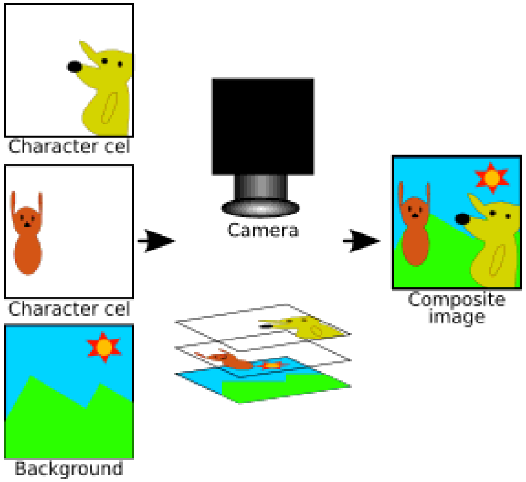
# Basics of Computer Animation
# Skinning/Enveloping

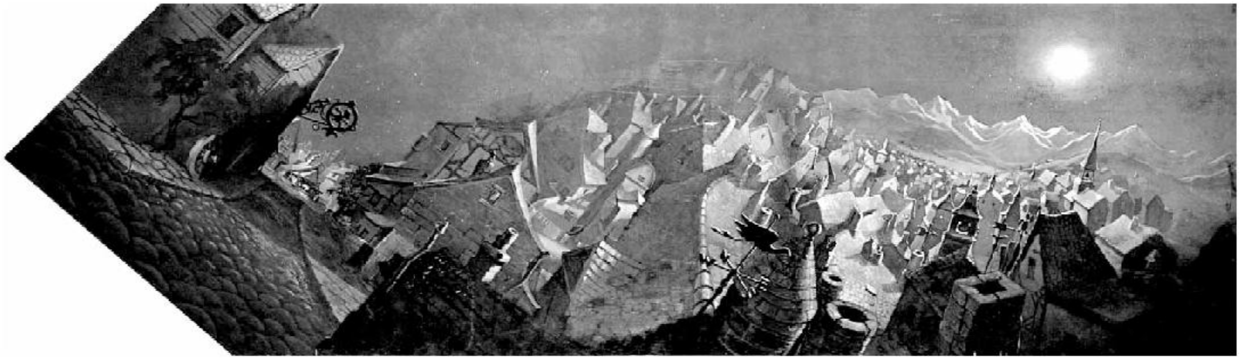
Many slides courtesy of Jovan Popovic, Ronen Barzel, and Jaakko Lehtinen

# Traditional Animation

- Draw each frame by hand
  - great control, but tedious
- Reduce burden with cel animation
  - Layer, keyframe, inbetween, …
  - Example: Cel panoramas (Disney's Pinocchio)







From ACM © 1997 "Multiperspective panoramas for cel animation."

# Traditional Animation Principles

- The in-betweening, was once a job for apprentice animators. Splines accomplish these tasks automatically. However, the animator still has to draw the keyframes. This is an art form and precisely why the experienced animators were spared the in-betweening work even before automatic techniques.
- The classical paper on animation by John Lasseter from Pixar surveys some the standard animation techniques:
- "*Principles of Traditional Animation Applied to 3D Computer Graphics*," **SIGGRAPH'87**, pp. 35-44.
- See also The Illusion of Life: Disney Animation, by Frank Thomas and Ollie Johnston.

# Example: Squash and Stretch

- **Squash**: flatten an object or character by pressure or by its own power

- **Stretch**: used to increase the sense of speed and emphasize the squash by contrast

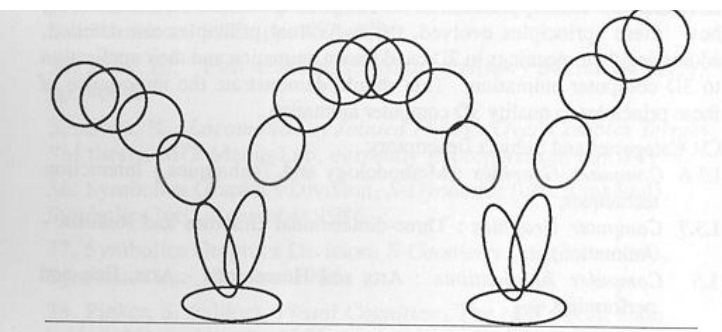


FIGURE 2. Squash & stretch in bouncing ball.

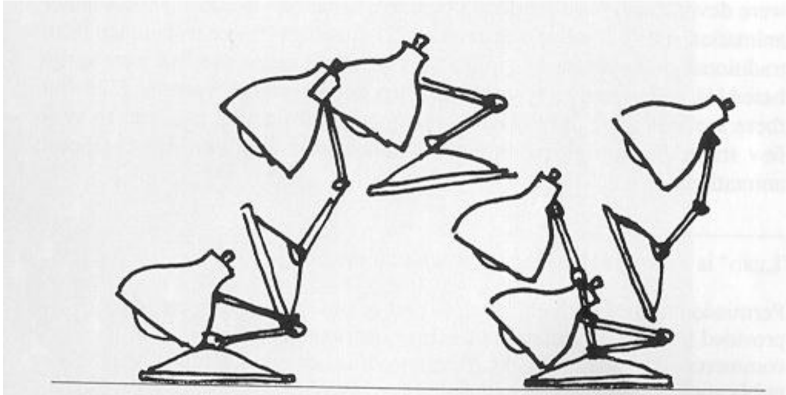


FIGURE 3. Squash & stretch in Luxo Jr.'s hop.



Image adapted from: Lasseter, John. "Principles of Traditional Animation applied to 3D Computer Animation." ACM SIGGRAPH Computer Graphics 21, no. 4 (July 1987): 35-44.

# Example: Timing

- Timing affects weight:
  - Light object move quickly
  - Heavier objects move slower

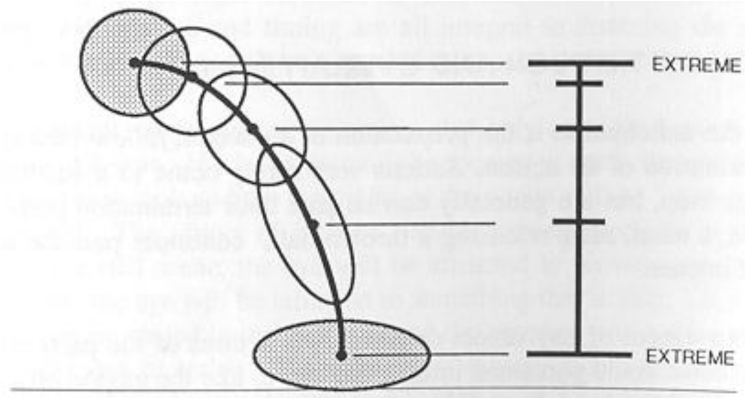


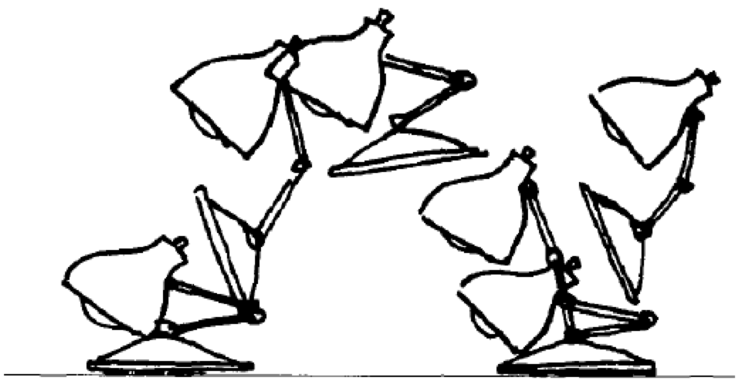
FIGURE 9. Timing chart for ball bounce.



- Timing completely changes the interpretation of the motion.

# Computer Animation

- How do we describe and generate motion of objects in the scene?





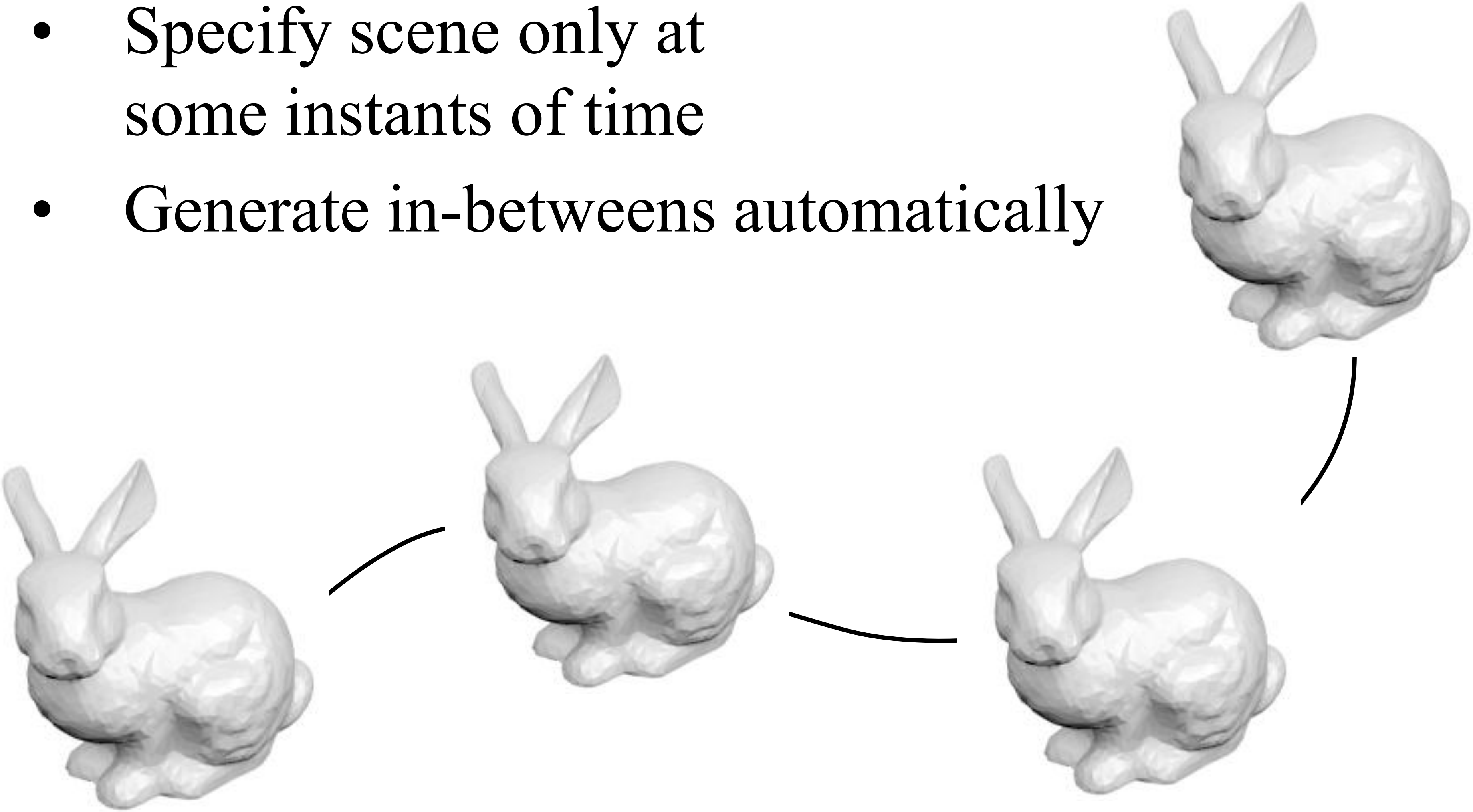
- Two very different contexts:
  - Production (offline)
    - Can be hardcoded, entire sequence know beforehand
  - Interactive (e.g. games, simulators)
    - Needs to react to user interaction, sequence not known

# Plan

- Types of Animation (overview)
  - Keyframing
  - Procedural
  - Physically-based
- Animation Controls
- Character Animation
  using skinning/enveloping

# Types of Animation: Keyframing

- Specify scene only at some instants of time
- Generate in-betweens automatically

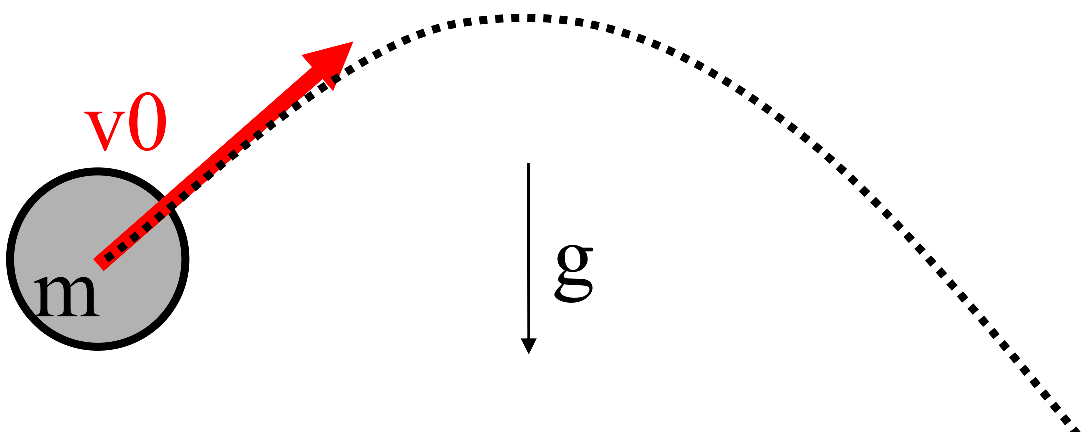


# Types of Animation: Procedural

- Describes the motion algorithmically
- Express animation as a function of small number of parameters
- Example
  - a clock/watch with second, minute and hour hands
  - express the clock motions in terms of a “seconds” variable
    - the clock is animated by changing this variable
- Another example: Grass in the wind, tree canopies, etc.

# Types of Animation: Physically-Based

- Assign physical properties to objects
  - Masses, forces, etc.
- Also procedural forces (like wind)
- Simulate physics by solving equations of motion
  - Rigid bodies, fluids, plastic deformation, etc.
- Realistic but difficult to control



# Another Example

- Physically-Based Character Animation
  - Specify keyframes, solve for physically valid motion that interpolates them by “spacetime optimization”

- Anthony C. Fang and Nancy S. Pollard, 2003. Efficient Synthesis of Physically Valid Human Motion, ACM Transactions on Graphics 22(3) 417-426, Proc. SIGGRAPH 2003.http://graphics.cs.cmu.edu/nsp/projects/spacetime/spacetime.html

# Plan

- Types of Animation (overview)
  - Keyframing
  - Procedural
  - Physically-based
- Animation Controls
- Character Animation
  using skinning/enveloping

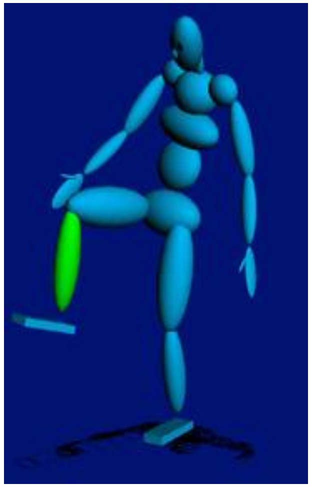
# Because we are Lazy...

- Animation is (usually) specified using some form of low-dimensional **controls** as opposed to remodeling the actual geometry for each frame.

Can you think of examples?

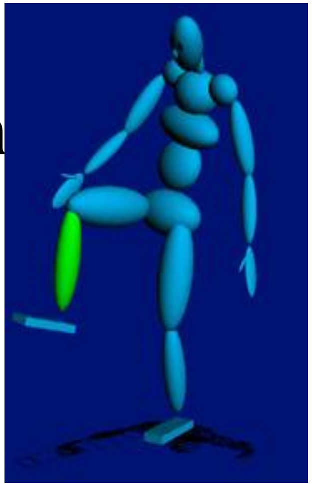
# Because we are Lazy...

- Animation is (usually) specified using some form of low-dimensional **controls** as opposed to remodeling the actual geometry for each frame.
  - Example: The joint angles (bone transformations) in a hierarchical character determine the pose
  - Example: A rigid motion is represented by changing the object-to-world transformation (rotation and translation).



# Because we are Lazy...

- Animation is (usually) specified using some form of low-dimensional **controls** as opposed to remodeling the actual geometry for each frame.
  - Example: The joint angles (bone transformations) in a hierarchical character determine the pose
  - Example: A rigid motion is represented by changing the object-to-world transformation (rotation and translation).



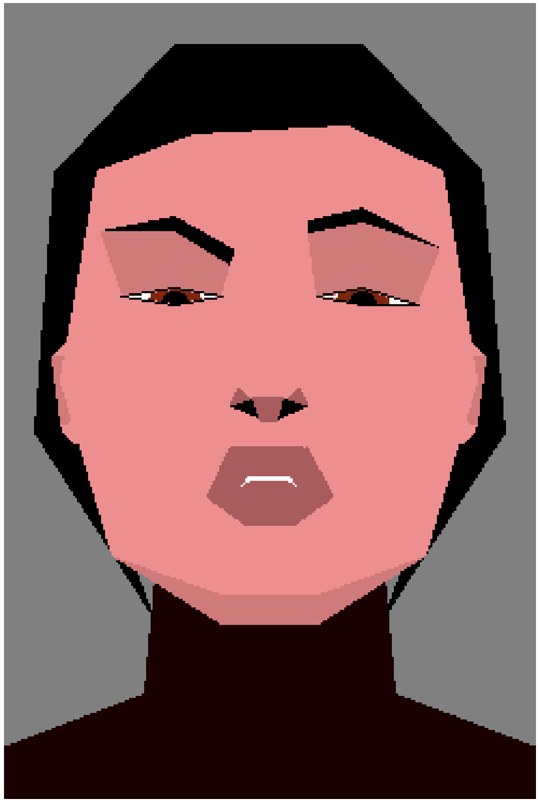


**"Blendshapes" are keyframes that are just snapshots of the entire geometry.**

# Example of Higher-Level Controls

- Ken Perlin's facial expression applet http://mrl.nyu.edu/~perlin/experiments/facedemo/

- Lower-level controls are mapped to semantically meaningful higher-level ones
  - "Frown/smile" etc.

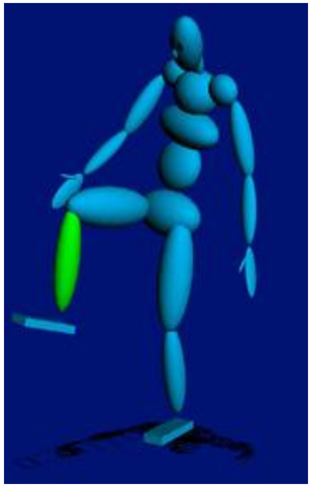


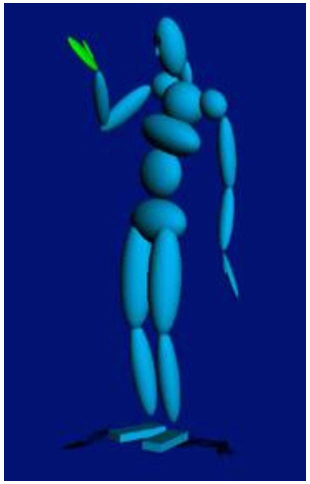
Building 3D models and their animation controls is a major component of every animation pipeline.

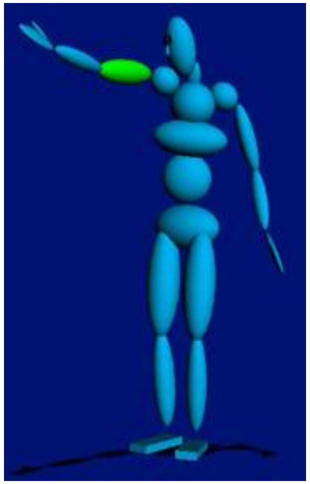
Building the controls is called “rigging”.

# Articulated Character Models

- Forward kinematics describes the positions of the body parts as a function of joint angles
  - Body parts are usually called “bones”
  - Angles are the low-dimensional control.
- Inverse kinematics specifies constraint locations for bones and solves for joint angles.





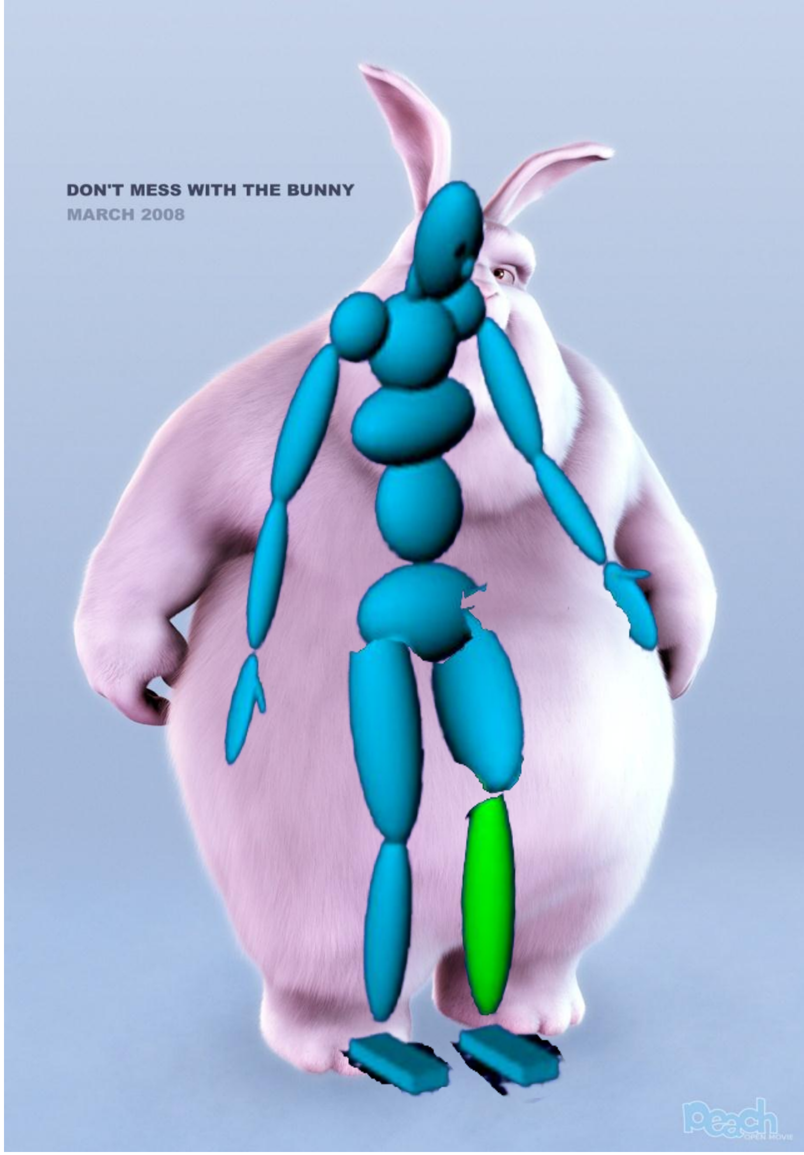


# Skinning Characters

- Embed a skeleton into a detailed character mesh

# Skinning Characters

- Embed a skeleton into a detailed character mesh
- Animate "bones"
  - Change the joint angles over time
  - Keyframing, procedural, etc.
- Bind skin vertices to bones
  - Animate skeleton, skin will move with it

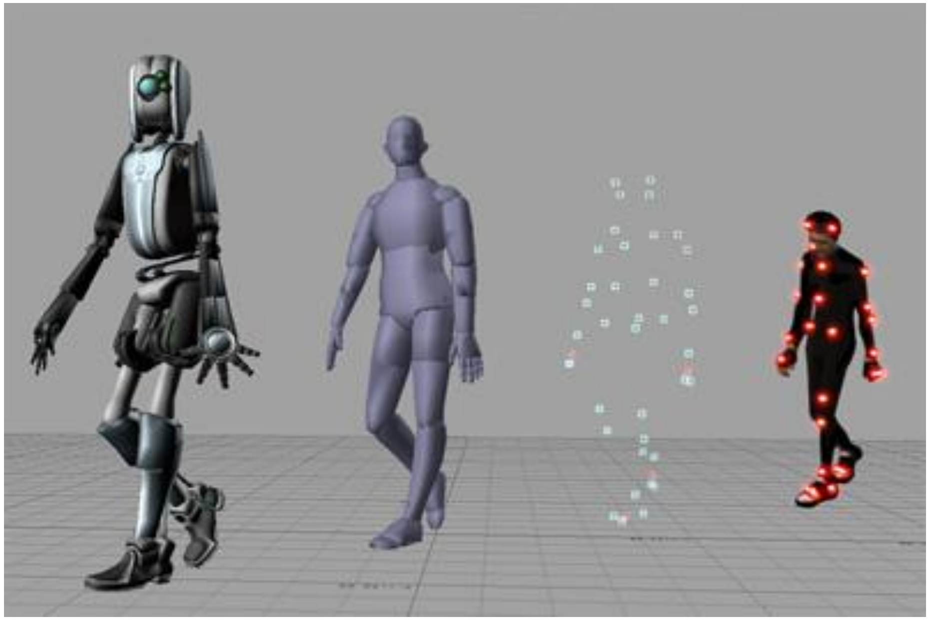


# Motion Capture

- Usually uses optical markers and multiple high-speed cameras
- Triangulate to get marker 3D position
  – (Again, structure from motion and projective geometry, i.e., homogeneous coordinates)
- Captures style, subtle nuances and realism
- But need ability to record someone

# Motion Capture

- Motion capture records 3D marker positions
  – But character is controlled using animation controls that affect bone transformations!
- Marker positions must be translated into character controls (“retargeting”)



# Questions?

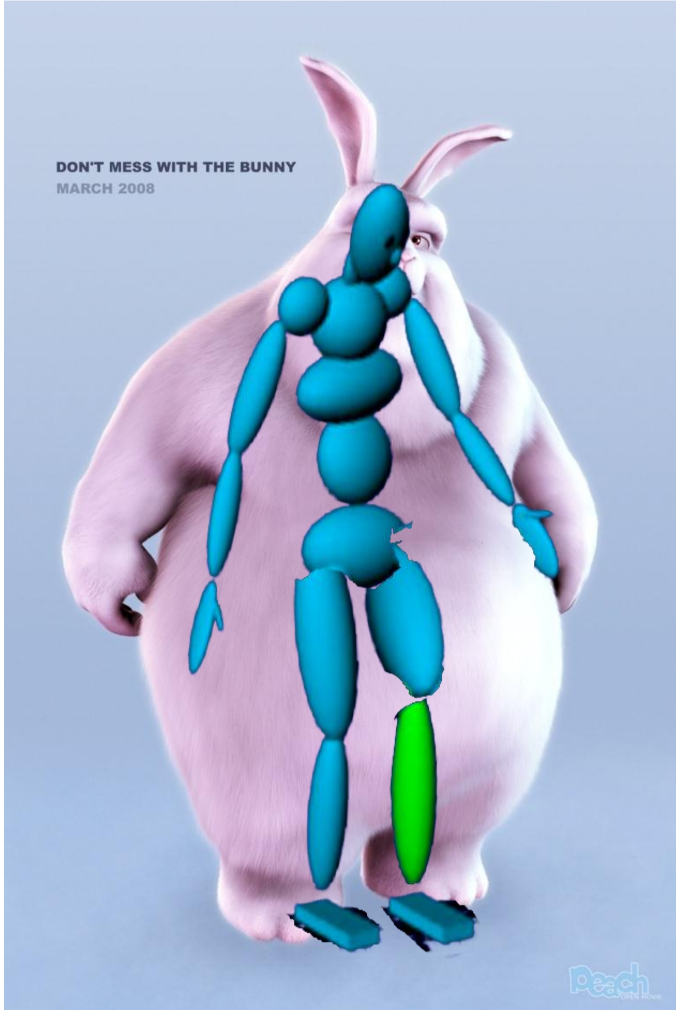
# Plan

- Types of Animation (overview)
  - Keyframing
  - Procedural
  - Physically-based
- Animation Controls
- Character Animation
  using skinning/enveloping

# Skinning/Enveloping

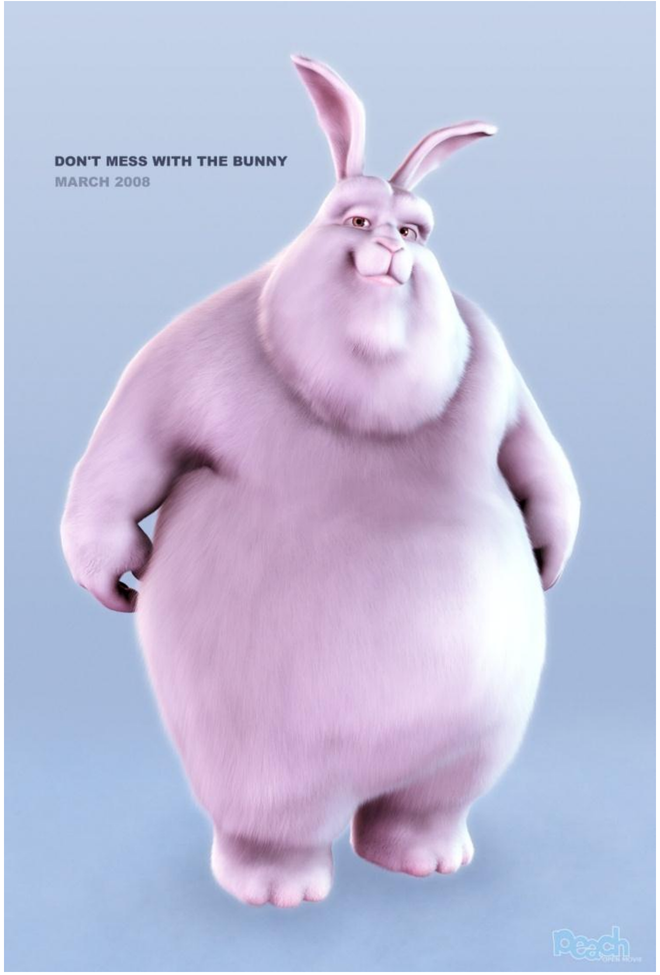
# Skinning

- We know how to animate a bone hierarchy
  - Change the joint angles, i.e., bone transformations, over time (keyframing)



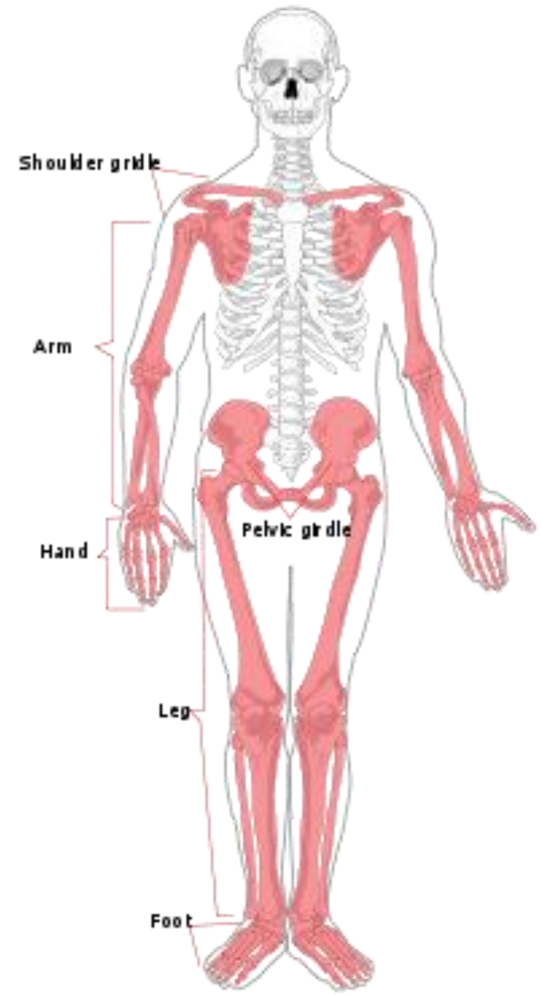
# Skinning

- We know how to animate a bone hierarchy
  – Change the joint angles, i.e., bone transformations, over time (keyframing)
- Embed a skeleton into a detailed character mesh
- Bind skin vertices to bones
  – Animate skeleton, skin will move with it
  – But how?



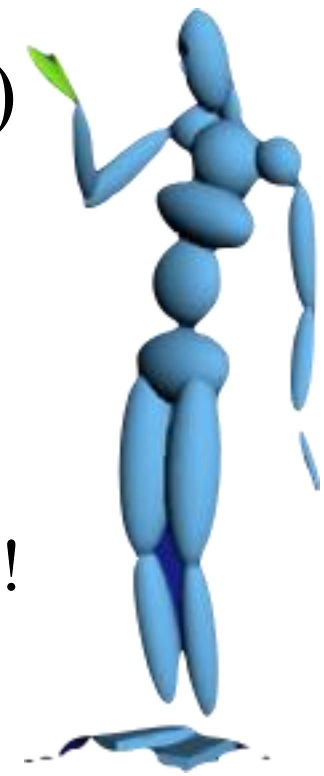
# Skinning/Enveloping

- Need to infer how skin deforms from bone transformations.
- Most popular technique: Skeletal Subspace Deformation (SSD), or simply Skinning
  - Other aliases
    - vertex blending
    - matrix palette skinning
    - linear blend skinning

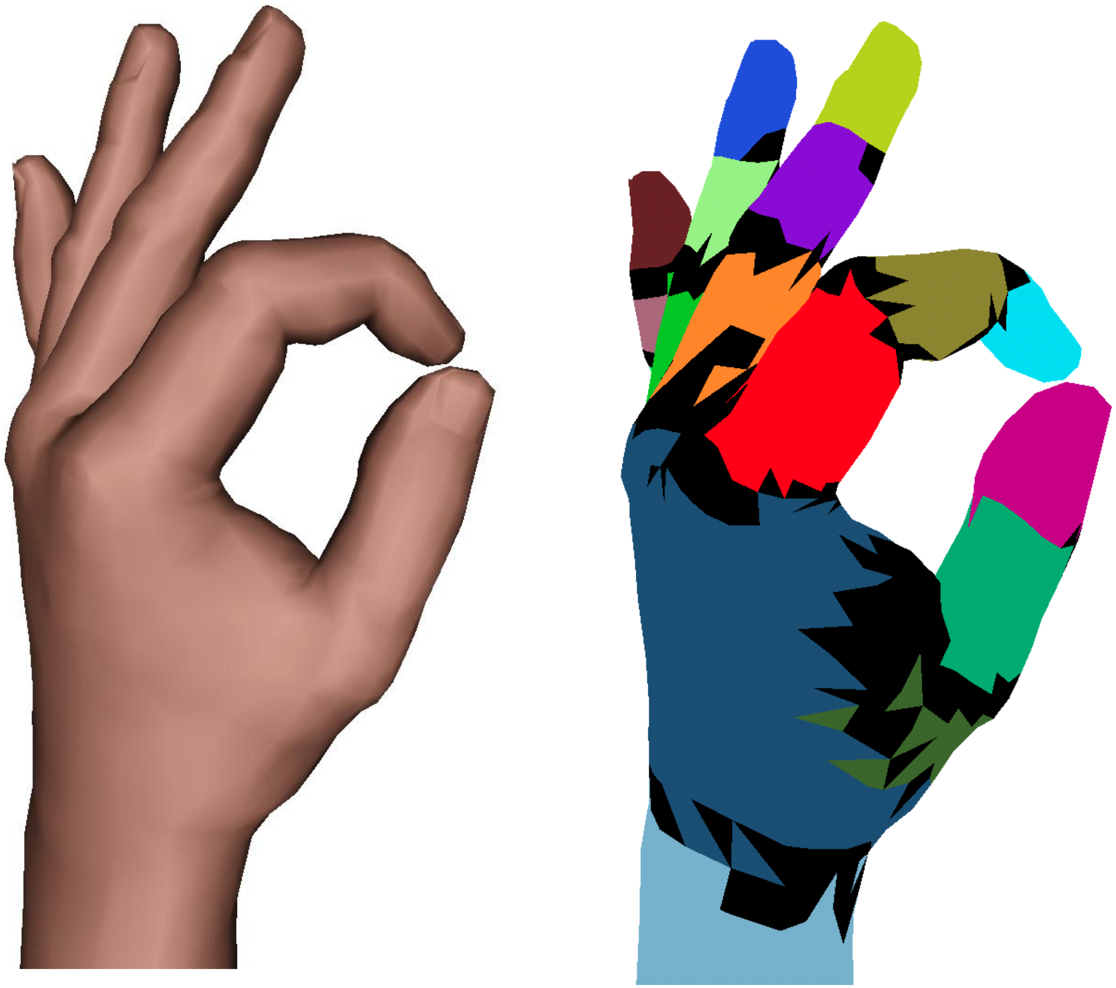


# SSD / Skinning

- Each bone has a deformation of the space around it (rotation, translation)
  - What if we attach each vertex of the skin to a single bone?
    - Skin will be rigid, except at joints where it will stretch badly
  - Let's attach a vertex to many bones at once!
    - In the middle of a limb, the skin points follow the bone rotation (near-rigidly)
    - At a joint, skin is deformed according to a "weighted combination" of the bones



# Example



Colored triangles are attached to 1 bone

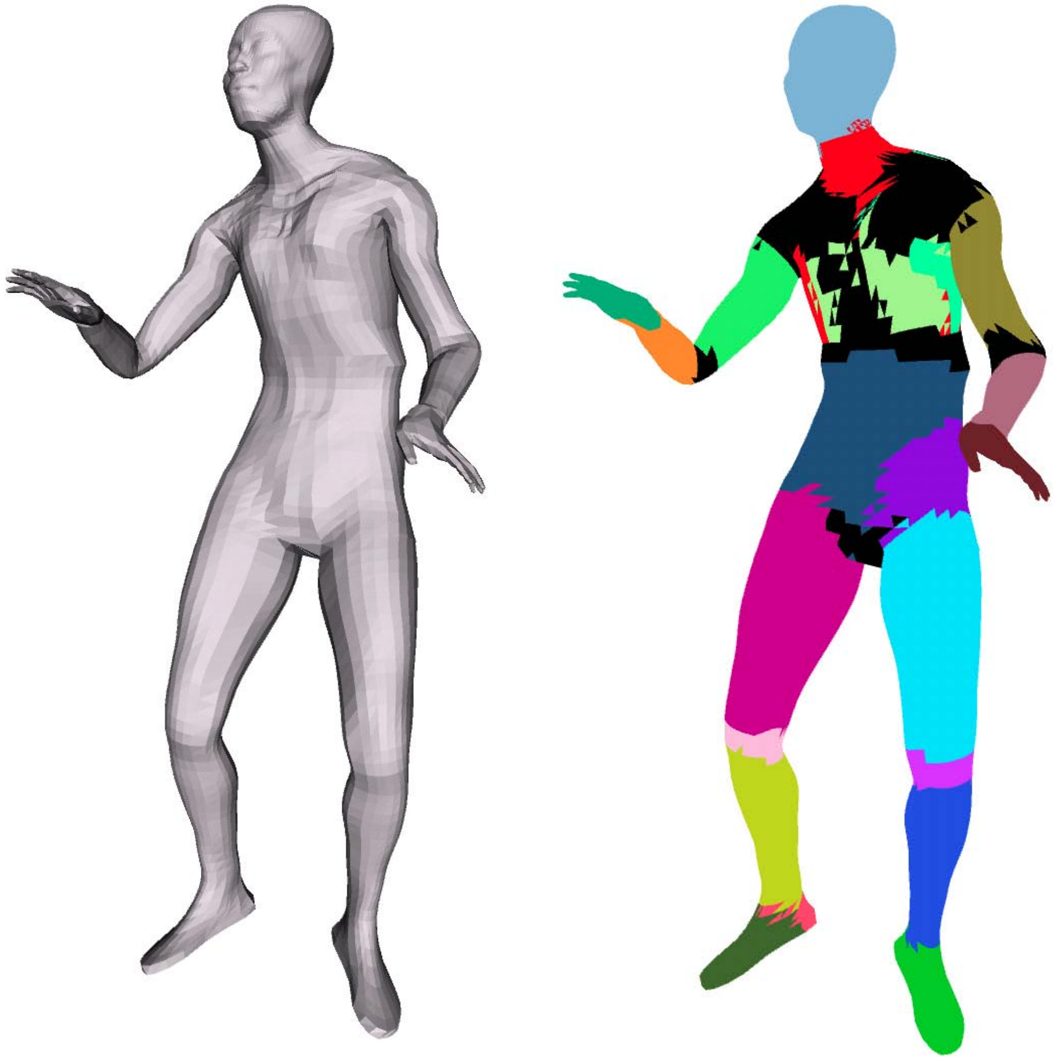
Black triangles are attached to more than 1

Note how they are near joints



James & Twigg, Skinning Mesh Animations, 2005, used with permission from ACM, Inc.

# Example



Colored triangles are attached to 1 bone

Black triangles are attached to more than 1

Note how they are near joints



James & Twigg, Skinning Mesh Animations.

# Vertex Weights

- We'll assign a weight $w_{ij}$ for each vertex $\mathbf{p}_i$ for each bone $\mathbf{B}_j$.
  - "How much vertex *i* should move with bone *j*"
  - $w_{ij} = 1$ means $\mathbf{p}_i$ is rigidly attached to bone *j*.

# Vertex Weights

- We'll assign a weight $w_{ij}$ for each vertex $\mathbf{p}_i$ for each bone $\mathbf{B}_j$.
  - "How much vertex $i$ should move with bone $j$"
  - $w_{ij} = 1$ means $\mathbf{p}_i$ is rigidly attached to bone $j$.

Figure 8: Top: heat equilibrium for two bones. Bottom: the result of rotating the right bone with the heat-based attachment



From Automatic Rigging and Animation of 3D Characters.

# Vertex Weights

- We'll assign a weight $w_{ij}$ for each vertex $\mathbf{p}_i$ for each bone $\mathbf{B}_j$.
  - "How much vertex *i* should move with bone *j*"
  - $w_{ij} = 1$ means $\mathbf{p}_i$ is rigidly attached to bone *j*.
- Weight properties
  - Usually want weights to be non-negative

# Vertex Weights

- We'll assign a weight $w_{ij}$ for each vertex $\mathbf{p}_i$ for each bone $\mathbf{B}_j$.
  - "How much vertex $i$ should move with bone $j$"
  - $w_{ij} = 1$ means $\mathbf{p}_i$ is rigidly attached to bone $j$.
- Weight properties
  - Usually want weights to be non-negative
  - Also, want the sum over all bones to be 1 for each vertex

# Vertex Weights cont'd

- We'll assign a weight $w_{ij}$ for each vertex $\mathbf{p}_i$ for each bone $\mathbf{B}_j$.
  - "How much vertex *i* should move with bone *j*"
  - $w_{ij} = 1$ means $\mathbf{p}_i$ is rigidly attached to bone *j*.
- We'll limit the number of bones *N* that can influence a single vertex
  - *N*=4 bones/vertex is a usual choice
  - **Why?**

# Vertex Weights cont'd

- We'll assign a weight $w_{ij}$ for each vertex $\mathbf{p}_i$ for each bone $\mathbf{B}_j$.
  - "How much vertex *i* should move with bone *j*"
  - $w_{ij} = 1$ means $\mathbf{p}_i$ is rigidly attached to bone *j*.
- We'll limit the number of bones *N* that can influence a single vertex
  - $N$=4 bones/vertex is a usual choice
  - Why? You most often don't need very many.
  - Also, storage space is an issue.
  - In practice, we'll store *N* (bone index *j*, weight $w_{ij}$) pairs per vertex.

# How to compute vertex positions?

# Linear Blend Skinning

- **Basic Idea 1**: Transform each vertex **pi** with each bone as if it was tied to it rigidly.

# Linear Blend Skinning

- **Basic Idea 1**: Transform each vertex **pi** with each bone as if it was tied to it rigidly.
- **Basic Idea 2**: Then blend the results using the weights.

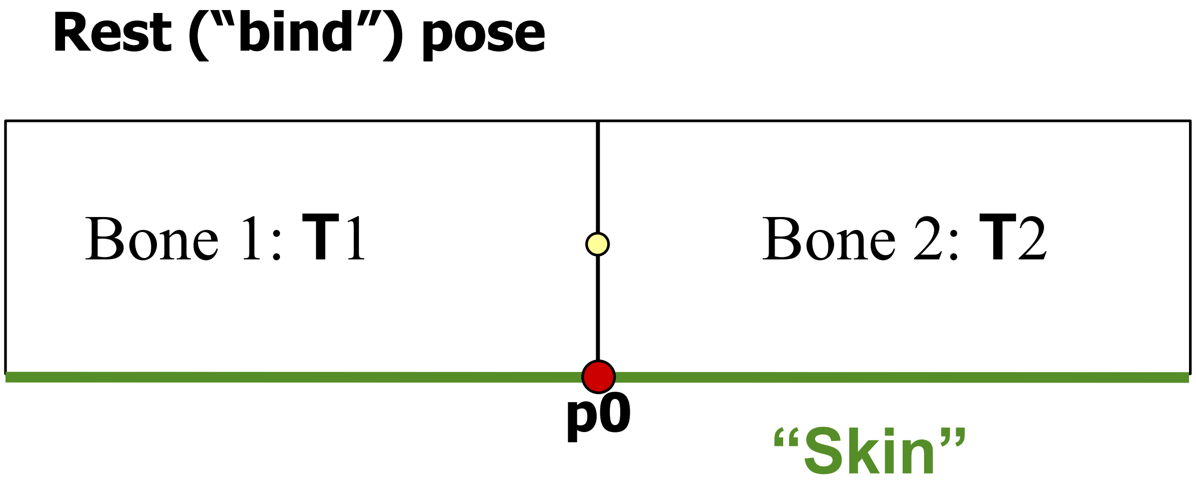
# Computing Vertex Positions

- **Basic Idea 1**: Transform each vertex **pi** with each bone as if it was tied to it rigidly.
- **Basic Idea 2**: Then blend the results using the weights.

"
$$\boxed{\begin{gathered} \boldsymbol{p}'_{ij} = \boldsymbol{T}_j\, \boldsymbol{p}_i \\ \boldsymbol{p}'_i = \sum_j w_{ij} \boldsymbol{p}'_{ij} \end{gathered}}$$
"

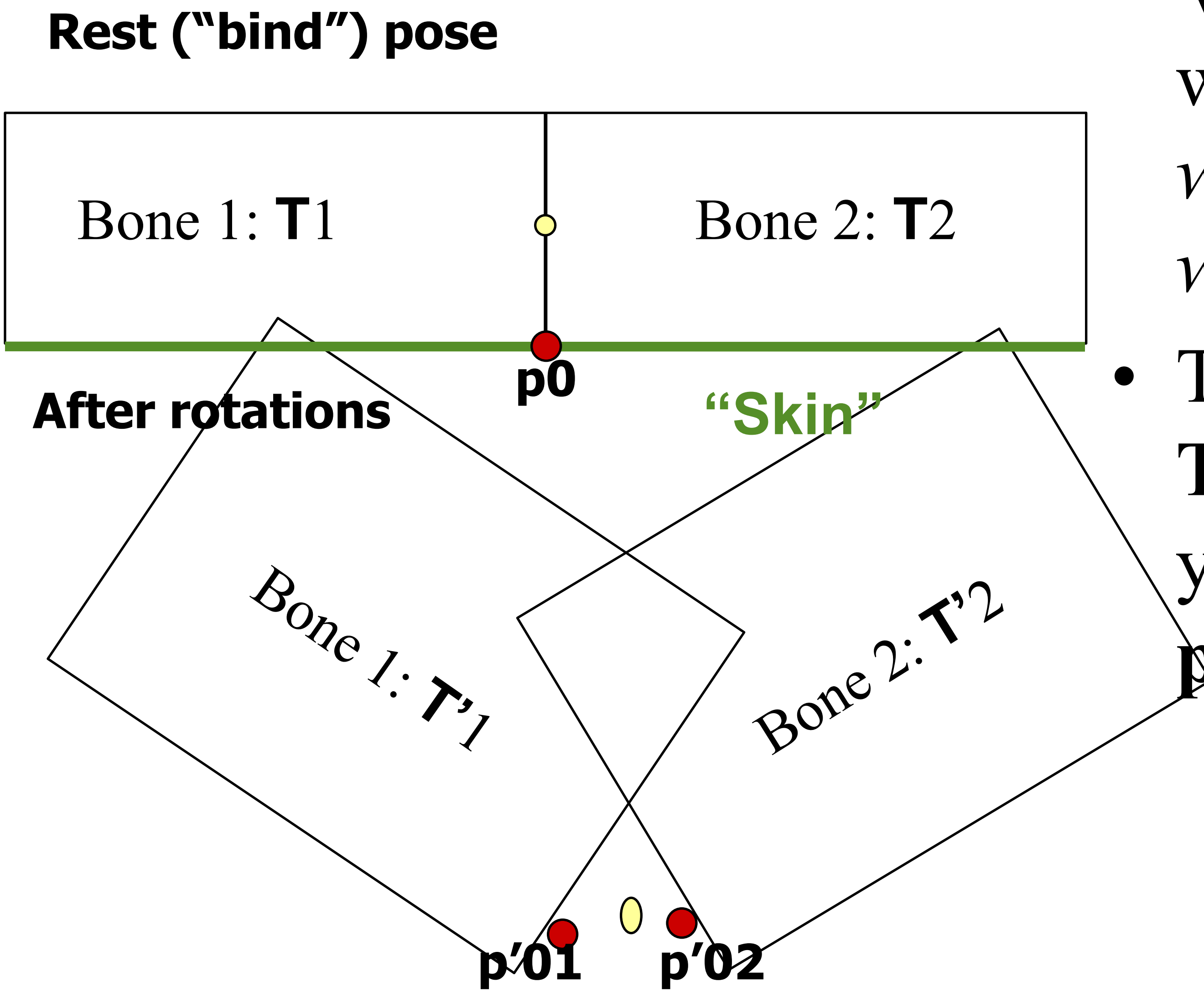
**p**'ij is the vertex i transformed using bone j.
**T**j is the current transformation of bone j.
**p**'i is the new skinned position of vertex i.

# Computing Vertex Positions



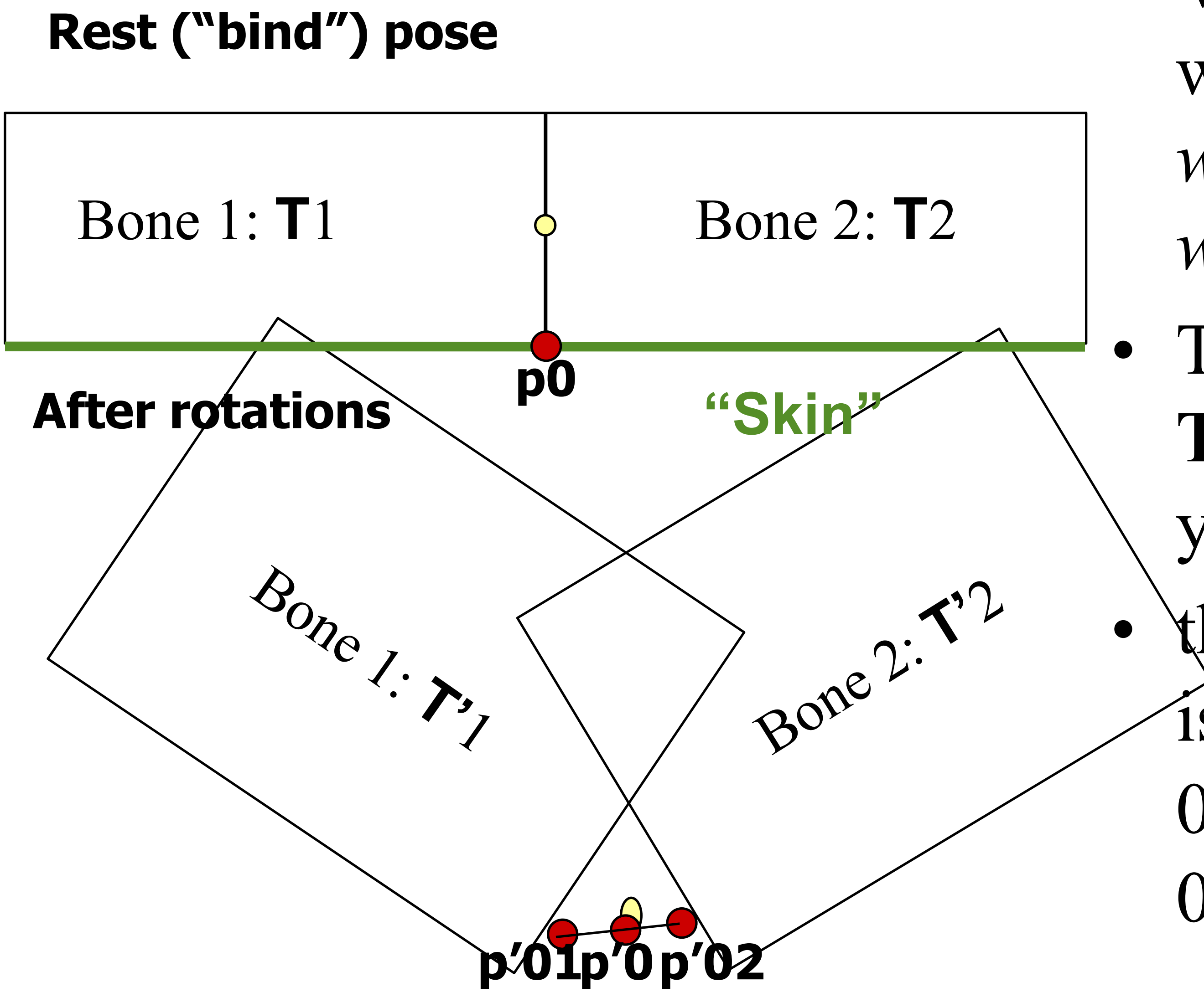

- Vertex **p**0 has weights $w01=0.5$, $w02=0.5$

# Computing Vertex Positions



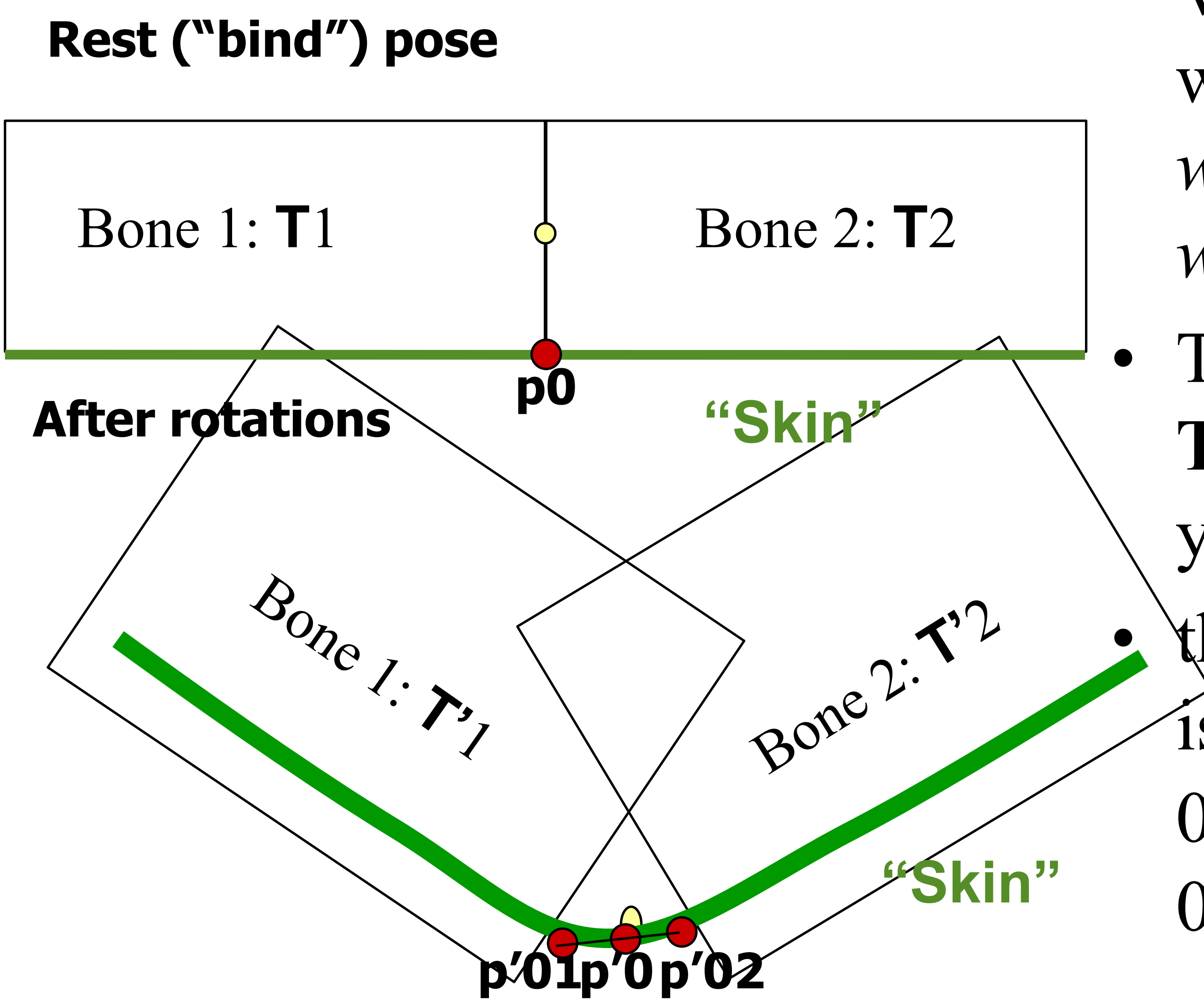

- Vertex **p**0 has weights $w01$=0.5, $w02$=0.5
- Transform by **T'**1 and **T'**2 yields **p**'01, **p**'02

# Computing Vertex Positions



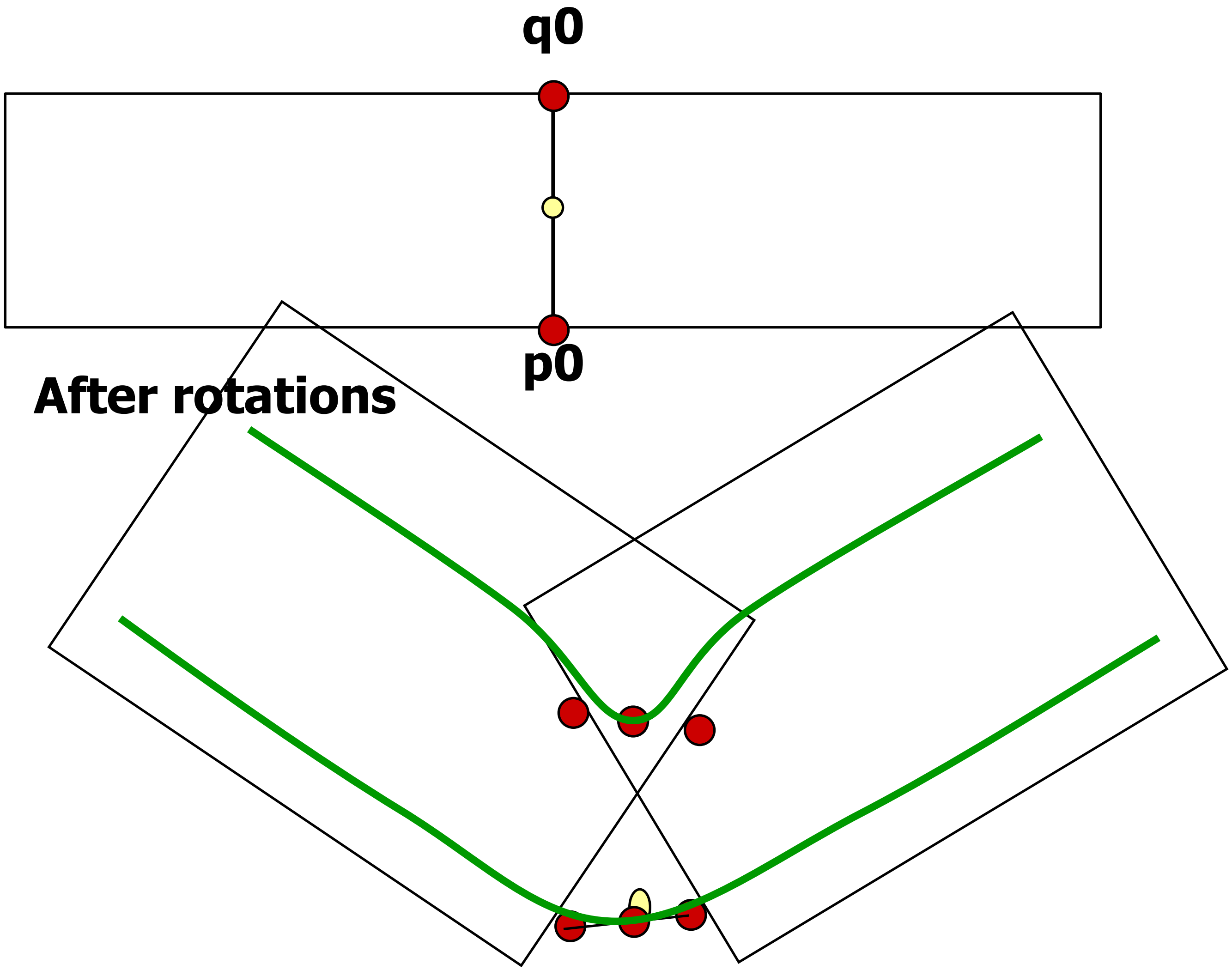

- Vertex **p**0 has weights *w*01=0.5, *w*02=0.5
- Transform by **T**'1 and **T**'2 yields **p**'01, **p**'02
- the new position is **p**'0= 0.5***p'**1 + 0.5***p**'2

# Computing Vertex Positions



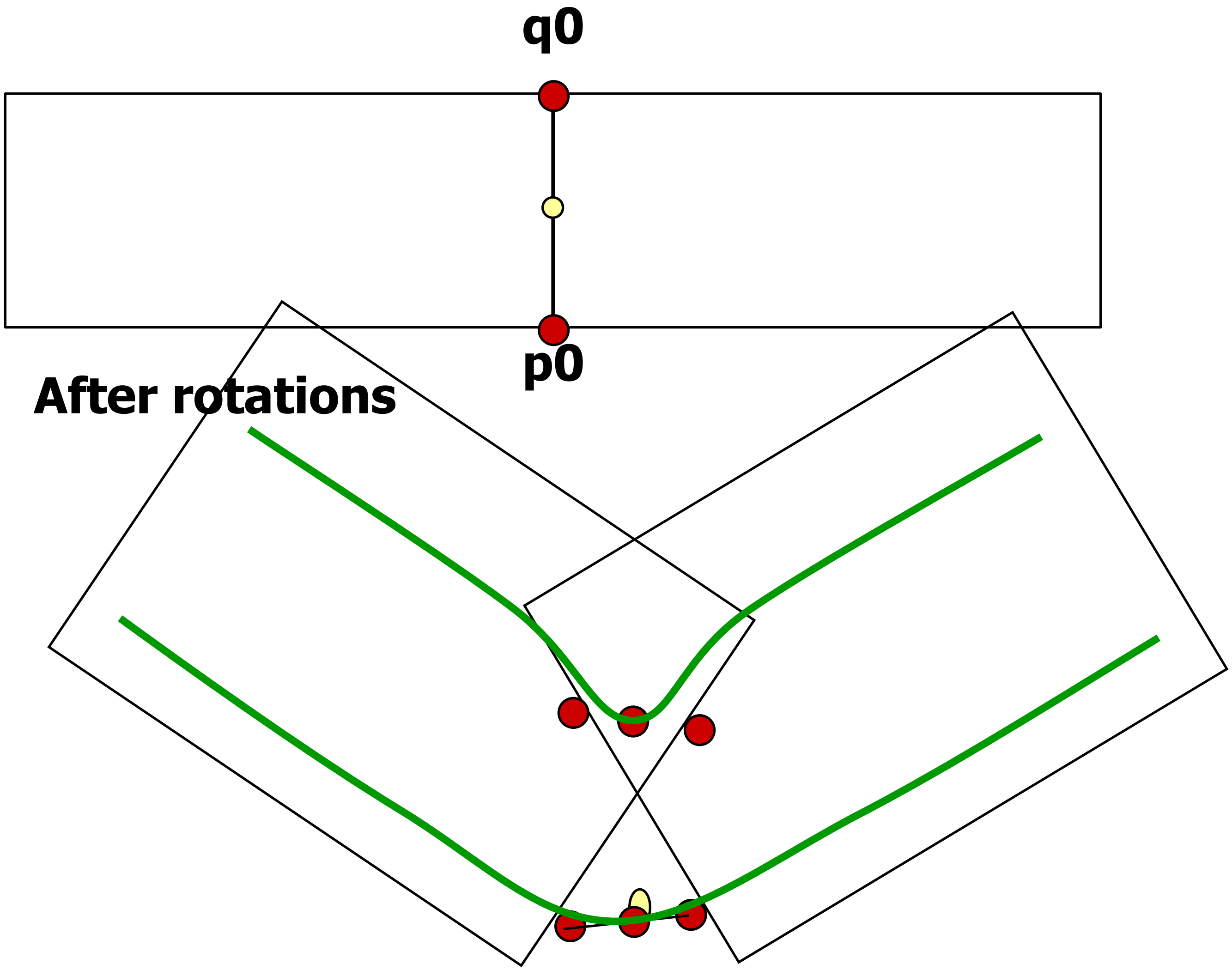

- Vertex **p**0 has weights *w*01=0.5, *w*02=0.5
- Transform by **T**'1 and **T**'2 yields **p**'01, **p**'02
- the new position is **p**'0= 0.5***p'**1 + 0.5***p**'2

# SSD is Not Perfect



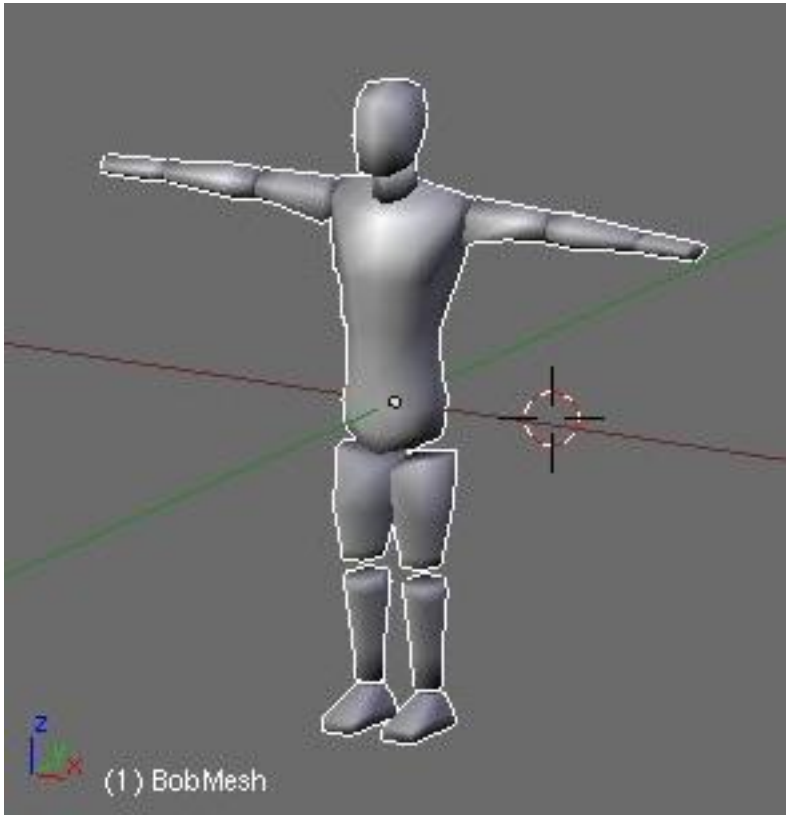
# SSD is Not Perfect

Questions?



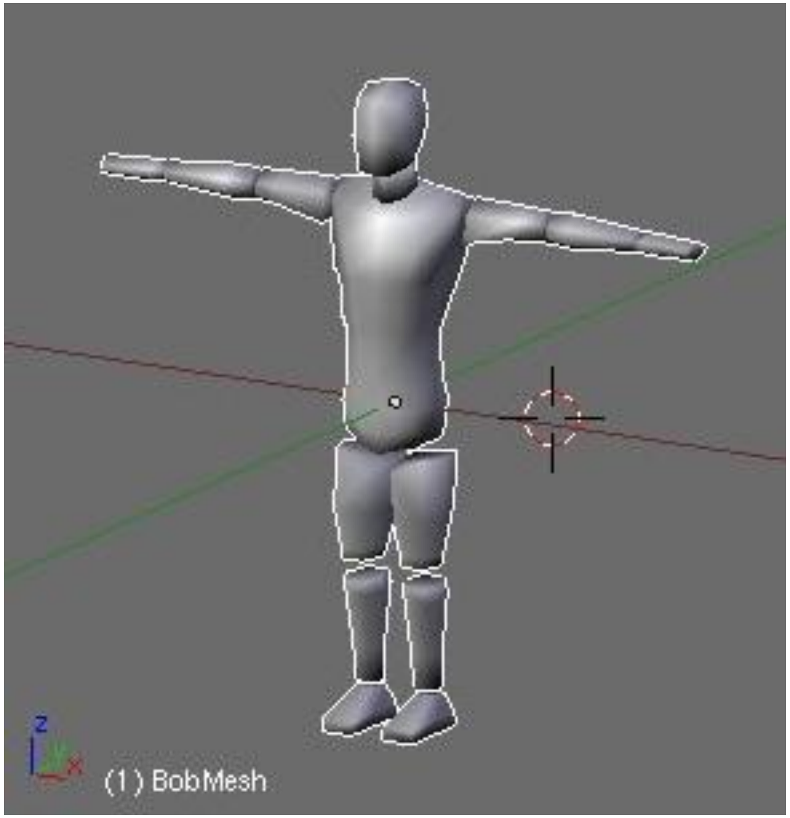
# Bind Pose

- We are given a skeleton and a skin mesh in a default pose
  - Called “bind pose”
  - Undeformed vertices **pi** are given in the object space of the skin
    - a “global” coordinate system, no hierarchy



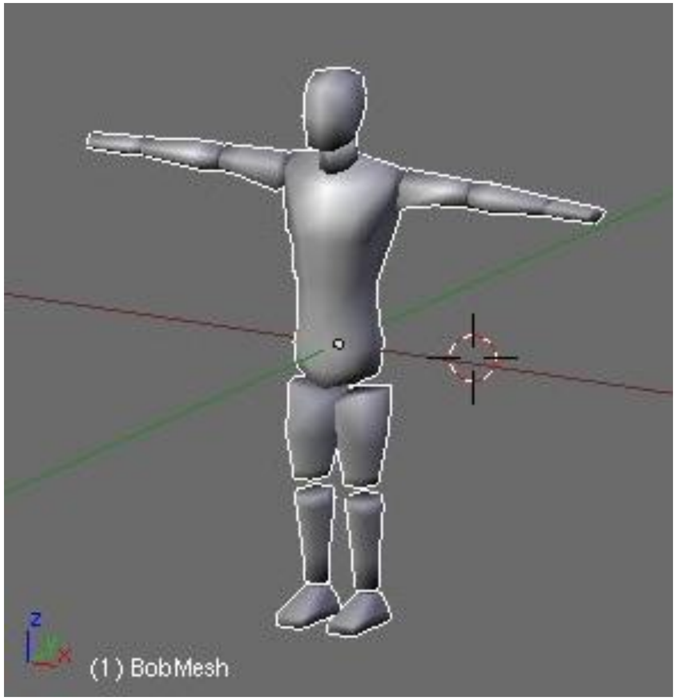
# Bind Pose

- We are given a skeleton and a skin mesh in a default pose
  - Called "bind pose"
  - Undeformed vertices $\mathbf{p}i$ are given in the object space of the skin
- Previously we conveniently forgot that in order for $\mathbf{p}'ij = \mathbf{T}j\ \mathbf{p}i$ to make sense, coordinate systems must match up.



# Coordinate Systems

- Undeformed vertices **pi** are given in the object space of the skin
- **Tj** is in local bone coordinate system
  - according to skeleton hierarchy

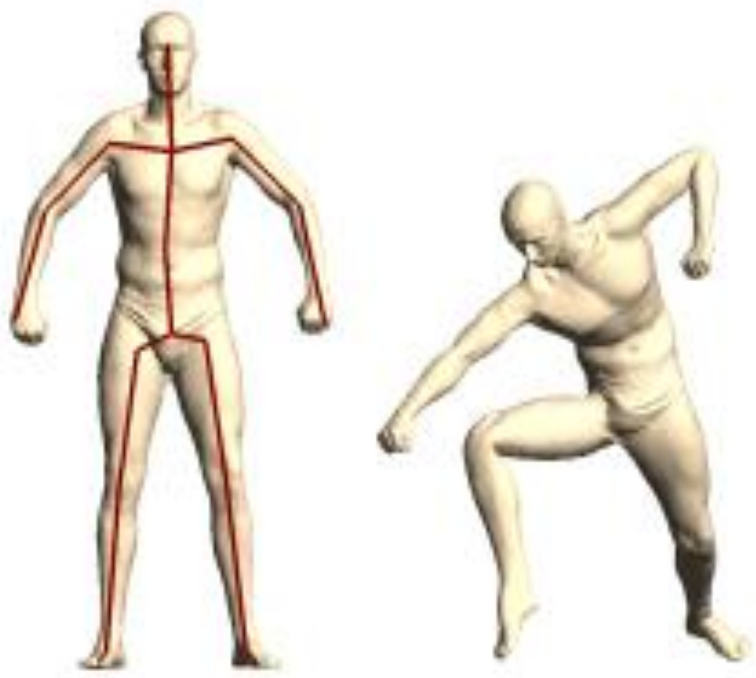


# Bind Pose cont'd

- In the rigging phase, we line the skeleton up with the undeformed skin.
  - This gives some "rest pose" bone transformations **B**j from local bone coordinates to global
  - **B**j concatenates all hierarchy matrices from node j up to the root

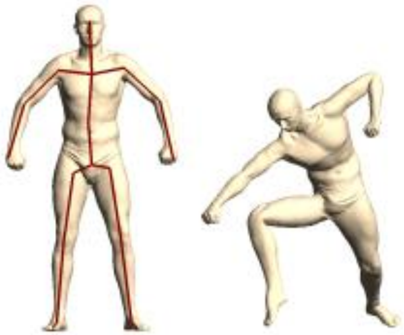
# Bind Pose cont'd

- When we animate the model, the bone transformations **Tj** change.



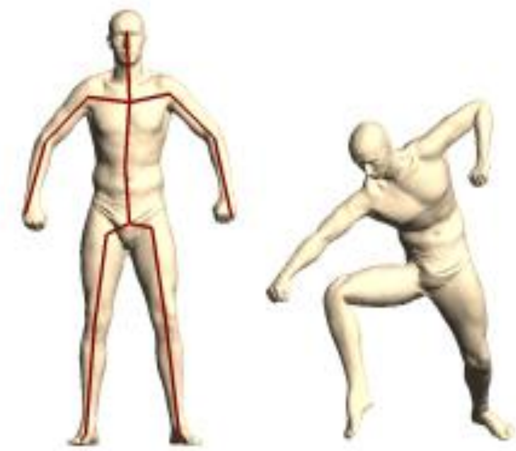
# Bind Pose cont'd

- When we animate the model, the bone transformations **Tj** change.
  - What is **Tj**? It maps from the local coordinate system of bone *j* to world space.
  - again, concatenates hierarchy matrices



# Bind Pose cont'd

- When we animate the model, the bone transformations **Tj** change.
  - What is **Tj**? It maps from the local coordinate system of bone *j* to world space.





- To be able to deform **pi** according to **Tj**, we must first express **pi** in the local coordinate system of bone j.
  - This is where the bind pose bone transformations **Bj** come in.

# Bind Pose cont'd

- To be able to deform **p**i according to **T**j, we must first express **p**i in the local coordinate system of bone j.
  - This is where the bind pose bone transformations **B**j come in.

$$\boxed{\boldsymbol{p}'_{ij} = \boldsymbol{T}_j \boldsymbol{B}_j^{-1} \boldsymbol{p}_i}$$

This maps **p**i from bind pose to the local coordinate system of bone j using **B**-1j, and then to world space using **T**j.

# Bind Pose cont'd

$$\boxed{\boldsymbol{p}'_{ij} = \boldsymbol{T}_j \boldsymbol{B}_j^{-1} \boldsymbol{p}_i}$$

This maps **p**i from bind pose to the local coordinate system of bone j using **B**-1j, and then to world space using **T**j.

What is **T**j **B**-1j? It is the relative change between the bone transformations between the current and the bind pose.

# Bind Pose cont'd

**What is the transformation when the model is still in bind pose?**

$$\boxed{\mathbf{p}'_{ij} = \mathbf{T}_j \mathbf{B}_j^{-1} \mathbf{p}_i}$$

This maps **p**i from bind pose to the local coordinate system of bone j using **B**-1j, and then to world space using **T**j.

What is **T**j **B**-1j? It is the relative change between the bone transformations between the current and the bind pose.

# Bind Pose cont'd

**What is the transformation when the model is still in bind pose?**

**The identity!**

$$\boxed{\boldsymbol{p}'_{ij} = \boldsymbol{T}_j \boldsymbol{B}_j^{-1} \boldsymbol{p}_i}$$

This maps **p**i from bind pose to the local coordinate system of bone j using **B**-1j, and then to world space using **T**j.

What is **T**j **B**-1j? It is the relative change between the bone transformations between the current and the bind pose.

# Bind Pose cont'd

**What is the transformation when the model is still in bind pose?**

**The identity!**

$$\boxed{\boldsymbol{p}'_{ij} = \boldsymbol{T}_j \boldsymbol{B}_j^{-1} \boldsymbol{p}_i}$$

This maps **p**i from bind pose to the local coordinate system of bone j using **B**-1j, and then to world space using **T**j.

What is **T**j **B**-1j? It is the relative change between the bone transformations between the current and the bind pose.

Questions?

# Bind Pose & Weights

- We then figure out the vertex weights $w_{ij}$.
  - How? Usually paint by hand!
  - We'll look at much cooler methods in a while.

Figure 8: Top: heat equilibrium for two bones. Bottom: the result of rotating the right bone with the heat-based attachment



From Automatic Rigging and Animation of 3D Characters.

# Skinning Pseudocode

- Do the usual forward kinematics
  - get a matrix **T**j(t) per bone
    (full transformation from local to world)
- For each skin vertex **p**i

$$\boldsymbol{p}'_i = \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \boldsymbol{p}_i$$

# Skinning Pseudocode

- Do the usual forward kinematics
  - get a matrix **Tj**(t) per bone
    (full transformation from local to world)
- For each skin vertex **pi**

$$\boldsymbol{p}_i' = \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \boldsymbol{p}_i$$

**Do you remember how to treat normals?**

# Skinning Pseudocode

- Do the usual forward kinematics
  - get a matrix **Tj**(t) per bone
    (full transformation from local to world)
- For each skin vertex **pi**

$$\boldsymbol{p}_i' = \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \boldsymbol{p}_i$$

- Inverse transpose for normals!

$$\boldsymbol{n}_i' = \left( \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \right)^{-\mathrm{T}} \boldsymbol{n}_i$$

# Skinning Pseudocode

- Do the usual forward kinematics
- For each skin vertex **p**i

$$\boldsymbol{p}'_i = \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \boldsymbol{p}_i$$

- Note that the weights & bind pose vertices are constant over time
  - Only matrices change (small number of them, one per bone)
  - This enables implementation on GPU “vertex shaders” (little information to update for each frame)

# Hmmh...

- This is what we do to get deformed positions

$$\boldsymbol{p}_i' = \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \boldsymbol{p}_i$$

# Hmmh...

- This is what we do to get deformed positions

$$\boldsymbol{p}_i' = \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \boldsymbol{p}_i$$

- But wait...

$$\boldsymbol{p}_i' = \left( \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \right) \boldsymbol{p}_i$$

# Hmmh...

- This is what we do to get deformed positions

$$\boldsymbol{p}_i' = \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \boldsymbol{p}_i$$

- But wait...

$$\boldsymbol{p}_i' = \left( \sum_j w_{ij} \boldsymbol{T}_j(t) \boldsymbol{B}_j^{-1} \right) \boldsymbol{p}_i$$

- Rotations are not handled correctly (!!!)

# Indeed... Limitations

- Rotations really need to be combined differently (quaternions!)

Figure 2: The 'collapsing elbow' in action, c.f. Figure 1.

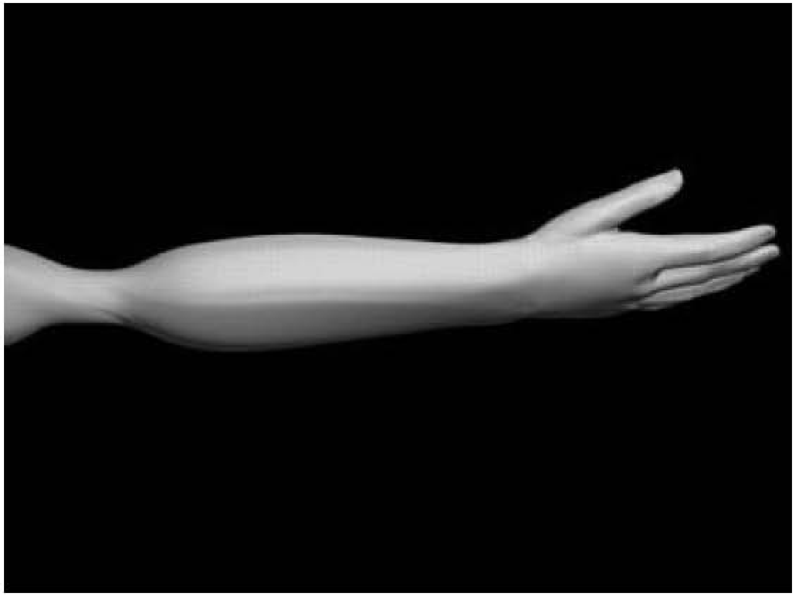
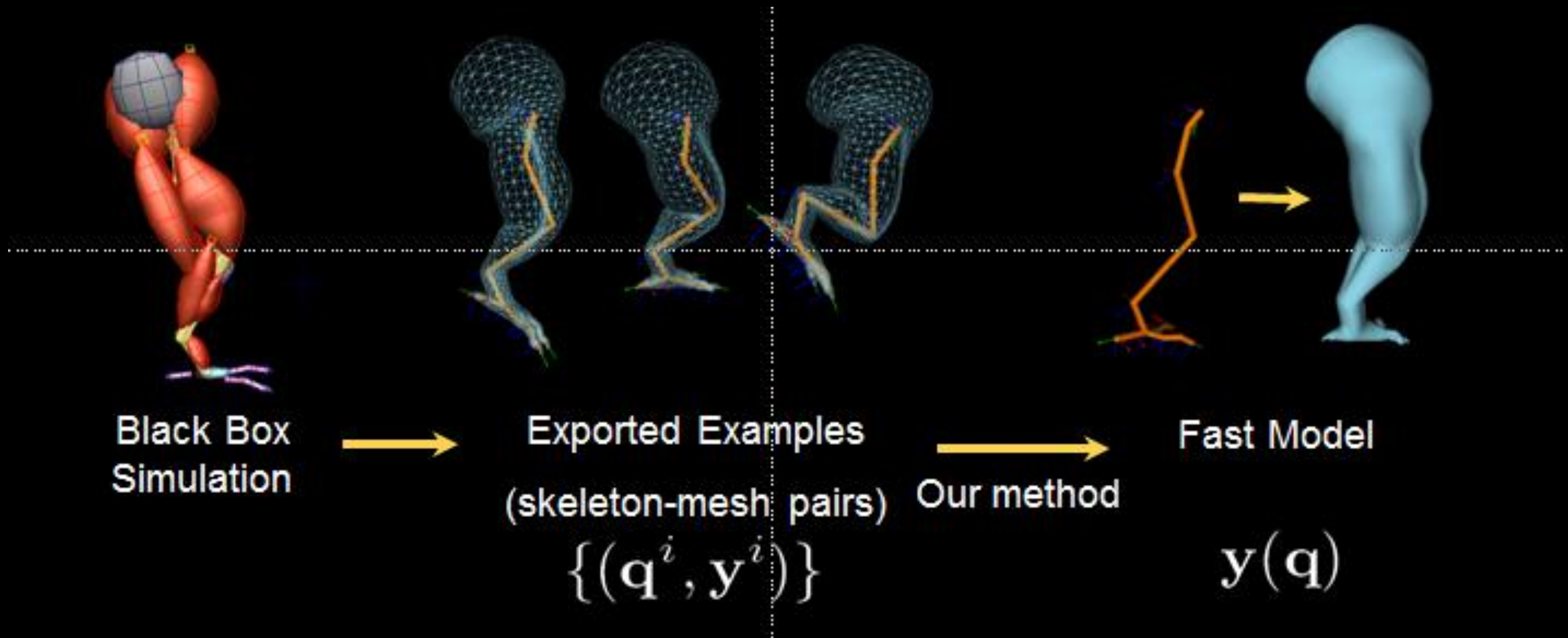


Figure 3: The forearm in the 'twist' pose, as in turning a door handle, computed by SSD. As the twist approaches 180° the arm collapses.



- From: Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation, J. P. Lewis, Matt Cordner, Nickson Fong

# Real-time enveloping with rotational regression
# Wang, Pulli, Popovic
# We learn a fast model from exported examples.



# Figuring out the Weights

- Usual approach: Paint them on the skin.
- Can also find them by optimization from example poses and deformed skins.
  - Wang & Phillips, SCA 2002

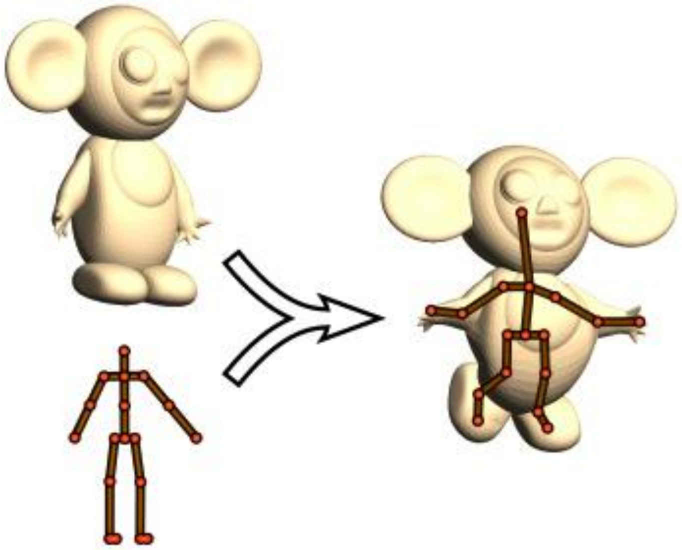
Figure 8: Top: heat equilibrium for two bones. Bottom: the result of rotating the right bone with the heat-based attachment



From Automatic Rigging and Animation of 3D Characters.

# Super Cool: Automatic Rigging

- When you just have some reference skeleton animation (perhaps from motion capture) and a skin mesh, figure out the bone transformations and vertex weights!
- Ilya Baran, Jovan Popovic: Automatic Rigging and Animation of 3D Characters, SIGGRAPH 2007
-



# Super Cool: Automatic Rigging

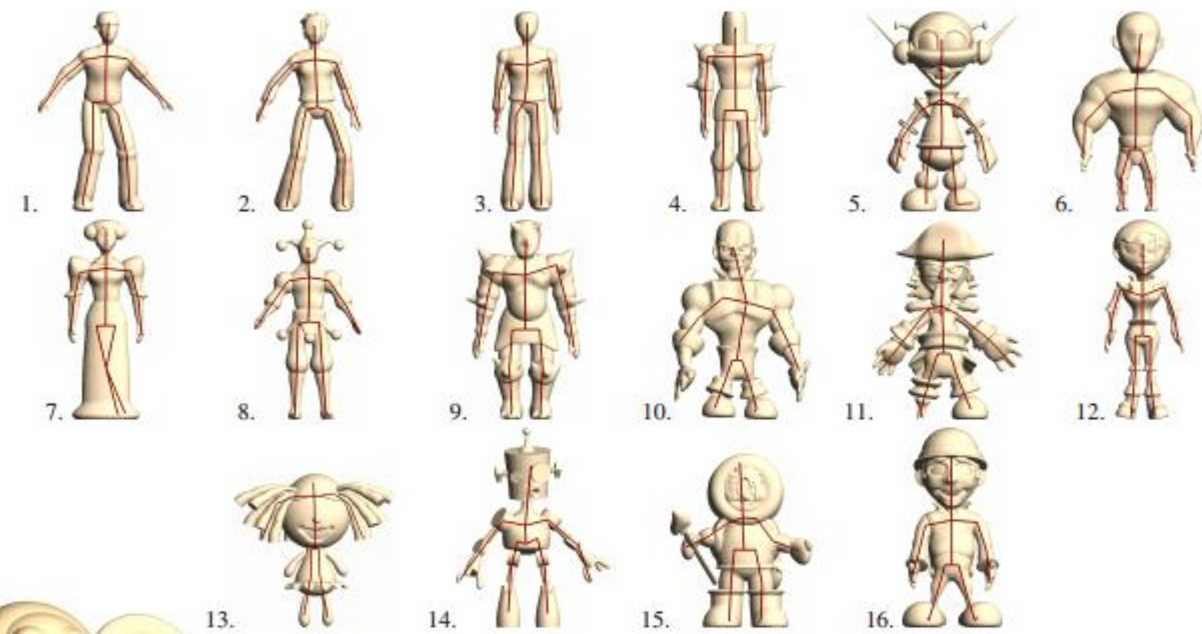
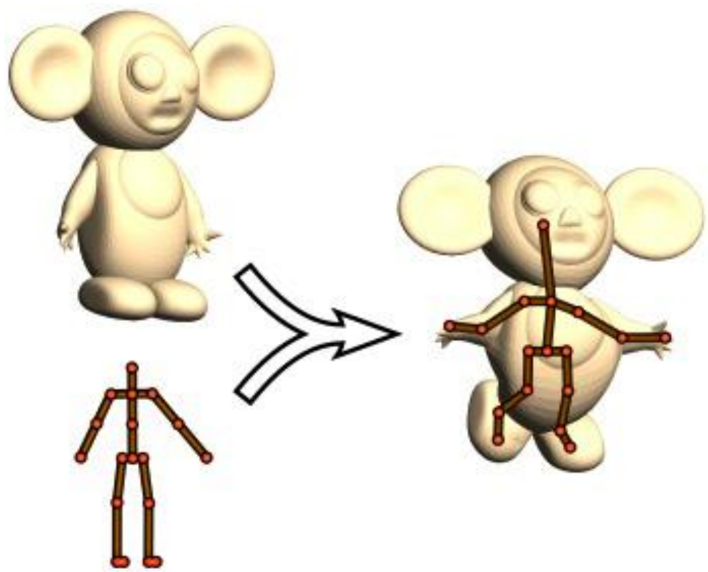



Figure 9: Test Results for Skeleton Embedding





From <u>Automatic Rigging and Animation of 3D Characters</u> by Baran and Popovic, used with permission from ACM, Inc.

# The Other Direction

## Skinning Mesh Animations

Doug L. James Christopher D. Twigg

Carnegie Mellon University

igure 1: **Stampede!** *Ten thousand skinned mesh animations (SMAs) synthesized in graphics hardware at interactive rates. All SMAs are eformed using only traditional matrix palette skinning with well-chosen nonrigid bone transforms. Distant SMAs are simplified.*



From Skinning Mesh Animations.

# That's All for Today!

- Further reading
  - http://www.okino.com/conv/skinning.htm
- Take a look at any video game – basically all the characters are animated using SSD/skinning.

6.837 Computer Graphics
Fall 2012