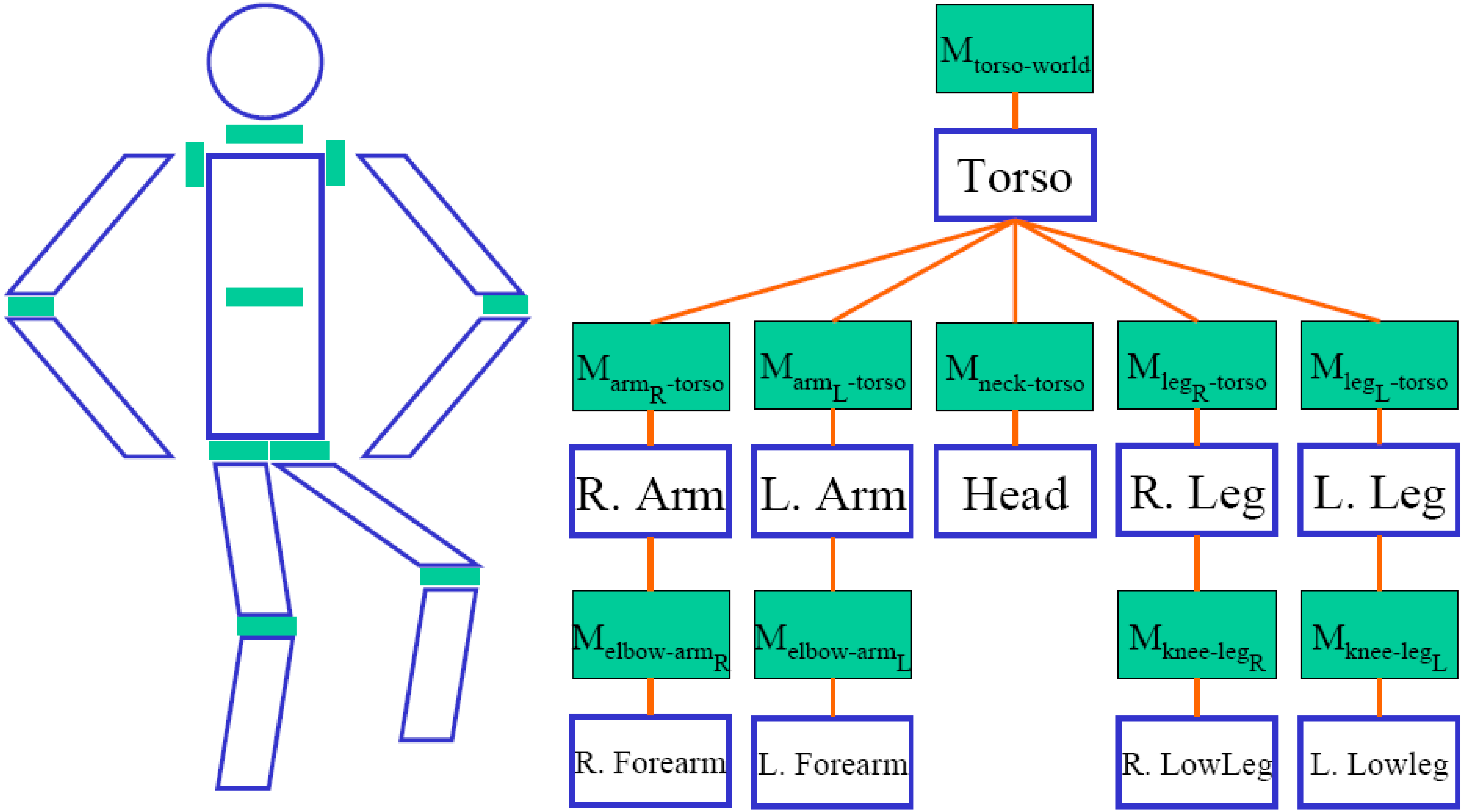


Character Animation: Skeleton-based

- Skeleton: hierarchical joints and bones
  - Joint degrees of freedom (rotation axes)
  - Limits of movement
- Skin
  - Smoothing/blending
- Binding
  - Correspondence between skeleton and skin geometries
- Motion Blending
  - Cross dissolves
  - State machines

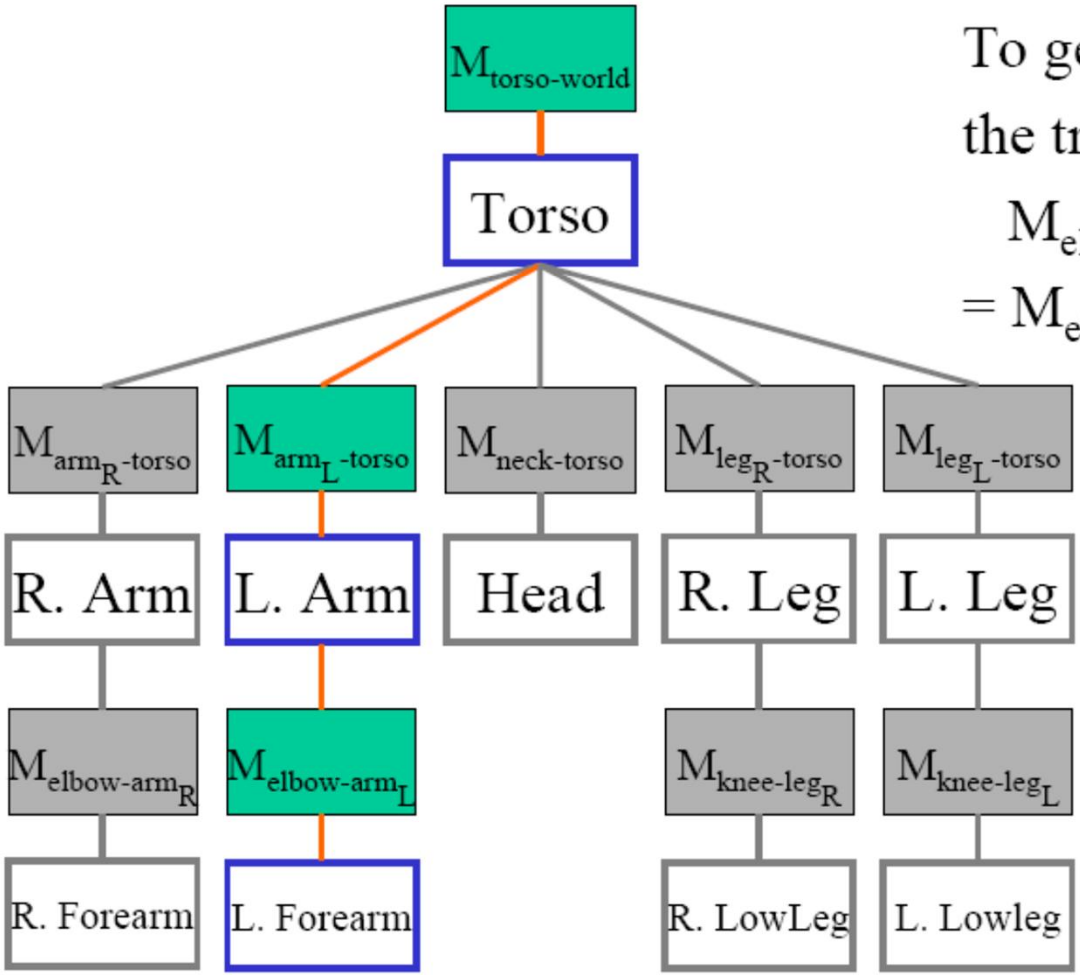


Character Skeleton



Copyright © Amitabh Varshney

Character Skeleton



Forward Kinematics:

To get the position of left arm, the transformation matrix is:

$$M_{elbow\text{-}arm_L}\ M_{arm_L\text{-}torso}\ M_{torso\text{-}world}$$
$$= M_{elbow\text{-}\cancel{arm_L}}\ M_{\cancel{arm_L}\text{-}\cancel{torso}}\ M_{\cancel{torso}\text{-}world}$$
$$= M_{elbow\text{-}world}$$

Character Skeleton

- Each matrix is parameterized by its degrees of freedom and constrained to lie within its limits:
  - For instance: $M_{\text{knee-Rleg}} = R_x(\phi)$, $-180^o \leq \phi \leq 0^o$
  - Each joint matrix also encodes a *joint offset* (length of the bone), thus in fact, $M_{\text{knee-Rleg}} = T(len)\, R_x(\phi)$
- The joint's motion parameter (say $\phi(t)$) is changed from frame to frame to generate the animation
- The parameter changes (velocities, accelerations, etc) are specified by a higher-level animation system – keyframe, mocap, or procedural

Character Skeleton

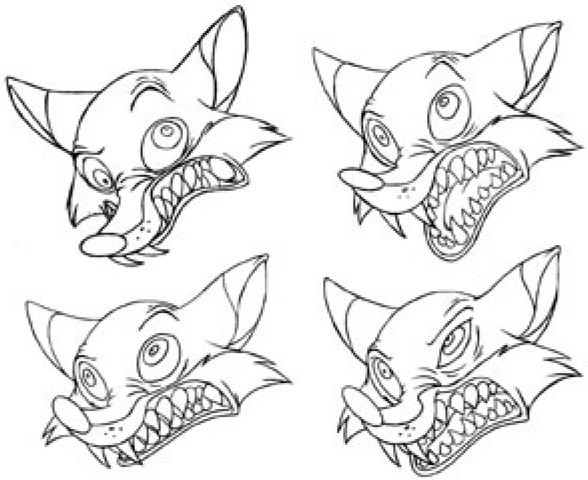
- *Pose:* a list of parameters $\phi = (\phi_1, \phi_2, \ldots, \phi_n)$ defining the various joint angles of the skeleton
- *Channel:* A sequence of parameters for a single joint angle $\phi_i(t)$
  - Often use parametric curves (Bezier, B-spline, Hermite, …) to edit, interpolate, approximate, or compress
- *Animation:* An array of poses $\phi(t)$ or an array of channels $(\phi_1(t), \phi_2(t), \ldots, \phi_n(t))$
  - Tradeoff memory access coherence vs CPU computation

Character Skeleton – Animation

- Key-Frame Animation
- Motion Capture

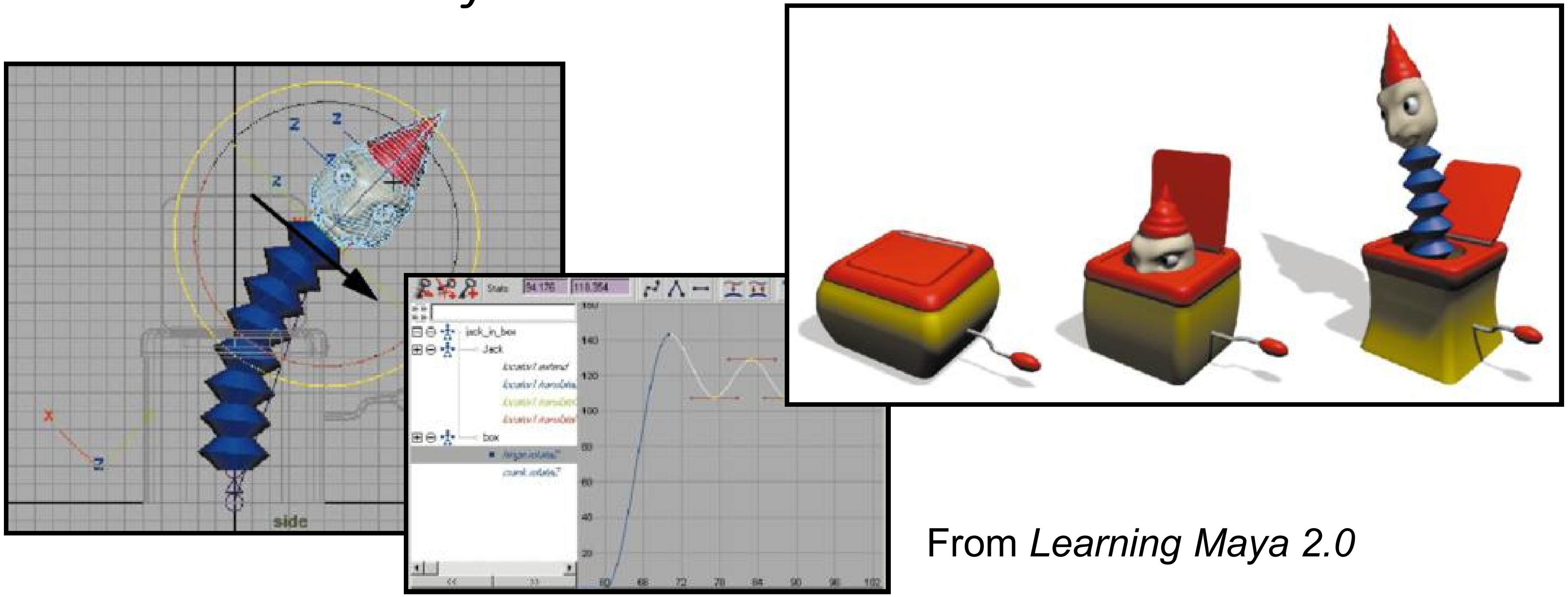
Traditional Keyframe Animation

- Traditional approach to animation in movies
  - Main animator draws a few key frames
  - Assistant animators draw interpolating frames



Key-Frame Animation

- Requires a highly skilled user
- Computers interpolate vertex coordinates between key frames



From *Learning Maya 2.0*

Lerp and Slerp

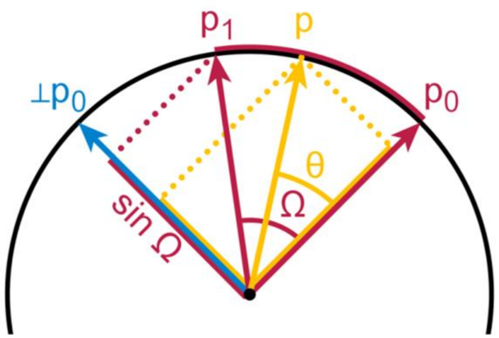
- Linear Interpolation (in Cartesian coordinates): Lerp

$$p(\alpha) = (1-\alpha)p_0 + \alpha p_1 \qquad 0 \le \alpha \le 1$$

- Spherical Linear Interpolation (on surface of a sphere): Slerp

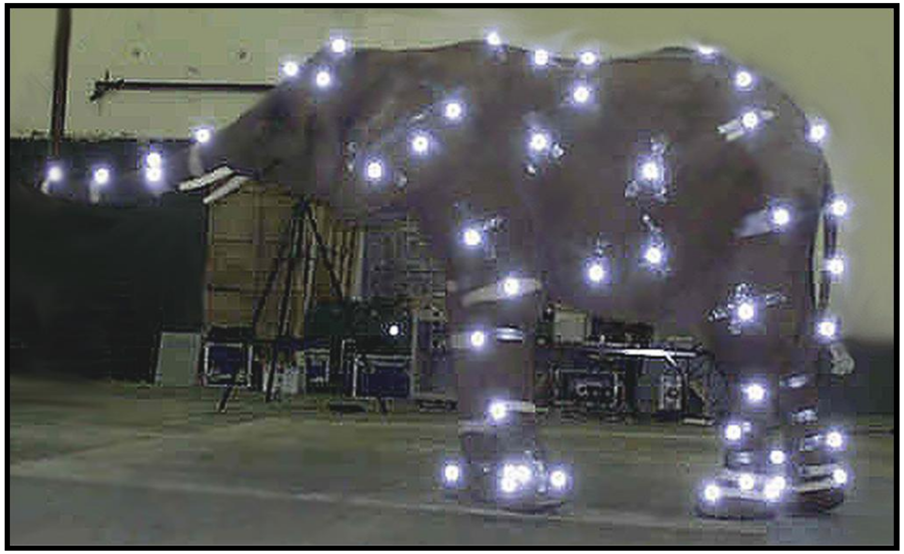
$$p(\alpha) = \frac{\sin((1-\alpha)\Omega)}{\sin\Omega} p_0 + \frac{\sin(\alpha\Omega)}{\sin\Omega} p_1$$

$$\Omega = \cos^{-1}(p_0 \cdot p_1)$$



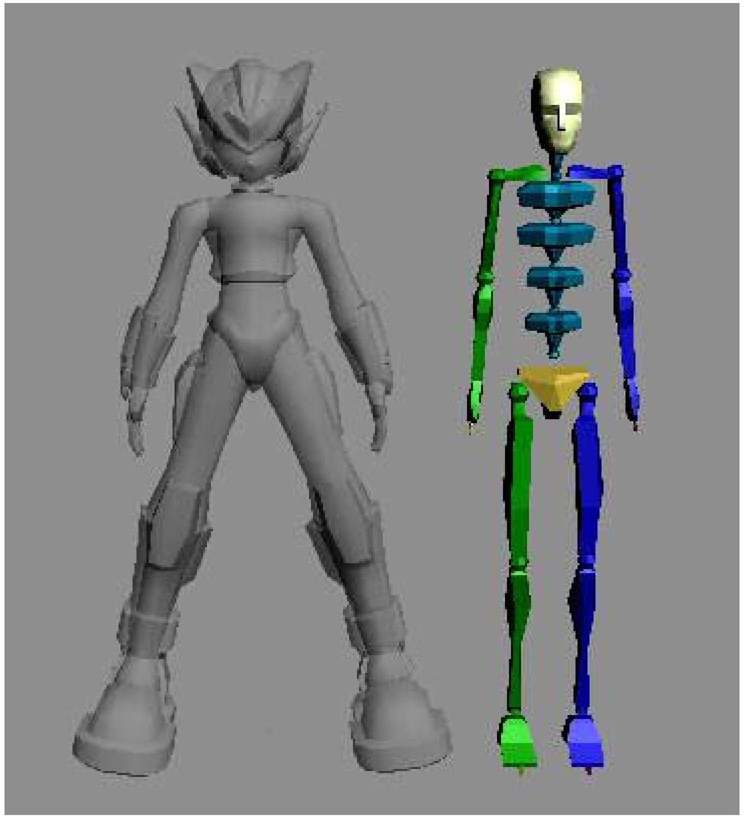
Motion Capture

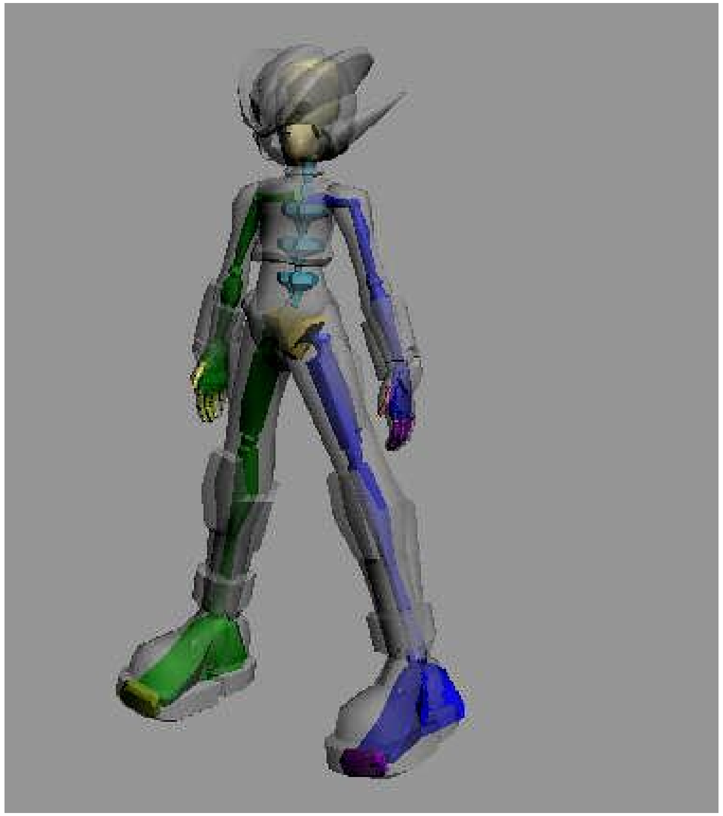
- Markers/sensors placed on subject
- Record and playback the motion
- Time-consuming clean-up
- Real-time, extremely flexible, easy to set-up



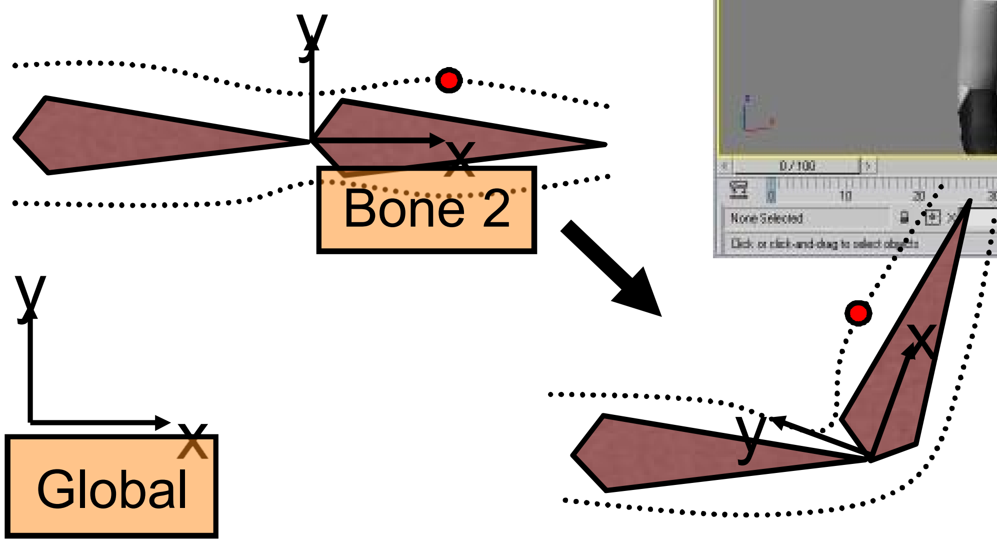
Motion Capture

Skin and Bones

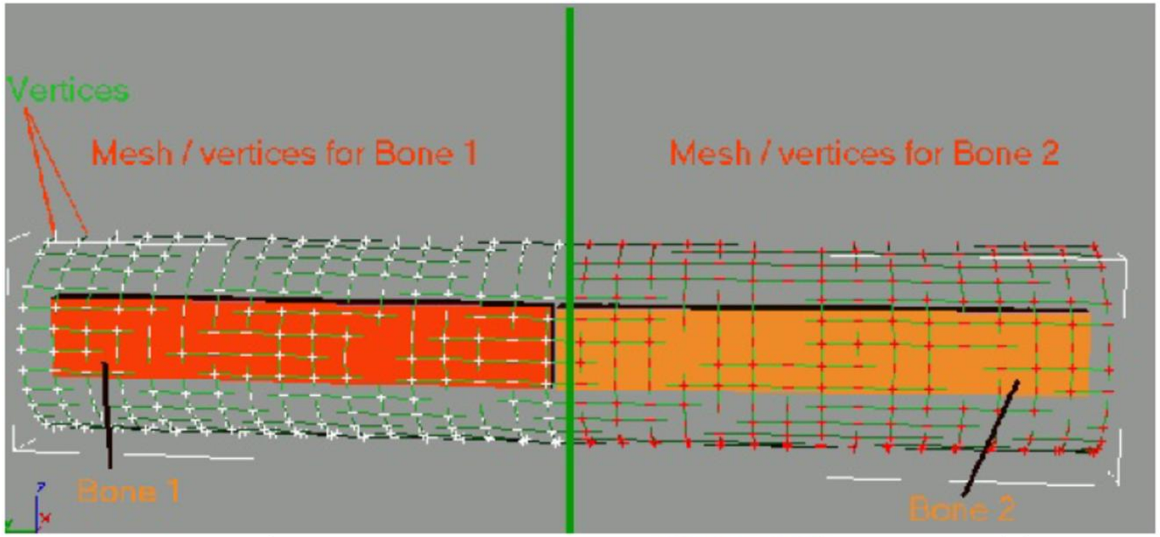


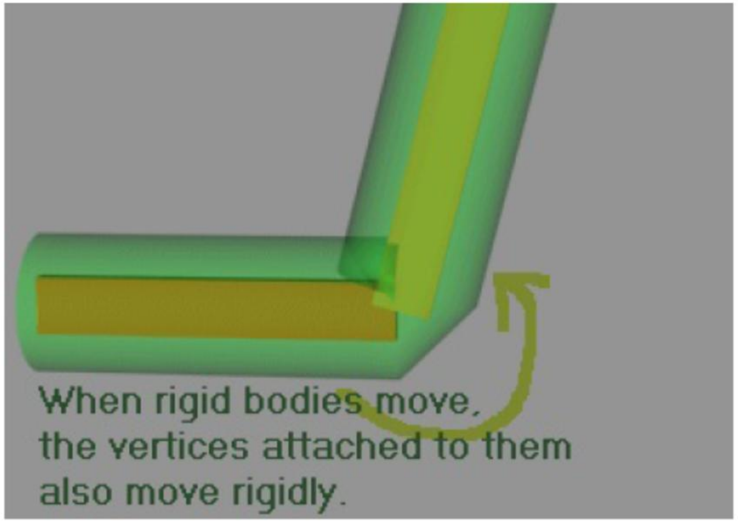


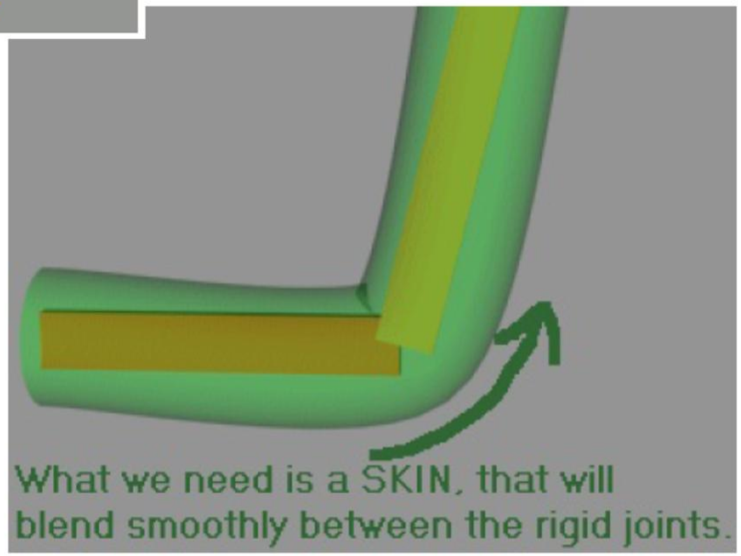
Skin and Bones



Skin and Bones







Skinning

- *Rigid Skin:* Every vertex is associated with exactly one joint:

  $\mathbf{v}'(t) = \mathbf{M}_{\mathbf{joint}}(t) \cdot \mathbf{v}$

- *Smooth Skin:* Each vertex is associated with multiple (usually two) joints:

  $\mathbf{v}'(t) = w_1\, \mathbf{M}_{\mathbf{joint1}}(t) \cdot \mathbf{v} + w_2\, \mathbf{M}_{\mathbf{joint2}}(t) \cdot \mathbf{v} + \ldots + w_n\, \mathbf{M}_{\mathbf{joint}n}(t) \cdot \mathbf{v}$

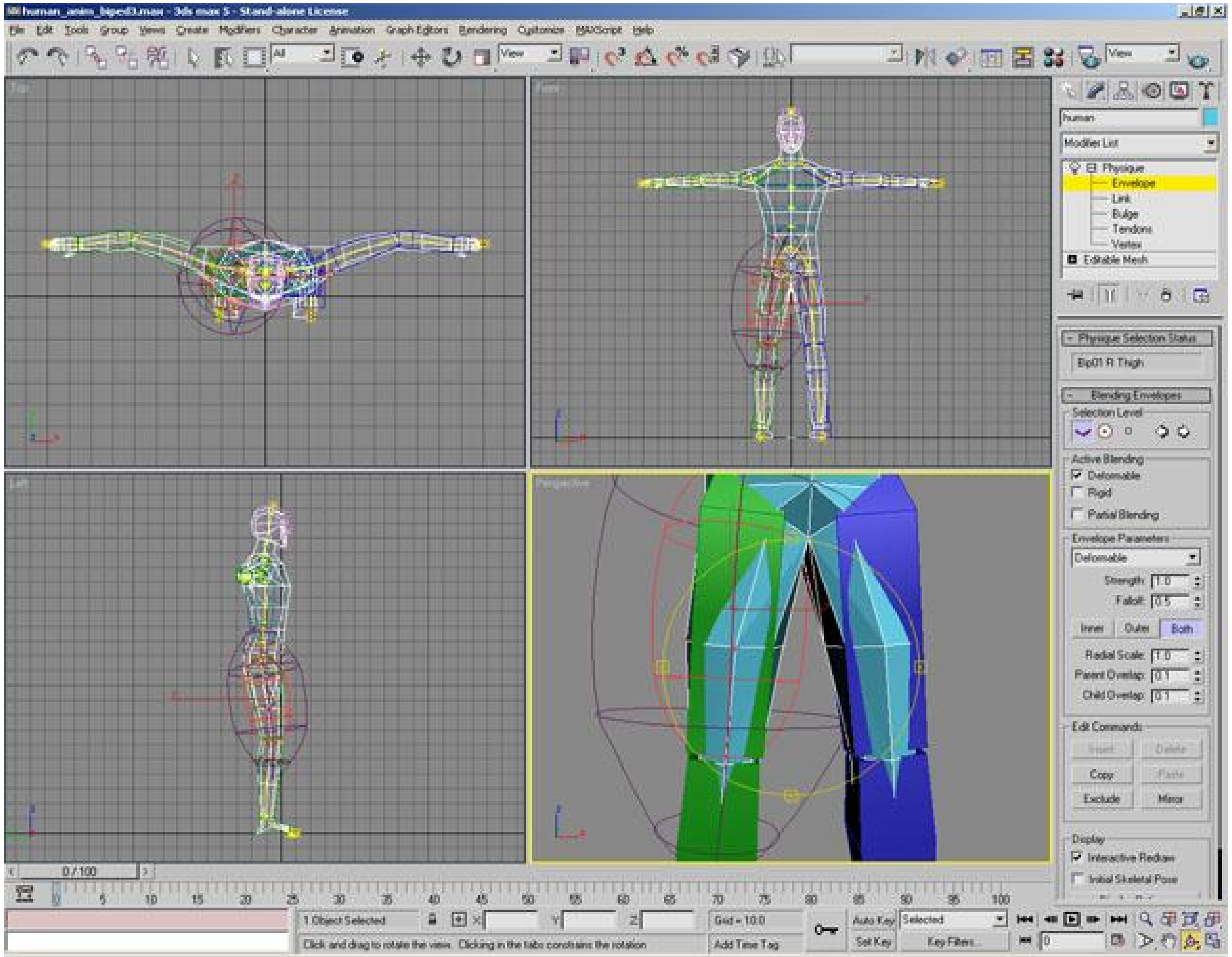
  where $w_1 + w_2 + \ldots + w_n = 1$

- GPU support (blending matrices) for smooth skin

Binding Skin with Bones

- Set correspondence
  - Proximity: assign each skin vertex to the closest bone(s)
  - Manual: create bounding volumes (spheres, ellipsoids, cubes) for each bone to enclose skin vertices that belong to it (3DS Max Envelope)
  - Automatic/manual weights

Envelope in 3DS Max



Motion Blending

- Characters in games have a library of actions that need to be sequenced (generally seamlessly) on demand and at interactive rates
  - Sitting, walking, running
  - Sword fighting
  - Passing, kicking
- *Motion blending:*
  - Interpolate between ending pose of one action and starting pose of another action

Motion Blending

- Simplest blend: Cross Dissolve
  - *Lerp* or *Slerp* the two poses

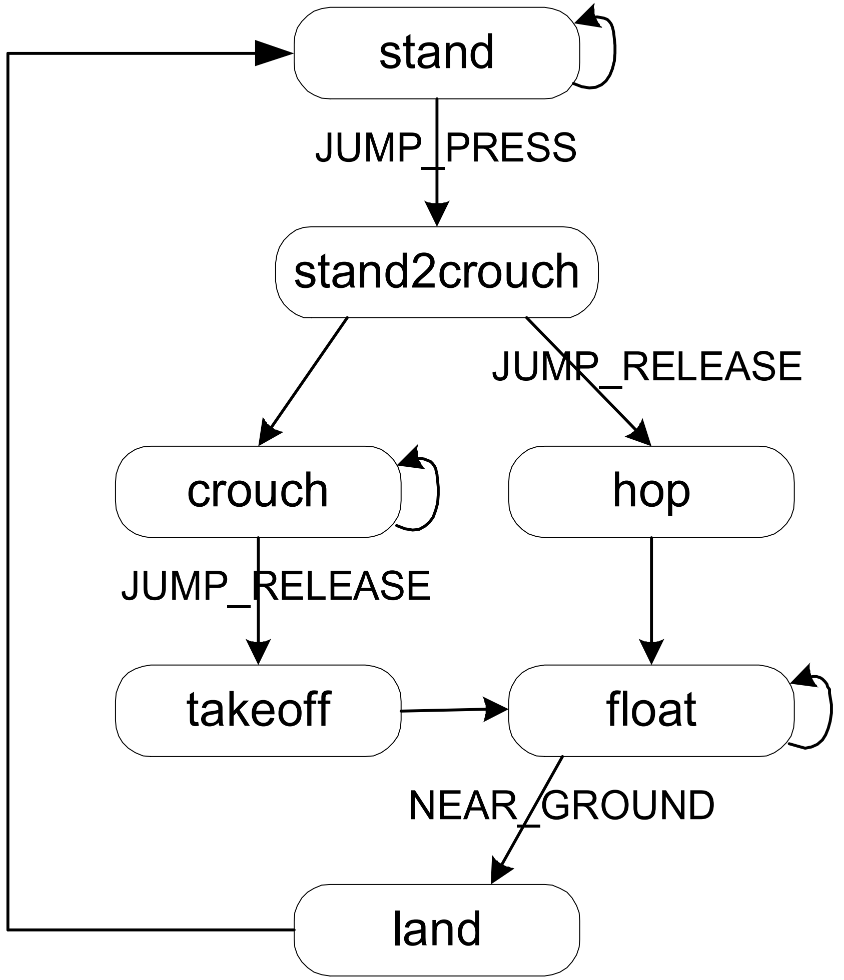
- *Challenges*
  - Ensuring phase synchronization
    - Running to Kicking
  - Adapting to changes in velocity
    - Walking to Running
  - Mixing rotations and translations
    - Sitting to Walking

Animation State Machine

- Consider a finite state machine of states representing animation clips and transitions representing motion blends

- Enables complex motion sequences
  - *Sitting* to *Walking* should have an intermediate stage of *Standing*

Animation State Machine

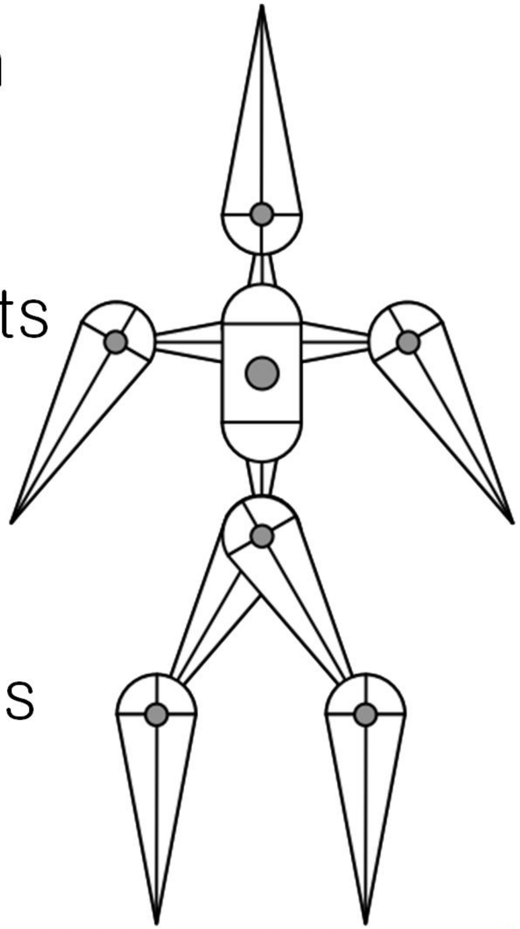


Animation State Machine

- Provides an object-oriented way to build complex motions
  - Encapsulate each simpler move (state machine) as a state of the more complex move/machine
- Allows manual fine tuning of motion transitions between selected states
- Simplifies design of animated games
  - Transitions triggered by user events, game AI, randomness, …

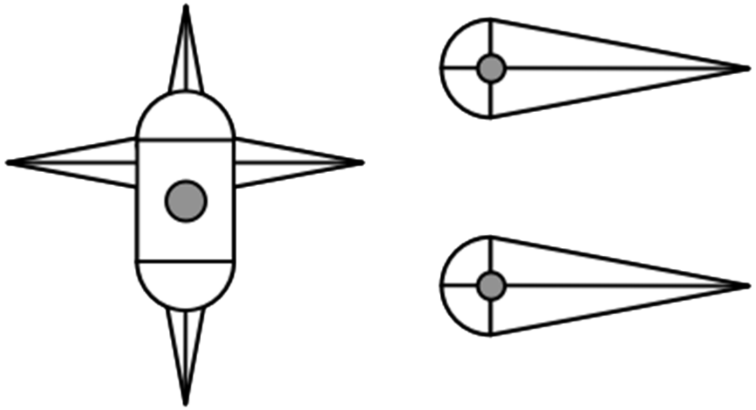
Kinematics

- Forward Kinematics
  - Given a hierarchical scene graph for an articulated structure (root location, lengths, joint angles), find the locations of all end points
- Inverse Kinematics
  - Given the positions of the root, end points, and lengths, find a self-consistent set of joint angles



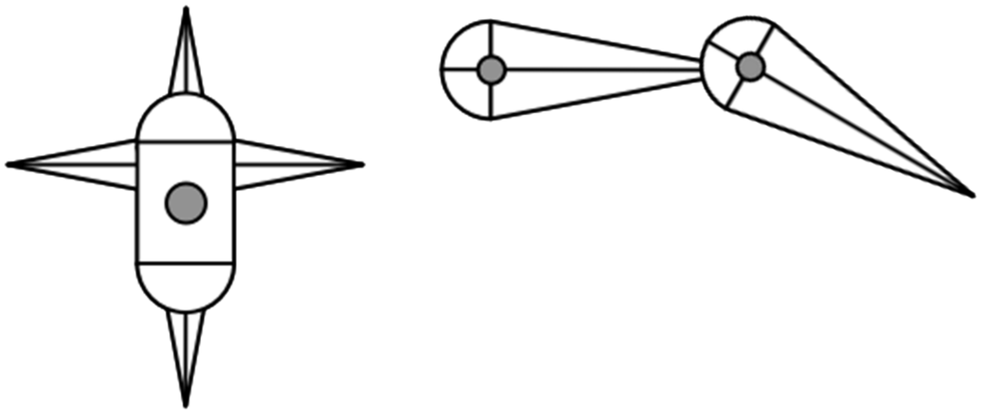
Forward Kinematics

- Composite transformations up the hierarchy



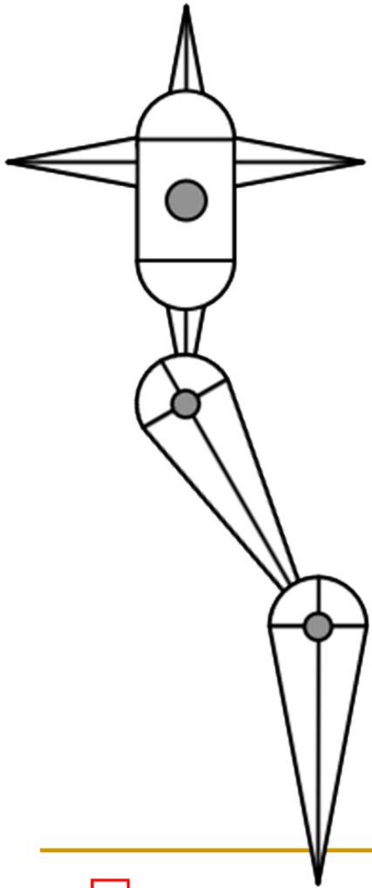
Forward Kinematics

- Composite transformations up the hierarchy



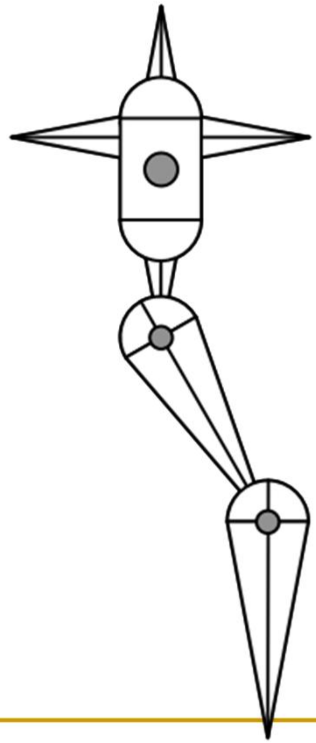
Forward Kinematics

- Composite transformations up the hierarchy



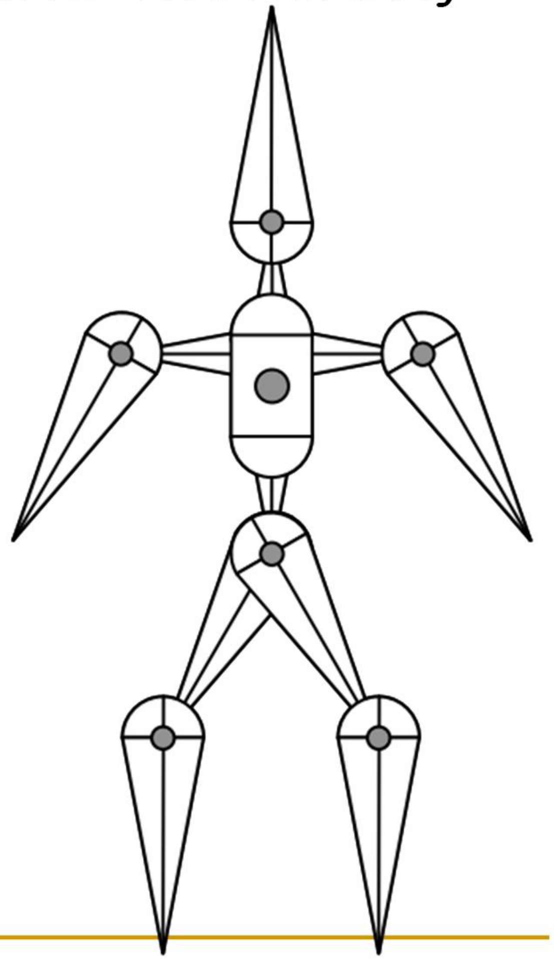
Forward Kinematics

- Composite transformations up the hierarchy



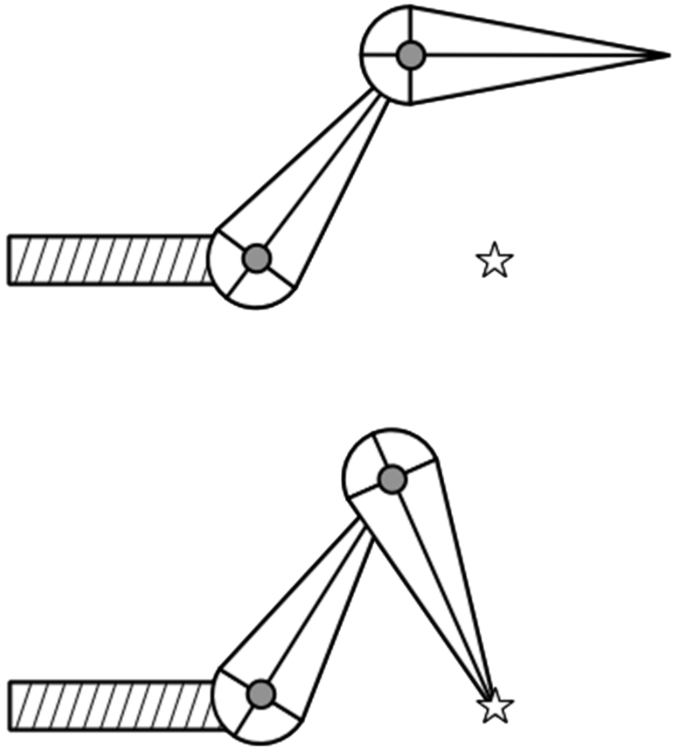
Forward Kinematics

- Composite transformations up the hierarchy



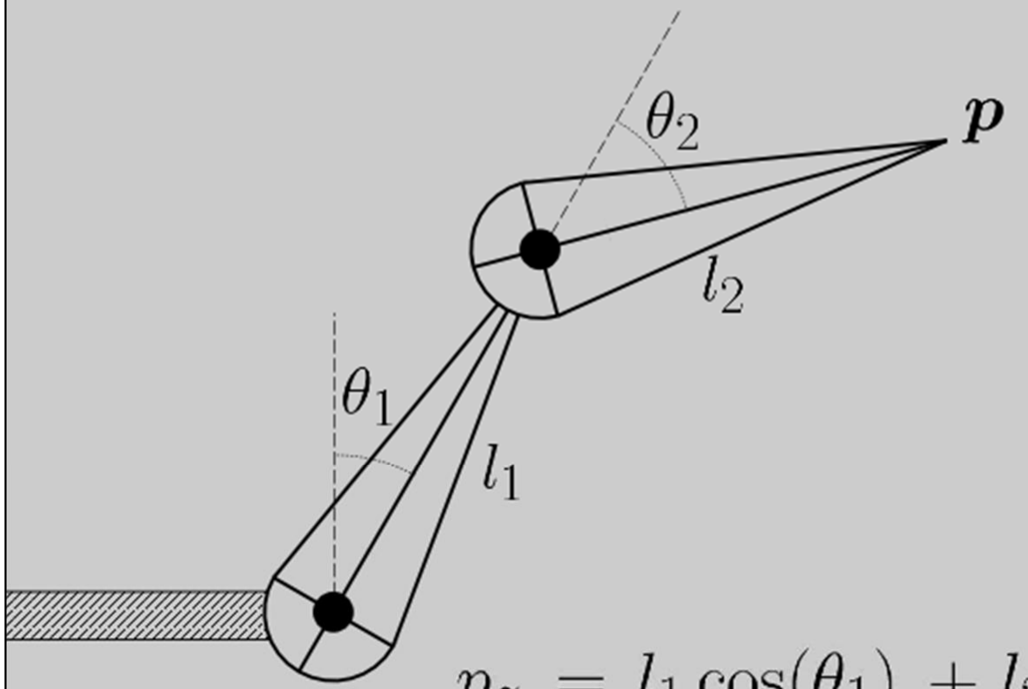
Inverse Kinematics

- Given
  - Root transformation
  - Initial configuration
  - Desired end point location
- Find
  - Interior parameter settings



Inverse Kinematics

- A simple two segment arm in 2D



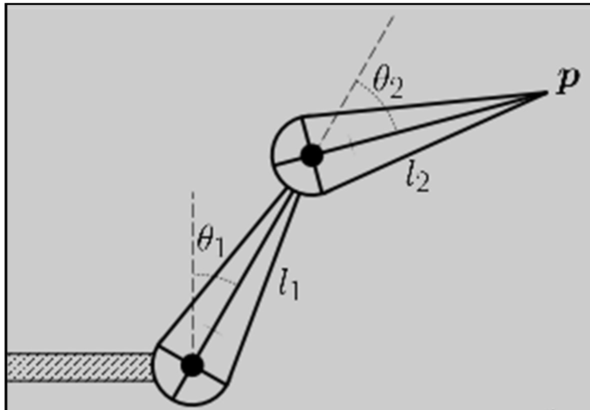
$$p_z = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

$$p_x = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

Warning: Z-up Coordinate System

Inverse Kinematics

- Direct IK: solve for the parameters

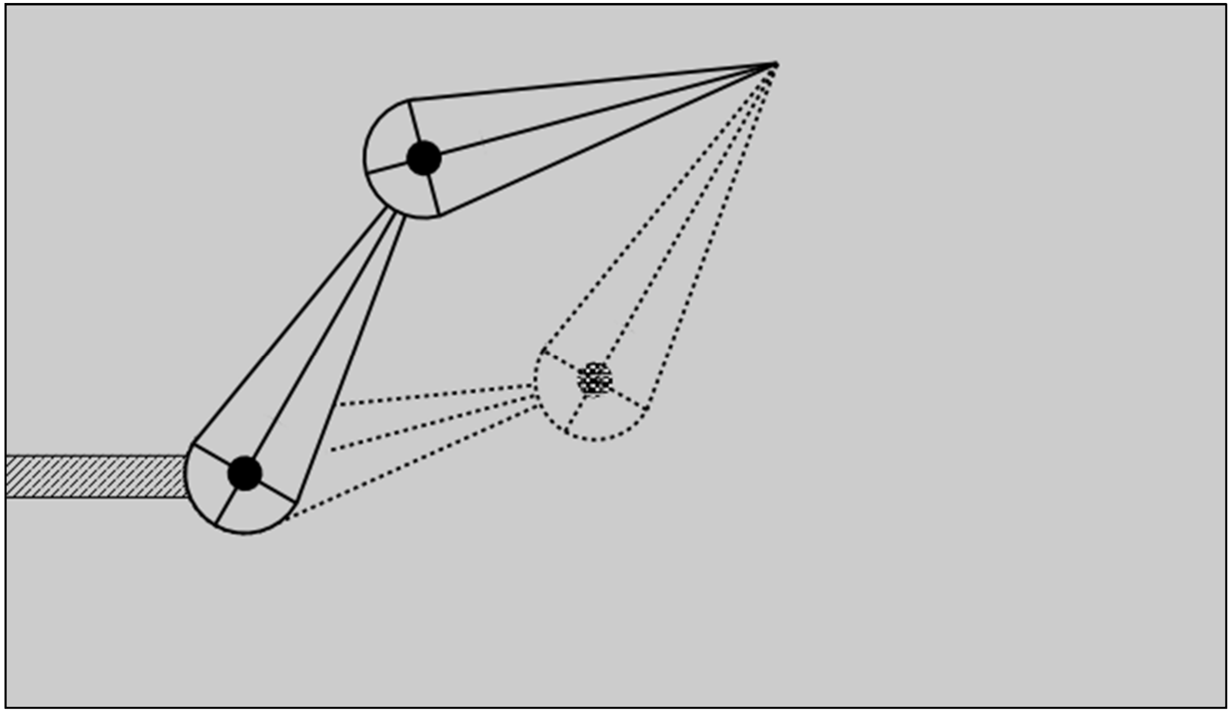


$$\theta_2 = \cos^{-1}\left(\frac{p_z^2 + p_x^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)$$

$$\theta_1 = \frac{-p_z l_2 \sin(\theta_2) + p_x(l_1 + l_2\cos(\theta_2))}{p_x l_2 \sin(\theta_2) + p_z(l_1 + l_2 \cos(\theta_2))}$$

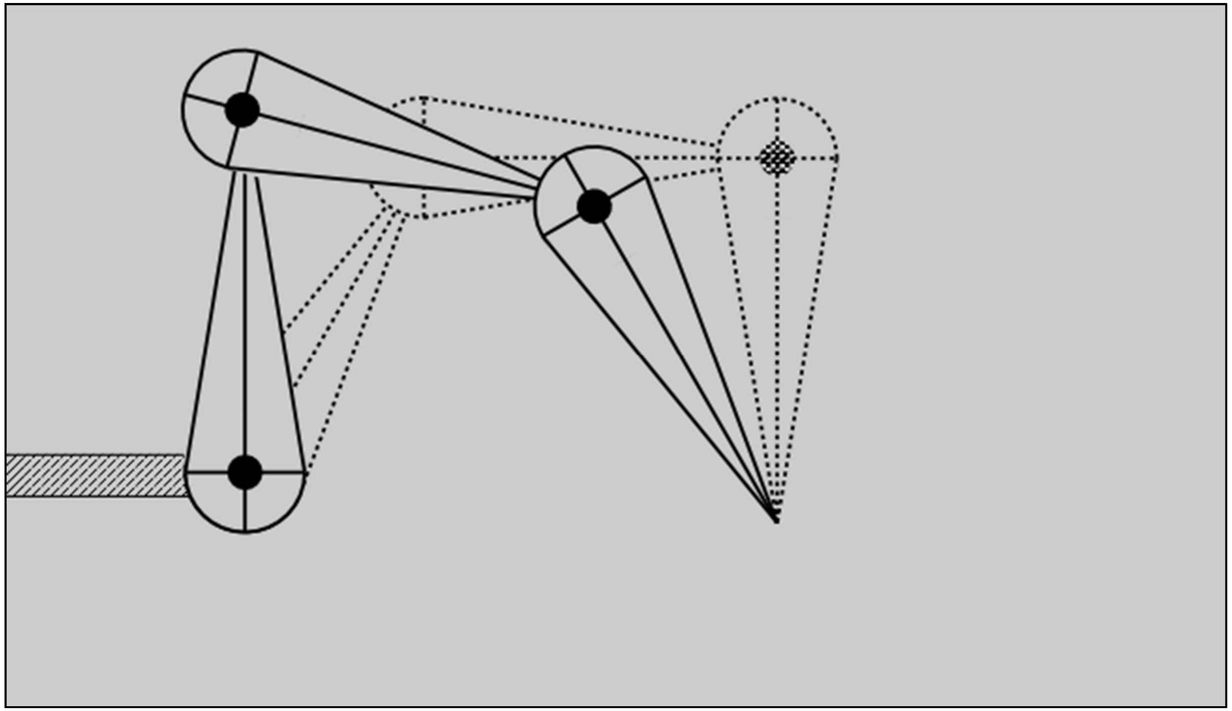
Inverse Kinematics

- Why is the problem hard?
  - Multiple solutions separated in configuration space



Inverse Kinematics

- Why is the problem hard?
  - Multiple solutions connected in configuration space



Inverse Kinematics

- Why is the problem hard?
  - Solutions may not always exist