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
Blend shapes
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

$$p'_i = \sum_j w_j p_{ij}$$

Works well for relatively small motions
- Often used for facial animation
- Runs in real time; popular for games
We can separate automatic skeleton construction into two (not quite independent) tasks: determining the skeletal topology and the skeletal embedding. The topological problem asks how many bones make up the skeleton and how they are connected to each other. The embedding problem asks where bones are situated in space, or, equivalently, where bone joints are located within the shape. These adjustments may even reach back to the construction of the control structure.

A desired pose requires more degrees of freedom than previously imagined, or the center of rotation at a joint may need to be altered to improve a pose. The locomotion of humans and many other animals depends on their internal skeletons. The arrangement of a creature's bones and topology imply the geometry and topology of its inner skeleton. Physical notions into mathematic quantities. The powerful paradigm of constrained energy optimization proves useful for modeling constraints and rest energies.

Traditional riggers construct a 3D skeleton as a hierarchy of directed line-segments. This forms a graphical tree, where nodes are connecting two or more bones. The parent-child relationship is maintained as the skeleton is constructed. Later it will provide a first-order approximation of its outward appearance. Effects due to muscles beneath the skin (e.g. flexing bicep) should not be ignored, but for the sake of our discussion we briefly mention ideas for automatic selection of control points, curves and regions.

Skinning generalizes to other handle types besides skeletons. We will also touch on how cages can be computed automatically, and briefly mention ideas for automatic selection of control points, curves and regions. We instead focus on general methods, in particular, those which utilize geometry. These methods rely heavily on the prior knowledge that the skeleton has a known topology and even a known general state as input. As output, we expect a tree of line segments embedded in the object.
Mesh skinning

A simple way to deform a surface to follow a skeleton
This document is meant to be a living document (though, perhaps not an immortal one). This version was compiled on 8 December 2015. If you find clumsy typos or audacious mistakes, please email alecjacobson@gmail.com.

1 Introduction

The traditional character skinning pipeline is labor intensive. We may break the pipeline into three main steps to see where professional riggers and animators spend the most time, and consequently where modern automatic methods will take over or assist.

Let us recall the linear blend skinning formula so we may track how each step affects a character’s deformation (see Figure 1). The new position of a point $v_0$ on the shape is computed as the weighted sum of handle transformations applied to its rest position $v$:

$$v_0 = \sum_{j=1}^{n} w_j(v) T_j \cdot v.$$  (1)

In the first step, a professional rigger must build a control structure inside, on or around the character. In the most traditional case, this control structure is a hierarchy of rigid bones (directed line segments) referred to as a skeleton. We will see how automatic skinning methods invite new control structures such as cages, loose points, or selected rigid regions. We will refer to the set of bones, points, cage vertices etc. as handles. The set of $n$ handles defines the range of the sum in Equation (1).

Second, the rigger paints skinning weights for each bone. This process is iterative, combining expertise with trial and error. Riggers paint by adding or subtracting weight values, check the effect on a set of canonical pose, adjust the weights, smooth the weights, and repeat. Furthermore, this process is typically conducted using a 2D mouse interface over a perspective projection on a 2D display: many view adjustments interrupt and prolong the painting process. These weights are defined for each handle $j$ at any point $v$ on the shape and enter inside our summation as $w_j(v)$ in Equation (1).

Finally, the animator poses the character by applying transformations to each handle. In the traditional case of a skeleton, these transformations are often stored as relative rotations relating the change of each bone from its “rest” or “identity” pose to its current pose in the basis of that bone’s parent. The absolute transformation may be recovered by following the skeleton’s forward kinematics tree. In our skinning formula, the absolute transformation is the affine matrix $T_j$ that takes handle $j$ from its rest pose in world space to its current pose.
Mesh skinning math: setup

**Surface has control points** $p_i$
- 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

\[ p'_i = \sum_j w_{ij} M_j p_i \]

\[ = \left( \sum_j w_{ij} M_j \right) p_i \]
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
Surface collapses on the inside of bends and in the presence of strong twists

- Average of two rotations is not a rotation!

[Figure 4.1]

[References: Lewis et al. SG’00, Mohr & Gleicher SG’03]
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

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

Unit dual quaternions

- inherits quaternion constraint: $\|q_0\| = 1$
- adds one more constraint: $q_0 \cdot q_\epsilon = 0$
- a 6D manifold embedded in 8D
- represents rigid motions with nice properties

Skinning by blending dual quaternions works well
While this method is straightforward and very common, the normals computed this way are not always a good approximation of the true normals which we would obtain by averaging normals of adjacent triangles. This is because for skinned models (either linear or spherical blend skinning [Kavan and Thalmann 2005] and log-matrix blending [Cordier and Magnenat-Thalmann 2005] and similarly, by blending unit dual quaternions. In either case, the normals are computed as:

\[ \hat{v}_i = M_i^T \cdot \mathbf{v}_i \]

In order to compare computational performance, we have implemented both CPU and GPU versions of dual quaternion skinning. The visual results confirm that our DLB method is indeed free of all artifacts that dual quaternion skinning is more than twice as fast (and also slightly slower than both linear and direct quaternion blending, we believe that this is not a high price to pay for the elimination of artifacts.

For dual quaternions: twist

\[ X_j = (1 - \tau_i) X_j^0 + \tau_i X_j^1 \]

where \( X_j^0 \) and \( X_j^1 \) are the two dual quaternions that describe the transformation of vertex \( j \) with weights \( \tau_i \) and \( 1 - \tau_i \) respectively. As discussed in Section 2.2, some artifacts are better visualized on a simple model spread deformations along the length of a bone.

\[ \mathbf{v}_i = \frac{1}{\sum \tau_i} \sum \tau_i M_i \mathbf{v}_i \]

\[ M_i = \mathbf{R}_i \mathbf{S}_i \mathbf{R}_i^{-1} \]

The challenge of calculating more accurate normals of skinned surfaces has been opened by Merry et al. [2006b] and further refined by Tarini et al. [2014]. The key idea is to assume that skinning weights are continuous functions, which allows us to define the Jacobian of the skinning transformation:

\[ \mathbf{J}_i = \frac{\partial M_i}{\partial \mathbf{v}_i} = \frac{\partial \mathbf{R}_i}{\partial \mathbf{v}_i} \mathbf{S}_i + \mathbf{R}_i \frac{\partial \mathbf{S}_i}{\partial \mathbf{v}_i} \mathbf{R}_i^{-1} \]

If we compute our normals as the gradient of a weight in a particular vertex:

\[ \hat{v}_i = \tau_i \frac{\partial v_i}{\partial \mathbf{v}_i} + (1 - \tau_i) \frac{\partial v_i}{\partial \mathbf{v}_i} \]

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

Figure 15: Comparison of direct quaternion blending (left) and dual quaternion skinning (right). Our implementation of spherical blend skinning [Kavan and Thalmann 2005] produces very natural skin deformations.

A 2D capsule object demonstrating that skinned normals can be a poor approximation of true, geometric normals.