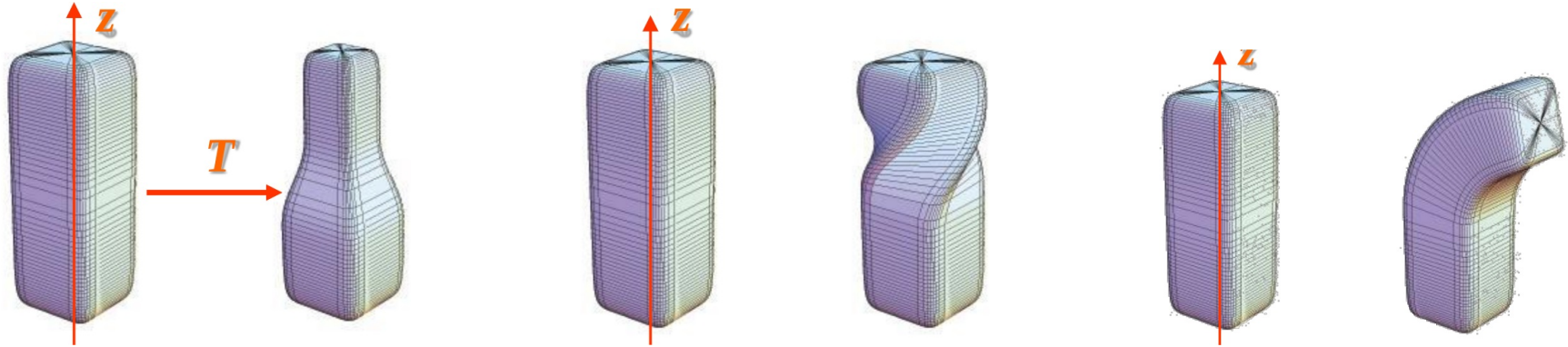


Character Animation

How to deform/animate character

We can handle explicitly some limited surface/space deformation

- Simple: Translation, Rotation, Scaling
- More complex: Stretch, Twist, Bend



But global deformations are not adapted to articulated characters

⇒ Multiples rotations, with smooth deformations around the articulations

How to deform/animate character

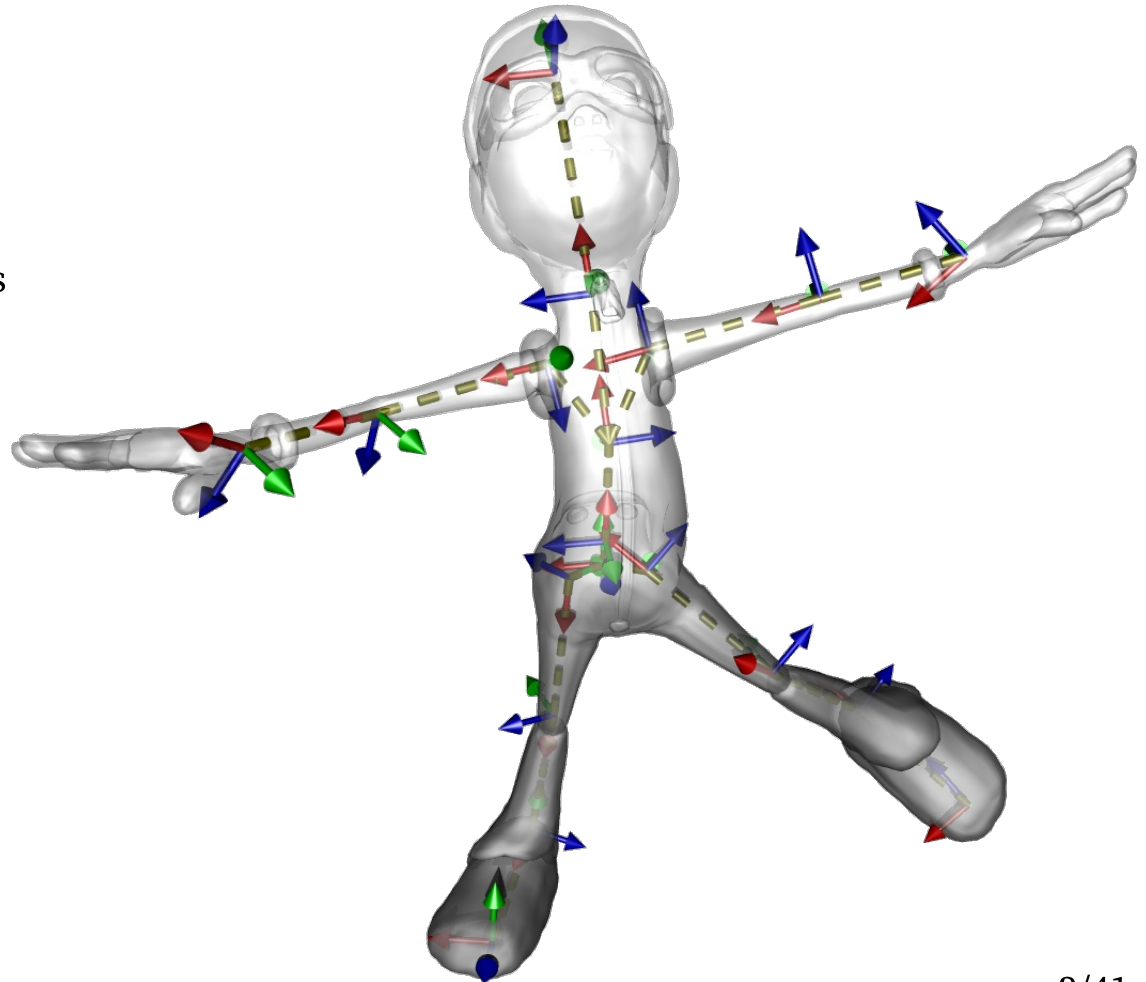
Idea => Use a **Skeleton**

Animation Skeleton

- Set of frames (position,orientation)
- Describes the non-rigid parts of the characters
its degrees of freedom

Terminology animation skeleton

- **Joint** = A frame
- **Bone** = Segment between two joints

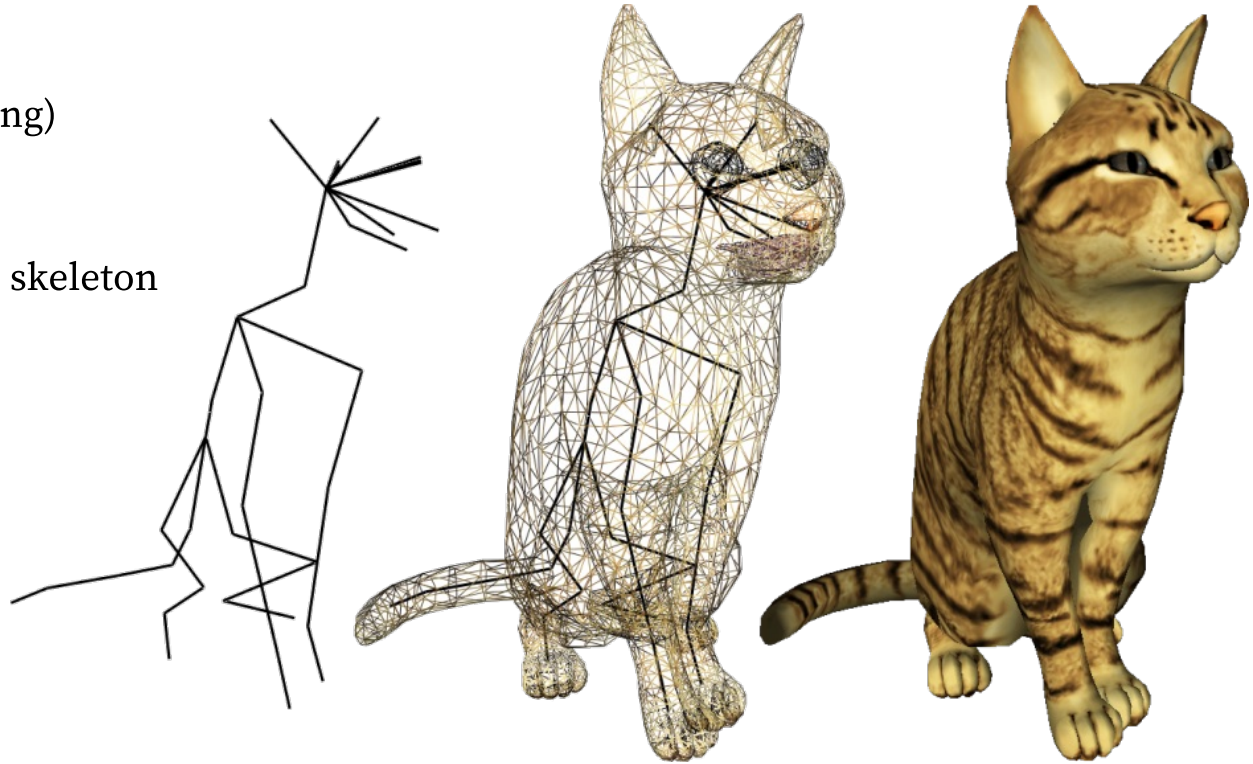


How to deform/animate character

Animating a character

- 1- Animate its skeleton (/posing)
- 2- Deform its *skin* accordingly (= skinning)

Note: Animation Skeleton != Anatomical skeleton



Character Animation

Skeletal Animation

Skeleton structure

Characteristics

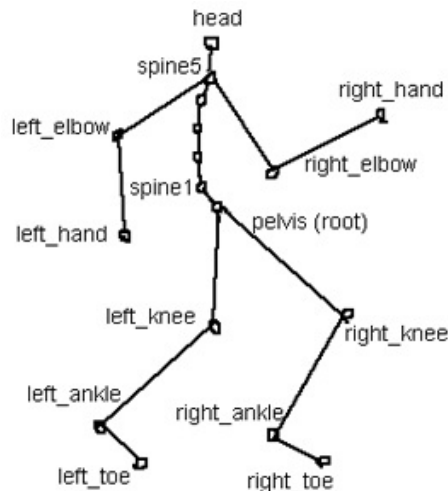
- Hierarchical representation

All children follow the transformation of the parent

Need to define a root: usually at the hips/pelvis

- Convenient to express local deformation with respect to the parent

ex. Rotate knee from 20°



Converting local to global frames/joint coordinates

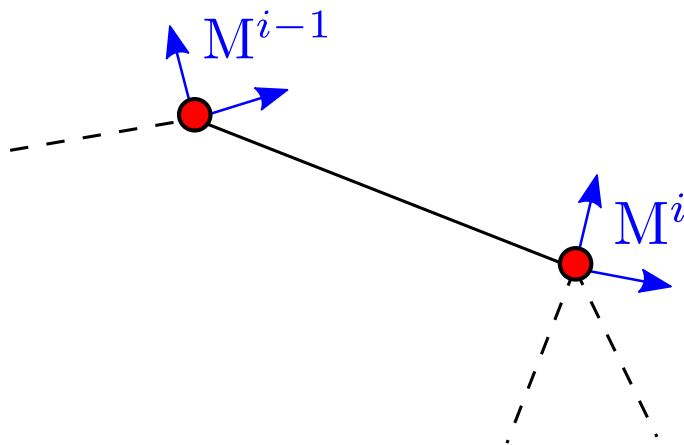
- With 4x4 matrices M

$$M_{global}^i = M_{global}^{i-1} M_{local}^i$$

- With translation t , rotation R

$$R_{global}^i = R_{global}^{i-1} R_{local}^i$$

$$t_{global}^i = t_{global}^{i-1} + R_{global}^{i-1} t_{local}^i$$



Encoding hierarchical skeleton

- Simplest encoding based on index within vector

```
Geometry=[M0, M1, M2, M3, M4, M5]
```

```
Parent = [-1,0,1,1,0,4]
```

- Convert local coordinates to global coordinates

```
local (Geometry) <- std::vector of rotation (r), translation (p)
```

```
global (Geometry) <- std::vector of rotation (r), translation (p)
```

```
global[0] = local[0];
```

```
for(size_t k=1; k<N; ++k)
```

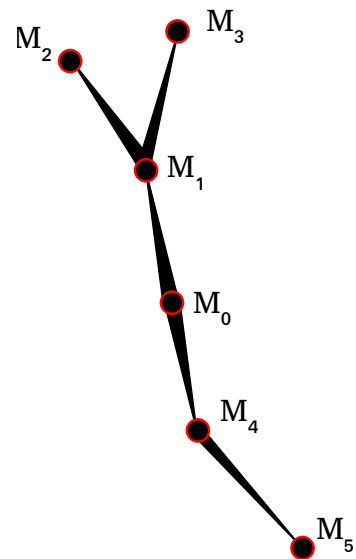
```
{
```

```
    int parent = Parent[k];
```

```
    global[k].r = global[parent].r * local[k].r;
```

```
    global[k].p = global[parent].r * local[k].p + global[parent].p;
```

```
}
```

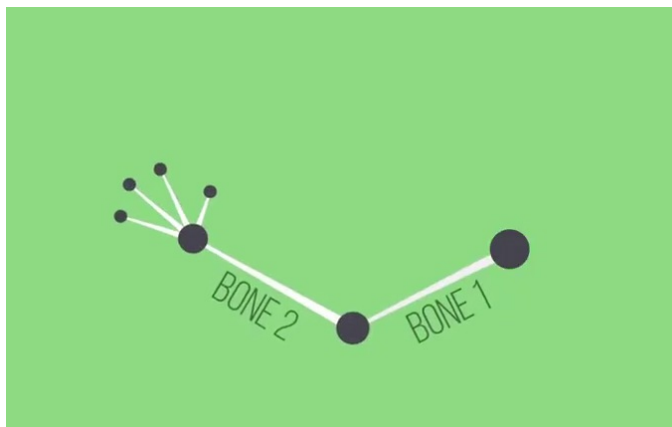
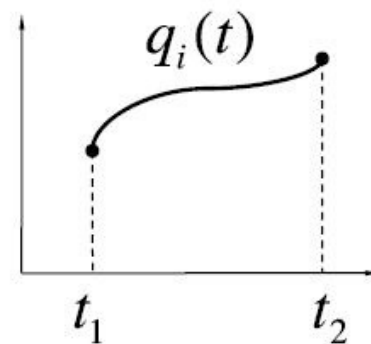
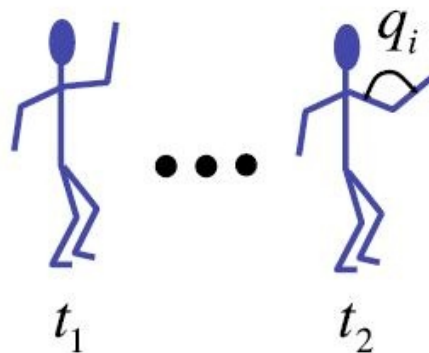


Forward kinematics

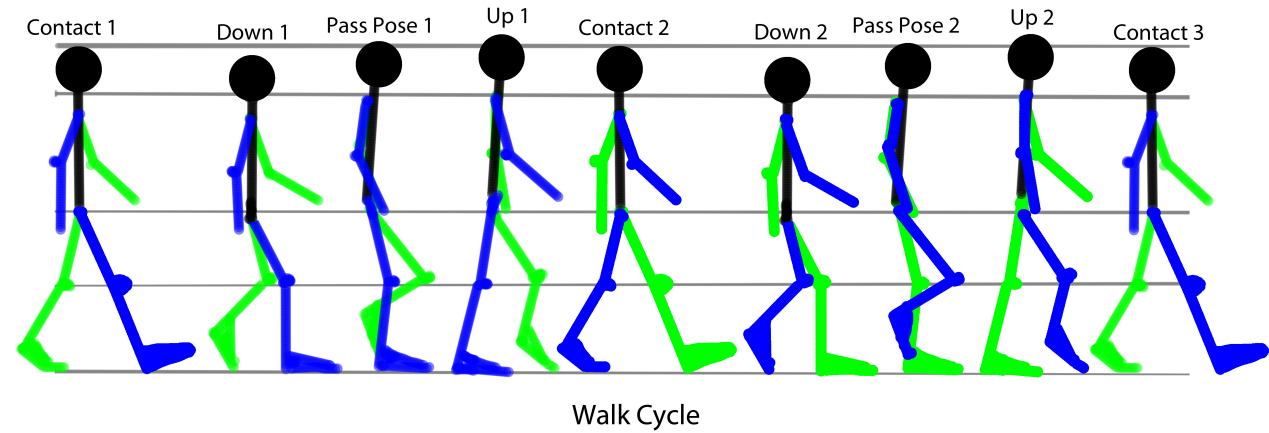
FK - Forward Kinematics

- Each joint angle is set manually
- Adapted to set orientation of specific parts
- Interpolate rotations during animation

(+) Generates curved trajectory naturally



KF limits



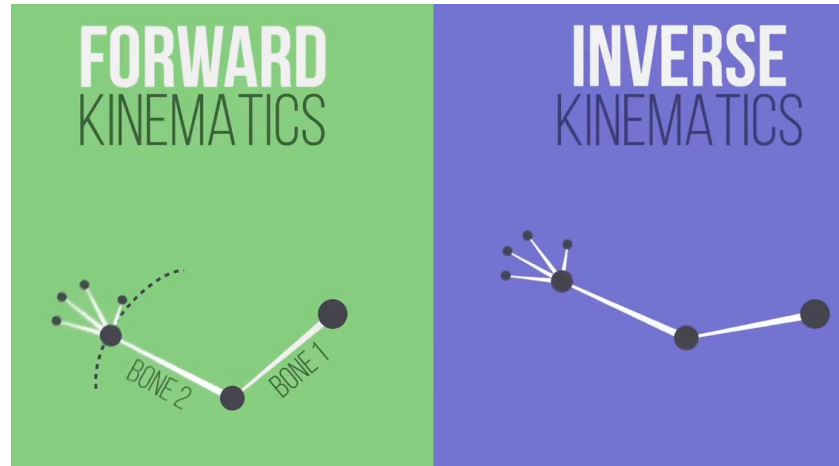
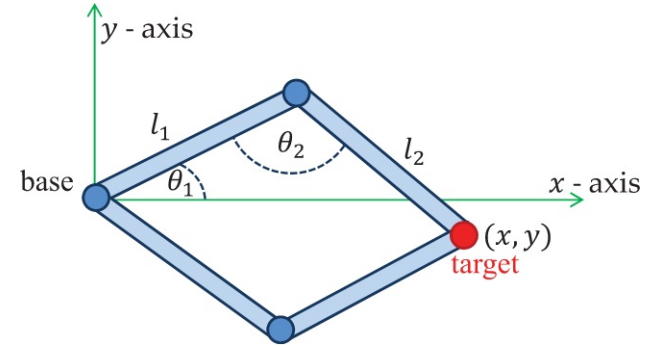
Inverse Kinematics

IK: Inverse Kinematics

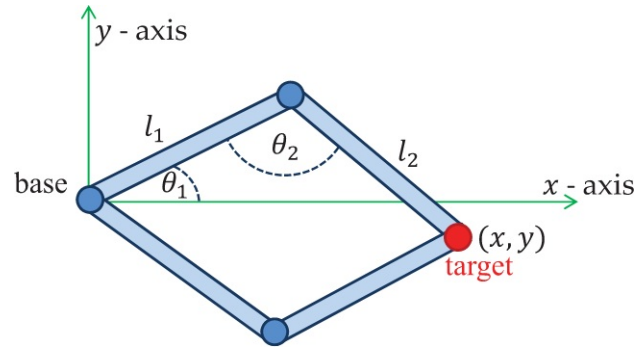
- Describe position (/and orientation) of *end-effectors* (contact, walking, etc)
- Compute joint angles reaching this position

$$p_k = p_0 + \sum_{i=0}^{k-1} l_i R_i u_i$$

R_i can be expressed with various rotation parameters (Matrices, Euler angles, axis/angle, quaternions, etc)



IK Example with two bones



[A. Aristidou et al. Inverse Kinematics Techniques
in Computer Graphics: A Survey. STAR EG. 2017.]

In general the general case $p_k = f(\theta_i)$

- Look for $\theta_i = f^{-1}(p_k)$
- f is a non linear function
- There may exists multiple solutions (or none)
- Solutions may exhibits discontinuities
- Closed form solution are not available

Two solutions defined by

$$\cos(\theta_1) = \frac{l_1^2 + x^2 + y^2 - l_2^2}{2l_1 \sqrt{x^2 + y^2}}$$

$$\cos(\theta_2) = \frac{l_1^2 + l_2^2 - (x^2 + y^2)}{2l_1 l_2}$$

Inverse Kinematics

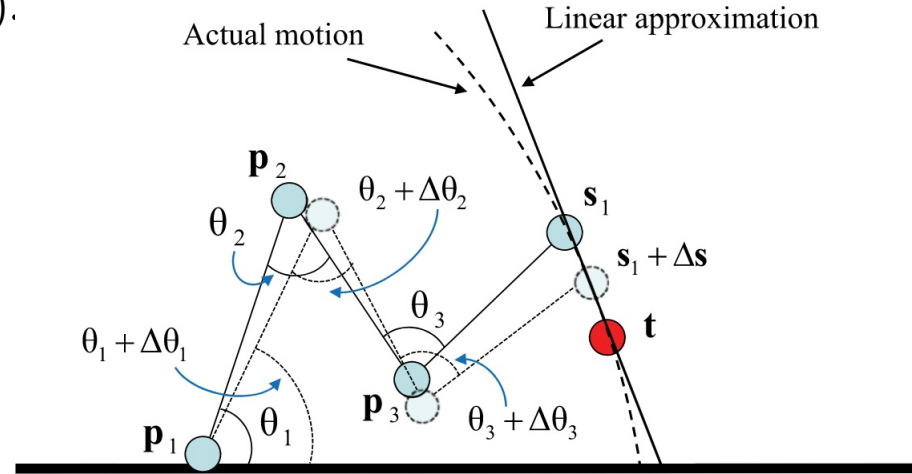
Numerical inversion of $p = f(\theta)$, $\theta = (\theta_0, \dots, \theta_{N-1})$.

Consider small step size $p \rightarrow p + \Delta p$

$$\Delta p \simeq \underbrace{\left(\frac{\partial f}{\partial \theta} \right)}_{\text{J}} \Delta \theta$$

J - Jacobian matrix.

→ Not square ($3 \times N$), not invertible.



Possible solution - **Pseudo inverse**

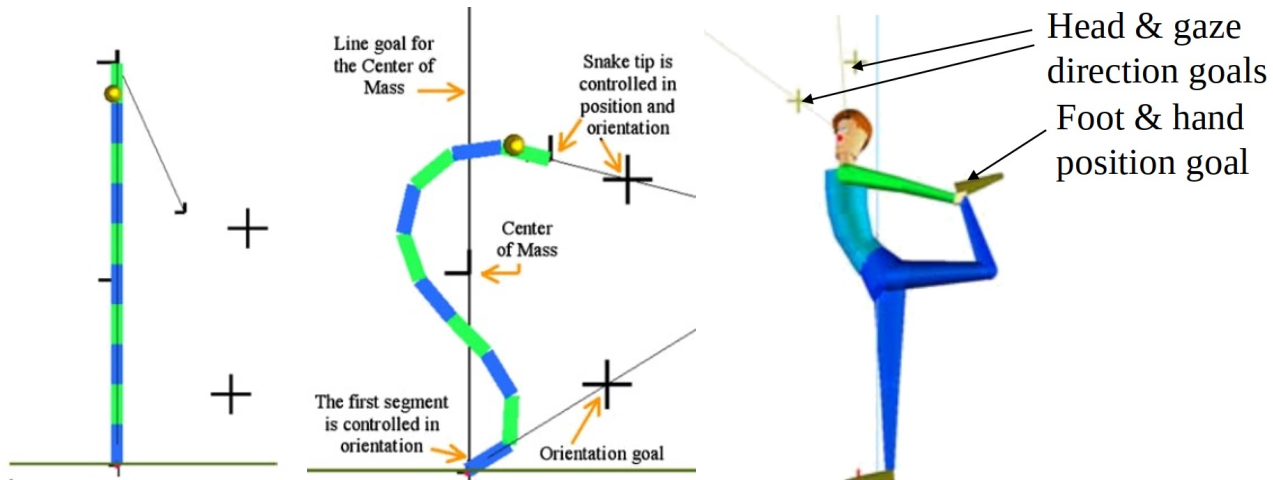
$$\begin{cases} \Delta\theta = J^+ \Delta p \\ J^+ = J^T (J J^T)^{-1} \end{cases}$$

- Check that $\mathbf{J}\mathbf{J}^+ = \mathbf{I}_d$
- But check that $\Delta\theta = \mathbf{J}^+ \Delta p + (\mathbf{I} - \mathbf{J}^+ \mathbf{J}) \mathbf{y}$ is also solution, for any vector $\mathbf{y}$.
- $\mathbf{y}$: Secondary task

Inverse Kinematics

Example of secondary task use

Center of mass above the sustentation polygon



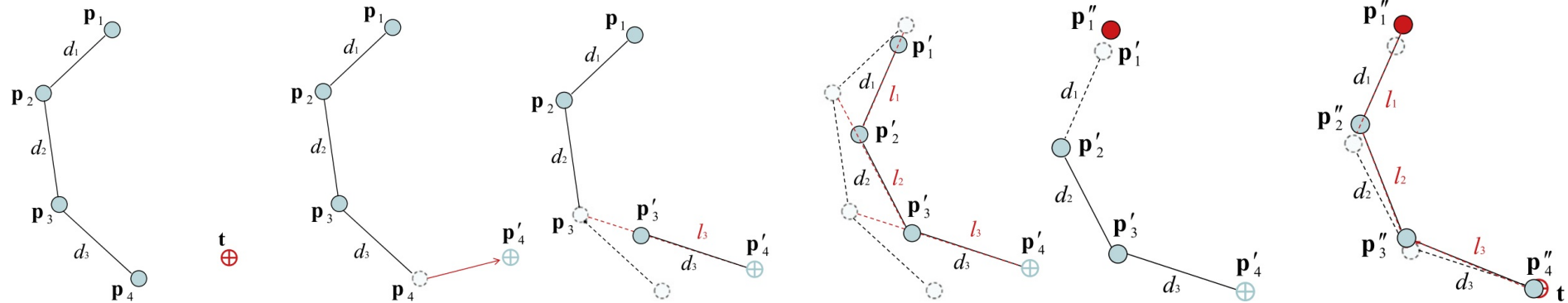
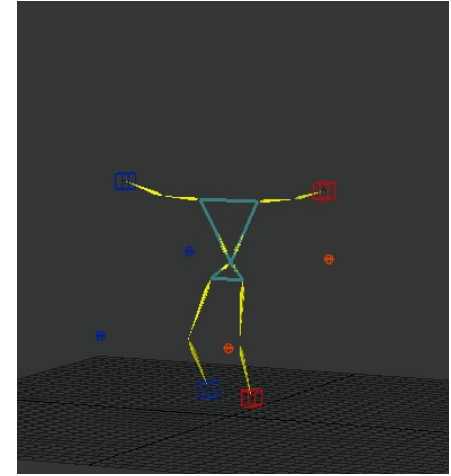
P. Baerlocher, R. Boulic. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. TVC 2004.

Inverse Kinematics

Heuristic alternative

ex. FABRIK.

A. Aristidou, J. Lasenby. A fast iterative solver for the Inverse Kinematics problem. Graphical Models 2011.

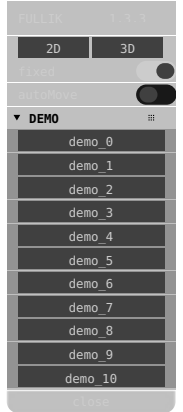


- Very fast, works ok for simple chain
- Doesn't handle global constraints well

Inverse Kinematics

Example

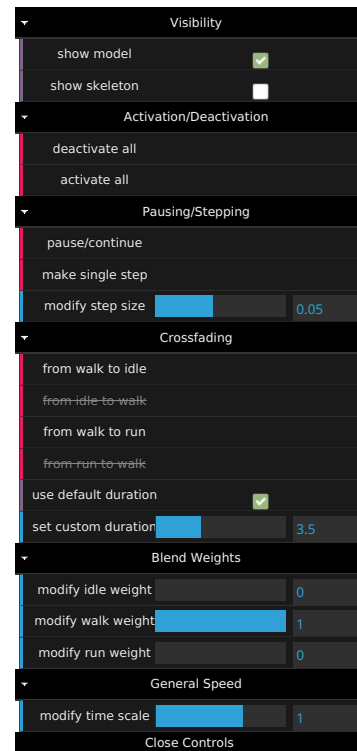
Demo 2 - Chain with fixed base and multiple unconstrained bones.



Blending skeleton animation

[three.js](#)

[realitymeltedown.com](#)



Motion graphs

Also called Move Trees

- Stores multiple precomputed animation

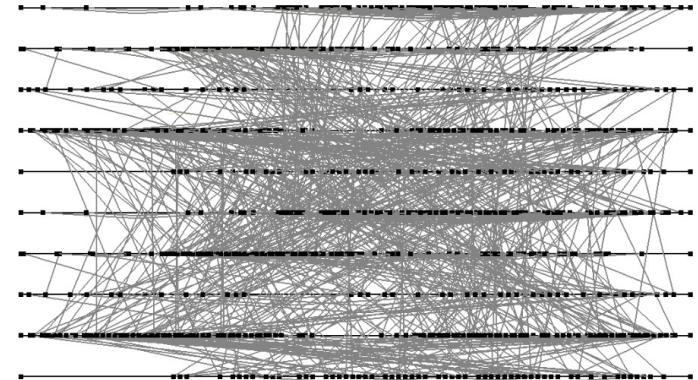
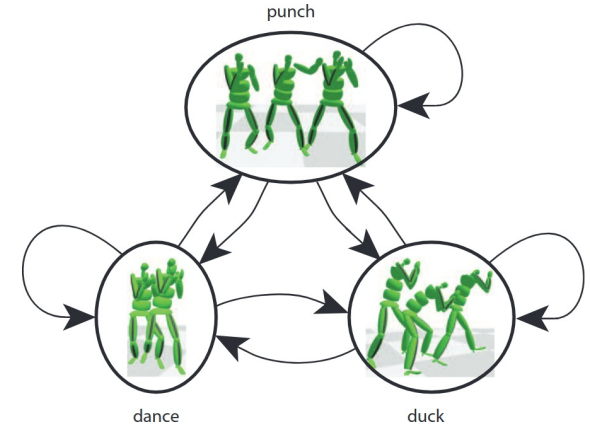
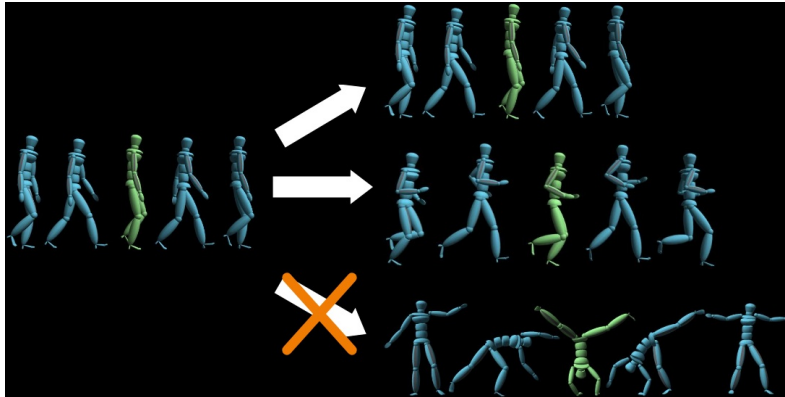
Manually design, motion capture, etc

- Find optimal transitions between different motions

- [Mizuguchi et al., Data driven motion transitions for interactive games, EG short paper, 2001]

- [Kovar et al., Motion Graphs, ACM SIGGRAPH 2002]

- [Heck and Gleicher, Parametric Motion Graphs, ACM SIGGRAPH 2007]



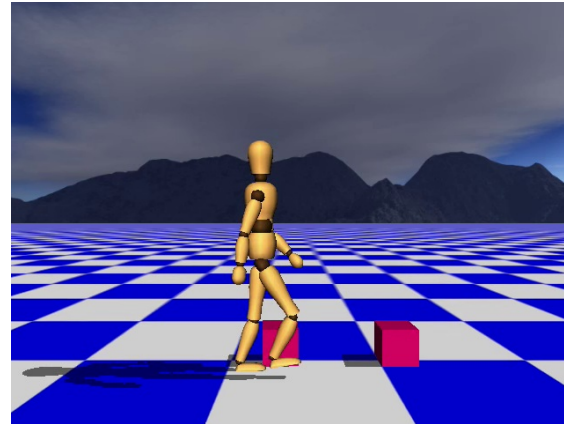
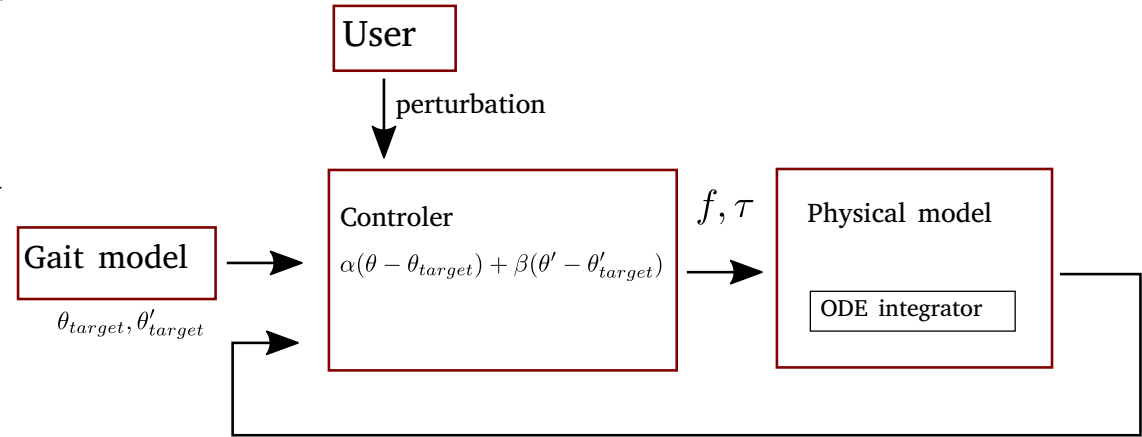
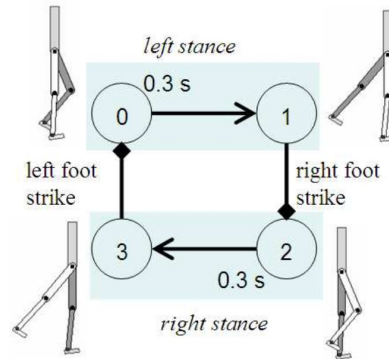
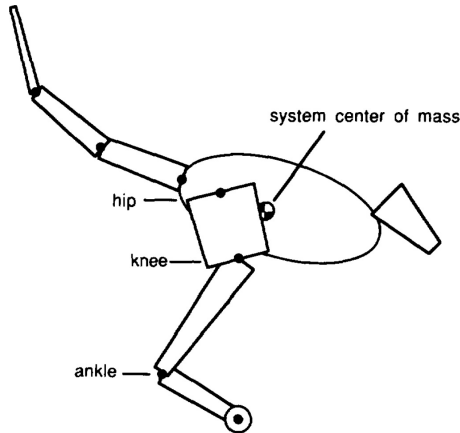
Controllers

Mix between predefined motions and physics →
allow user perturbations

- Define target motions
- Use proportional derivative controllers (with feedback loop) to control forces and torques

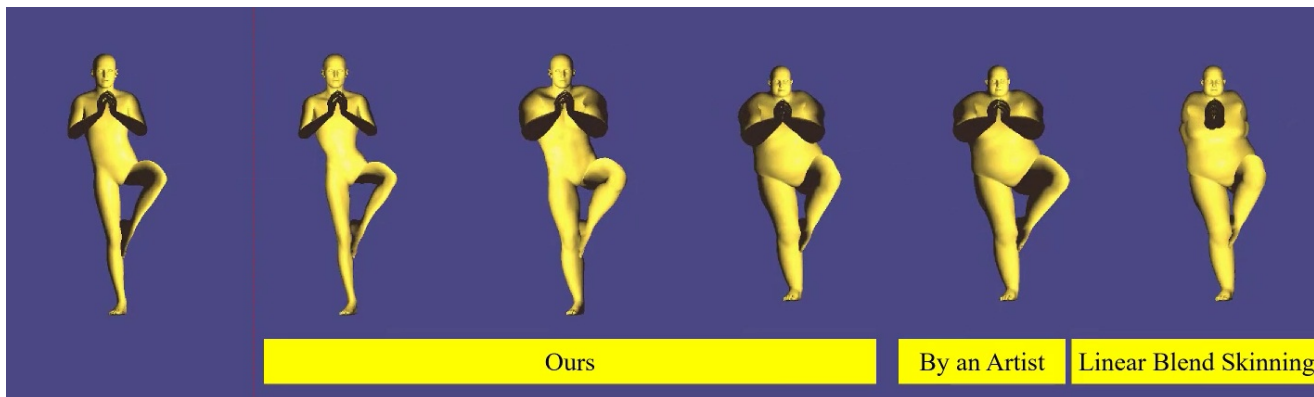
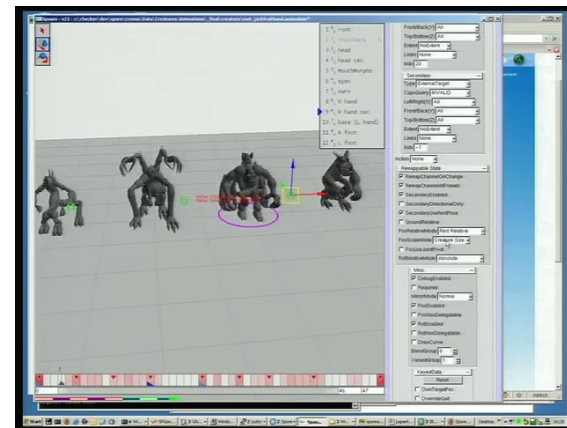
- [M. Raibert and J. Hodgins. Animation of Dynamic Legged Locomotion, ACM SIGGRAPH 2001]

- [K. Yin et al., SIMBICON: Simple Biped Locomotion Control, ACM SIGGRAPH 2007]



Motion transfert

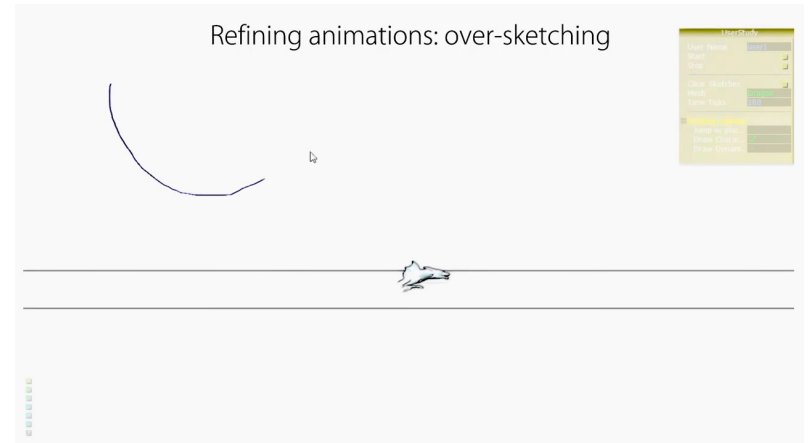
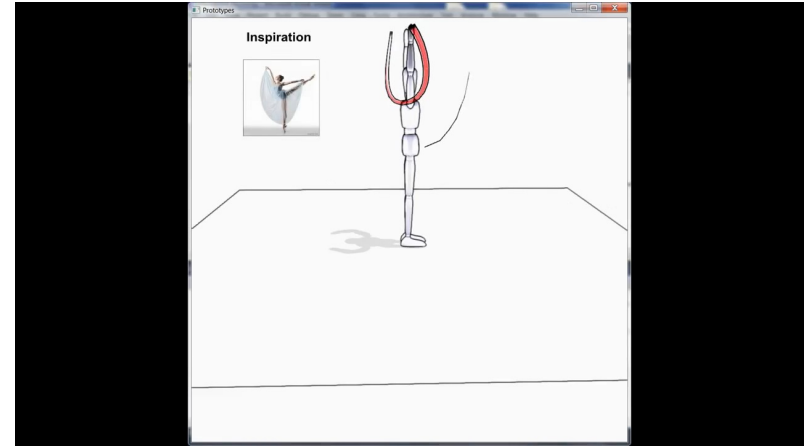
- Local coordinates mapping from skeletal motions
 - [C. Hecker et al., Real-time Motion Retargeting to Highly Varied User-Created Morphologies, SIGGRAPH 2008] (Spore)
- Including shape morphology
 - [Z. Liu et al., Surface based Motion Retargeting by Preserving Spatial Relationship, MIG 2018]



Animation design

Based on the *Line of Action*

- [Guay et al., The Line of Action: an Intuitive Interface for Expressive Character Posing, ACM SIGGRAPH Asia 2013]
- [Guay et al., Space-time sketching of character animation, ACM SIGGRAPH 2015]
- [Choi et al., SketchiMo: Sketch-Based Motion Editing for Articulated Characters, ACM SIGGRAPH 2016]



Skeletal animation and learning

Reinforcement learning from muscles

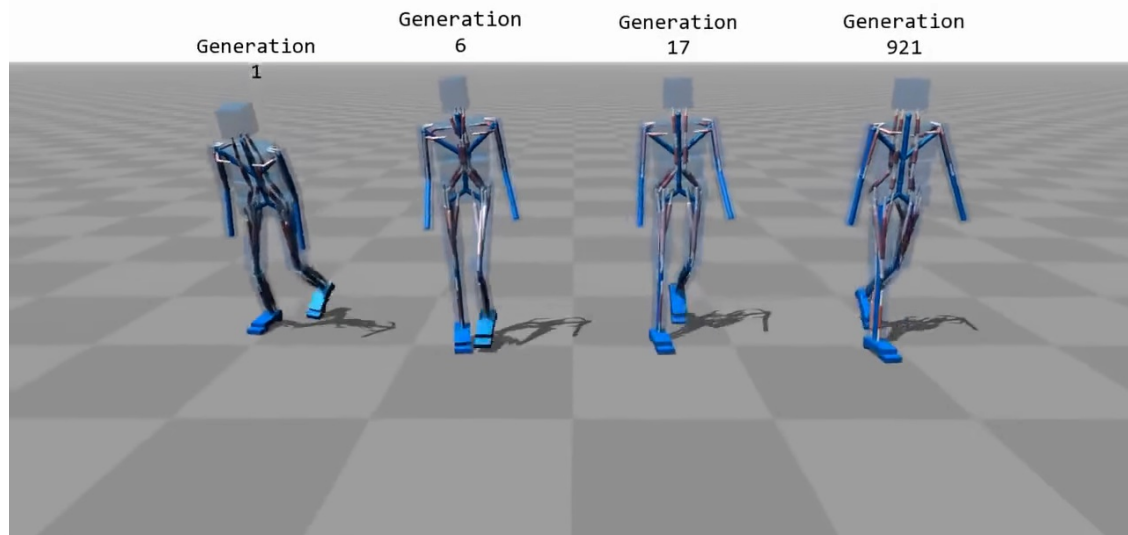
[Geijtenbeek et al., Flexible Muscle-Based Locomotion for Bipedal Creatures, ACM TOG 2013]

Learning physics-based controllers

[Won et al. How to Train Your Dragon: Example-Guided Control of Flapping Flight, ACM SIGGRAPH Asia 2017]

Neural networks for character control from large data base of motion capture data

[Holden et al., Phase-Functioned Neural Networks for Character Control, ACM TOG 2017]



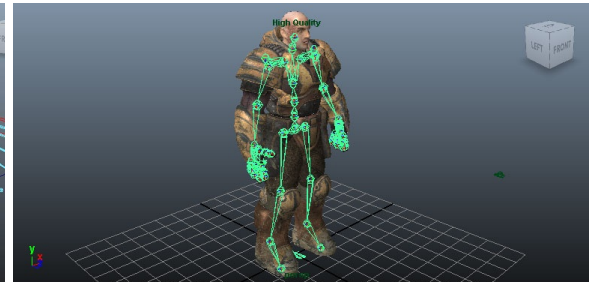
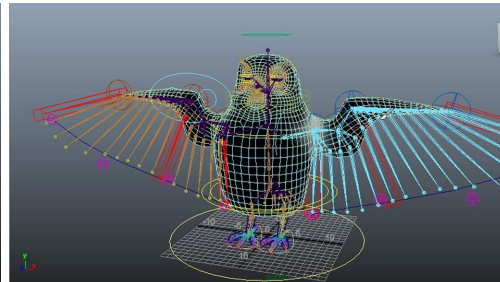
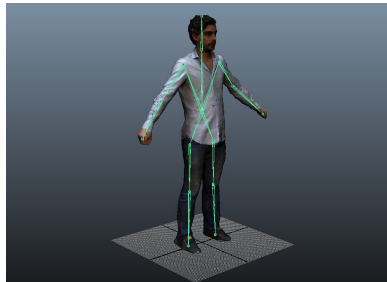
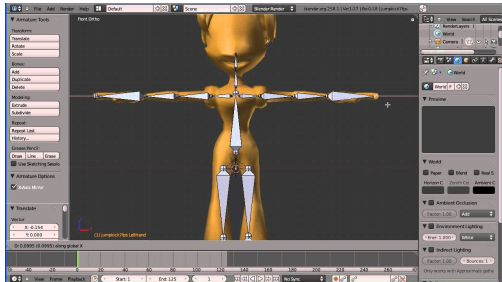
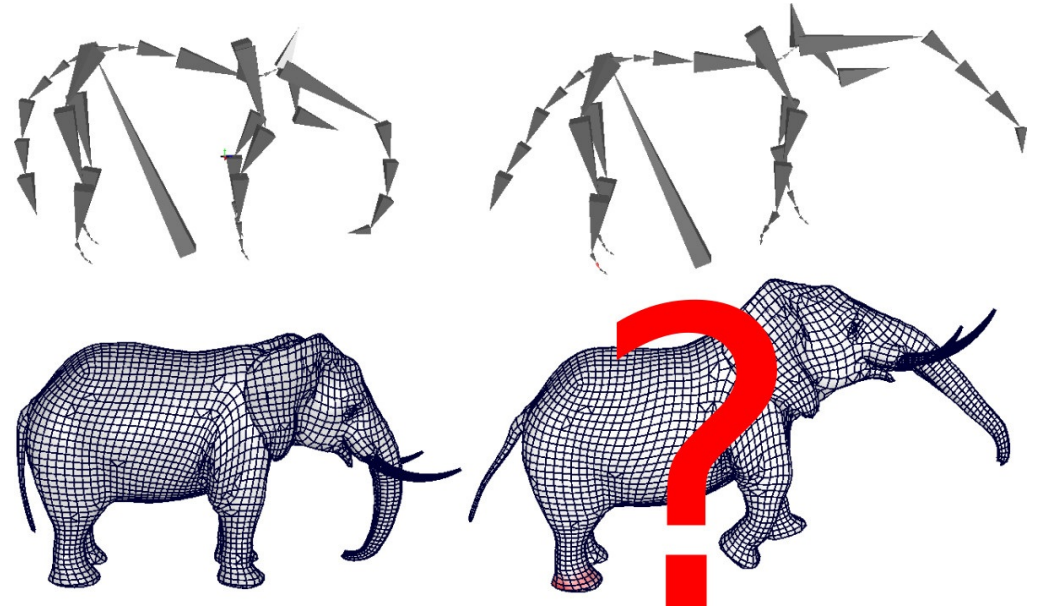
Character Animation

Skin deformation

Animating skin

Animated character = A Skeleton + a Mesh

- We saw how to pose/animate skeleton
- Objective: How to deform the surrounding mesh given the skeleton



Brute force physically-based modeling

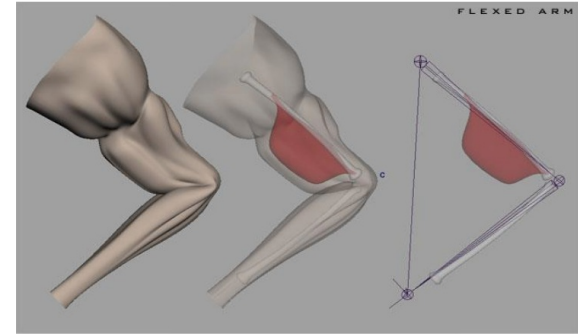
- Model muscles attached to bones
- Simulate muscle deformation using FEM
- Add an elastic skin layer on top of muscles

(+) Detailed dynamic results

(-) Computational time

(-) Tedious to setup and control

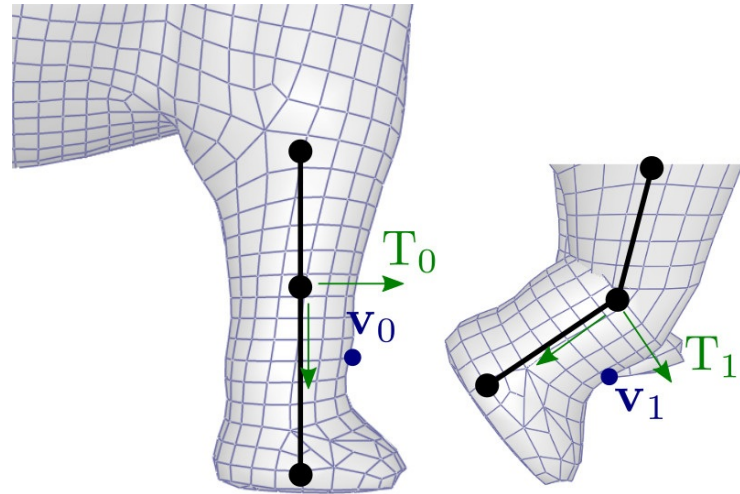
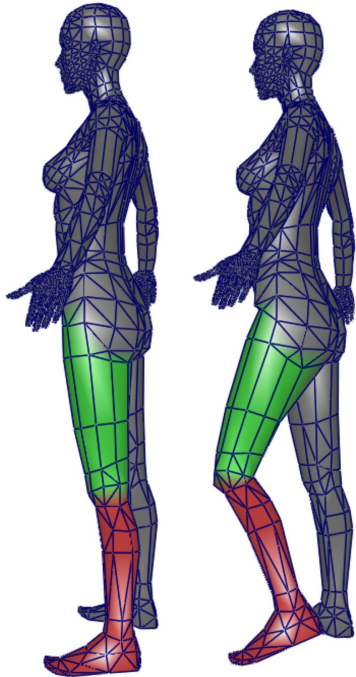
[Teran et al., Creating and Simulating Skeletal Muscle from the Visible
Human Data Set. IEEE TVCG 2005]



30 muscles, 10×10^6 tetrahedrons

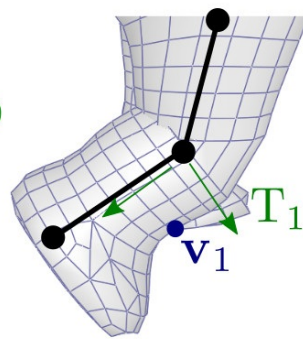
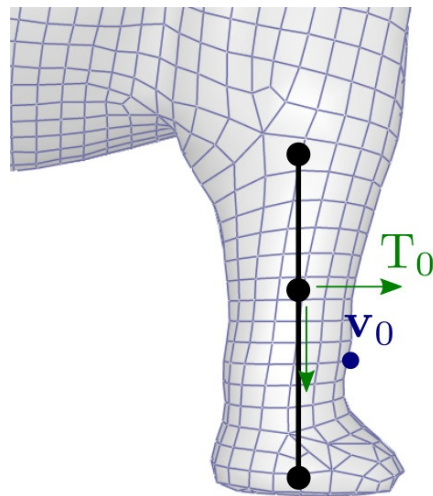
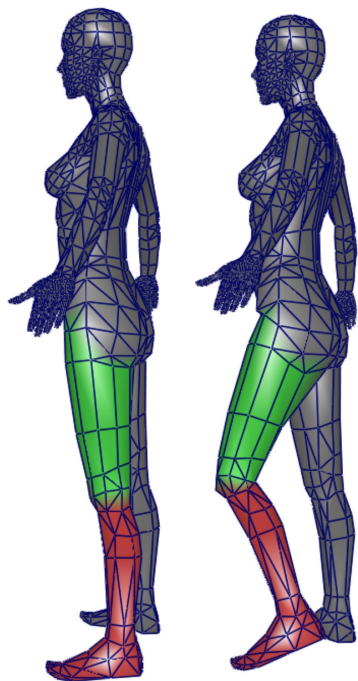
Geometrical approach: Rigid Skinning

- Associate mesh parts rigidly to nearby bone
- Compute vertex deformation using geometry



Geometrical approach: Rigid Skinning

- Associate mesh parts rigidly to nearby bone
- Compute vertex deformation using geometry



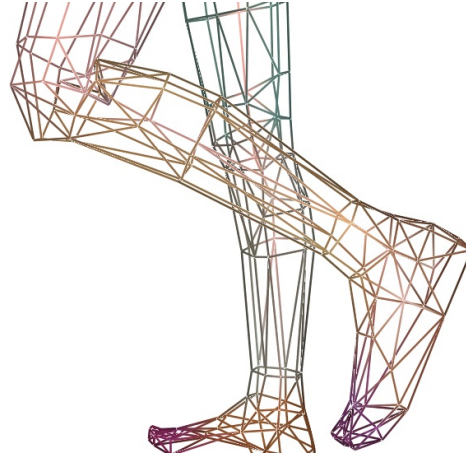
$$v_0^{local} = v_1^{local}$$

$$T_0^{-1} v_0 = T_1^{-1} v_1$$

$$v_1 = T_1 T_0^{-1} v_0 = M v_0$$

Rigid skinning pro/cons

- (+) Skeleton is easy to built
- (+) Skeleton interaction is intuitive to model rigid articulations
- (-) Discontinuities



Smooth skinning

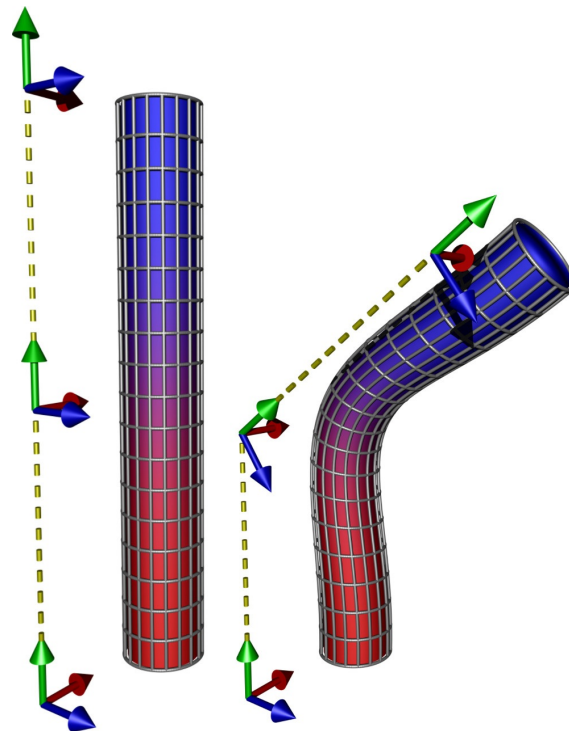
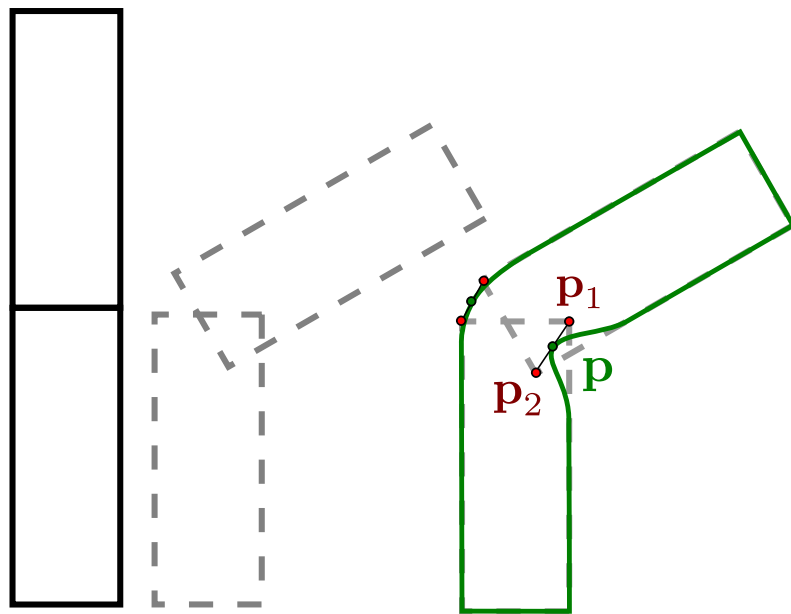
Idea: Interpolate positions between associated frames

Ex. middle vertex position of a bending cylinder

$$p = 0.5p_1 + 0.5p_2$$

$$p = 0.5 M_1 p_0 + 0.5 M_2 p_0$$

$$p = (0.5 M_1 + 0.5 M_2) p_0$$



Smooth skinning - formulation

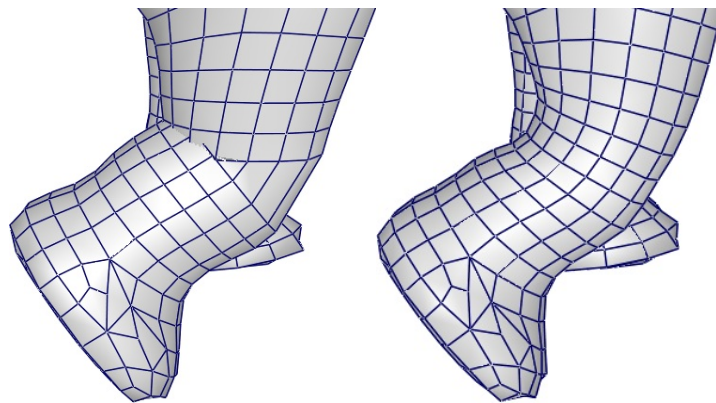
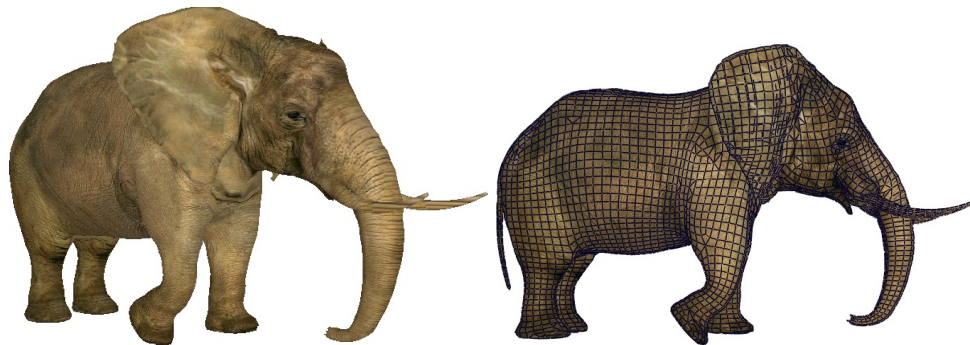
General formula

$$p = \left(\sum_i \omega_i M_i \right) p_0$$

$$p = \left(\sum_i \omega_i T_i (T_i^0)^{-1} \right) p_0$$

ω_i barycentric weights: $0 \leq \omega_i \leq 1$, $\sum_i \omega_i = 1$

Also called **L**inear **B**lend **S**kinning (LBS)



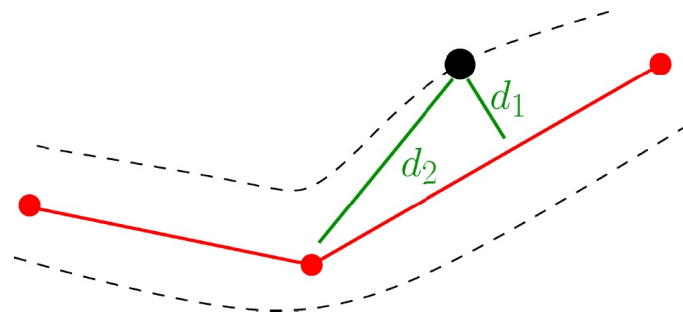
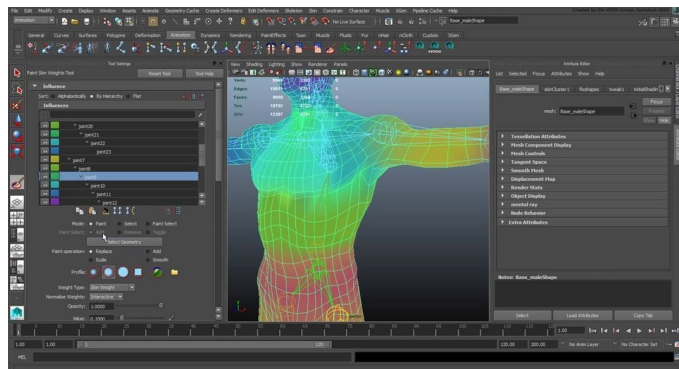
Rigid skinning

Smooth skinning (LBS)

Skinning Weights

How to compute skinning weights ?

- Paint them manually
- Automatic computation



Associating bones and skinning weights (or any animation handle) to mesh is called **Rigging**

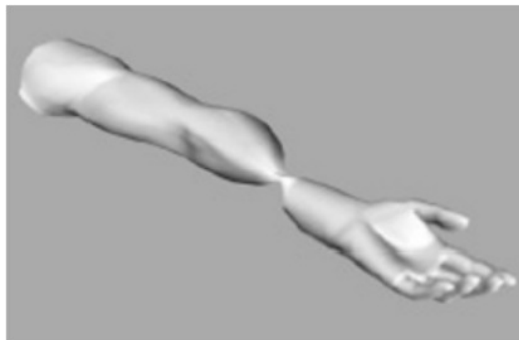
Linear Blend Skinning

Widely used for character animation (in all modeling/animation software)

- Direct, intuitive control (through skeleton)
- Very efficient (matrix multiplication, GPU)

(-) But has well known defects:

- Artifacts for large angles
- Complex rigging settings



Candy wrapper

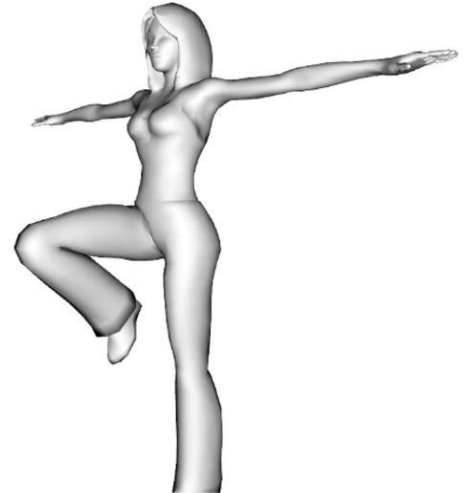
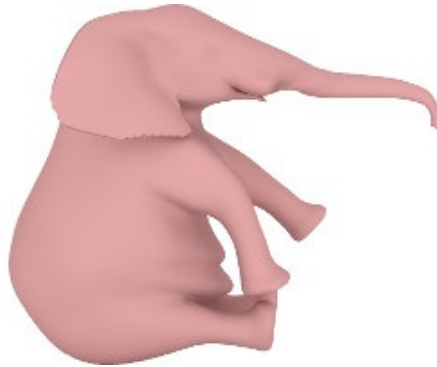
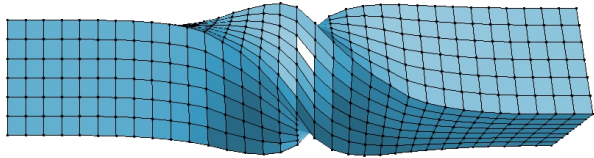


Collapsing elbow

Improving LBS

Study cases

- 1- Improving the interpolation
- 2- Preserving a constant volume
- 3- Improving rigidity and avoiding self collisions



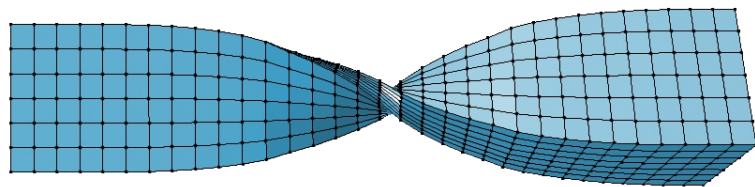
1. Skinning Interpolation

Problem: Remember that linear interpolation of rigid deformation matrices may lead to artifacts

Skinning equation $p = \sum_i \omega_i M_i p_0$

$\Rightarrow$ *Linear interpolation of rigid frames (translation + rotation)*

Leads to non rigid deformation



Cannot use directly quaternion (each vertex should have a different center of rotation).

Dual quaternion

Idea: Use of **dual quaternions**

[L. Kavan et al. Geometric Skinning with Approximate Dual Quaternion Blending, ACM TOG 2008]

Dual quaternion = Generalization of quaternion to dual number

- $\hat{q} = q_0 + \epsilon q_\epsilon$

- ϵ : dual number $\epsilon^2 = 0$

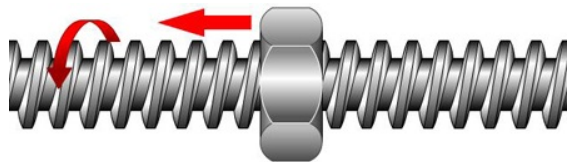
dual number commonly used to model infinitesimal quantity (ex. used for automatic differentiation)

A dual quaternion $\hat{q} = q_0 + \epsilon q_\epsilon$ encodes a rigid transformation as a **screw motion**

Screw motion: rotation about an axis followed by a translation in the direction of this axis

- q_0 : pure rotation component

- q_ϵ : encodes translation component (dual part)



Dual quaternion

Given a quaternion q_0 and a translation t , its associated dual quaternion is

$$\hat{q} = q_0 + \frac{\epsilon}{2} q_t q_0, q_t = (t_x, t_y, t_z, 0)$$

Similarly to quaternion: angle/axis correspondance

$$-\hat{q} = \cos(\hat{\theta}/2) + \hat{n} \sin(\hat{\theta}/2)$$

$$-\hat{\theta} = \theta_0 + \epsilon \theta_\epsilon$$

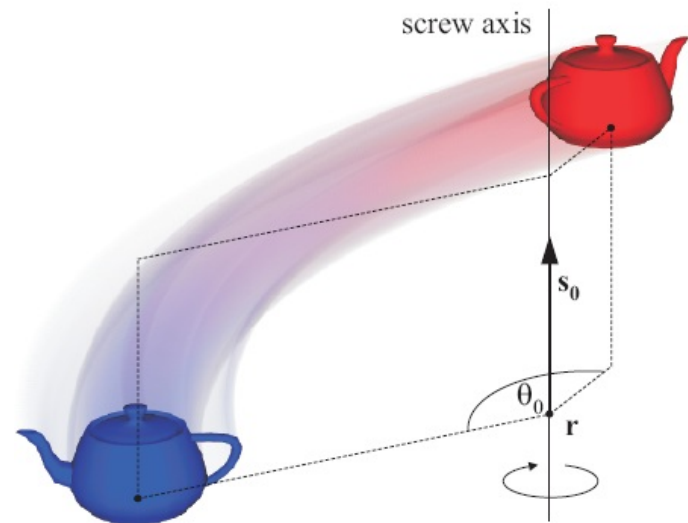
- θ_0 : angle of rotation

- $\theta_\epsilon = t \cdot n_0$: amount of translation along n_0

$$-\hat{n} = n_0 + \epsilon n_\epsilon$$

- n_0 : axis of rotation

- $n_\epsilon = \frac{1}{2}((n_0 \times t) \cotan(\theta_0/2) + t) \times n_0$: called the moment of the rotation axis.

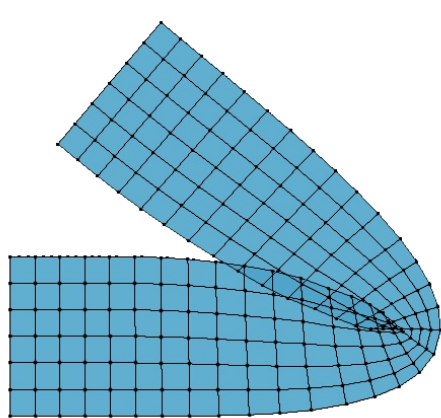


Dual Quaternion Skinning (DQS)

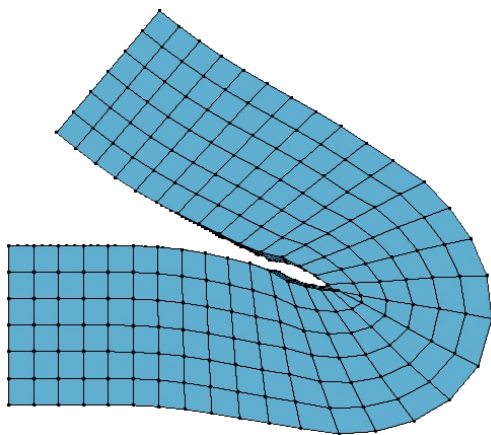
- Encode rigid transformation M into dual quaternion $\hat{q}_i$
- Perform blending in the dual quaternion space (ScLERP)

$$\hat{q} = \sum_i \omega_i \hat{q}_i$$

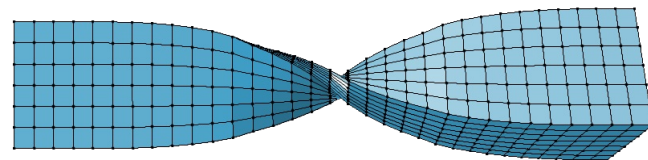
- Normalize resulting dual quaternion $\hat{q}_n = \hat{q} / \|\hat{q}\|$
- Apply $\hat{q}_n$ to p_0



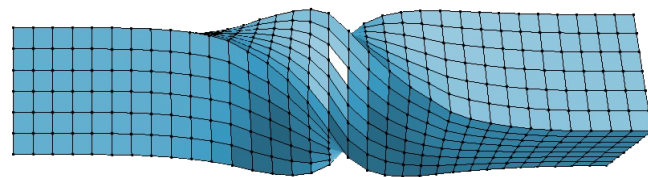
LBS



DQS



LBS



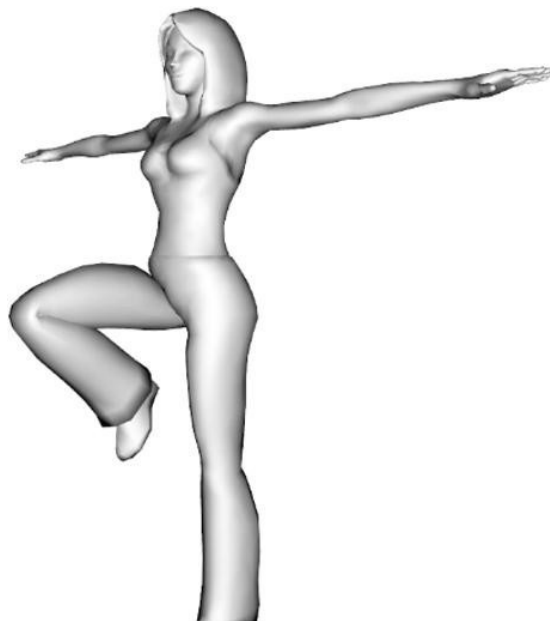
DQS

Dual Quaternion VS LBS

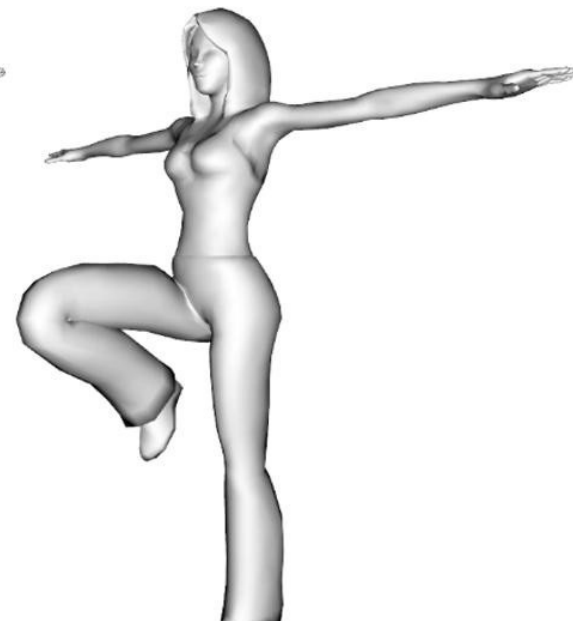
Dual quaternion

- (+) Fully solves *candy wrapper* artifact
- (+) Almost as efficient as LBS
- (-) But may create artificial bulge

Not always preferred to LBS



LBS



DQS

2. Volume preserving skinning

- Optimization based model using tetrahedrons

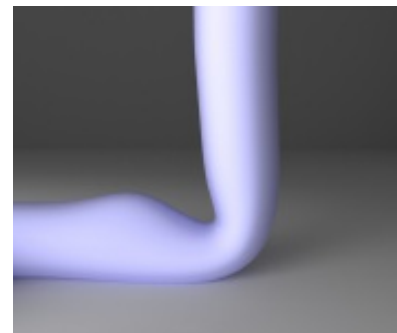
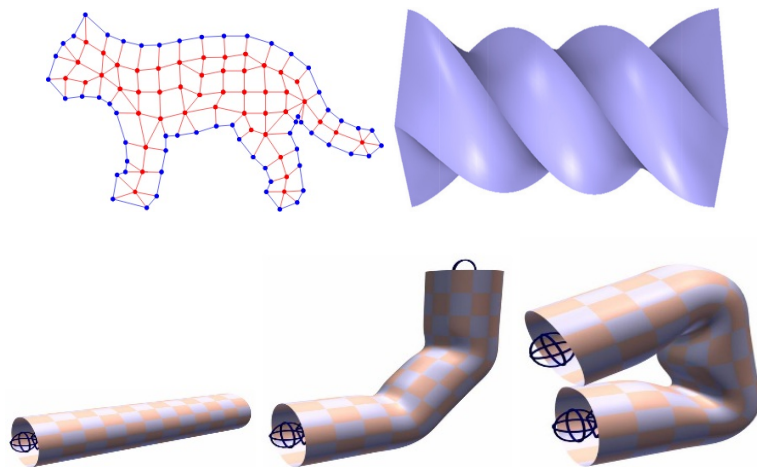
[K. Zhou et al., Large Mesh Deformation Using the Volumetric Graph Laplacien, ACM SIGGRAPH 2005]

- Specific deformaters

[A. Angelidis and K. Singh, Kinodynamic skinning using volume-preserving deformations, SCA 2007]

- Direct geometrical volume preservation

[D. Rohmer et al., Exact volume preserving skinning with shape control, SCA 2009]



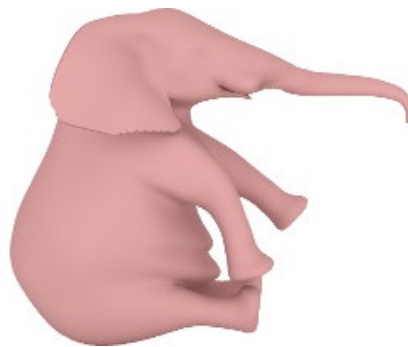
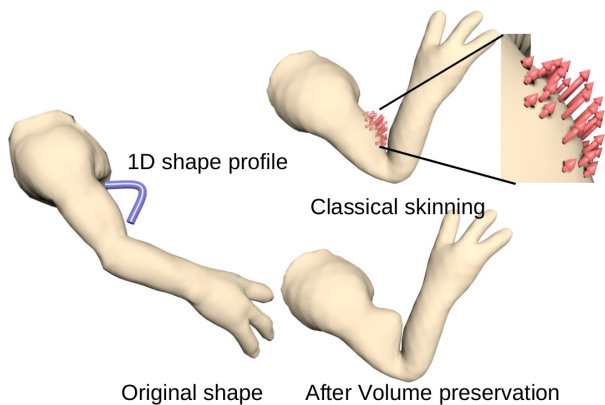
Geometrical constant volume skinning

$$\begin{cases} \min \sum_k \frac{\|u_k\|^2}{\gamma_k} \\ \text{subject to } V(p_k + u_k) = V_{objective} \end{cases}$$

- u_k deformation vector

- γ_k weights

$$- V = \sum_{faces(l,m,n)} \frac{z_l + z_m + z_n}{6} \begin{vmatrix} x_m - x_l & y_m - y_l \\ x_n - x_l & y_n - y_l \end{vmatrix}$$



[SCA 2009]

3. Improving rigidity and avoiding self collision

Idea: Extend implicit surface modeling to skinning deformation

1- Approximate the surface around each bone using implicit surfaces

Blend implicit surfaces to get a continuous surface

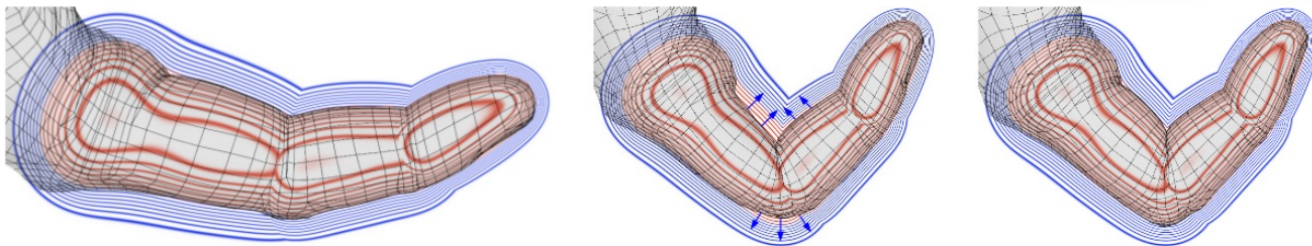
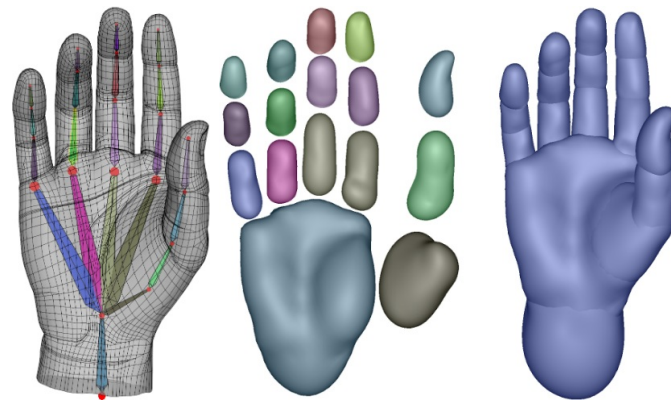
store exact isovalue for each vertex

2- Deform the mesh using LBS/DQS

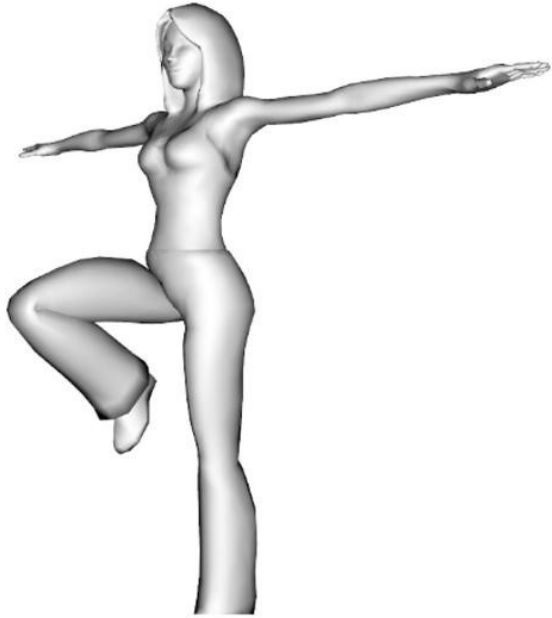
Deform each implicit surface rigidly with its associated bone

3- Project vertices back onto their original isovalues

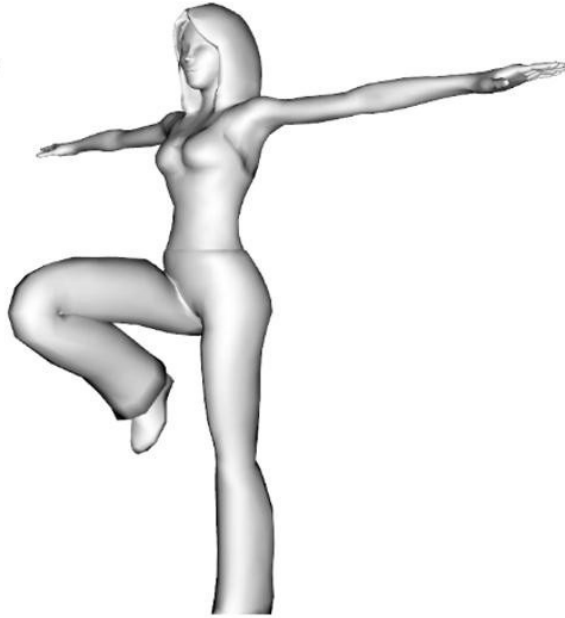
Stop projection when intersection is detected



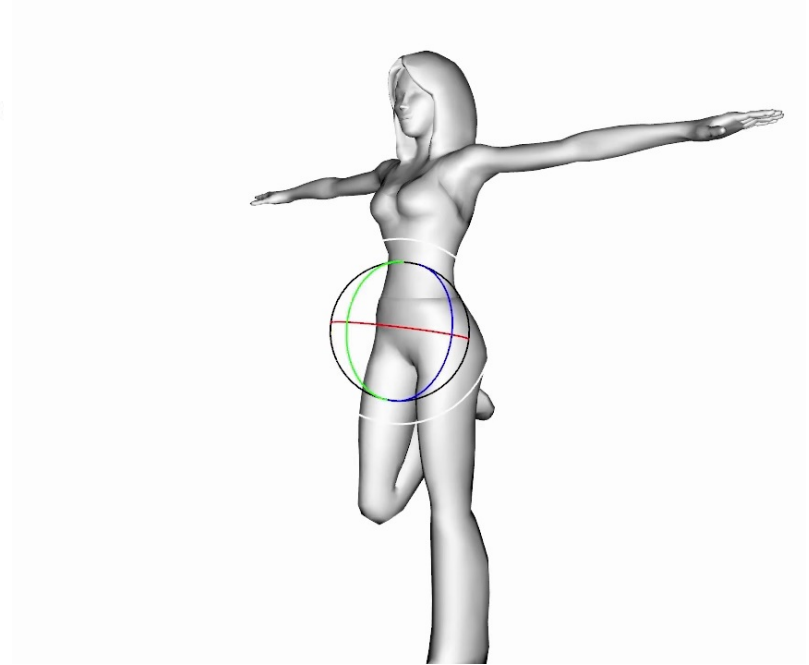
Implicit Skinning



LBS



DQS



Implicit Skinning

[R. Vaillant et al., Implicit Skinning: Real-Time Skin Deformation with Contact Modeling, ACM SIGGRAPH 2013]