# Character Animation

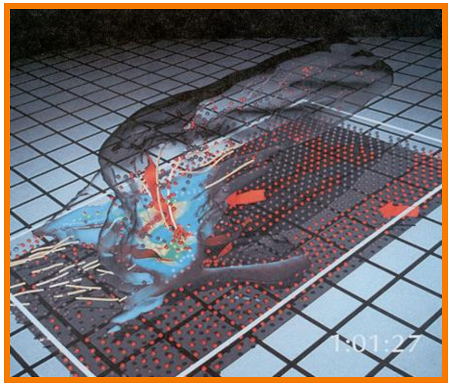
COS 426, Spring 2014
Princeton University

# Computer Animation

- Animation
  - Make objects change over time according to scripted actions

Pixar

- Simulation / dynamics
  - Predict how objects change over time according to physical laws



University of Illinois

# Computer Animation

- Describing how 3D objects (& cameras) move over time

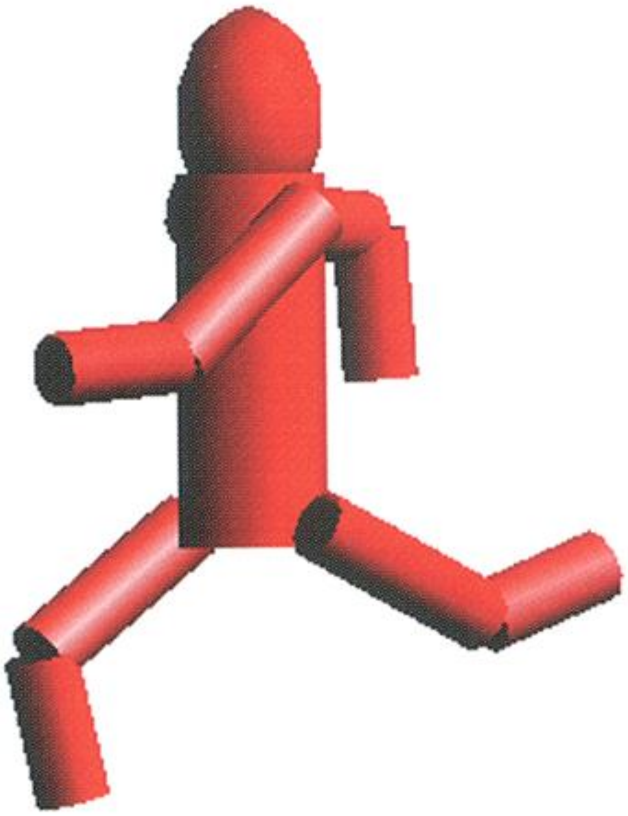
Pixar

# Computer Animation

- Challenge is balancing between …
  - Animator control
  - Physical realism

# Character Animation Methods

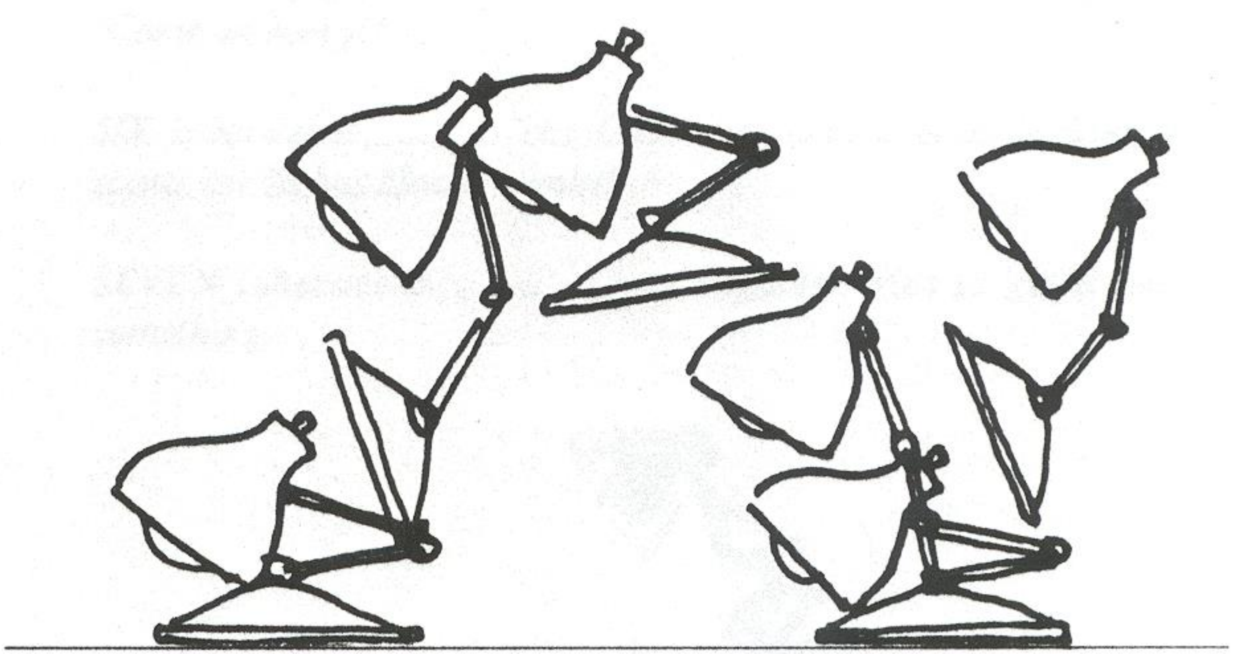
- Keyframing / Forward Kinematics
- Inverse Kinematics
- Dynamics
- Motion capture



Angel Plate 1

# Keyframe Animation

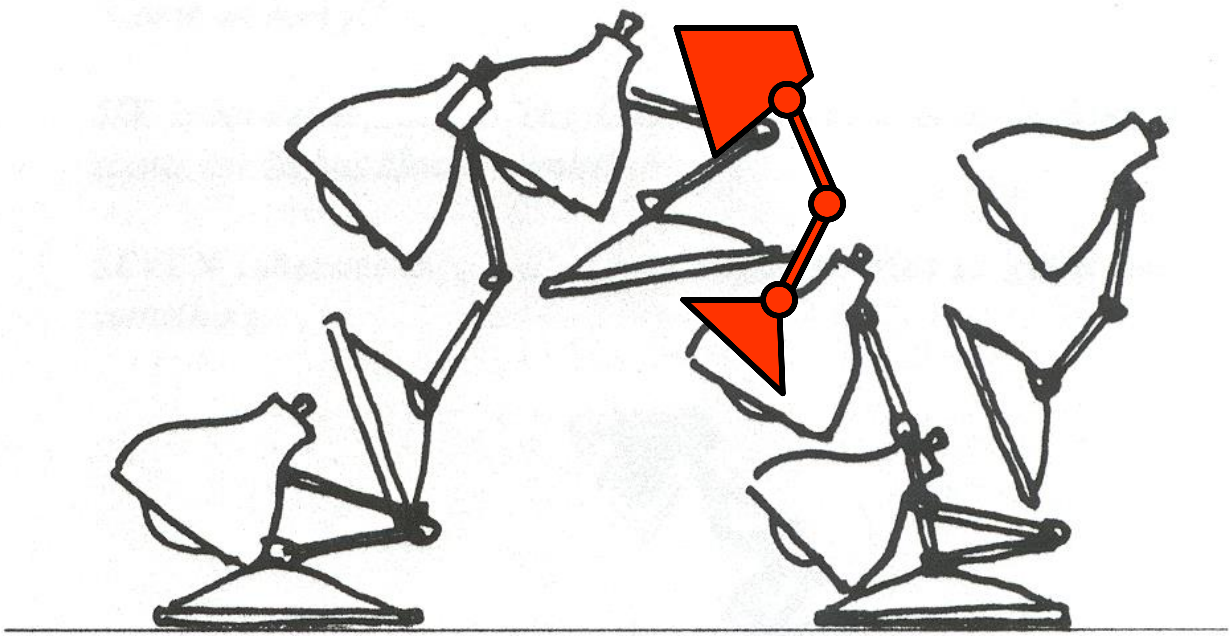
- Define character poses at specific time steps called “keyframes”



Lasseter `87

# Keyframe Animation

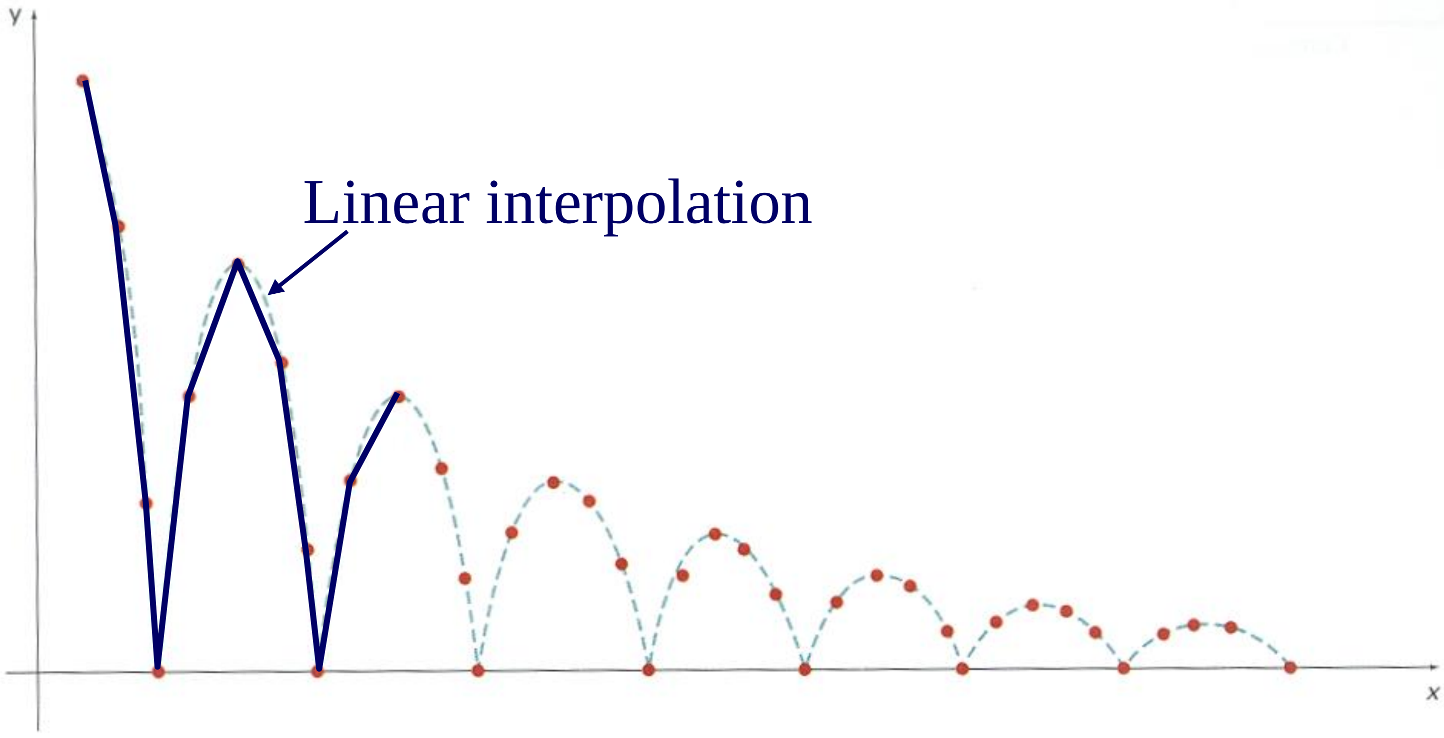
- Interpolate variables describing keyframes to determine poses for character in between



Lasseter `87

# Keyframe Animation

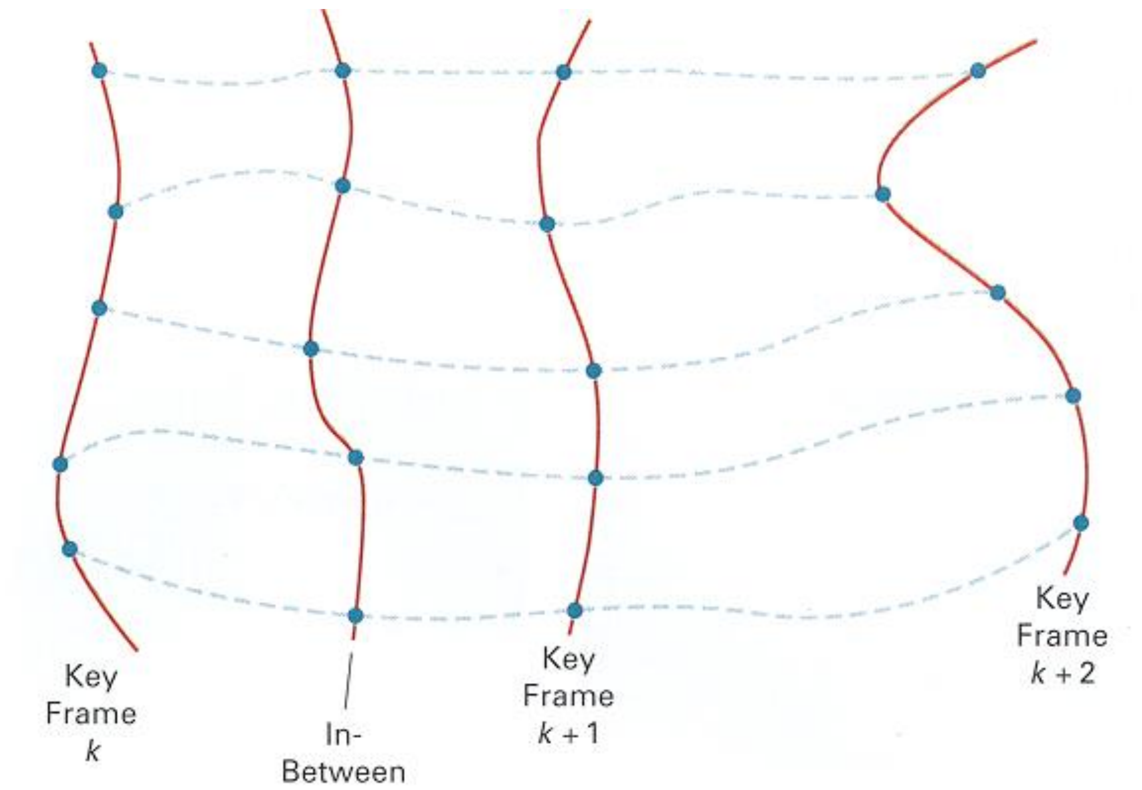
- Inbetweening:
  - Linear interpolation - usually not enough continuity




H&B Figure 16.16

# Keyframe Animation

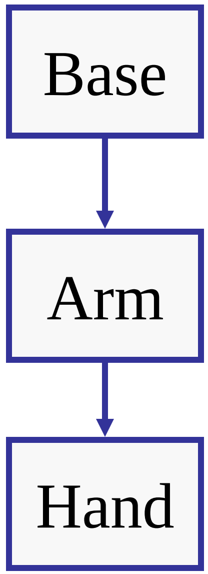
- Inbetweening:
  - Spline interpolation - maybe good enough



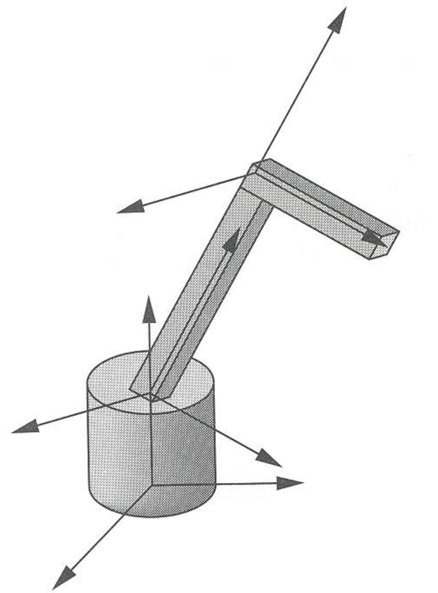

H&B Figure 16.11

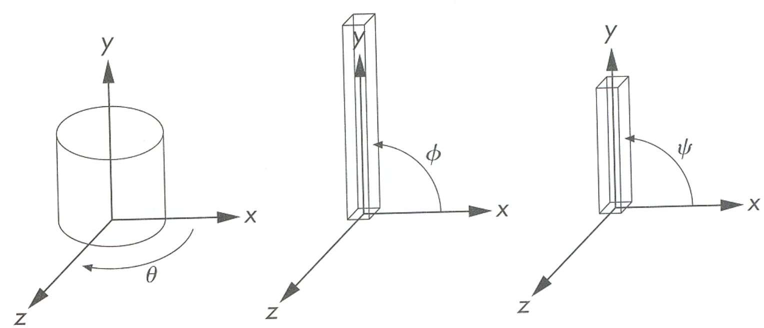
# Articulated Figures

- Character poses described by set of rigid bodies connected by “joints”




Scene Graph

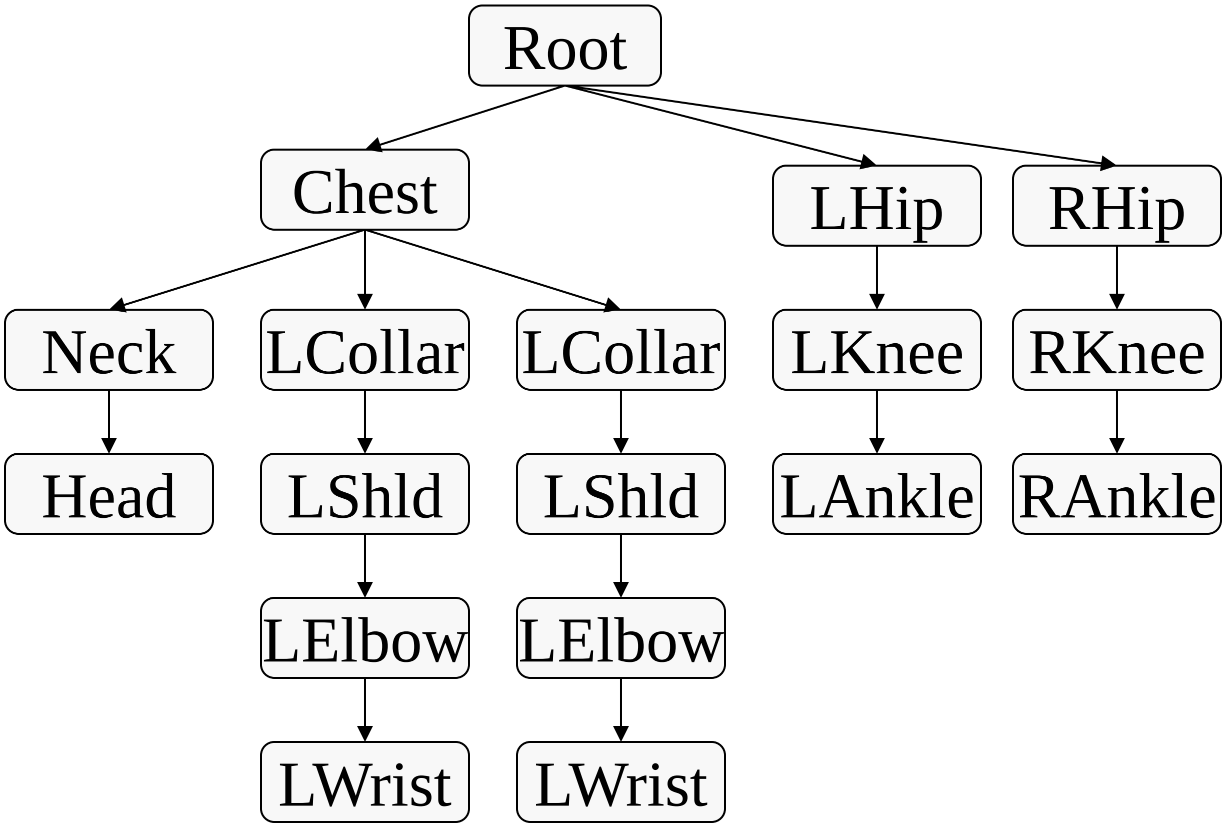


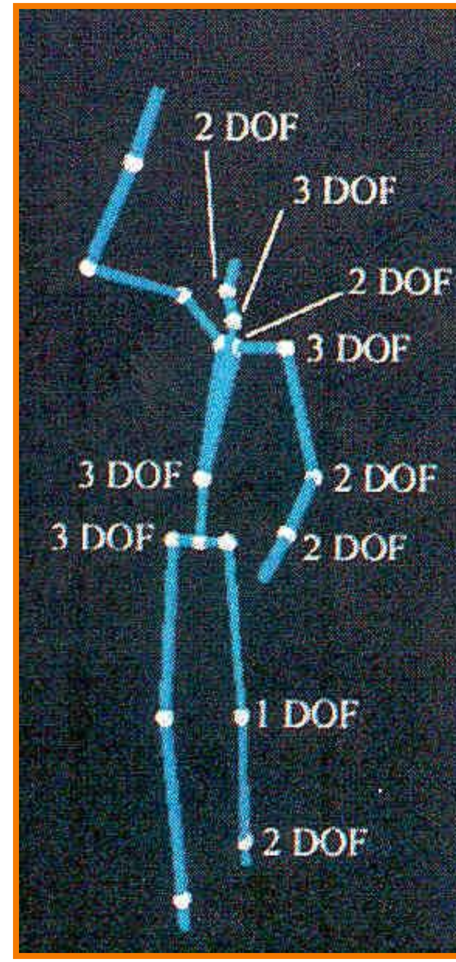



Angel Figures 8.8 & 8.9

# Articulated Figures

- Well-suited for humanoid characters

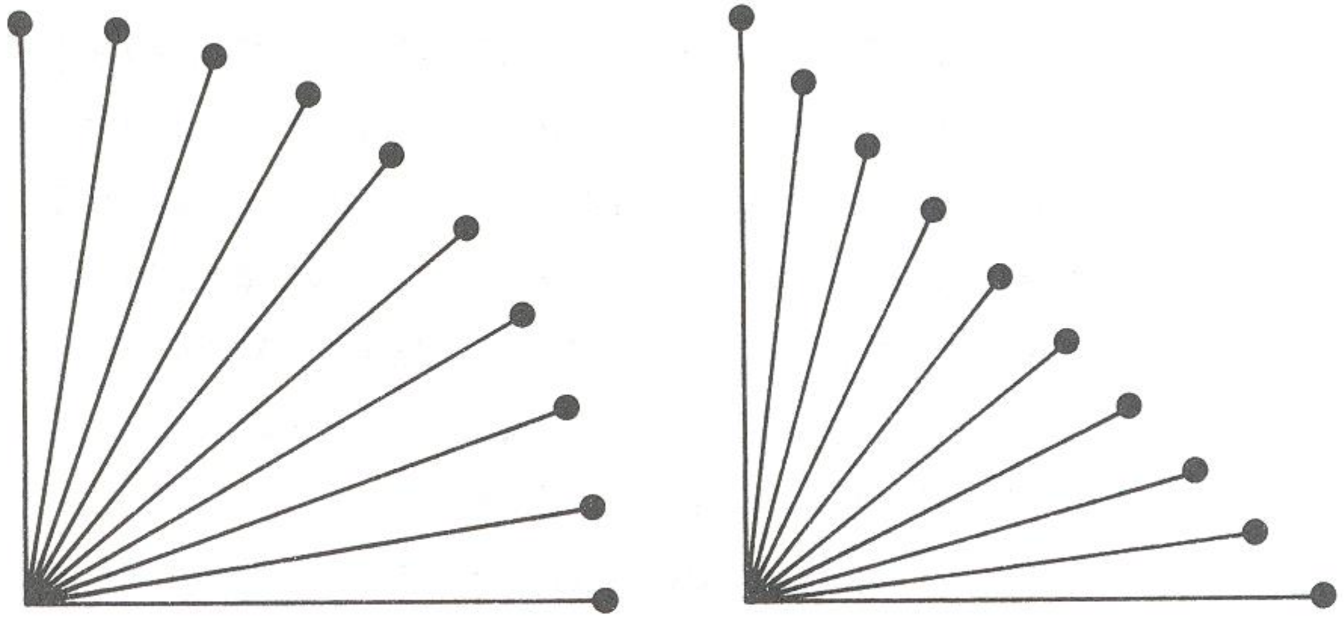



Rose et al. `96

# Articulated Figures

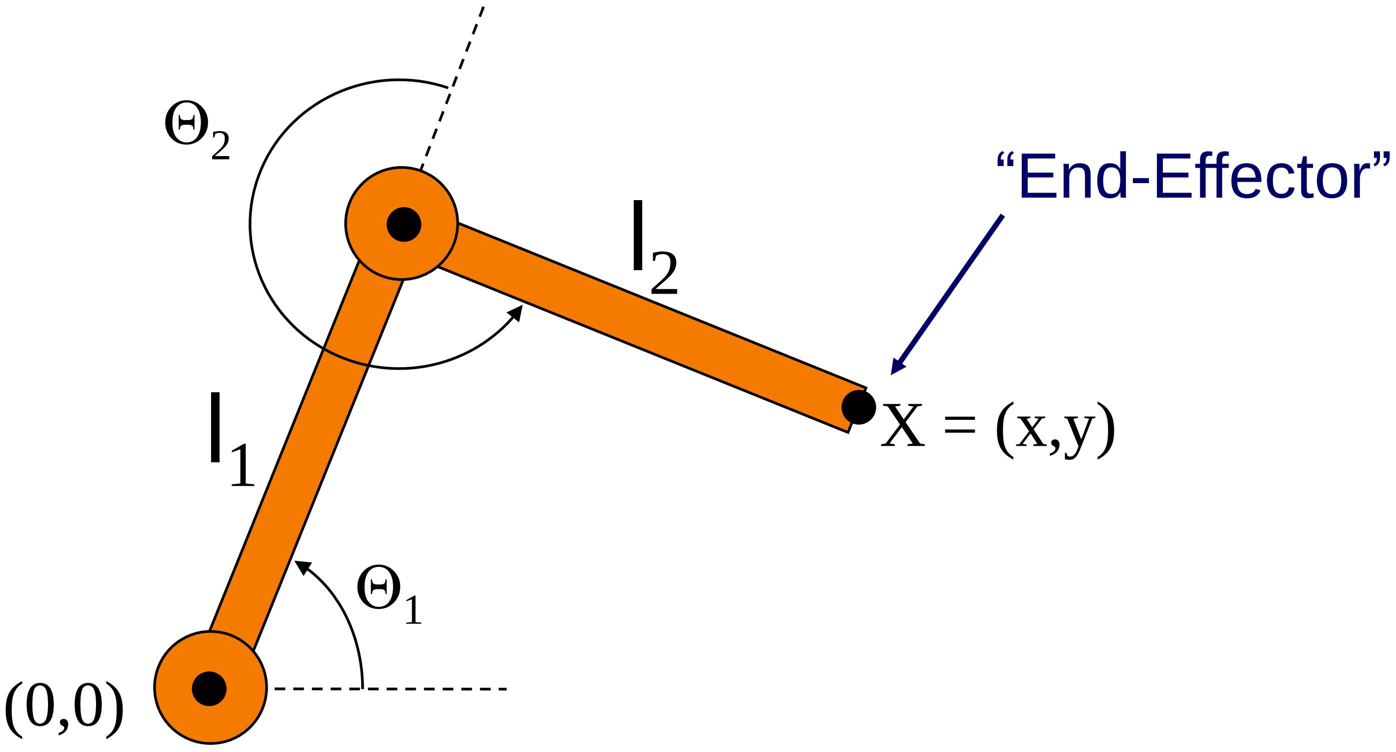
- Animation focuses on joint angles



Watt & Watt

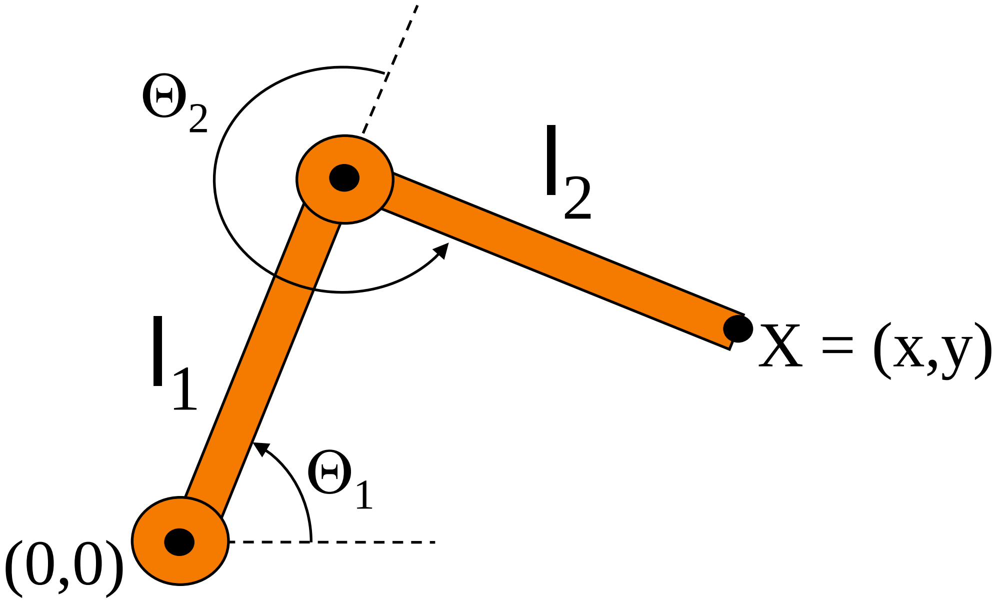
# Forward Kinematics

- Describe motion of articulated character



# Forward Kinematics

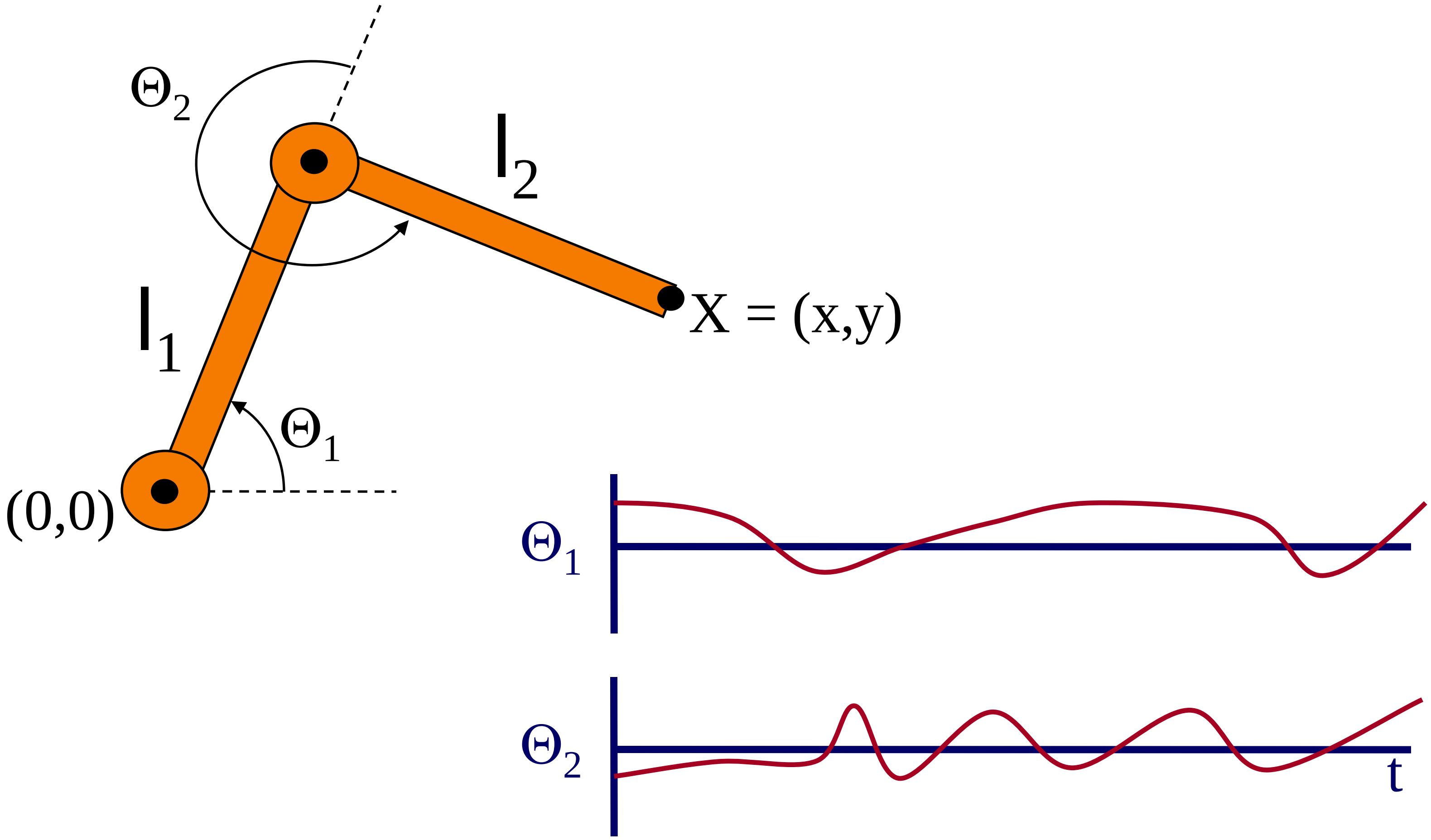
- Animator specifies joint angles: $\Theta_1$ and $\Theta_2$
- Computer finds positions of end-effector: X




$$X = (l_1 \cos \Theta_1 + l_2 \cos(\Theta_1 + \Theta_2), l_1 \sin \Theta_1 + l_2 \sin(\Theta_1 + \Theta_2))$$

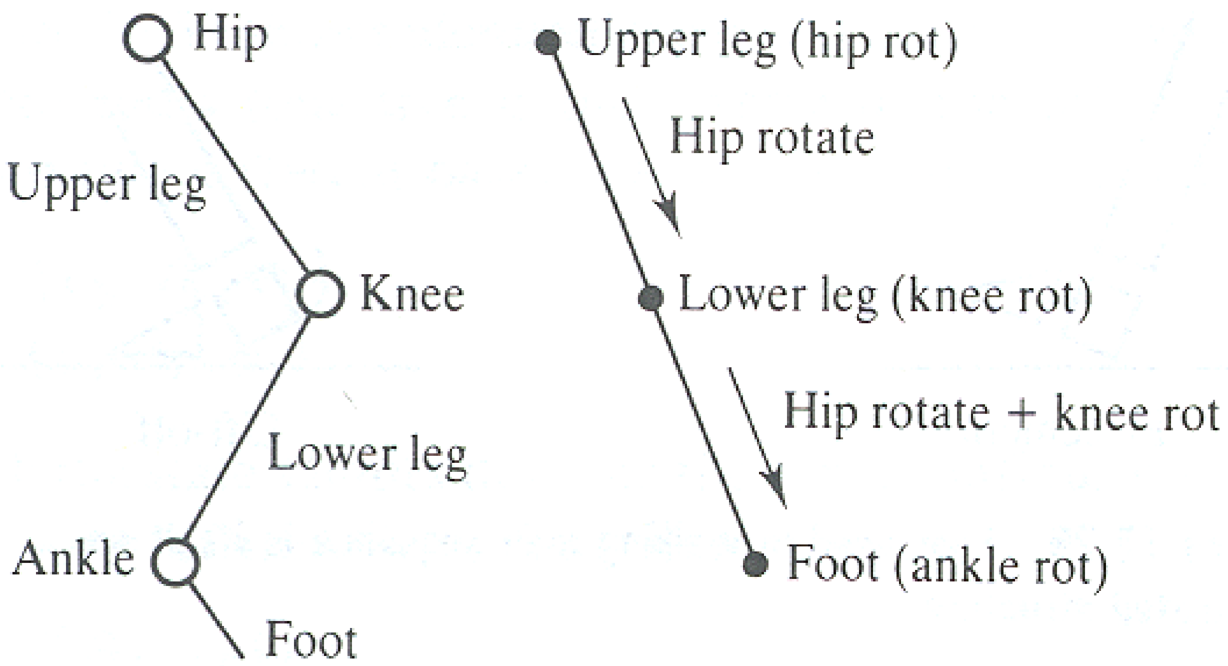
# Forward Kinematics

- Joint motions specified e.g. by spline curves



# Example: Walk Cycle

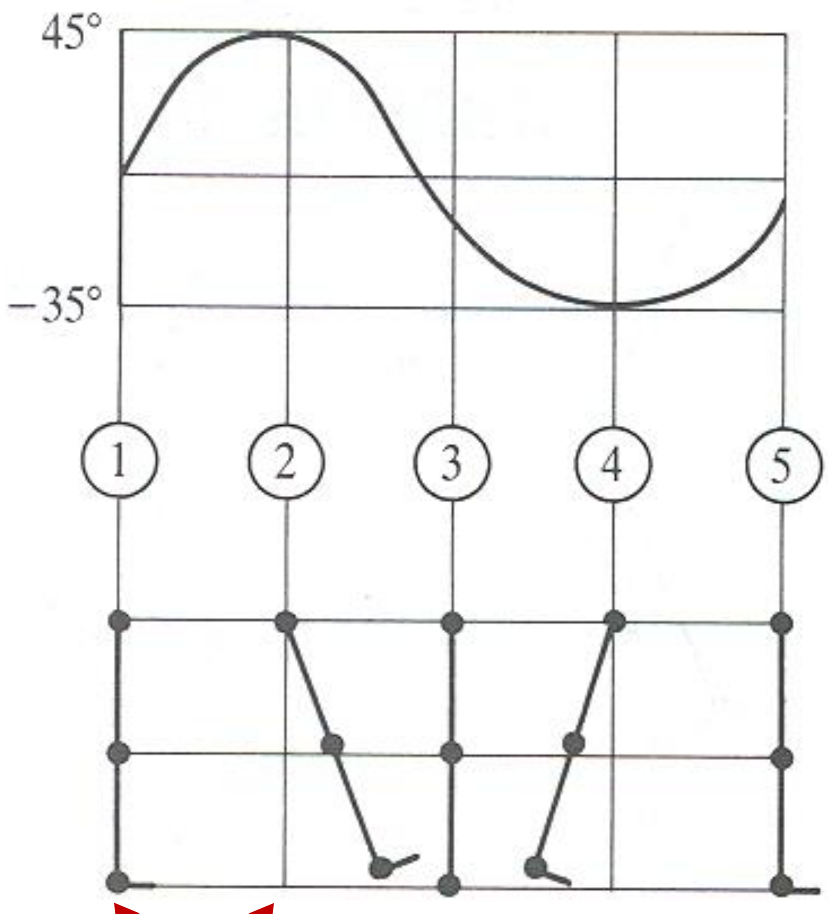
- Articulated figure:




Watt & Watt

# Example: Walk Cycle

- Hip joint orientation:

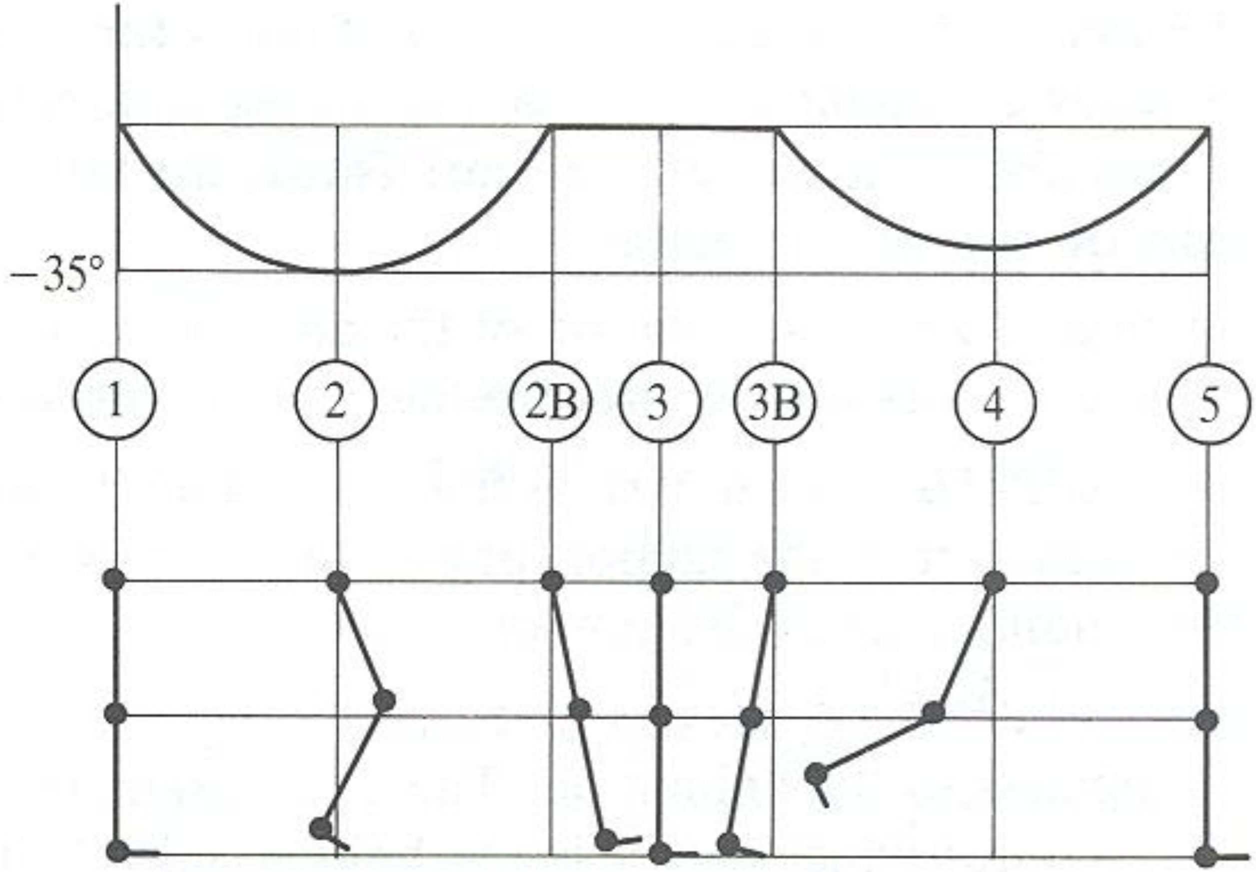



Keyframes

Watt & Watt

# Example: Walk Cycle

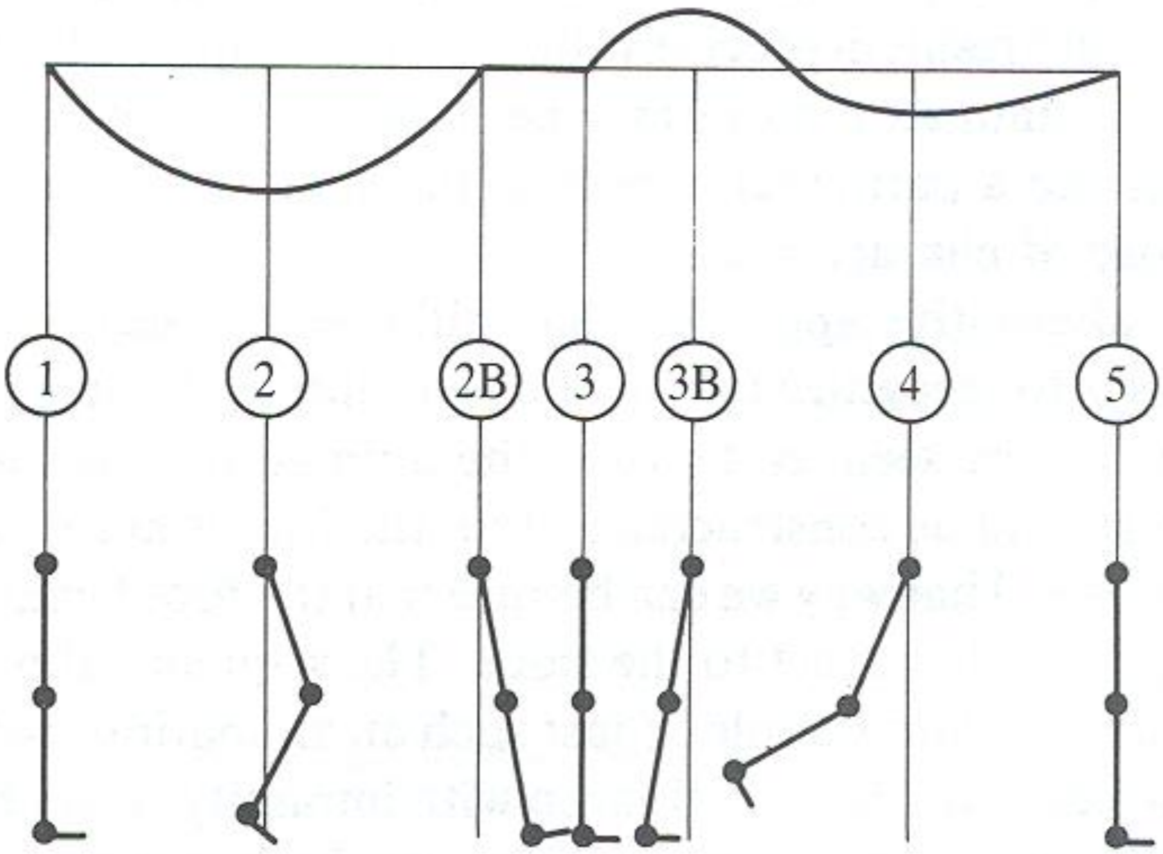
- Knee joint orientation:




Watt & Watt

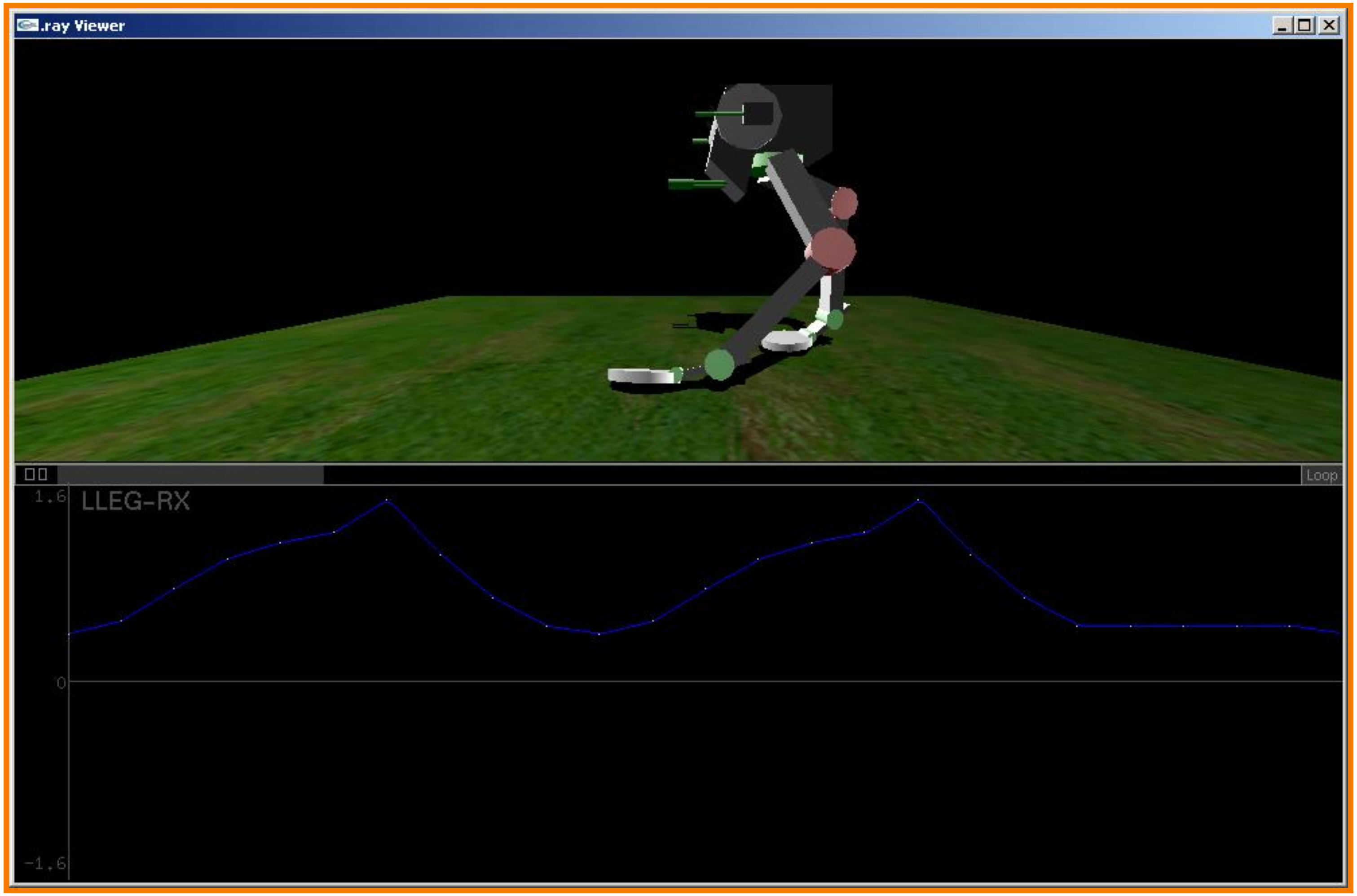
# Example: Walk Cycle

- Ankle joint orientation:




Watt & Watt

# Example: Robot



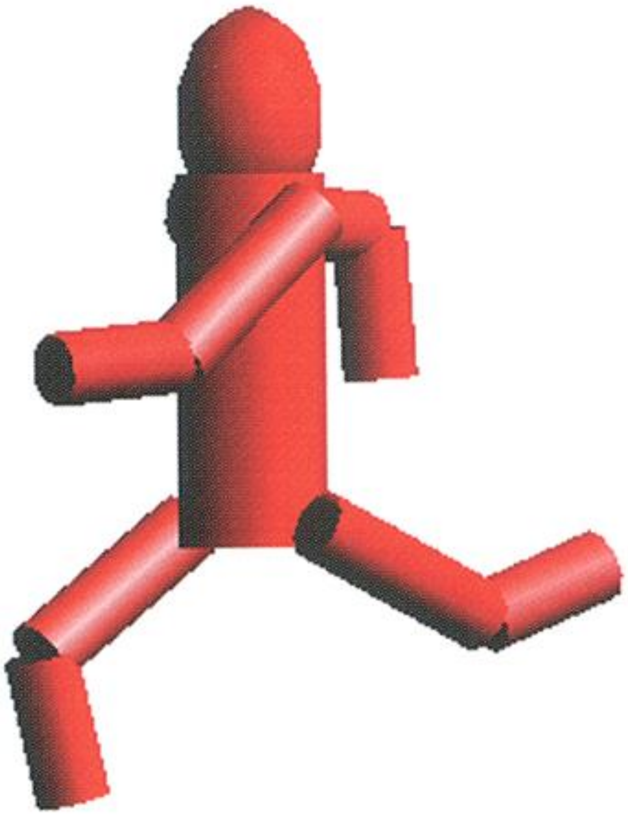

Mihai Parparita, COS 426, Princeton University, 2003

# Example: Ice Skating

(Mao Chen, Zaijin Guan, Zhiyan Liu, Xiaohu Qie, CS426, Fall98, Princeton University)

# Character Animation Methods

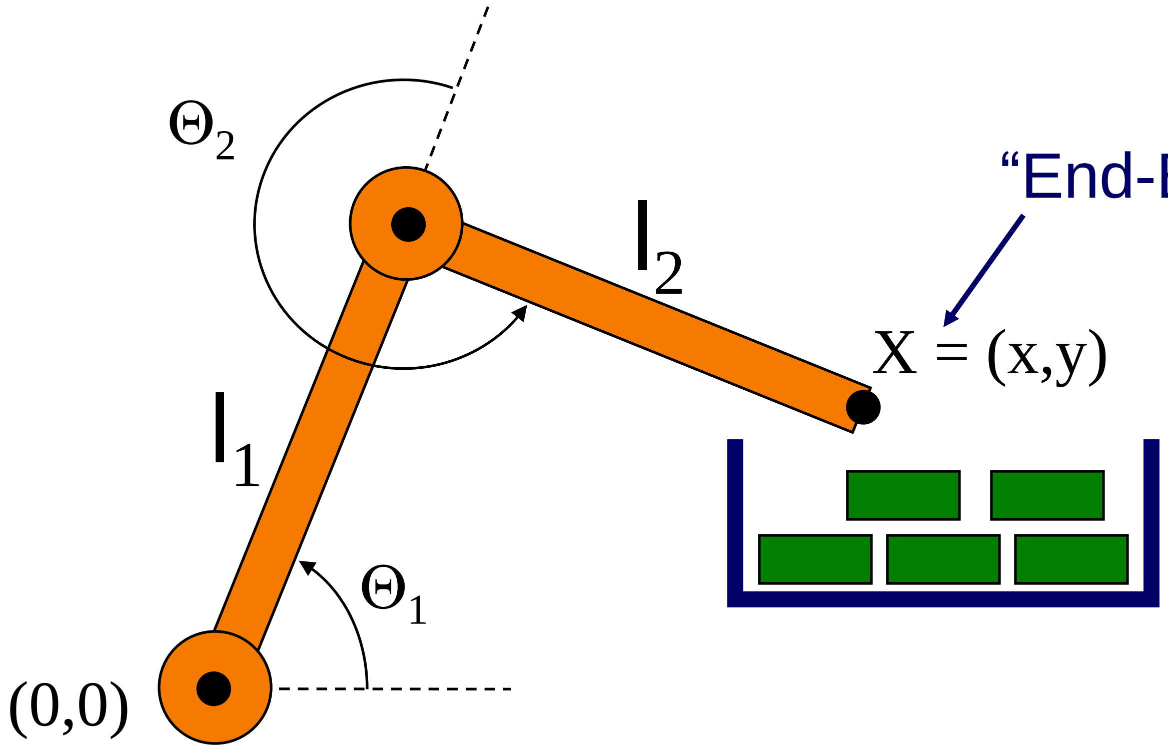
- Keyframing / Forward Kinematics
- Inverse Kinematics
- Dynamics
- Motion capture



Angel Plate 1

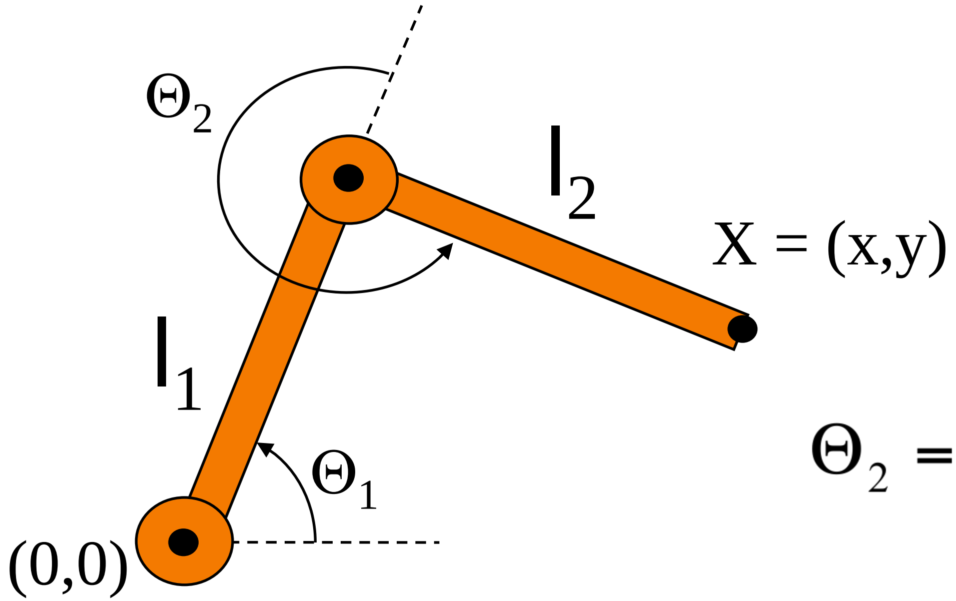
# Inverse Kinematics

- What if animator knows position of “end-effector”?



# Inverse Kinematics

- Animator specifies end-effector positions: X
- Computer finds joint angles: $\Theta_1$ and $\Theta_2$:

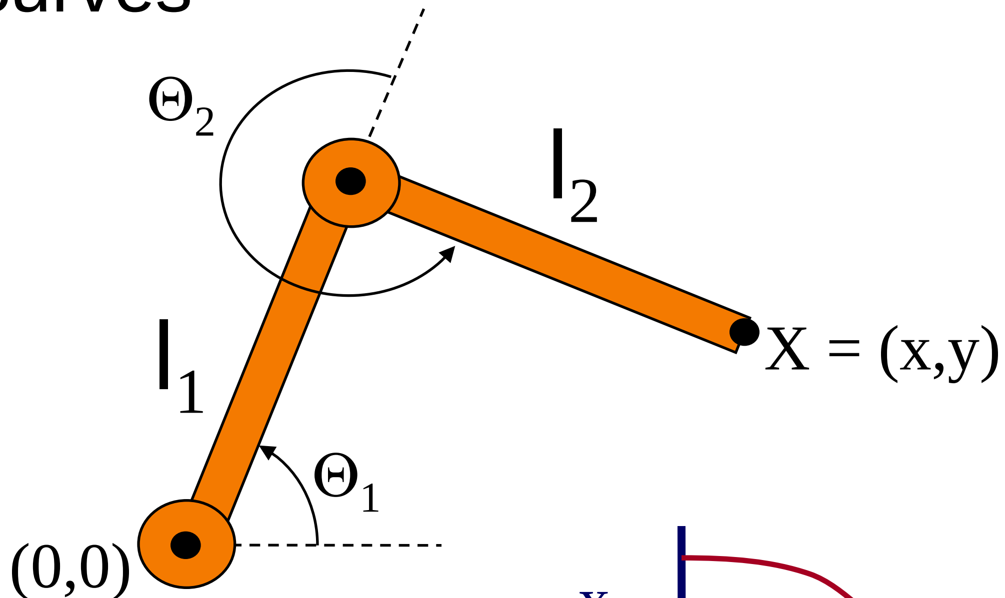


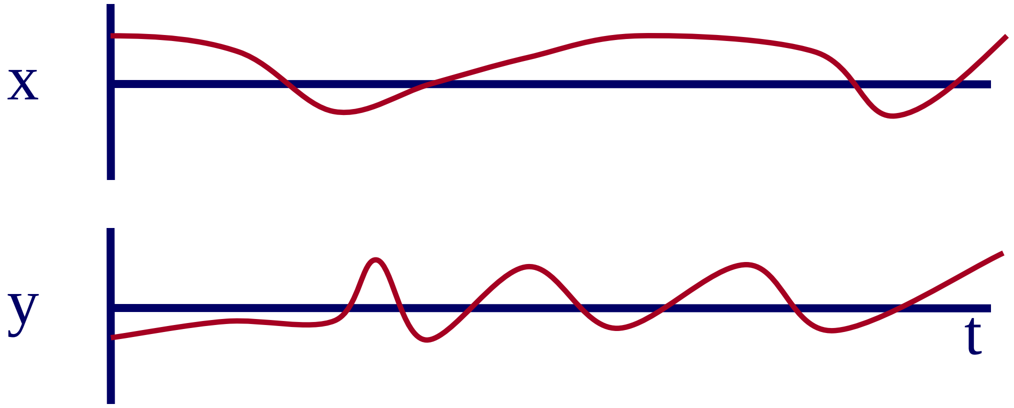

$$\Theta_2 = \cos^{-1}\left(\frac{x^2 + x^2 - l_1^{\ 2} - l_2^{\ 2}}{2 l_1 l_2}\right)$$

$$\Theta_1 = \frac{-(l_2 \sin(\Theta_2)x + (l_1 + l_2\cos(\Theta_2))y}{(l_2 \sin(\Theta_2))y + (l_1 + l_2\cos(\Theta_2))x}$$

# Inverse Kinematics

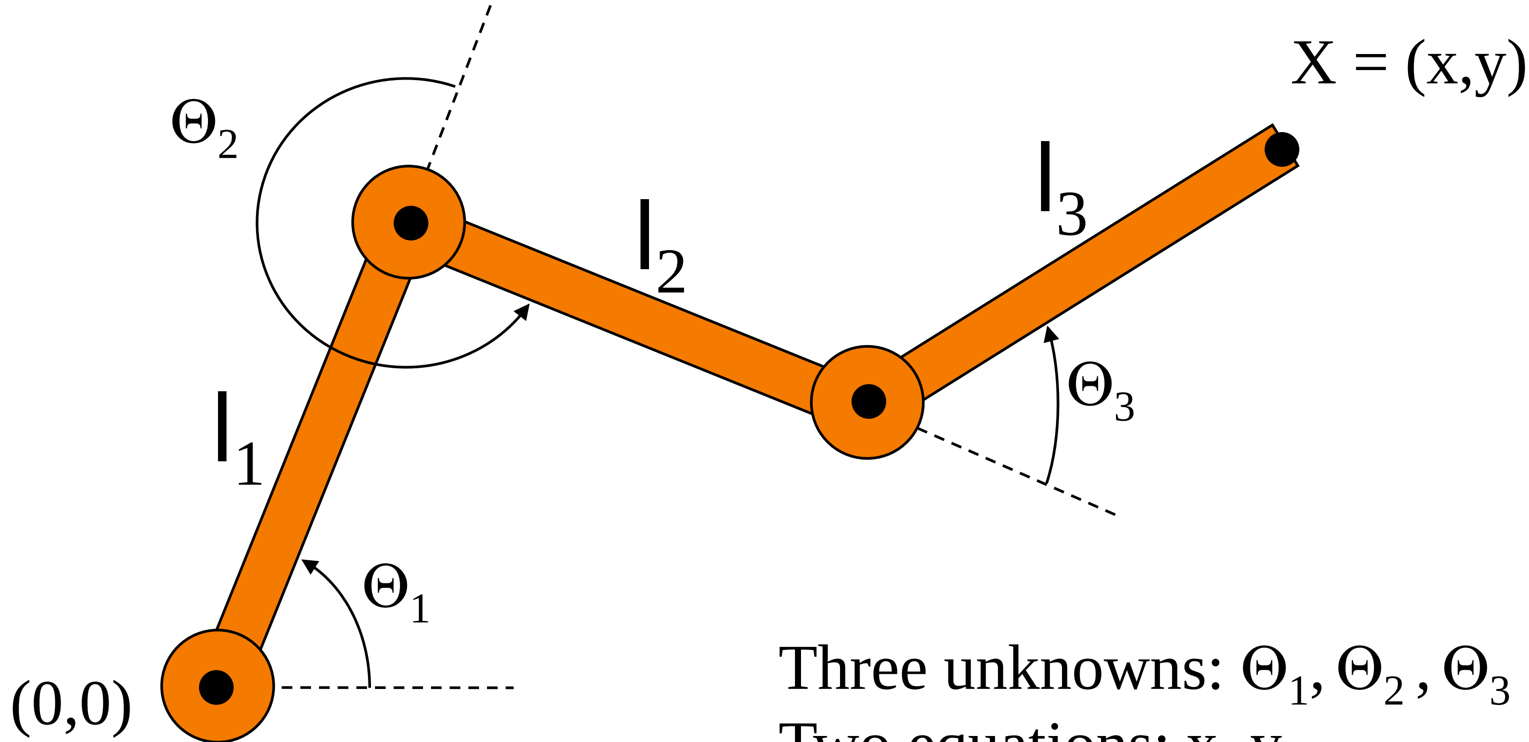
- End-effector postions can be specified by spline curves



# Inverse Kinematics

- Problem for more complex structures
  - System of equations is usually under-constrained
  - Multiple solutions

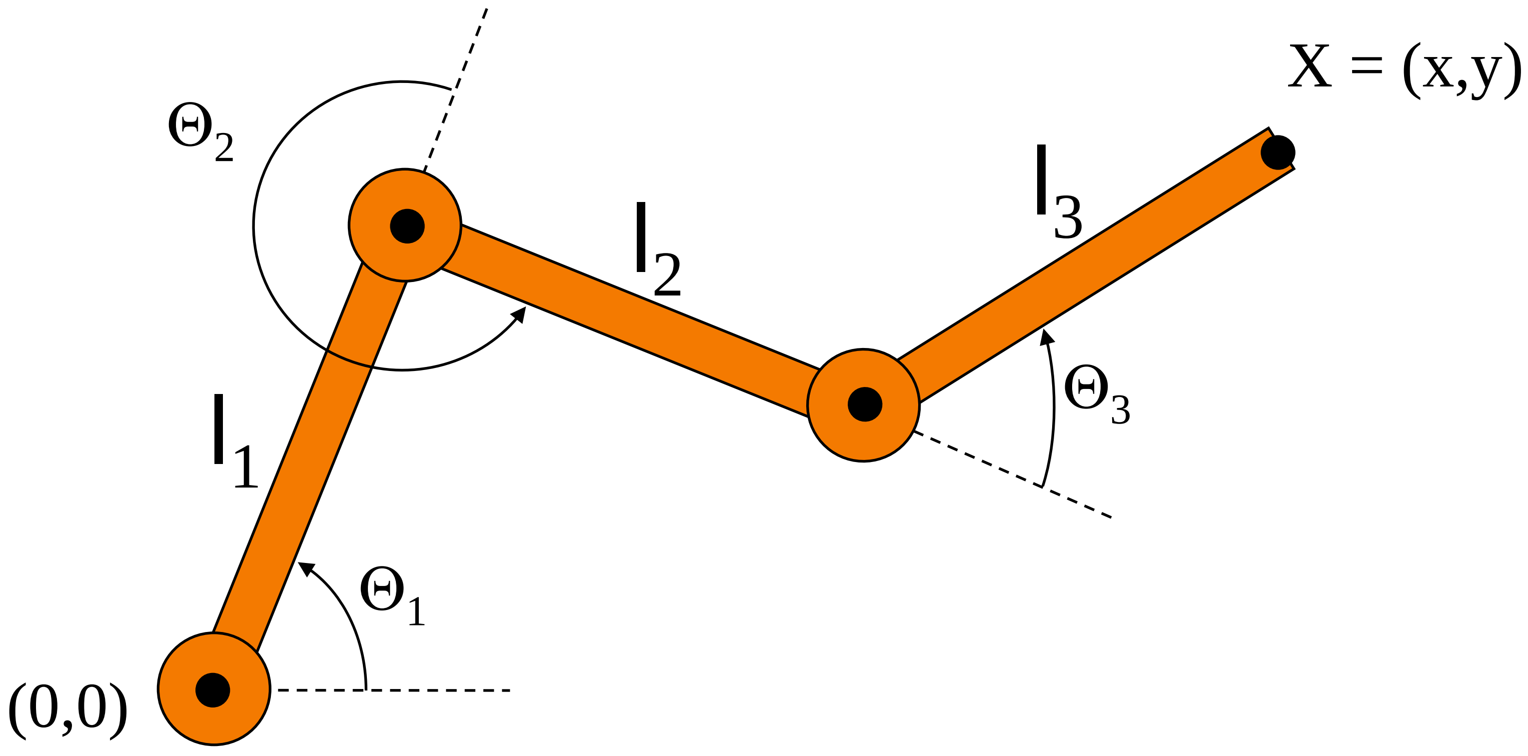



Three unknowns: $\Theta_1, \Theta_2, \Theta_3$

Two equations: x, y

# Inverse Kinematics

- Solution for more complex structures:
  - Find best solution (e.g., minimize energy in motion)
  - Non-linear optimization



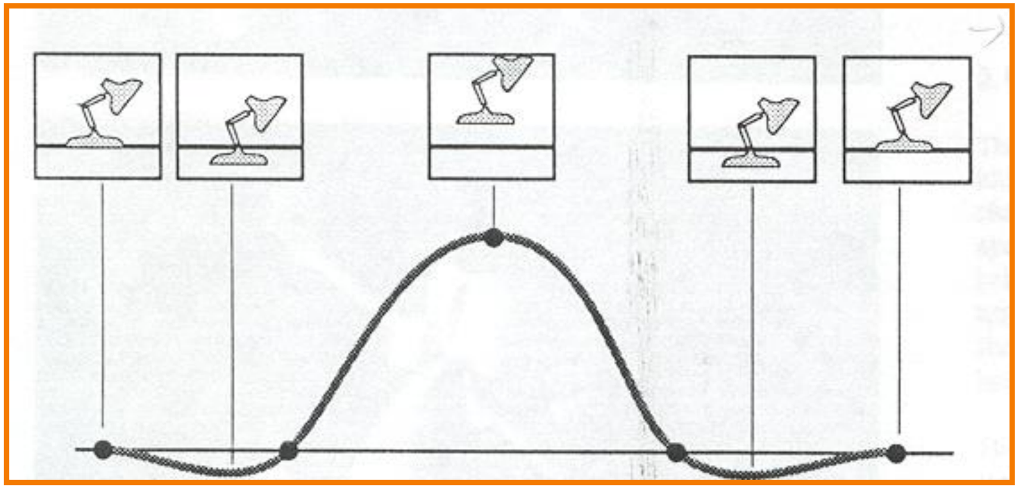
# Example: Ball Boy

“Ballboy”

Fujito, Milliron, Ngan, & Sanocki
Princeton University

# Kinematics

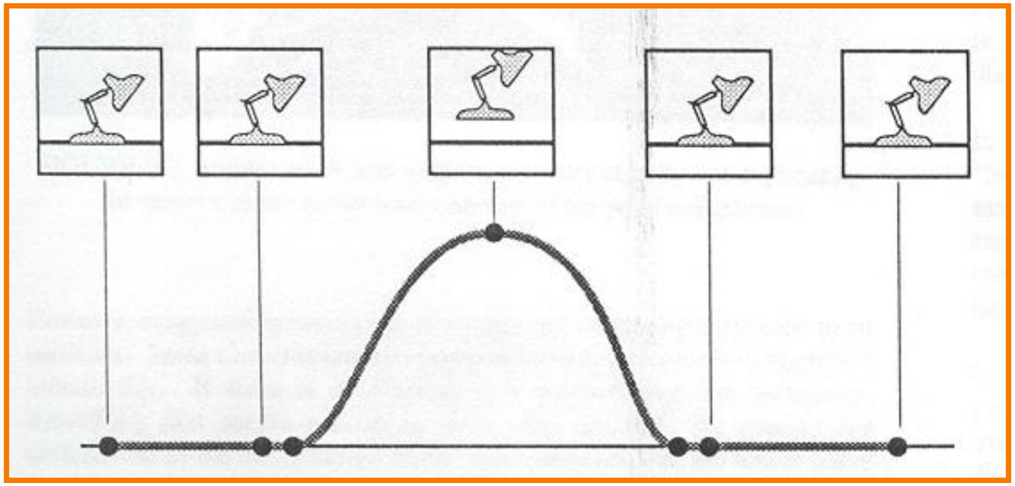
- Advantages
  - Simple to implement
  - Complete animator control
- Disadvantages
  - Motions may not follow physical laws
  - Tedious for animator



Lasseter `87

# Kinematics

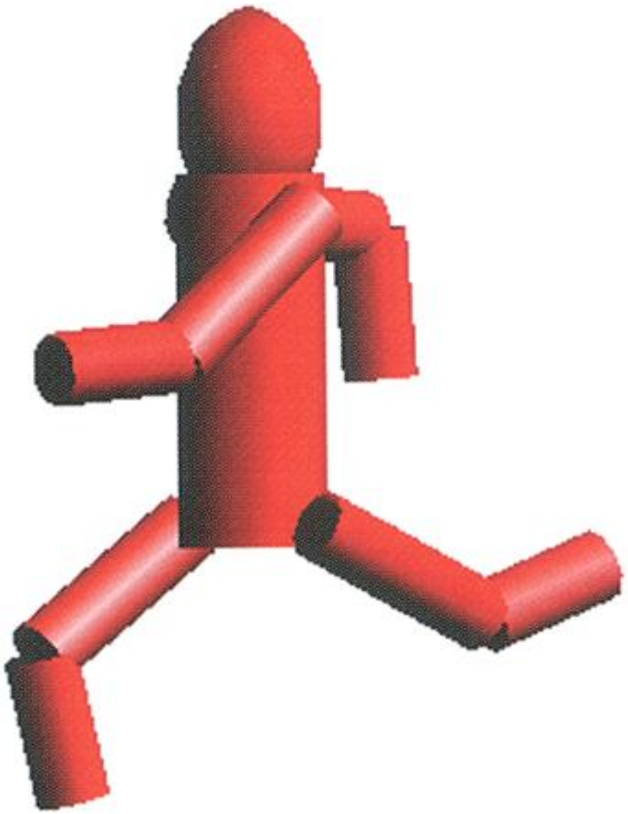
- Advantages
  - Simple to implement
  - Complete animator control
- Disadvantages
  - Motions may not follow physical laws
  - Tedious for animator



Lasseter `87

# Character Animation Methods

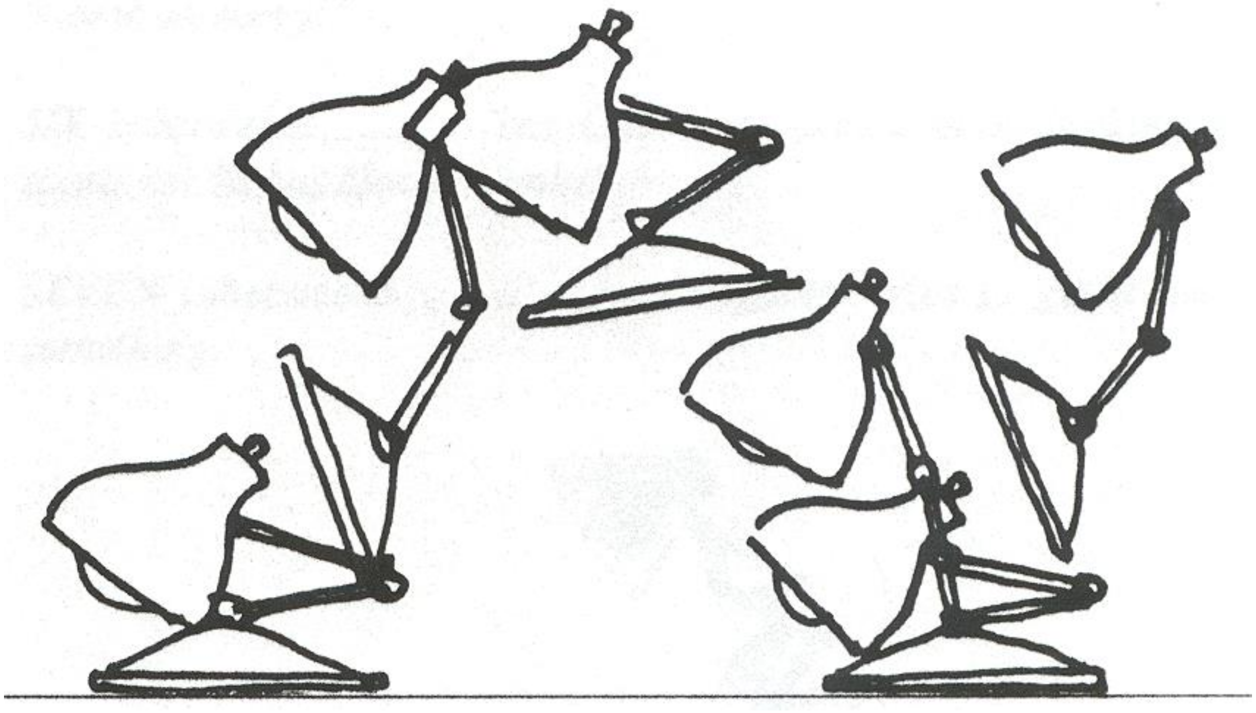
- Keyframing / Forward Kinematics
- Inverse Kinematics
- Dynamics
- Motion capture



Angel Plate 1

# Dynamics

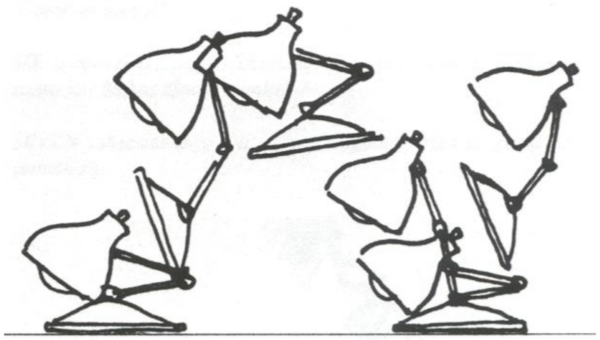
- Simulation of physics ensures realism of motion



Lasseter `87

# Spacetime Constraints

- Animator specifies constraints:
  - What the character's physical structure is
    - » e.g., articulated figure
  - What the character has to do (keyframes)
    - » e.g., jump from here to there within time t
  - What other physical structures are present
    - » e.g., floor to push off and land
  - How the motion should be performed
    - » e.g., minimize energy



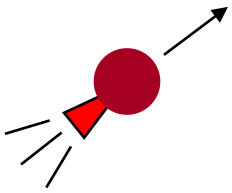
# Computer Animation

Pixar

# Spacetime Constraints

- Computer finds the "best" physical motion satisfying constraints

- Example: particle with jet propulsion
  - **x**(t) is position of particle at time t
  - **f**(t) is force of jet propulsion at time t
  - Particle's equation of motion is:



$$mx'' - f - mg = 0$$

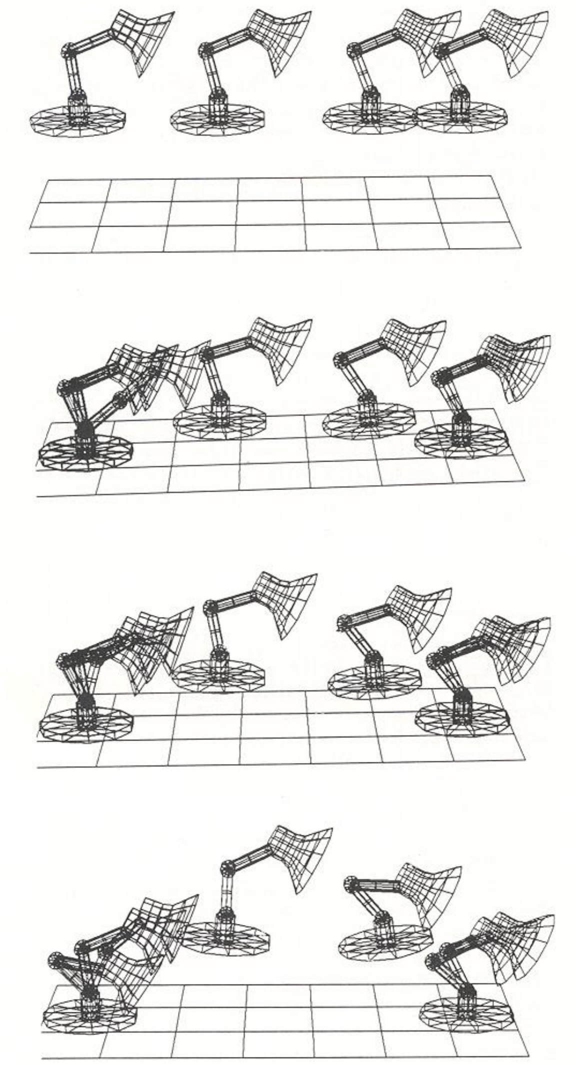
  - Suppose we want to move from a to b within $t_0$ to $t_1$ with minimum jet fuel:

$$\text{Minimize } \int_{t_0}^{t_1} \left| f(t)^2 \right| dt \text{ subject to } x(t_0)=a \text{ and } x(t_1)=b$$

Witkin & Kass `88

# Spacetime Constraints

- Solve with iterative optimization methods



Witkin & Kass `88

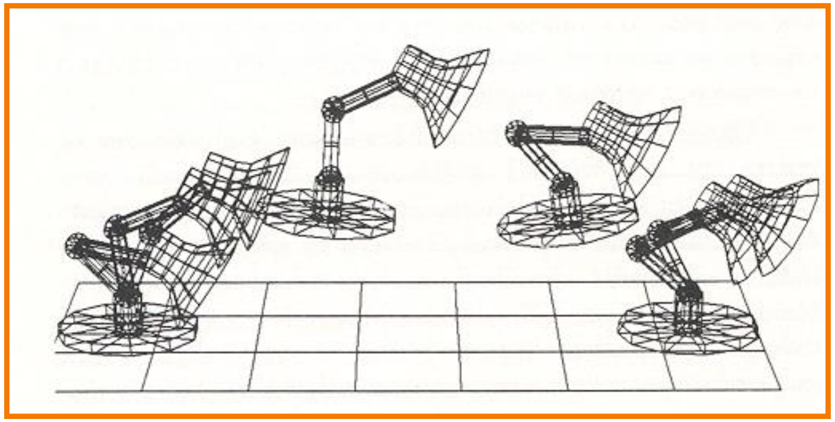
# Spacetime Constraints

- Advantages:
  - Free animator from having to specify details of physically realistic motion with spline curves
  - Easy to vary motions due to new parameters and/or new constraints

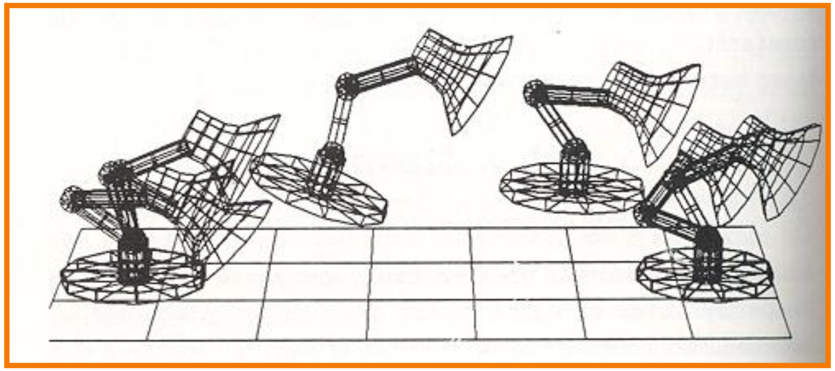
- Challenges:
  - Specifying constraints and objective functions
  - Avoiding local minima during optimization

# Spacetime Constraints

- Adapting motion:



Original Jump

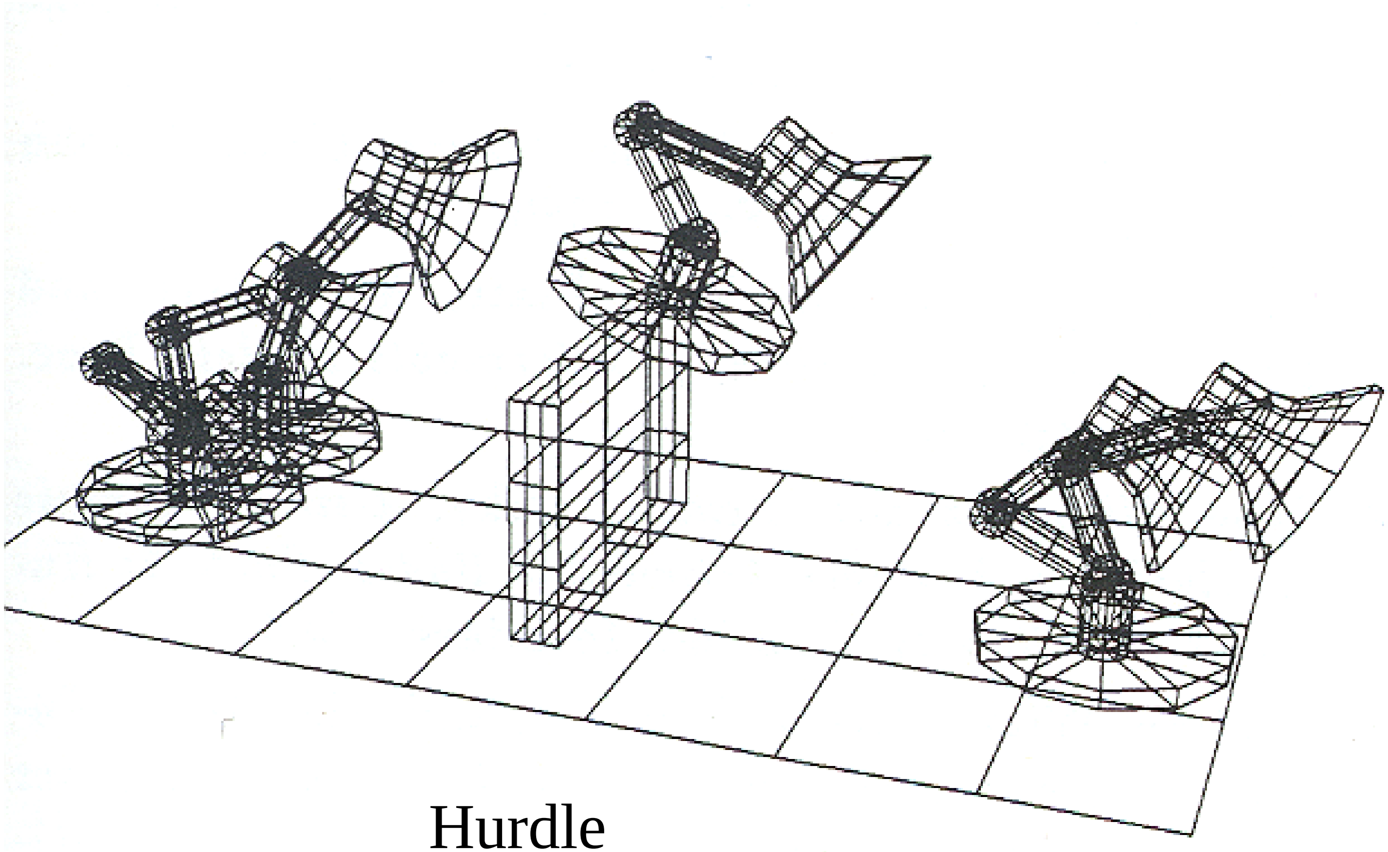


Heavier Base

Witkin & Kass `88

# Spacetime Constraints

- Adapting motion:

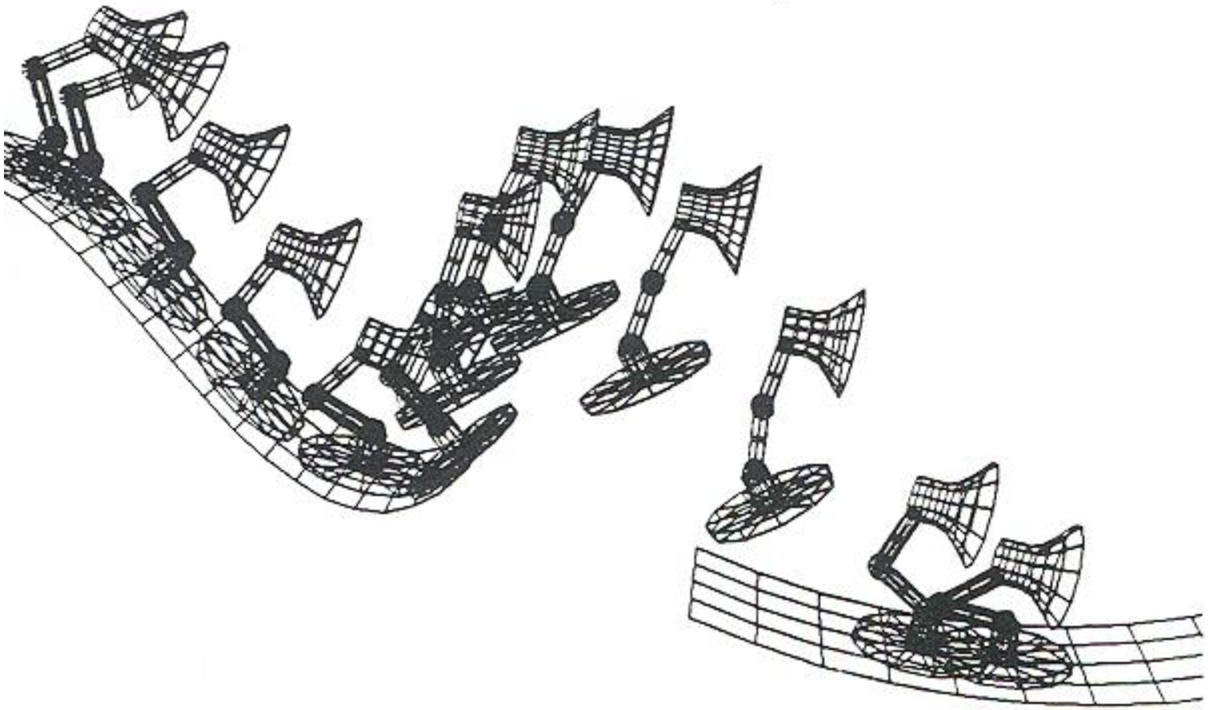


Hurdle

Witkin & Kass `88

# Spacetime Constraints

- Adapting motion:



Ski Jump

Witkin & Kass `88

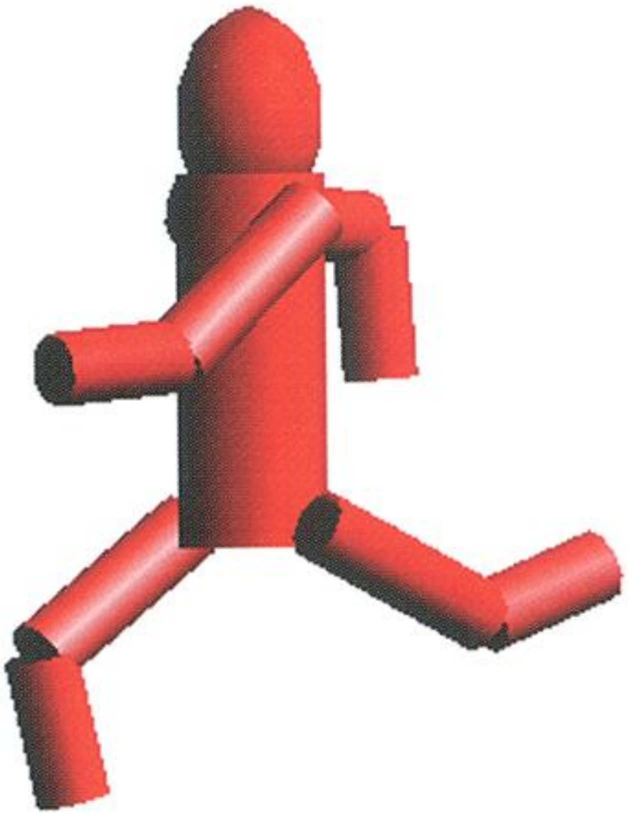
# Spacetime Constraints

- Advantages:
  - Free animator from having to specify details of physically realistic motion with spline curves
  - Easy to vary motions due to new parameters and/or new constraints

- Challenges:
  - Specifying constraints and objective functions
  - Avoiding local minima during optimization

# Character Animation Methods

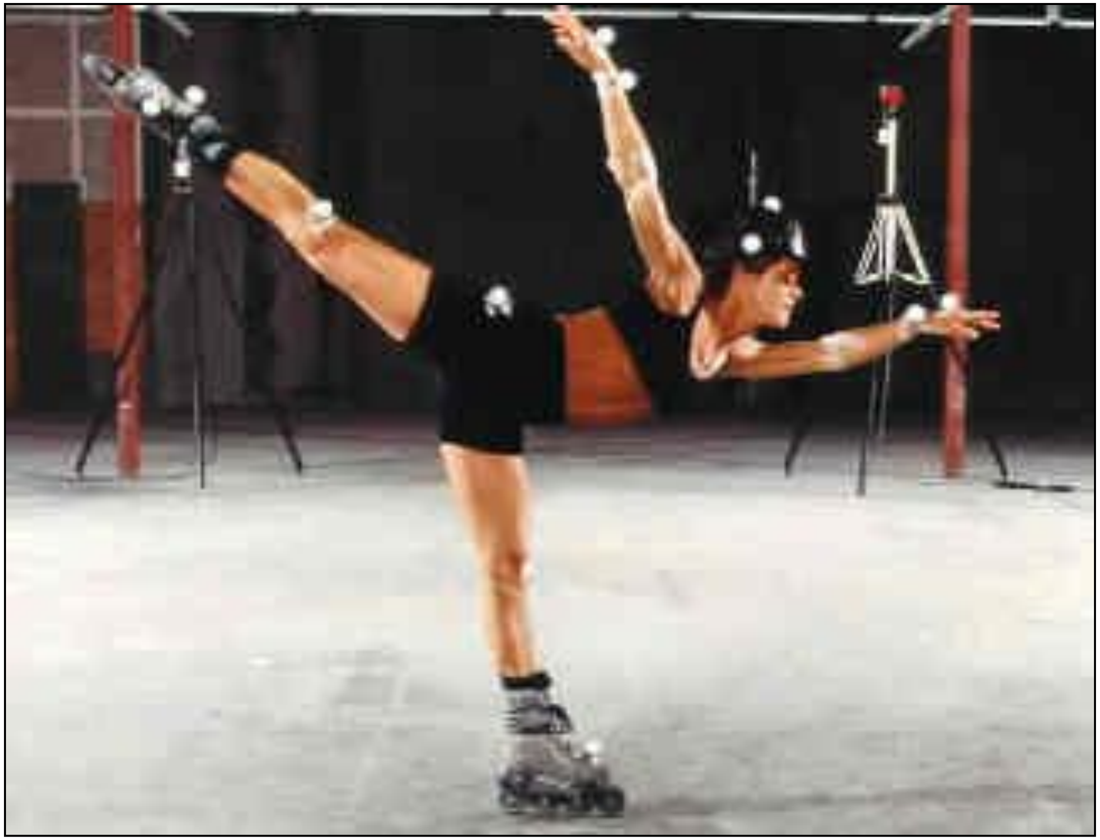
- Keyframing / Forward Kinematics
- Inverse Kinematics
- Dynamics
- Motion capture



Angel Plate 1

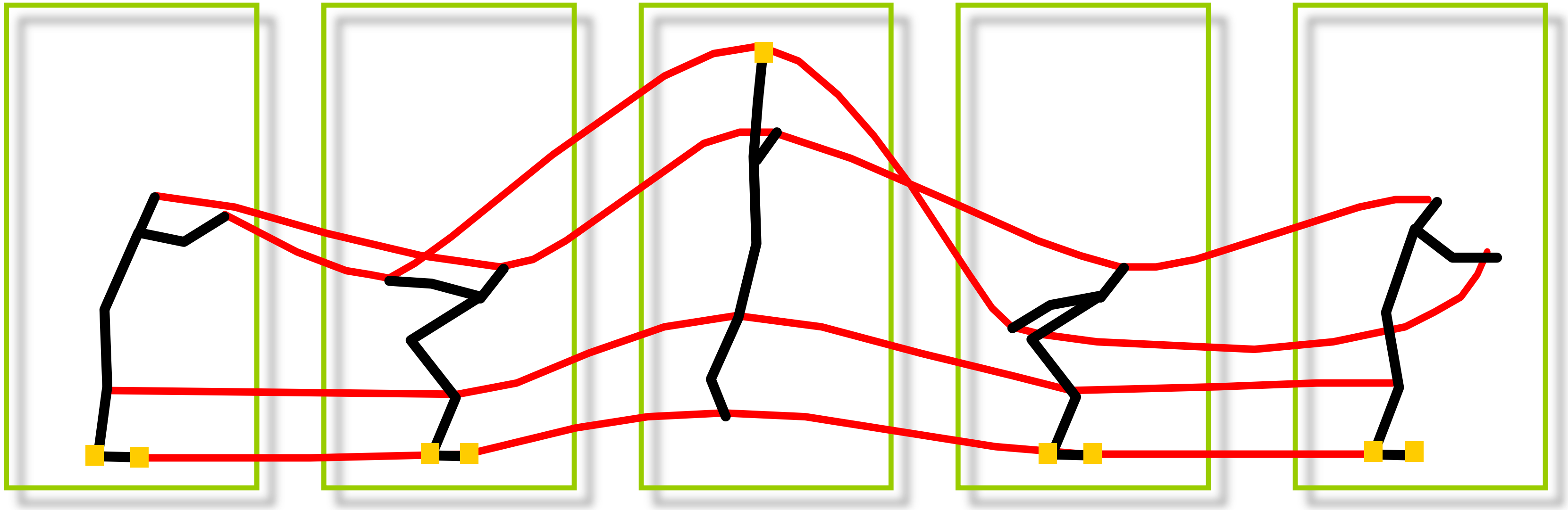
# Motion Capture

- Measure motion of real characters and then simply “play it back” with kinematics



# Motion Capture

- Measure motion of real characters and then simply "play it back" with kinematics



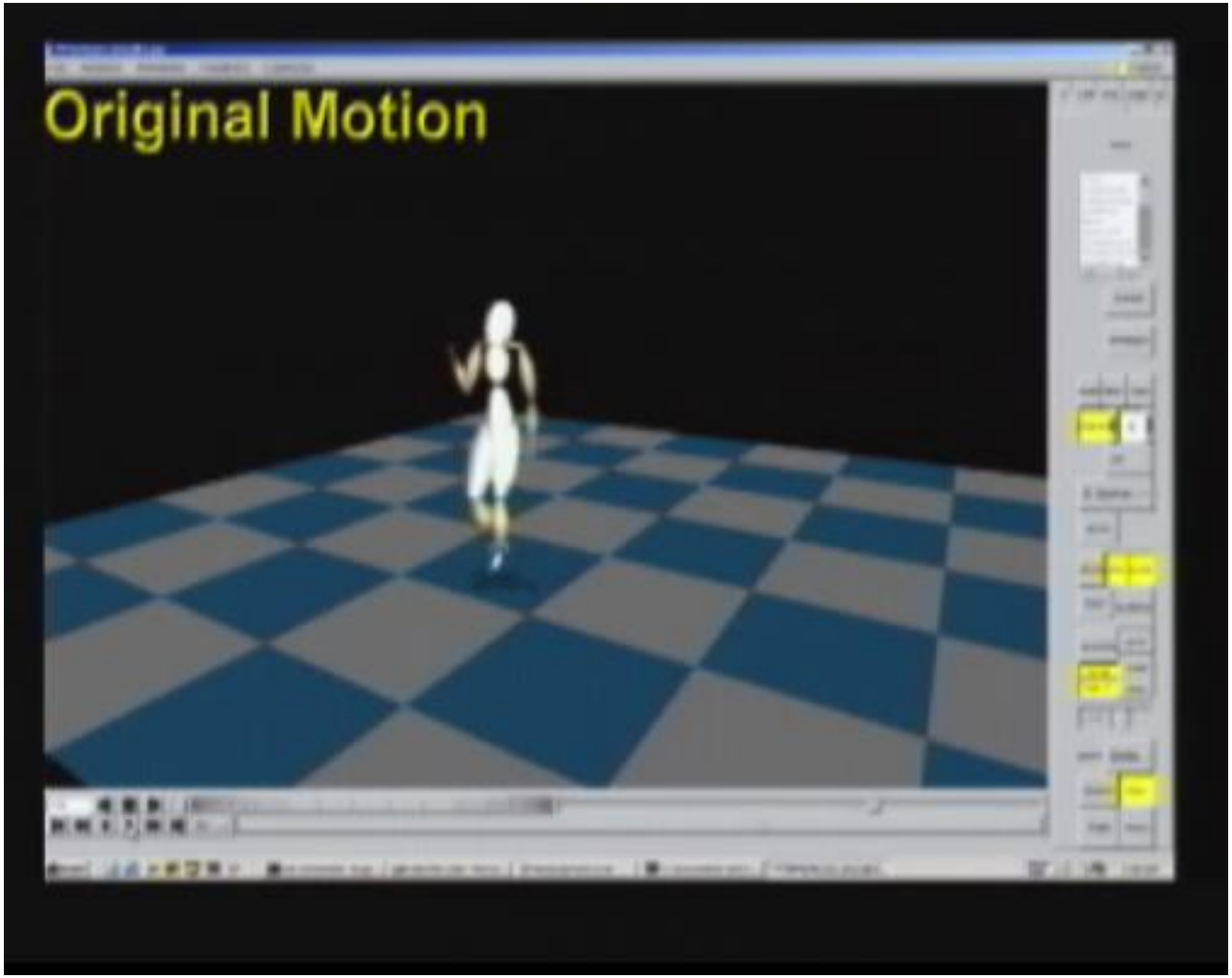
Captured Motion

# Motion Capture

- Advantage:
  - Physical realism
- Challenge:
  - Animator control

# Motion Capture

- Editing motion:

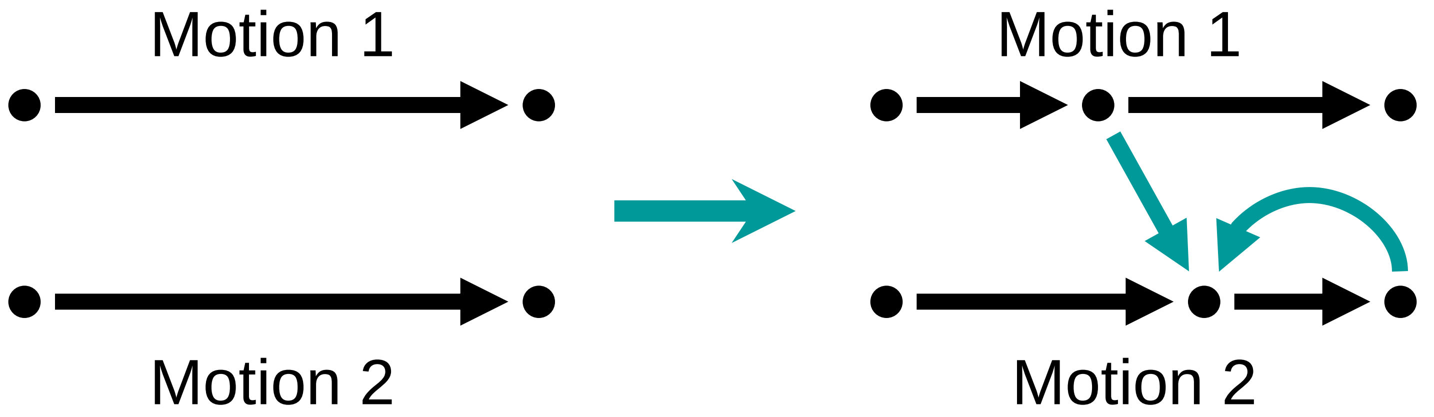


# Motion Capture

- Editing motion:

# Motion Capture

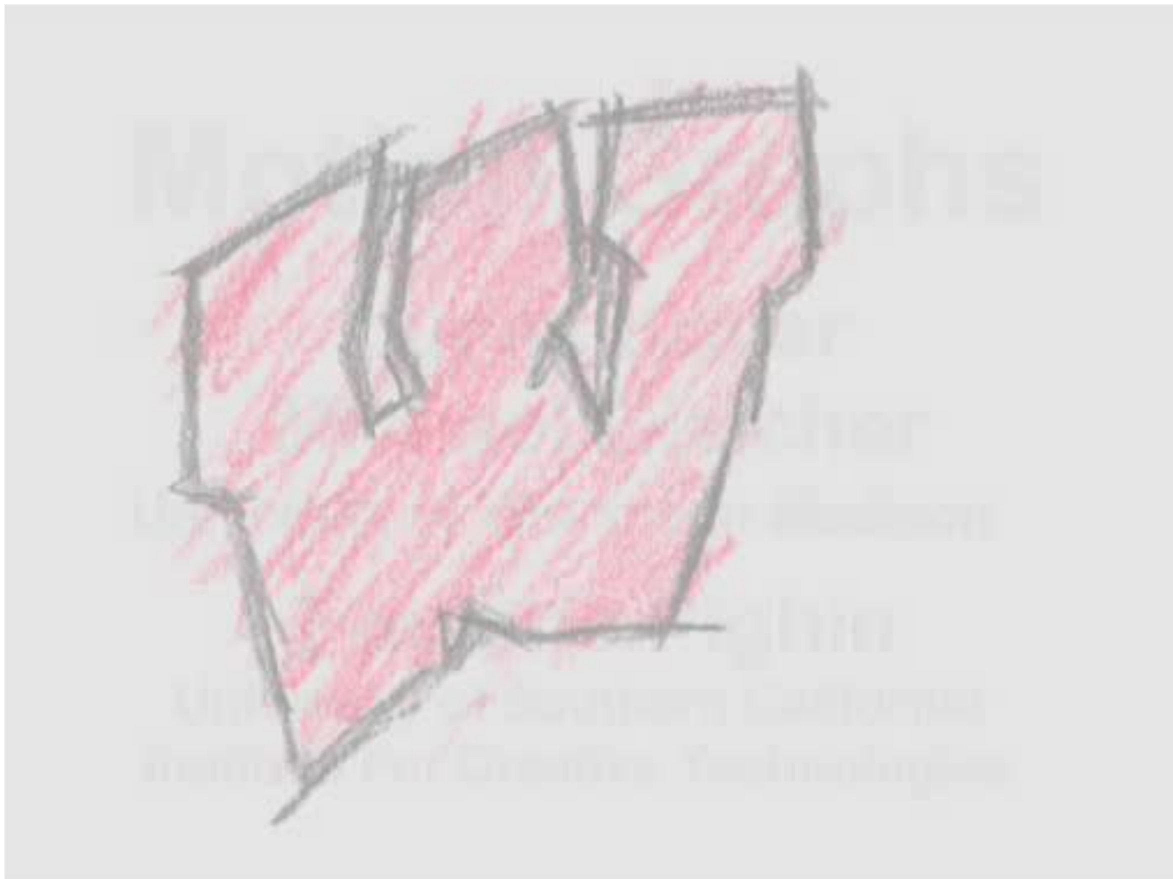
- Motion graphs:




Kovacs & Gleicher

# Motion Capture

- Motion graphs:



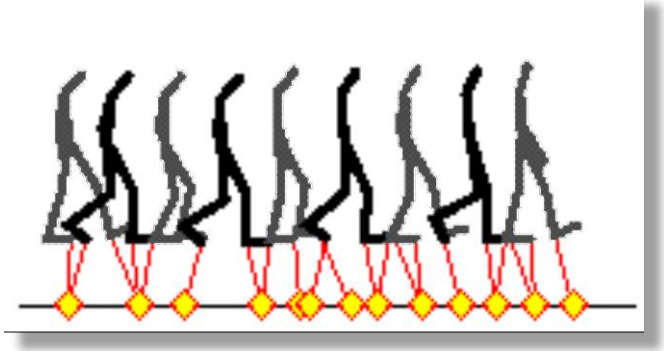
Kovacs & Gleicher

# Motion Capture

- Retargeting motion:

Original motion data + constraints:

New character:



New motion data:

Gleicher

# Motion Capture

- Retargeting motion:

Gleicher

# Motion Capture

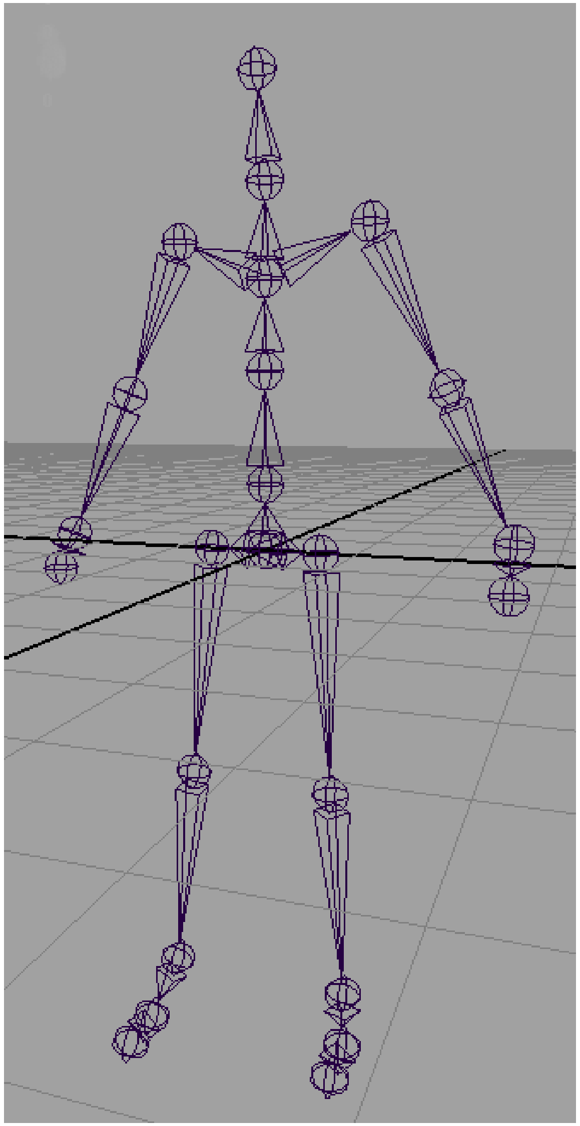
- Morphing motion:

# Beyond Skeletons…

- Skinning
- Motion blur

# Kinematic Skeletons

- Hierarchy of transformations ("bones")
  - Changes to parent affect all descendent bones
- So far: bones affect objects in scene or parts of a mesh
  - Equivalently, each point on a mesh acted upon by one bone
  - Leads to discontinuities when parts of mesh animated
- Extension: each point on a mesh acted upon by more than one bone



# Linear Blend Skinning

- Each vertex of skin potentially influenced by all bones
  - Normalized weight vector $w^{(v)}$ gives influence of each bone transform
  - When bones move, influenced vertices also move
- Computing a transformation $T_v$ for a skinned vertex
  - For each bone
    - » Compute global bone transformation $T_b$ from transformation hierarchy
  - For each vertex
    - » Take a linear combination of bone transforms
    - » Apply transformation to vertex in original pose

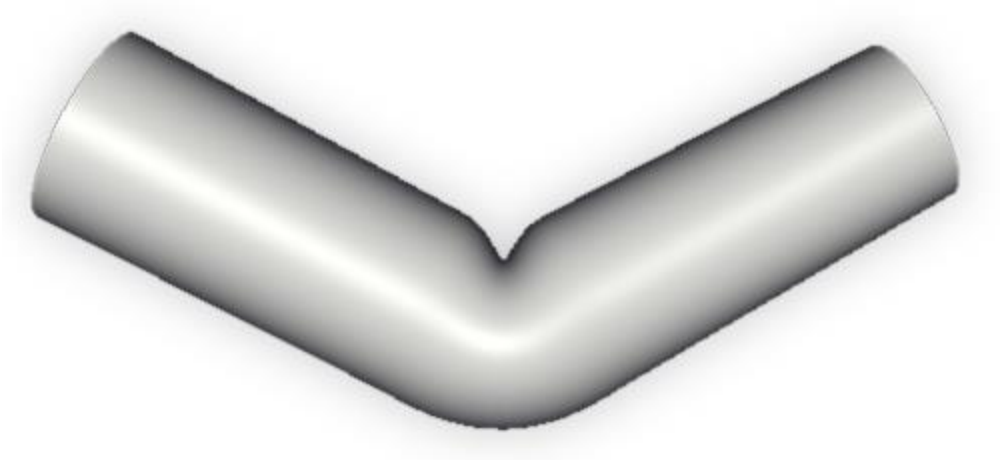
$$T_v = \sum_{b \in B} w_b^{(v)} T_b$$

- Equivalently, transformed vertex position is weighted combination of positions transformed by bones

$$v_{transformed} = \sum_{b \in B} w_b^{(v)} \left( T_b v \right)$$

# Assigning Weights: "Rigging"

- Painted by hand
- Automatic: function of relative distances to nearest bones
  - Smoothness of skinned surface depends on smoothness of weights!

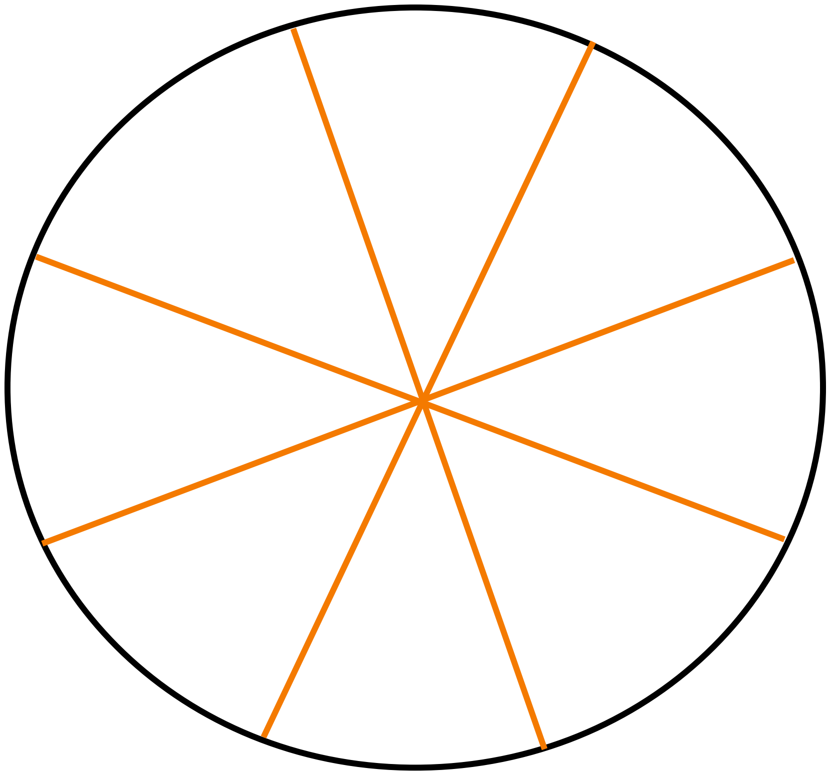


# Beyond Skeletons…

- Skinning
- Motion blur

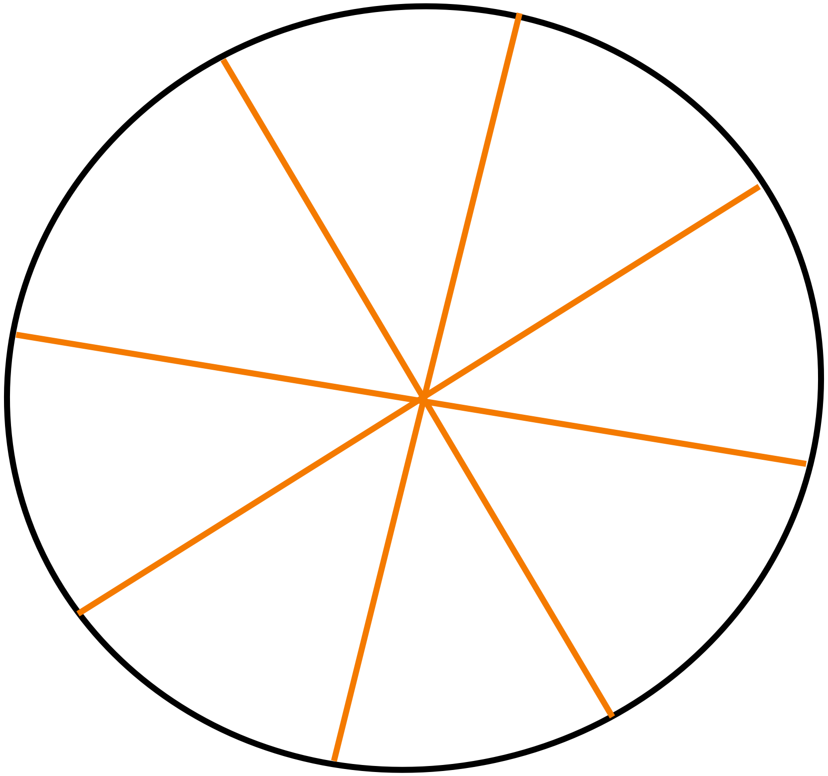
# Temporal Aliasing

- Artifacts due to limited temporal resolution
  - Strobing
  - Flickering



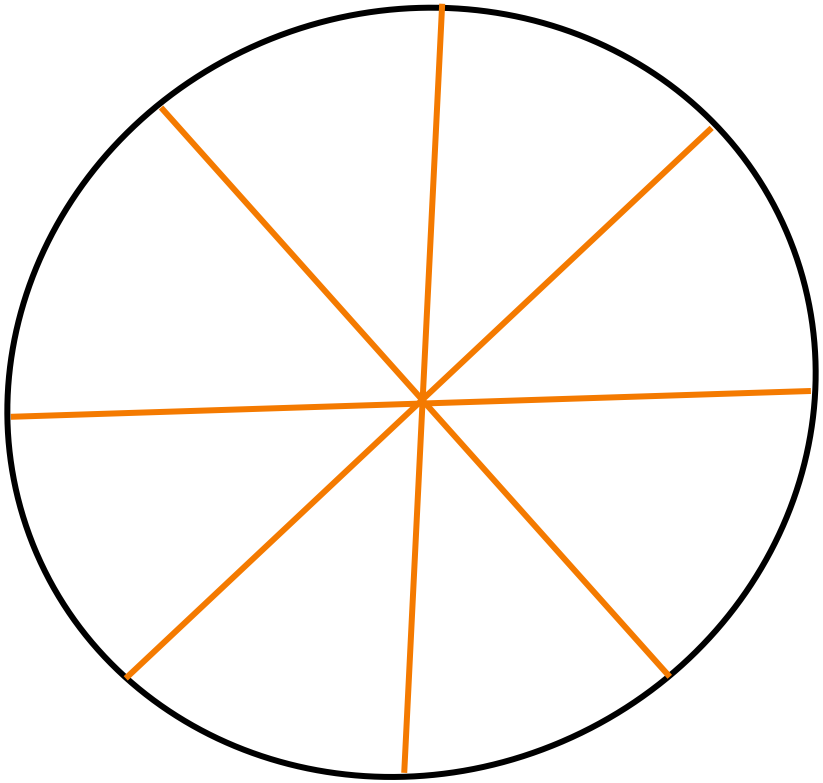
# Temporal Aliasing

- Artifacts due to limited temporal resolution
  - Strobing
  - Flickering



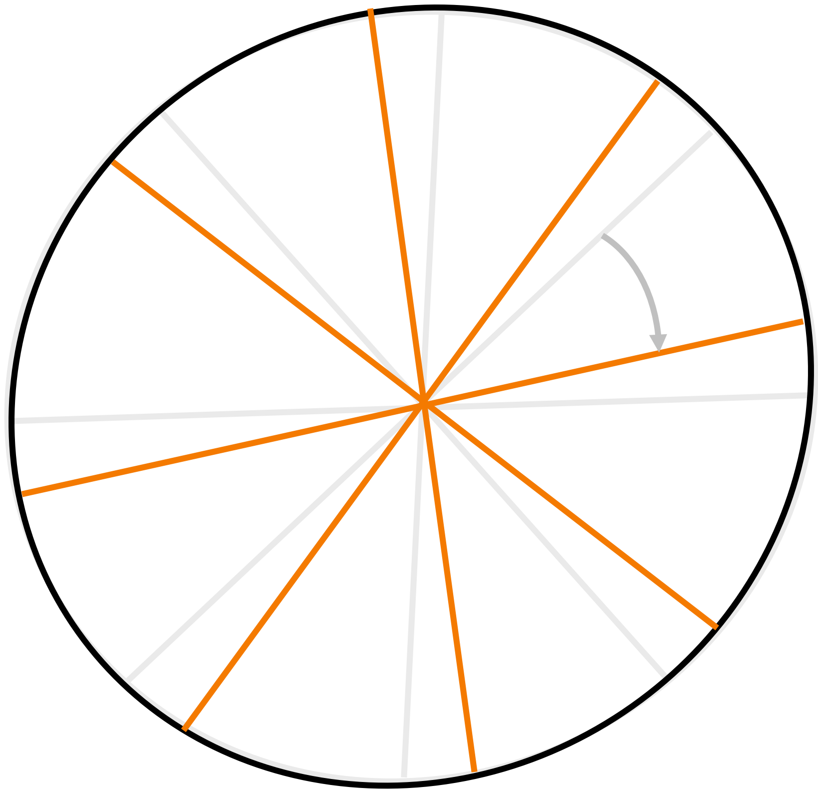
# Temporal Aliasing

- Artifacts due to limited temporal resolution
  - Strobing
  - Flickering



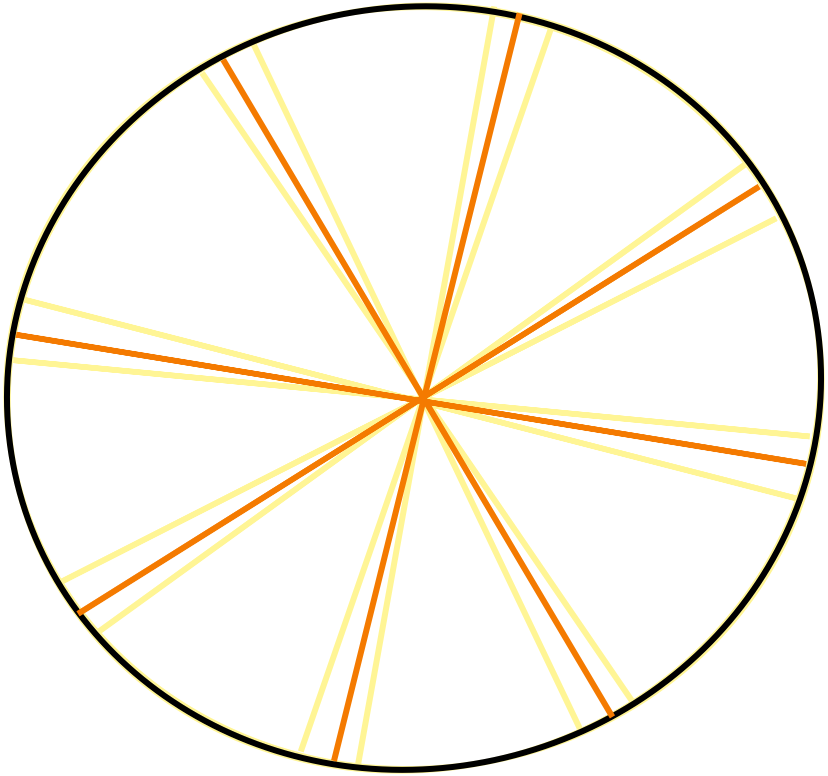
# Temporal Aliasing

- Artifacts due to limited temporal resolution
  - Strobing
  - Flickering



# Motion Blur

- Composite weighted images of adjacent frames
  - Remove parts of signal under-sampled in time



# Summary

- Kinematics
  - Animator specifies poses (joint angles or positions) at keyframes and computer determines motion by kinematics and interpolation
- Dynamics
  - Animator specifies physical attributes, constraints, and starting conditions and computer determines motion by physical simulation
- Motion capture
  - Compute captures motion of real character and provides tools for animator to edit it