10 Mesh Animation

Steve Marschner

CS5625 Spring 2022

Basic surface deformation methods

Blend shapes: make a mesh by combining several meshes

Mesh skinning: deform a mesh based on an underlying skeleton

Both use simple linear algebra

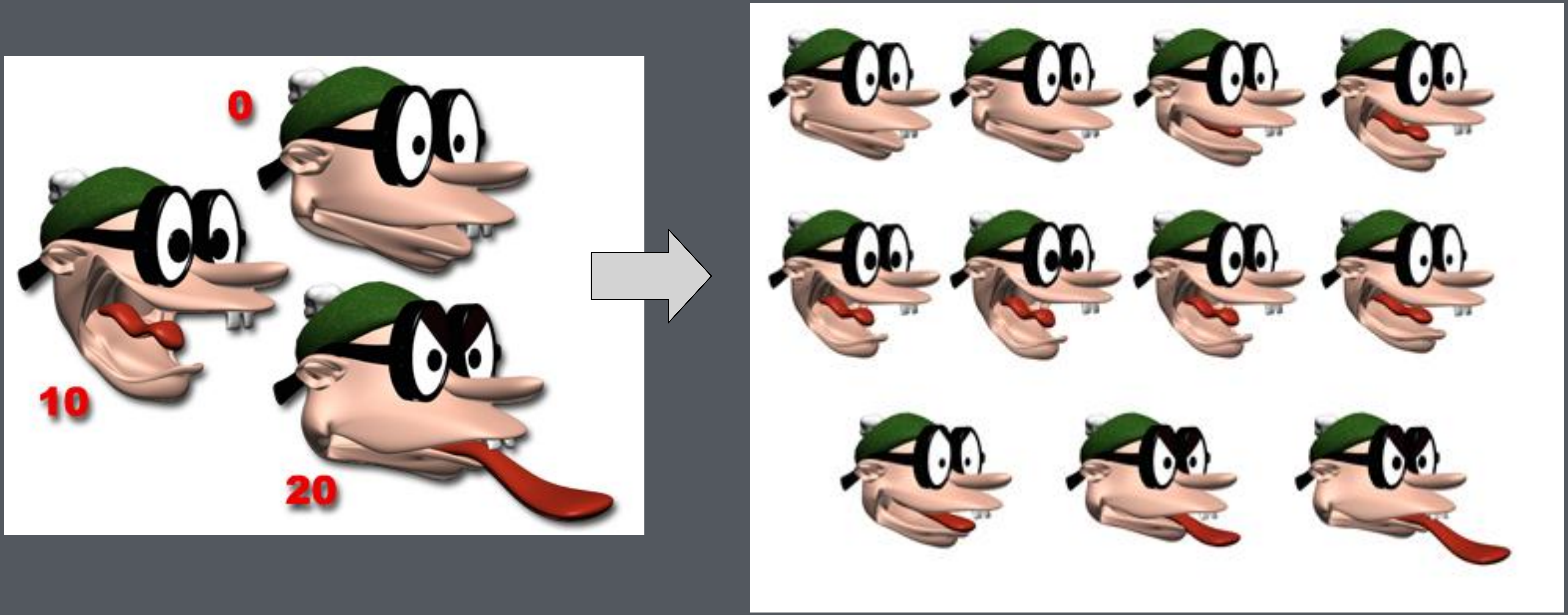
- Easy to implement—first thing to try
- Fast to run—used in games

The simplest tools in the offline animation toolbox

Blend shapes

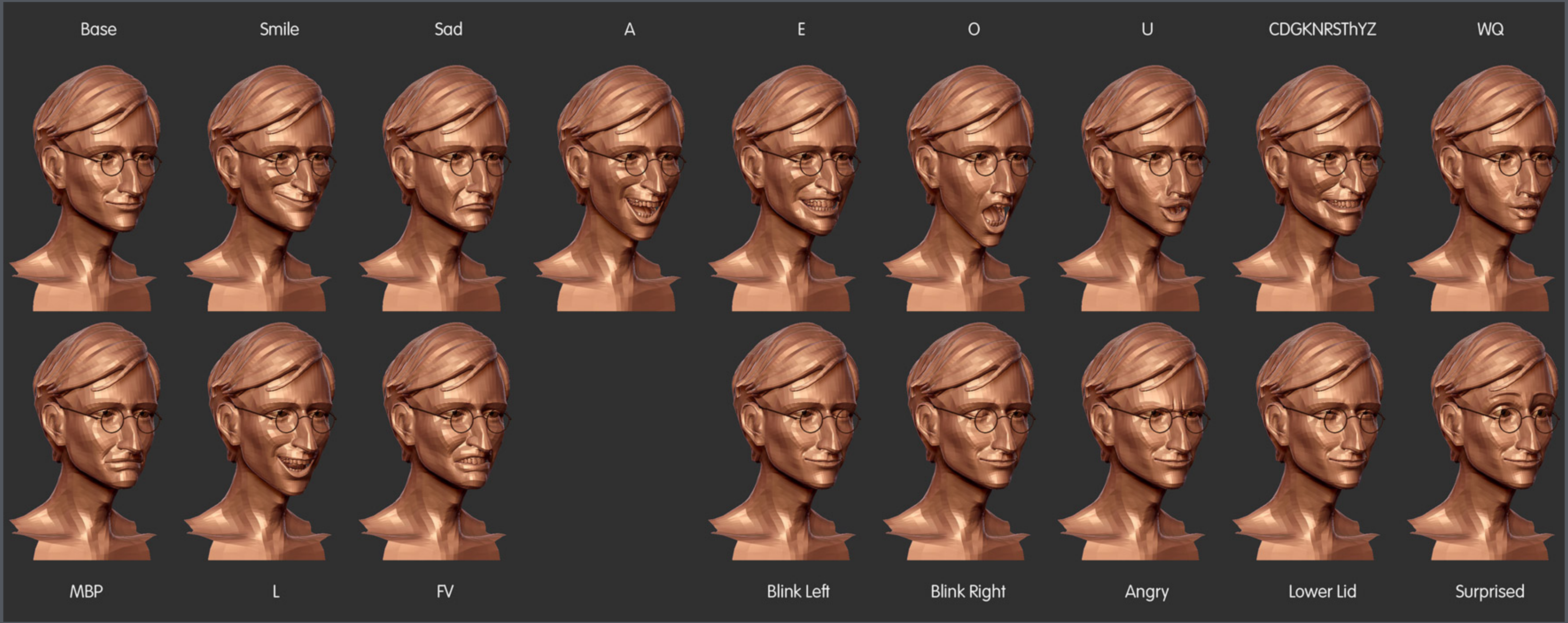
Simply interpolate linearly among several key poses

- Aka. blend shapes or morph targets



[3D Studio Max example]

Blend shapes



José Alves da Silva—*Corlyorn Family* (Vodafone campaign)

Blend shapes math

Simple setup

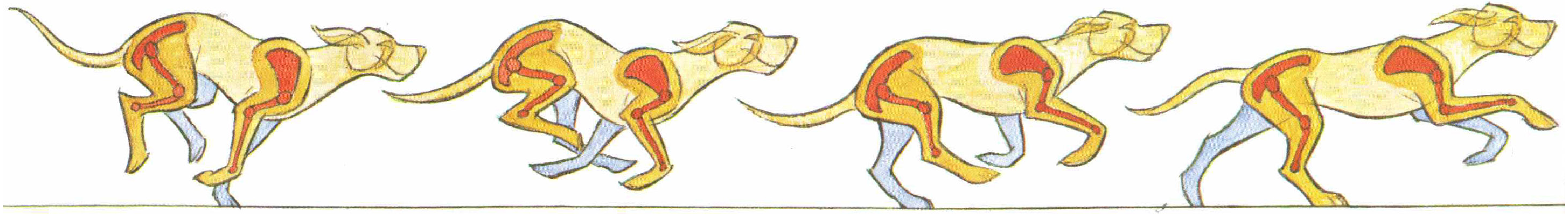
- User provides key shapes: a position for every control point in every shape
  - p_{ij} for point *i*, shape *j*
- Per frame: user provides a weight w_j for each key shape
  - Must sum to 1.0

Computation of deformed shape

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

Works well for relatively small motions

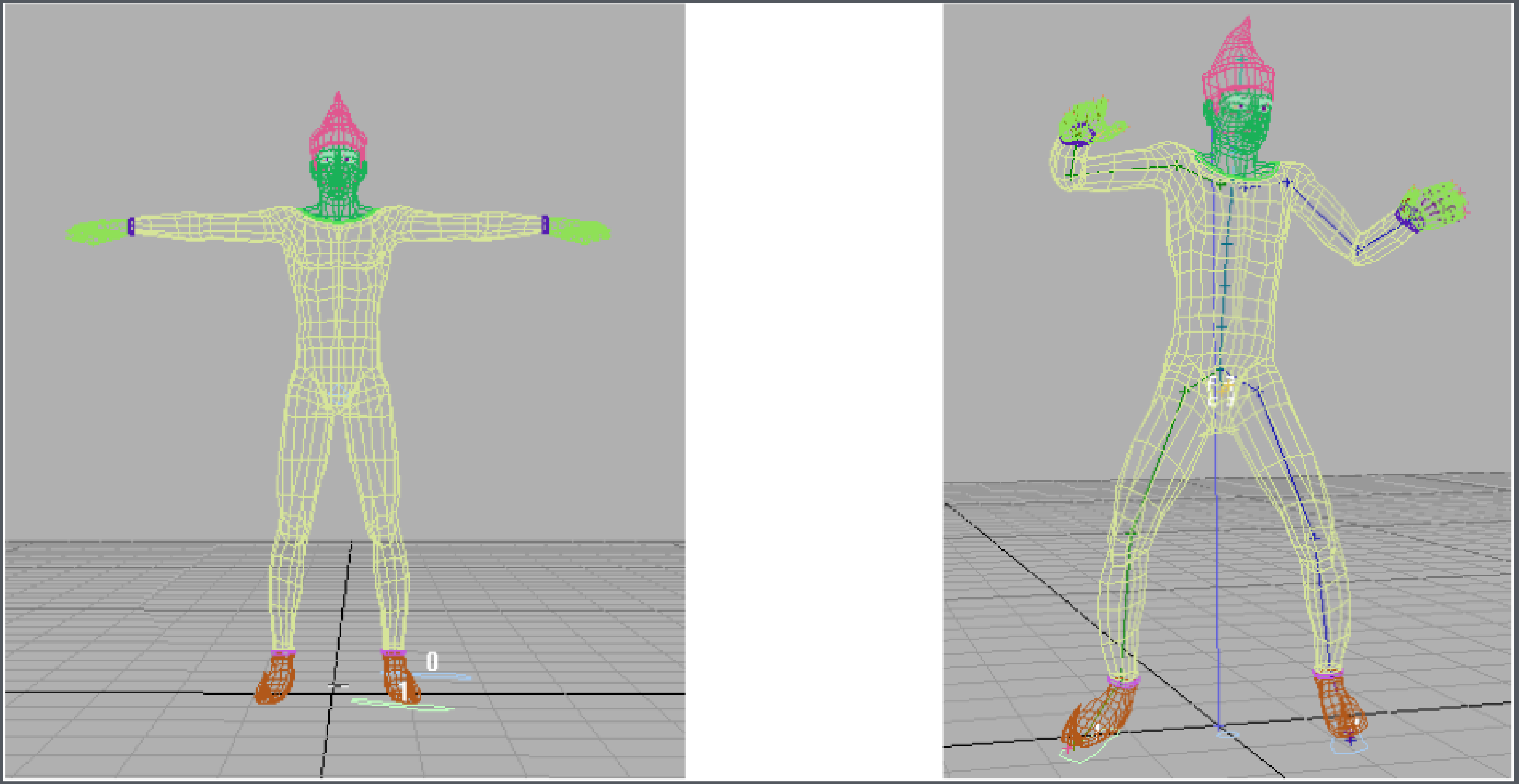
- Often used for for facial animation
- Runs in real time; popular for games



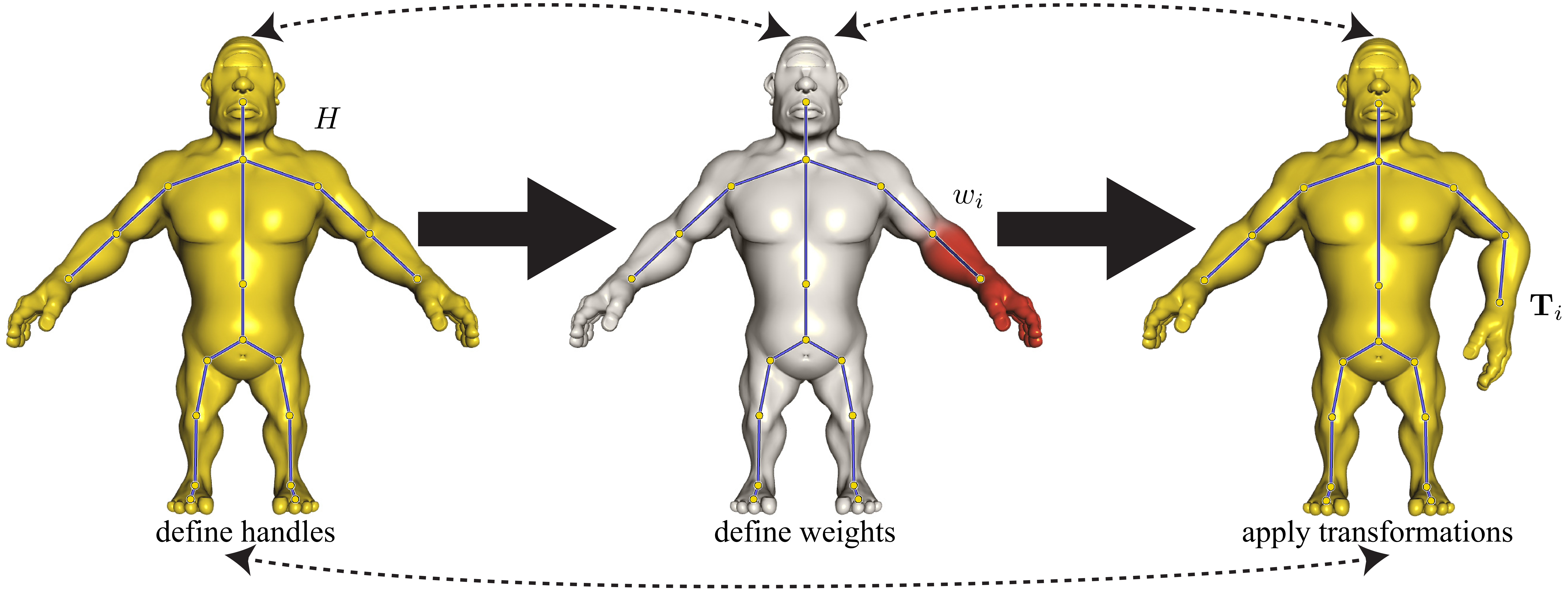
P. Blair, *Cartoon Animation*.

Mesh skinning

A simple way to deform a surface to follow a skeleton



[Sébastien Dominé | NVIDIA]



[Jacobson, SIGGRAPH 2014 course]

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

Points are transformed by a blended transformation

- Various ways to blend exist

Linear blend skinning

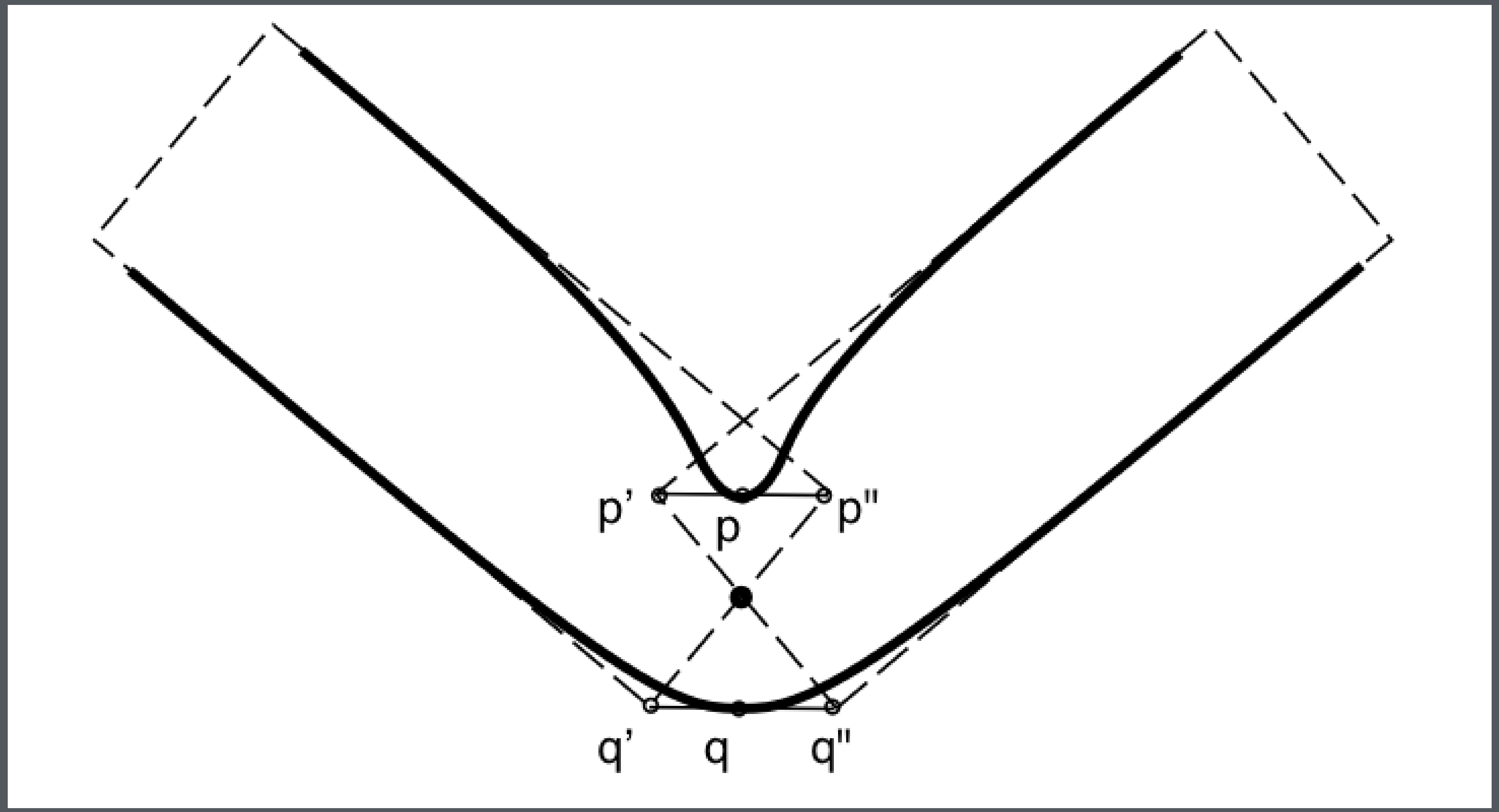
Simplest mesh skinning method

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

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

$$= \left(\sum_j w_{ij} M_j \right) \mathbf{p}_i$$



[Lewis et al. SIGGRAPH 2000]

Linear blend skinning in practice

In practice the bone transformations M_j are not given directly

- animators want to use transformation hierarchies to animate character position
- …and also to animate bones

Character mesh is modeled in a canonical pose called "bind pose"

- chosen for convenience and to keep all parts separated

Skeleton is created first to match bind pose

- this establishes proximity between bones and surface (which can be used to help author weights)

Skeleton is also animated over time

Linear blend skinning in practice

Skinning computations are done in coordinates of skeleton root

- mesh is modeled in these coordinates
- root node of skeleton defines these coordinates

Animated bone matrix *$M_j(t)$* has to operate on points in skeleton root coords

- need transform that carries bone *j* from its bind pose position to its animated position
- bind pose bone xf defined by bind pose xfs of bones: M_j^B
- animated bone xf defined by animated xfs of bones: $M_j^P(t)$
- bone xf in skeleton root coords for skinning equation: $M_j(t) = M_j^P(t)\,(M_j^B)^{-1}$

Deformed mesh is then computed in skeleton root coords

- still needs to be transformed to world coordinates by xfs above skeleton in scene graph

Linear blend skinning

Simple and fast to compute

- Can easily compute in a vertex shader

Used heavily in games

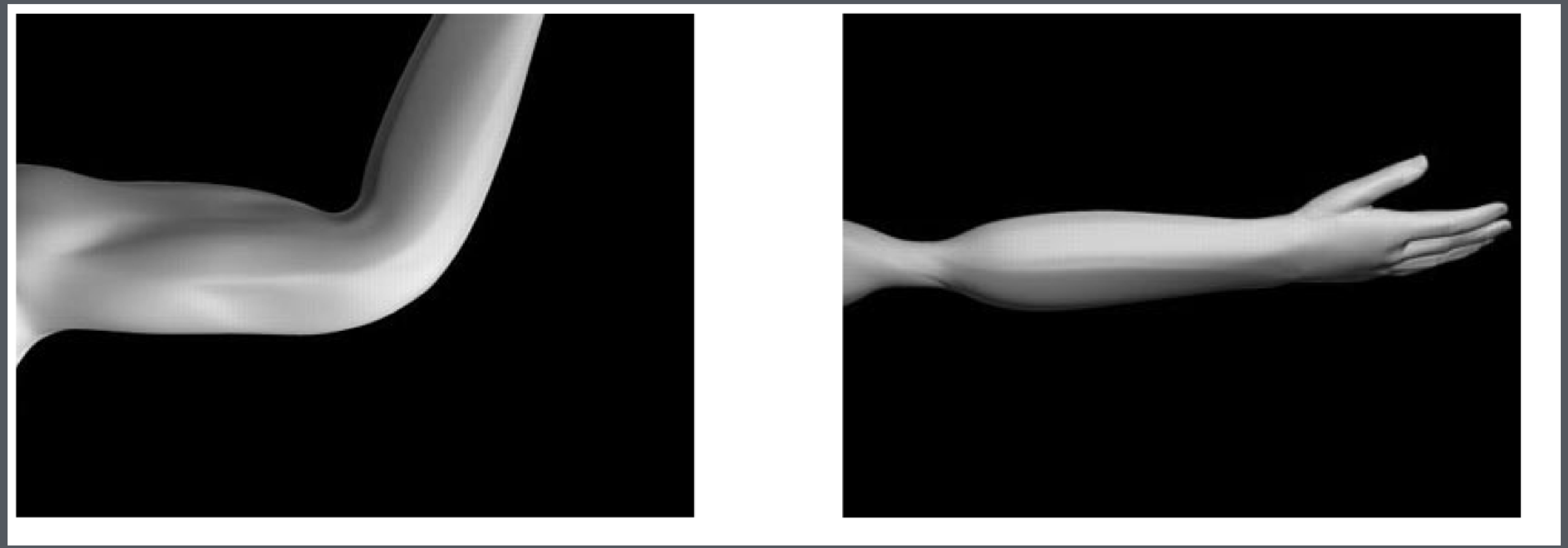
Has some issues with deformation quality

- Watch near joints between very different transforms

Linear 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!



[Lewis et al. SG'00]

[Mohr & Gleicher SG'03]

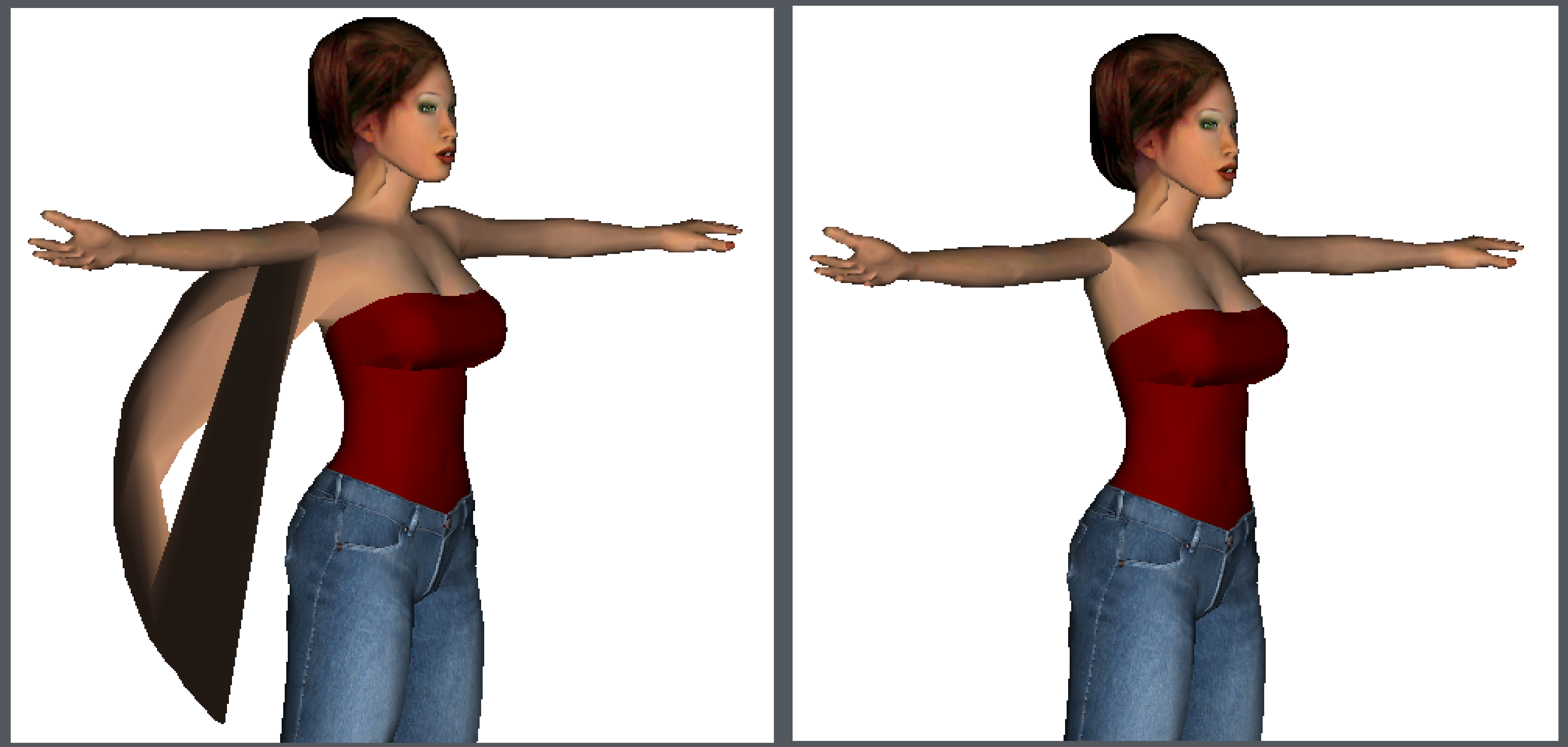
Dual quaternion skinning

Root problem of LBS artifacts: linear blend of rigid motions is not rigid

Blending quaternions is better

- proper spherical interpolation is hard with multiple weights
- just blending and renormalizing works OK

However, blending rotation and rotation center separately performs poorly



[Kavan et al. SG ’08]

Figure 6: Artifacts produced by blending rotations with respect to the origin (left) are even worse than those of linear blend skinning (right).

Dual quaternions

Combines quaternions (1, i, j, k) with dual numbers (1, ε)

- resulting system has 8 dimensions: 1, i, j, k, ε, εi, εj, εk
- write it as sum of two quaternions: $\hat{\mathbf{q}} = \mathbf{q}_0 + \epsilon \mathbf{q}_\epsilon$

Unit dual quaternions

- inherits quaternion constraint: $\|\mathbf{q}_0\| = 1$
- adds one more constraint: $\mathbf{q}_0 \cdot \mathbf{q}_\epsilon = 0$
- a 6D manifold embedded in 8D
- represents rigid motions with nice properties

Skinning by blending dual quaternions works well

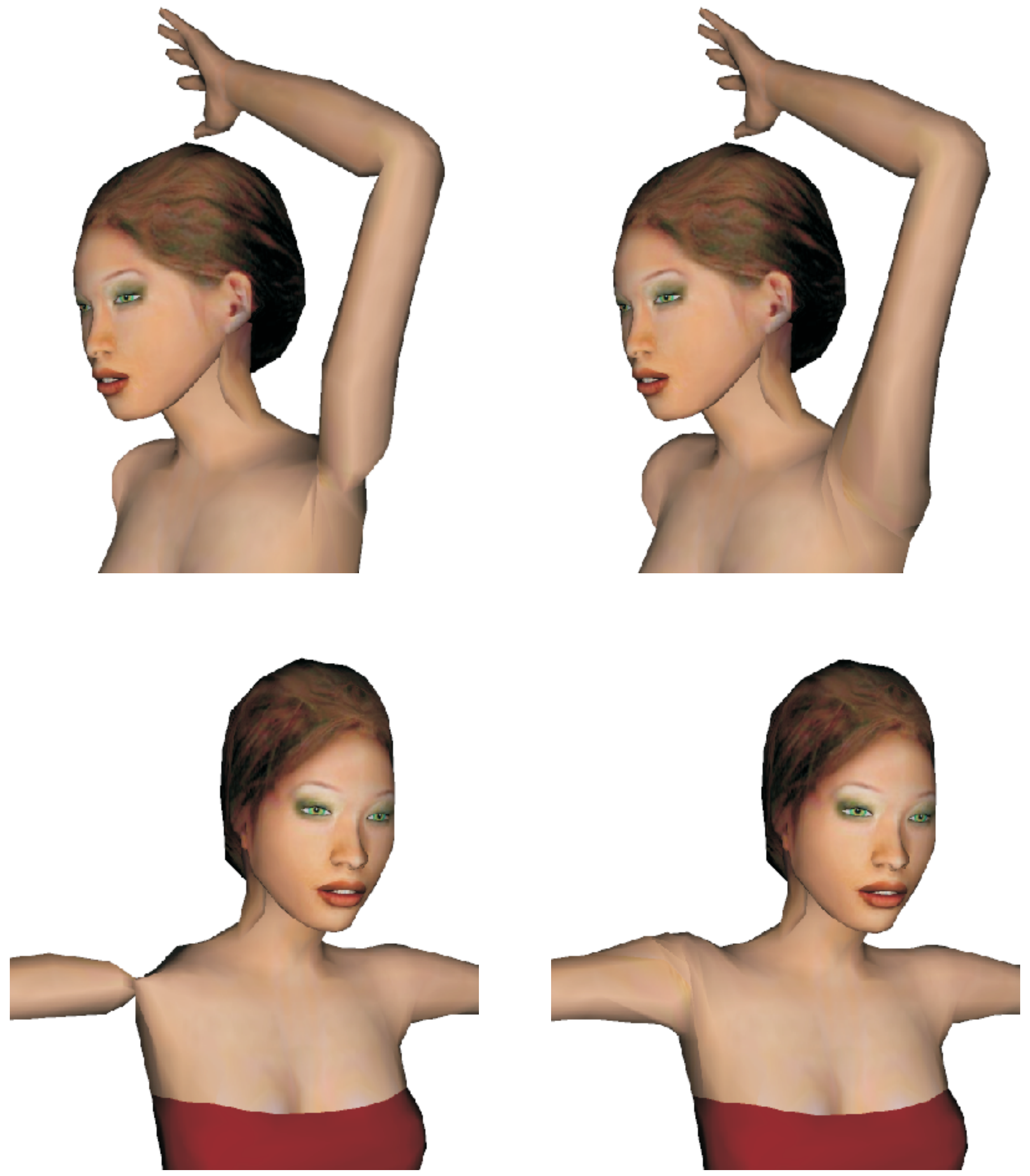
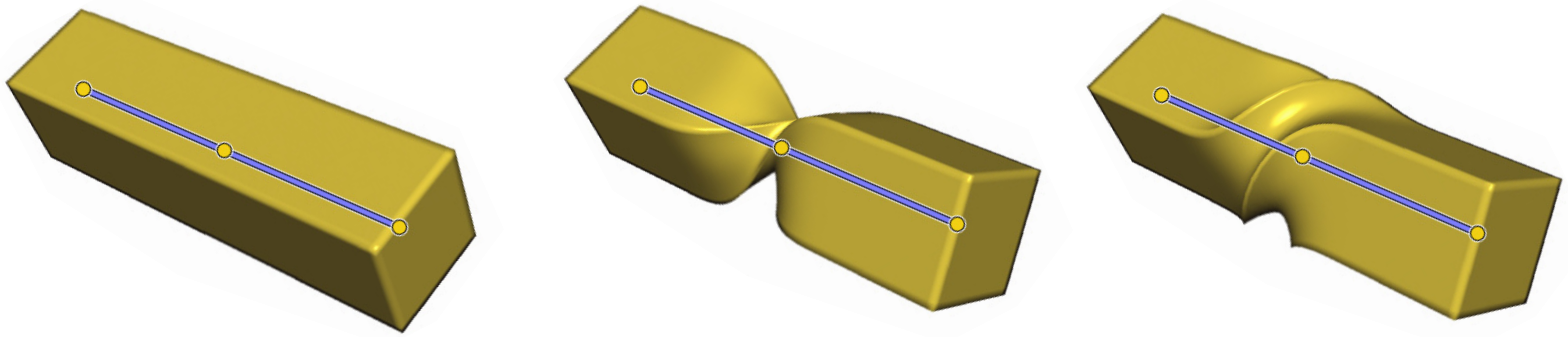


Figure 14: Comparison of linear (left) and dual quaternion (right) blending. Dual quaternions preserve rigidity of input transformations and therefore avoid skin collapsing artifacts.

[Kavan et al. SG '08]



Rest pose | Linear blend skinning | Dual quaternion skinning

[Kavan, SG'14 course]