Skinning Mesh Animations

Doug L. James Christopher D. Twigg

Carnegie Mellon University

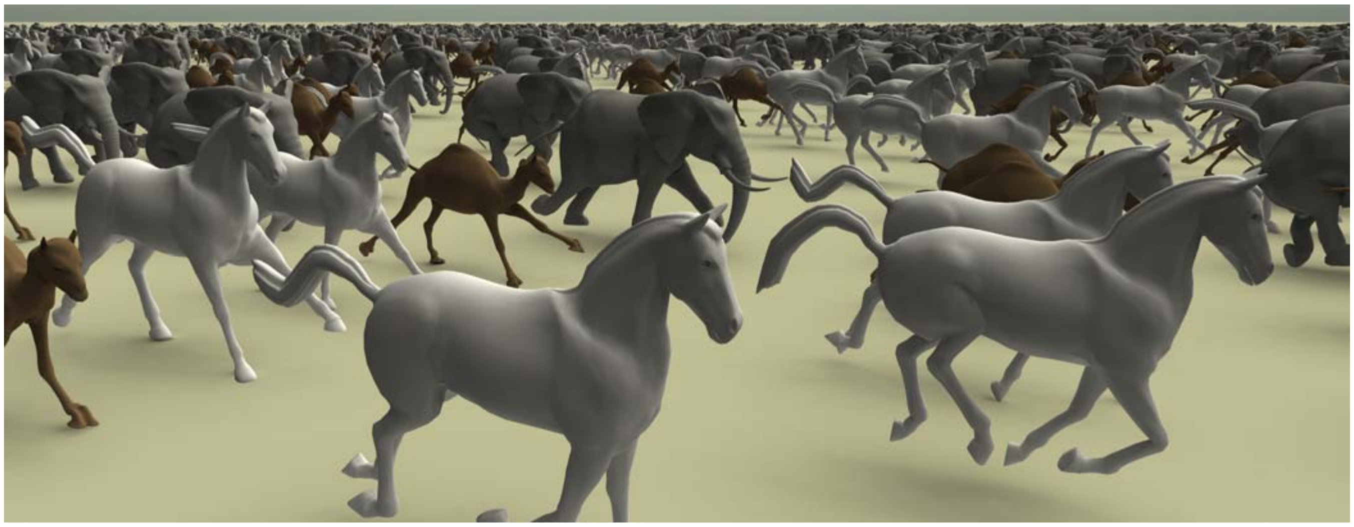


Figure 1: **Stampede!** *Ten thousand skinned mesh animations (SMAs) synthesized in graphics hardware at interactive rates. All SMAs are deformed using only traditional matrix palette skinning with well-chosen nonrigid bone transforms. Distant SMAs are simplified.*

Abstract

We extend approaches for skinning characters to the general setting of skinning deformable mesh animations. We provide an automatic algorithm for generating progressive skinning approximations, that is particularly efficient for pseudo-articulated motions. Our contributions include the use of nonparametric mean shift clustering of high-dimensional mesh rotation sequences to automatically identify statistically relevant bones, and robust least squares methods to determine bone transformations, bone-vertex influence sets, and vertex weight values. We use a low-rank data reduction model defined in the undeformed mesh configuration to provide progressive convergence with a fixed number of bones. We show that the resulting skinned animations enable efficient hardware rendering, rest pose editing, and deformable collision detection. Finally, we present numerous examples where skins were automatically generated using a single set of parameter values.

CR Categories: I.6.8 [SIMULATION AND MODELING]: Types of Simulation—Animation;

Keywords: deformation, skin, compression, mean shift, collision

1 Introduction

Animating articulated characters such as virtual humans is a fundamental operation in computer graphics and interactive applications. Techniques for rigging character skins by weighting vertices to an associated skeleton, or by interpolating example deformations, are widely used in video games and the computer animation industry. There are numerous reasons for their popularity: most skinning approaches are conceptually easy to understand and apply; they are capable of approximating interesting character shapes; and skinning can be hardware-accelerated on almost every commodity graphics card. However, the application of character skinning approaches has been almost entirely limited to objects with user-defined skeletons and rigid bones.

In this paper, we show that skinning techniques can also be used to automatically skin deformable mesh animations, without any need for specifying skeletons or bones. By estimating proxy bone transformations and vertex weights for deformable shape sequences, we produce skinned animations that approximate the original deformable animation. Note that we do not estimate a hierarchical kinematic skeleton, nor are there reconfigurable joint angles. Skinned meshes provide an approximate articulated mapping from undeformed to deformed geometry at all animation frames, and we further encode displacement corrections to provide progressive convergence. The resulting skinned animations support hardware-accelerated rendering on commodity graphics hardware. We present several examples of mesh animations that can be compiled to run on existing graphics hardware, including interactive examples involving millions of animated mesh triangles (see Figure 1). In addition to fast hardware rendering, skinned mesh animations provide a deformable shape parameterization that can aid subsequent processing: displacement mapping the skin's reference mesh results in modifications to the entire animation; the animation skin can be used for straight-forward level-of-detail (LOD) generation; and the skinned models define local reference frames that can be used to orient bounding volume hierarchies for reduced-coordinate collision detection algorithms [James and Pai 2004]. In summary, we show that a simple animation skinning technique can provide a familiar hardware-accelerated rendering format for mesh animations, as well as enable efficient animation processing.

2 Related Work

Character skinning has a long history in graphics, and related techniques have been used extensively to provide intuitive animation controls. We refer the reader to the related work section of [Lewis et al. 2000] for a nice summary. Most closely related to our work are several "skinning by example" approaches, that use input character poses to generate approximate and/or progressive skinning approximations. A key difference is that our input mesh examples are not required to have any skeletons, nor associated bone transforms.

Pose space deformation [Lewis et al. 2000] comprises one such family of approaches wherein character shapes (or local frame corrections) are interpolated as a function of character pose (see also [Allen et al. 2002]). For hardware rendering, EigenSkin is an approach for generating a compact pose space deformation model for complex shapes, such as a hand [Kry et al. 2002]. In that work, the reduced basis is generated by employing sensitivity analysis to move each joint in the hand independently, about a number of key hand poses, in order to infer the spatial structure of the hand displacement fields as a function of hand pose. A displacement correction model was then compressed in the rest pose to exploit redundancy. The resulting articulated reduced model included displacements and normals, and exhibited significant rank-reduction (much better than PCA on articulated data sets), but required a priori knowledge of joints and skinning weights. In comparison, our approach applies to general mesh animations as opposed to poseable articulated characters (see Figure 2).

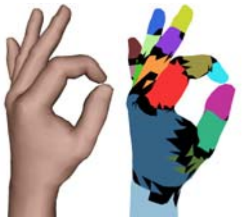


Figure 2: **One of 27 hand poses** *with core bone triangles for 17 estimated bones (*$\varepsilon = 0.05$*). Although our method can skin mesh sequences with pre-existing skeletons (§5), it's strength lies in optimizing non-hierarchical bone transforms for each frame (§4), and avoiding pose space interpolation of progressive skin corrections (§6).*

In contrast to pose space methods, simpler kinematic models, such as common *linear blend skinning* (c.f. *single-weight enveloping (SWE)*, *skeletal subspace deformation (SSD)*, etc.), can be fit to mesh examples, and do not require runtime data interpolation. Mohr and Gleicher [2003] estimated single-weight skins with rigid character bones, with provisions made for adding additional bones, and they also estimated vertex weights, but without nonnegativity constraints. *Multi-weight enveloping (MWE)* [Wang and Phillips 2002] provides better approximations than SWE, but at the cost of 12 weights/vertex/bone, instead of 1 weight/vertex/bone in SWE. Although we only use 1 weight/bone/vertex, our nonrigid affine bone transformations allow bones to squash and stretch to better approximate the mesh sequence. Like other approaches, MWE requires manual intervention to paint on bone influences, whereas our skin estimation approach is automatic.

Animation compression approaches have been explored in the last half decade [Lengyel 1999; Briceño et al. 2003; Guskov and Khodakovsky 2004], including PCA compression of shapes [Sloan et al. 2001], animations [Alexa and Müller 2000; Karni and Gotsman 2004], and parameterized animation compression [Hakura et al. 2000]. The goal of our approach is not animation compression per se, but rather hardware-accelerated rendering using a very simple and common vertex shader technique called *matrix palette skinning* [Lindholm et al. 2001]. Our skinned mesh animations are intended to produce skinned approximations with a large number of bone transforms (to support efficient hardware rendering). Nevertheless, our approximations can be quite good (even without using our articulated data reduction), so that our animation compression is quite competitive. Nevertheless, incremental coding approaches, such as [Gupta et al. 2002; Ibarria and Rossignac 2003], can be better suited to compression, and could be combined with our approach. Multi-resolution methods [Guskov and Khodakovsky 2004] provide powerful decorrelation tools, but involve multi-resolution mesh reparameterization that can be difficult for models with complex base domains, e.g., our high-genus bridge example.

Finally, special data structures exist for representing and displaying mesh animations [Shamir et al. 2000], however for real-time rendering, the bottleneck is often data transfer to the graphics card. Our approach avoids this bottleneck by having a single-weighted mesh resident in graphics hardware, and only sending a few hundred floats of bone transforms to hardware each frame.

3 Skinning Mesh Sequences

We seek to construct a vertex skinning transform, $\mathsf{T} = (\mathsf{T}_i)$, that approximates a sequence of input meshes. A key point is that the mesh sequences we approximate do not have predefined bone transformations, and therefore we must first estimate proxy bone transforms before estimating vertex weights.

Let the sequence of S meshes have deformed vertex positions $\mathbf{P} = (\mathsf{p}^1, \mathsf{p}^2, \ldots, \mathsf{p}^S)$, where $\mathsf{p}^t \in \mathbb{R}^{3N}$ for N vertices. For convenience, we will refer to the index t as "time," even though the sequence may not have a time interpretation. Given these meshes, we seek a skinning transformation T^t at each sequence step, t, that transforms the undeformed rest pose points, $\tilde{\mathsf{p}}$, to approximate p^t:

$$\mathsf{p}^t \approx \mathsf{T}^t \tilde{\mathsf{p}}, \quad t = 1 \ldots S, \tag{1}$$

with similar equations possible for vertex normals. We use linear blend skinning, where the transform for vertex i is

$$\mathsf{T}_i^t = \sum_{b \in \mathscr{B}_i} w_{ib} \bar{\mathsf{T}}_b^t, \tag{2}$$

but where the transforms $\bar{\mathsf{T}}_b^t$ need not be rigid. In the following sections, we first show how to compute robust estimates of proxy bone transformations, $\bar{\mathsf{T}}_b^t$, for our otherwise skeleton-free meshes (§4), and then (in §5) we estimate the skin's vertex-bone dependencies, $\mathscr{B}_i$, and corresponding vertex bone weights, w_{ib}. Note that if $\bar{\mathsf{T}}_b^t$ are already given, e.g., by a kinematic skeleton model, one may proceed directly to Section 5.

4 Identifying Near-Rigid Structures using Mean Shift on Rotation Sequences

Our first goal is to estimate proxy bone transformations $\{\bar{\mathsf{T}}_b^t\}$ for the mesh sequence. Our insight is that clustering triangles with similar rotation sequences reveals the near-rigid structure of the mesh animation. For reasons discussed in detail in §4.2, we use a nonparametric clustering approach based on the mean shift algorithm (see [Cheng 1995; Comaniciu and Meer 2002] for details). Note that the goal of this bone estimation section is not to partition the mesh into near-rigid components, but rather to estimate bone transformations $\{\bar{\mathsf{T}}_b^t\}$ for the mesh sequence.

4.1 Triangle Rotation Sequences

We estimate the near-rigid structure by identifying statistically significant rotation sequences of the mesh triangles. The rotation of triangle j at instant t relative to its rest configuration is computed using the Polar Decomposition [Shoemake and Duff 1992; Golub and Loan 1996], which is also common in large deformation kinematics (see [Etzmuss et al. 2003; Müller and Gross 2004]).

Let the indices of the triangle's vertices be i_1, i_2, and i_3, then the 3×3 nonorthogonal orientation matrix of triangle j at time t is $O_j^t = [\vec{v}_{21}\ \vec{v}_{31}\ \hat{n}]$ where $\vec{v}_{ab} = (\mathsf{p}_{i_a}^t - \mathsf{p}_{i_b}^t)$ and $\hat{n} = (\vec{v}_{21}\times\vec{v}_{31})/\|\vec{v}_{21}\times\vec{v}_{31}\|_2$. Similarly, let the nonorthogonal orientation matrix of rest triangle j be $\tilde{O}_j$. From the triangle's deformation gradient $F_j^t = O_j^t\left(\tilde{O}_j\right)^{-1} \in \mathbb{R}^{3\times3}$ (at time t), we extract the intrinsic triangle rotation, R_j^t, using the polar decomposition, $F_j^t = R_j^t W_j^t$ (see Figure 3). Here $W_j^t \in \mathbb{R}^{3\times3}$ is the symmetric right stretch matrix that stretches the triangle before it is rotated.

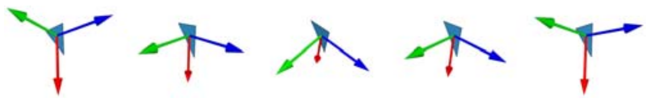


Figure 3: **Triangle rotation sequences** *as estimated using the polar decomposition. Rotation sequences represent each triangle motion as a high-dimensional point for subsequent mean shift clustering to estimate near-rigid components.*

Given the sequence of 3×3 relative rotation matrices for triangle j, we construct a point in $d = 9S$ dimensions, $\mathsf{z}_j \in \mathbb{R}^{9S}$, composed of concatenated rotation sequence vectors,

$$\mathsf{z}_j = \left(\mathrm{vec}(R_j^1),\ \ldots,\ \mathrm{vec}(R_j^S)\right) \tag{3}$$

where $\mathrm{vec}(R) : \mathbb{R}^{3x3} \to \mathbb{R}^9$ converts the row-ordered 3×3 rotation matrix, R, to a row-major 9-vector. We now estimate the statistically important near-rigid components by carefully clustering the triangle rotation sequence points, $\{\mathsf{z}_j\}$.

4.2 Mean Shift Clustering of Rotation Sequences

To estimate proxy bone transformations we seek to find (a) triangles that have similar rotation sequence points z, and also to discover (b) the number of different proxy bones required to resolve rotation sequence differences to some tolerance. Triangles can be partitioned using any number of clustering algorithms, but the quality of the resulting clusters for arbitrary skinning problems is unclear. Also, finding the right number of clusters for a certain skinning error can be cumbersome, e.g., using k-means clustering where "k" must be specified by the user. Given that we want to automatically skin a wide variety of input mesh sequences, we desire a robust clustering approach for estimating bone components.

Background on Mean Shift Clustering: Given a list of n points, $\mathsf{z}_j \in \mathbb{R}^d$, the fixed bandwidth mean shift algorithm constructs the sample point estimator (density function),

$$f(\mathsf{z}) = \frac{1}{n}\sum_{j=1}^{n} k\left(\left\|\frac{\mathsf{z}-\mathsf{z}_j}{h}\right\|^2\right), \tag{4}$$

where h is the bandwidth of some spherically symmetric kernel function, $k(\cdot)$. The gradient of this density function $f(\mathsf{z})$ is then efficiently computed and used in a hill climbing process to map each input point, z_j, to the nearest stationary point, $\bar{\mathsf{z}}_j$, of the density function (see Figure 4). For robust nonparametric clustering, these resulting modes, $\bar{\mathsf{z}}_j$, can be used to select cluster shapes using basins of attraction, and can therefore have very nontrivial shapes–unlike k-means clustering where points are simply assigned to the nearest cluster center. The single bandwidth parameter, h, allows the number of clusters to be chosen in terms of a length scale in the input point space.

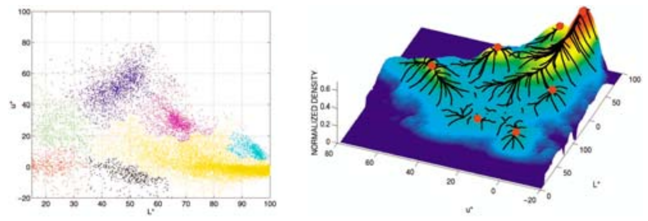
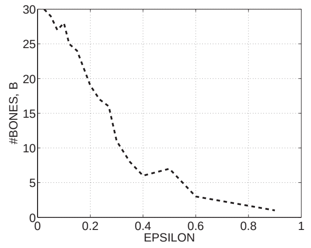


Figure 4: **Mean shift clustering of 2D points** *(images courtesy of [Comaniciu and Meer 2002] ©IEEE 2002) (Left) Input 2D points, with color-coded cluster output; note the interesting oblong cluster shapes. (Right) Related density field, with trajectories from the mean shift gradient ascent algorithm. Red dots indicate the final mode centers used for proximity-based classification of mean-shifted points.*

Why Choose Mean Shift for Bone Estimation? The mean shift clustering algorithm (see [Cheng 1995; Comaniciu and Meer 2002]) has many attractive features for bone estimation for skinning. First, mean shift clustering is *robust to outliers*, unlike k-means clustering which can be corrupted by a few bad points when computing each cluster's mean position. Robust approaches to compute averages are common when handling noisy real-world data, e.g., [Jones et al. 2003]. For bone estimation, robustness is important because mesh sequences can have many triangles with independent and/or atypical motions, such that their rotation sequence z points are statistical outliers.

Second, mean shift allows the number of clusters to be specified indirectly in terms of a physical scaling parameter, h, related to how different triangle rotations must be in order to belong to different bones. Specifically, we define a *single intuitive rotation tolerance parameter, ε,* such that $h \equiv 9S\varepsilon$ for an S-mesh sequence, and mean shift automatically determines the correct number of clusters. In fact, except where stated explicitly, *we skinned all examples in this paper using a single parameter value*: $\varepsilon = 0.05$. With this single parameter value, we found that mean shift identified characteristic clusters of various number, size and shape for all examples (see Figure 6), supporting our claim that the method automatically produces reasonable results without excessive parameter tuning. Furthermore, we found that values around $\varepsilon = 0.05$ tend to extract the most bones that would be useful, and decreasing ε does not tend to produce statistically significant clusters. In practice, fewer bones may be desired, e.g., for real-time rendering, and ε can be increased to reduce bones, at the cost of increasing skinning error.

Figure 5: **Number of bones, B, versus** ε *(horse example) shows the tendency to reduce bones with increasing rotation tolerance, ε. We observe that $\varepsilon = 0.05$ identifies useful near-rigid bones structure to skin almost all examples.*



One practical complication for mean shift clustering of rotation sequences represented as points in $d = 9S$ dimensions, is that efficient multidimensional range searching is required to find closest neighbors of a point to evaluate (4). Consequently, we recommend the approach taken by [Georgescu et al. 2003], wherein locality-sensitive hashing (LSH) is used for fast approximate nearest neighbor searches, and we use their publicly available mean shift implementation (see reference for website).

Associating Triangles to Core Bones: Given the input points $\{\mathsf{z}_j\}$, and h, the output of the mean shift clustering algorithm is each triangle's mean-shifted point, $\bar{\mathsf{z}}_j$, and B statistically significant modes, $\{\tilde{\mathsf{z}}_b\}_{b=1}^{B}$ associated with B discovered bones. Triangle j can then be associated with the bone whose mode $\tilde{\mathsf{z}}_b$ is closest to their mean-shifted point, $\bar{\mathsf{z}}_j$, with filtering performed for outlier points that are very far from statistically significant modes. In our implementation, we only assign triangles to their closest bone if $D_j \equiv \|\tilde{\mathsf{z}}_b - \bar{\mathsf{z}}_j\|_1 < h/4$, and *let $\mathscr{T}_b$ denote the core bone triangles strongly associated with bone b.* The total fraction of triangles associated with bones is denoted as the *near-rigid fraction, $\eta \in (0,1]$* ($\varepsilon = 0.05$ assumed hereafter), and are displayed as colored (non-black) triangles (see Figure 6). Finally, mesh sequences with higher η values tend to yield better skinning approximations.

4.3 Estimating Rigid Bones

Mean shift helps identify the number of bones, B, and the core triangles $\{\mathscr{T}_b\}_{b=1}^{B}$ most strongly associated with each of the B bones. It remains to estimate the average rotation of the core bone triangles, $\mathscr{T}_b$, at each mesh sequence, t. Unfortunately, the cluster mode centers, $\{\tilde{\mathsf{z}}_b\}_{b=1}^{B}$, are just vectors in $\mathbb{R}^{9S}$, and the extracted 3×3 matrix sequences are not even rotation matrices of $SO(3)$, and are therefore unsuitable for direct use as proxy bone rotations.

To estimate bone rotations, we use a convenient result by Moakher [2002], who proves that a meaningful *average rotation* can be computed as the arithmetic mean $\bar{R}$ of triangle rotations, followed by the unique projection onto $SO(3)$ given by the unique polar factor in the polar decomposition of $\bar{R}$ [Golub and Loan 1996]. This average rotation is meaningful provided that $\det(\bar{R}) > 0$. In practice, we observe that this latter condition is satisfied when $\varepsilon \ll 1$ for $h = \varepsilon 9S$. For very large ε choices, $\det(\bar{R}) > 0$ is not guaranteed, e.g., if $\bar{R}$ is the average of all mesh triangle rotations when $B = 1$. We use the area-weighted average triangle rotation, which is also easy to compute [Moakher 2002]. Finally, we note that averages of spherical quantities, such as quaternions, have been discussed in the graphics community for some time (see [Buss and Fillmore 2001]).

Lastly, given the rotation matrix for bone b at time, t, the translation component of the bone transformation $\bar{\mathsf{T}}_b^t$ is estimated using an area-weighted least squares fit to the motion of the centroids of the core bone triangles, $\mathscr{T}_b$.

4.4 Estimating Flexible Bones

Unlike in character animation where bones are clearly rigid skeletal components, there is no reason that our bones can not flex at each frame. Since much greater accuracy can be achieved with flexible bones, and mesh sequences are free to define "bones" as they wish, we advocate flexible bones in our implementation. Mathematically, instead of representing each bone transform $\bar{\mathsf{T}}$ by a rotation/translation pair, (R, v), we instead use (F, v) where $F = RW$ is a 3×3 matrix involving a stretch in addition to a rotation, and v is a (different) translation. Given the core bone triangles, $\mathscr{T}_b$, we use least squares to estimate the flexible bone transform sequence (F^t, v^t) that minimizes the distance from triangle centroids of the mesh sequence, and the reference model transformed using (F, v). This minimization can be done independently for each frame, and each bone, and the details are provided in Appendix A.

Discussion: Given that we prefer flexible bones, one might wonder why we cluster R when F is ultimately required. Why not cluster F sequences? The answer is that when a triangle is used to estimate an $F = RW$ matrix, the right stretch tensor, W, involves an incomplete observation of pre-stretch normal to the undeformed triangle. Consequently, triangles of different orientation will estimate different F matrices, even if they undergo the same spatial stretch and rotation. Therefore, rotations are a more suitable quantity to use to estimate near-rigid components with clustering.

5 Skin Estimation

Given a mesh sequence and matching proxy bone transformations, $\{\bar{\mathsf{T}}_b^t\}$, we now show how to estimate which bones can influence each vertex, and what the corresponding vertex weights are. We stress that the skin estimation procedure (in this section) accepts as input arbitrary bone transforms, such as those of a kinematic skeleton model, and is in no way restricted to mean shift bone estimation.

5.1 Estimating Bone Influences

Let $\mathscr{B}_i$ be the *vertex-bone influence set* containing bone indices that influence vertex i. In our implementation the *maximum number of per-vertex bones*, β, is specified by the user, since it depends on the intended application. For example, in hardware rendering it is favorable to keep the number of bones per-vertex relatively small, e.g., $\beta = 4$. Specifying the bones that influence each vertex is traditionally a task often performed by 3D artists, e.g., by so-called painting bone weights on a mesh. Assigning bones to each $\mathscr{B}_i$ influence set can be a difficult discrete model selection problem since it is often constrained by the maximum number of bones per vertex, β, or conditions on smoothness of the resulting mesh.

We use a simple but effective model selection approach that picks the best β bones for vertex i that have the smallest square errors when used to individually predict deformed positions. Specifically, bone b's sum of squared errors, when predicting vertex i's position sequence, is

$$\gamma_{bi} = \sum_{t=1\ldots S} \|\mathsf{p}_i^t - \bar{\mathsf{T}}_b^t \tilde{\mathsf{p}}_i\|_2^2, \quad b = 1\ldots B. \tag{5}$$

Because γ is only dependent on spatial quantities, bone influence sets also tend to have coherent spatial variations.

5.2 Estimating Vertex Weights

Given vertex-bone influence sets, $\{\mathscr{B}_i\}$, the associated weights are computed using a least squares approach. Least squares is often used for vertex weight estimation in "skinning by example" approaches, e.g., [Wang and Phillips 2002; Mohr and Gleicher 2003].

Weights are constrained by the mesh sequence approximation equations, $\mathsf{T}^t \tilde{\mathsf{p}} = \mathsf{p}^t$, which results in the over-constrained system of S equations with $B = |\mathscr{B}_i|$ unknowns,

$$\sum_{b \in \mathscr{B}_i} (\bar{\mathsf{T}}_b^t \tilde{\mathsf{p}}_i) w_{ib} = \mathsf{p}_i^t, \qquad t = 1\ldots S, \tag{6}$$

which is of the form

$$\mathbf{A}^{(i)} \mathbf{w}^{(i)} = \mathbf{b}^{(i)}, \quad i = 1\ldots N, \tag{7}$$

where for N vertices we have N different matrix problems. Additionally, we would like to enforce the affine constraint,

$$\sum_b w_{ib} = 1, \tag{8}$$

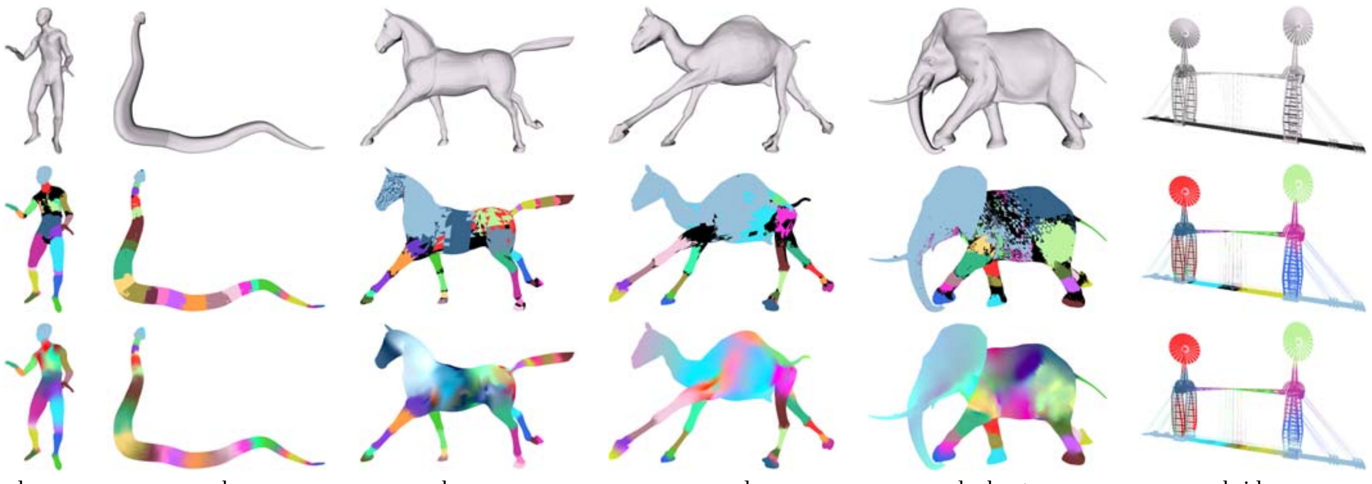


Figure 6: **Estimation of near-rigid bones and vertex weights:** *(Top) Reference triangle mesh of mesh sequence. (Middle) Estimated bones using mean shift classification of triangle rotation sequences into clusters. All examples use mean shift parameter, $\varepsilon = 0.05$. Here black denotes statistical outliers that are typically nonrigid triangles, e.g., shoulders (dance), floppy tails (camel), or bending joints. (Bottom) Vertex weighting to bones (TSVD weights).*

which could be done using constrained least squares. However, for reasons that will become clear shortly, in our implementation we will just consider the augmented system,

$$\begin{bmatrix} c\mathbf{A}^{(i)} \\ 1\dots1 \end{bmatrix} \mathbf{w}^{(i)} = \begin{pmatrix} c\mathbf{b}^{(i)} \\ 1 \end{pmatrix} \quad \Leftrightarrow \quad \tilde{\mathbf{A}}^{(i)}\mathbf{w}^{(i)} = \tilde{\mathbf{b}}^{(i)}, \qquad (9)$$

where $c = 1/\|\mathbf{b}^{(i)}\|_2$ is a scaling parameter, and the last row of ones has been added to account for (8).

Weight Over-Fitting: Direct solution of (7) (when A is nonsingular) can result in weights with potentially large positive and negative values. This is often referred to as over-fitting. Over-fitting can produce the best weights for the mesh sequence, and is adequate if only that mesh sequence is to be displayed. However, if the skinned model is to be able to approximate new poses (using new transforms), or support progressive corrections (as in §6), or allow rest pose editing (see Figure 15), or be simplified, then large (negative) weights should be avoided.

Computing Vertex Weights: We solve (9) using two different types of least squares methods:

1. **TSVD:** For cases where over-fitting is allowed, we solve (9) using Truncated Singular Value Decomposition (TSVD), truncating singular values, e.g., at $10^{-5}\|\tilde{\mathbf{A}}^{(i)}\|_2$, to avoid severe ill-conditioning. Higher thresholds can reduce over-fitting, but negative weights still occur at practical thresholds.

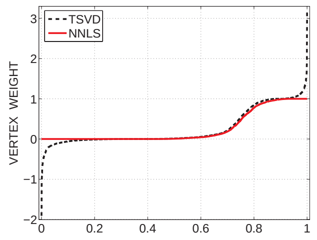
2. **NNLS:** To limit over-fitting and avoid regularization parameters entirely, we recommend positive vertex weights obtained by solving the nonnegative least squares (NNLS) problem,

$$\text{Solve} \quad \tilde{\mathbf{A}}^{(i)}\mathbf{w}^{(i)} = \tilde{\mathbf{b}}^{(i)} \quad \text{subject to} \quad \mathbf{w}^{(i)} \geq 0. \qquad (10)$$

We use the efficient NNLS iterative solution approach of [Lawson and Hanson 1974]. Because of the nonnegativity constraint, positive weights tend to be sparse. In general, we can bound negative weights, $\mathbf{w}^{(i)} \geq -\delta \leq 0$ by substituting $\mathbf{w}^{(i)} = (\tilde{\mathbf{w}}^{(i)} - \delta)$ into (9), and solving the modified problem

$$\text{Solve} \quad \tilde{\mathbf{A}}^{(i)}\tilde{\mathbf{w}}^{(i)} = \tilde{\mathbf{b}}^{(i)} + \tilde{\mathbf{A}}^{(i)}\delta \quad \text{subject to} \quad \tilde{\mathbf{w}}^{(i)} \geq 0. \qquad (11)$$

Figure 7: **Plot of sorted vertex weights** *computed using TSVD and NNLS (chicken example). TSVD allows negative weights that can lead to over-fitting. NNLS optimization provides a simple, efficient, and parameter-free way to get positive vertex weights that avoid over-fitting.*



Finally, we always normalize $(w_{ib})_{b\in\mathscr{B}_i}$ by its sum to ensure (8), with the resulting correction being negligible in all of our examples. Note that by not using the affine constraint (8) to eliminate one weight variable from the matrix equation (7) (as in [Mohr and Gleicher 2003]), we avoid preferential treatment of individual bone weights during TSVD or NNLS solves. These weights are compared in Figures 7 and 8.

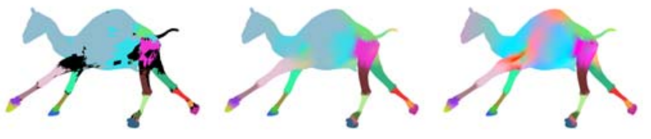


Figure 8: **Over-fitting comparison of TSVD and NNLS vertex weights:** *(Left) Core triangles; (Middle) Weights computed using NNLS have a similar coloration to the core triangles since their weights are nonnegative; (Right) Significant over-fitting is observed with TSVD weights since they can use large negative values to better approximate the data. Note: bone colors are weighted by vertex weights, so exaggerated colors indicate over-fitting (also see TSVD weights in Figure 6).*

6 Progressive Skin Corrections

Skinned animations may provide sufficient accuracy for many applications, however, they can always be improved by increasing the number of bones (same as decreasing ε). Another way of improving accuracy is by using data reduction to progressively add corrective displacements and normals. These two approaches, increasing bones or corrections, provide complementary improvements, with the latter being possible at runtime.

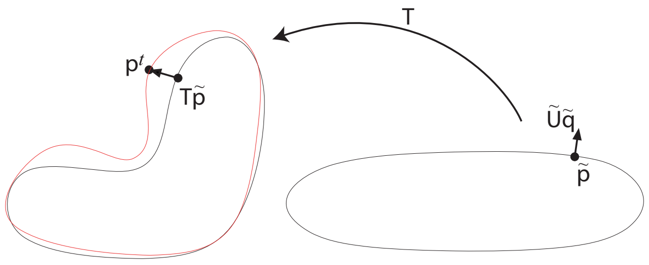

T
p^t
$\mathsf{T}\tilde{\mathsf{p}}$
$\tilde{\mathsf{U}}\tilde{\mathsf{q}}$
$\tilde{\mathsf{p}}$

Figure 9: **Skin corrections** *are defined in the rest pose.*

We use an approach analogous to EigenSkin [Kry et al. 2002] where the displacement and normal corrections are added in the rest configuration of the mesh prior to applying the skin transformation (see Figure 9). Related approaches also appear in the literature (see [Lewis et al. 2000]). Given the sequence of mesh points p^t, and their skinned approximations, $\mathsf{T}^t\tilde{\mathsf{p}}$, we obtain a sequence of S reference-frame displacement corrections, $((\mathsf{T}^t)^{-1}\mathsf{p}^t - \tilde{\mathsf{p}})$, that we perform data reduction on to obtain the representation, $((\mathsf{T}^t)^{-1}\mathsf{p}^t - \tilde{\mathsf{p}}) = \tilde{\mathsf{U}}\tilde{\mathsf{q}}^t$. Although any data reduction can be used, we use TSVD to generate a progressive approximation that converges to the exact mesh sequence as the rank of the displacement basis $\tilde{\mathsf{U}}$ is increased (see Figure 10). This amounts to the articulated shape model,

$$\mathsf{p}^t = \mathsf{T}^t(\tilde{\mathsf{p}} + \tilde{\mathsf{U}}\tilde{\mathsf{q}}^t), \quad t = 1\ldots S. \tag{12}$$

where a tilde denotes pre-transform quantities defined in the undeformed configuration of the mesh (see Figure 9). Similar equations exist for normals (c.f. [Kry et al. 2002]),

$$\mathsf{n} = \mathsf{F}^{-T}(\tilde{\mathsf{n}} + \tilde{\mathsf{N}}\tilde{\mathsf{q}}_n), \quad \text{or} \quad \mathsf{n} = \mathsf{F}(\tilde{\mathsf{n}} + \tilde{\mathsf{N}}'\tilde{\mathsf{q}}'_n). \tag{13}$$

As an alternative to progressive normal approximations, we can also use progressive approximations to other deformable appearance models. For example, in our implementation, we generate diffuse Precomputed Radiance Transfer (PRT) [Sloan et al. 2002] for SMAs (and translating ground planes). This is done by computing PRT transfer vectors for each mesh, then either computing TSVD of transfer vectors (as in [James and Fatahalian 2003]), or non-negative matrix factorization (NMF) [Lee and Seung 2000] of pre-lit vertex colors, to produce a low-rank, deformable, illumination approximation that can be efficiently computed in a vertex shader. Results are shown in Figure 1 for NMF-factorized, deformable, monochromatic PRT.

7 Results

Results for our examples are summarized in Table 1. Preprocess timings for are unoptimized, and were generated on an Intel Pentium 4 (2.0 GHz) processor with 1 GB RAM. Mean shift clustering clearly dominates all preprocess times, but is still under an hour in the worst case (ironically, the elephant example). Although the computational complexity of the approach is $O(nSB)$ for n triangles, S frames, and B bones, the memory complexity is $O(nS)$ for naïve in-core implementations, and can be inconvenient for large animations.

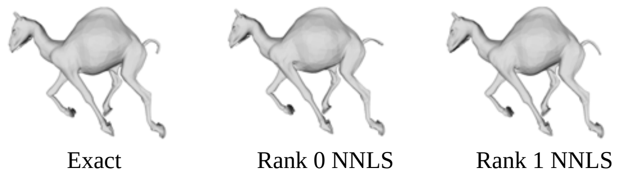


Figure 10: **Progressive displacement corrections:** *(Left) Exact camel mesh in a non-reference pose; (Middle) The uncorrected NNLS skinned model already provides a good resemblance ($B = 29$, $\beta = 4$, affine bones); (Right) One displacement correction removes the majority of distortion.*

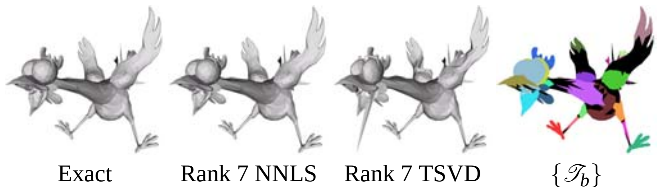


Figure 11: **Over-fitting impedes corrections:** *(Left) An extreme frame (#273) from the "Chicken Crossing" animation; (MidLeft) our approximation ($B = 22$, $\beta = 4$, affine bones) with NNLS weights and only 7 displacement corrections is almost indistinguishable; (MidRight) approximation with TSVD weights suffers from over-fitting, and exhibits unpredictable displacement artifacts (see spike near beak). (Right) Core bone triangles.*

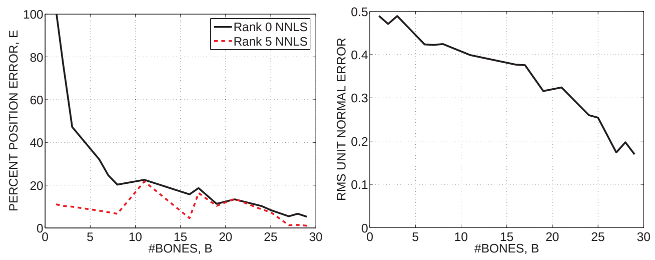


Figure 12: **Animation error with increasing bones,** *B for horse model (flexible bones, NNLS weights, $\beta = 4$). (Left) Percent relative position error, E, as a function of bones, B, shown for Rank 0 and Rank 5 corrections. While increasing the number of bones can improve accuracy, we observe that skin corrections are typically more effective when the near-rigid structure is resolved, e.g., at $B = 29$ bones ($\epsilon = 0.05$), or left alone at $B = 1$. Note that the $B = 1$ case is similar to the PCA compression approach of Alexa and Müller [2000]. (Right) The RMS unit normal error as a function of B.*

Animation compression is not our ultimate goal, however SMAs can provide compact approximations for quasi-articulated models. Results are shown in Figures 12 and 13, and Table 1.

Hardware implementation of SMAs is trivial given that any existing vertex program for matrix palette skinning can be used. Interactive performance can be achieved for very large scenes on an NVIDIA GeForce 6800 GT (see Figure 1). Mesh levels of detail are produced using mesh simplification with additional vertex attributes [Garland and Heckbert 1998] for skinning weights, displacement corrections, PRT coefficients, etc. (see Figure 14).

Comparison to vertex buffer objects: We compare the performance of our hardware-accelerated SMAs to an optimized OpenGL mesh sequence renderer based on Vertex Buffer Objects (VBO); each frame, the array of vertex positions and normals are transferred to static VBO memory, and then drawn as indexed geometry. Our test scene is composed of 1500 SMAs (499 camels, 483 elephants, and 518 horses) and involves 71.5 million triangles. The brute-force VBO approach (0.35 Hz) is more than six times slower than hardware-skinned SMAs with display lists (2.1 Hz). Also, SMAs can avoid excessive mesh storage, and the bottleneck of sending millions of triangles across our (8X) AGP bus.

Model	Frames, S	Vertices, N	Triangles, n	$t_{meanShift}$	t_{skin}	Bones, B	η	$E_{(Rigid,TSVD)}$	$E_{(Rigid,NNLS)}$	Compress	$E_{(Flex,TSVD)}$	$E_{(Flex,NNLS)}$	Compress
ball	100	5552	11100	3.0 min	0.6 min	1	1.00	0.78 (0.00)	0.78 (0.00)	72.7	$\underline{0.0002}$ (0)	0.0002 (0)	71.2
snake	121	9179	18354	8.0 min	1.3 min	27	0.98	0.20 (0.10)	0.24 (0.10)	38.2	$\underline{0.12}$ ($\underline{0.06}$)	0.17 ($\underline{0.06}$)	32.2
dance	181	7061	14118	9.1 min	1.4 min	21	0.93	1.21 (0.29)	1.34 (0.29)	50.4	$\underline{0.65}$ ($\underline{0.17}$)	0.77 (0.18)	40.3
bridge	30	135304	240316	7.2 min	4.6 min	14	0.92	20.3 (0.16)	27.1 (0.15)	12.8	$\underline{12.1}$ (0.10)	17.6 ($\underline{0.05}$)	12.8
camel	48	21887	43814	18.1 min	1.2 min	23	0.88	7.76 (0.98)	8.17 (0.93)	19.6	$\underline{3.11}$ ($\underline{0.54}$)	4.43 ($\underline{0.54}$)	18.9
elephant	48	42321	84638	46.2 min	2.4 min	25	0.85	10.6 (2.81)	12.5 (1.78)	20.0	$\underline{4.80}$ (1.01)	6.22 ($\underline{0.85}$)	19.6
horse	48	8431	16843	2.8 min	0.6 min	30	0.84	8.10 (1.46)	8.90 (1.38)	17.6	$\underline{3.62}$ (0.59)	4.53 ($\underline{0.53}$)	15.9
chicken	400	3030	5664	5.9 min	1.3 min	22	0.83	0.69 (0.20)	0.84 (0.17)	43.9	$\underline{0.39}$ (0.57)	0.45 ($\underline{0.14}$)	28.7
hands	26	1753	3470	0.1 min	0.1 min	17	0.79	2.63 (0.64)	2.78 (0.62)	8.9	$\underline{2.50}$ (0.59)	2.63 ($\underline{0.56}$)	7.9
elasticCow	204	2904	5804	2.4 min	0.7 min	18	0.41	$\underline{2.82}$ ($\underline{1.54}$)	3.27 (1.55)	38.6	3.09 (1.94)	3.18 (2.02)	27.6
clothHorse	53	8431	16843	7.1 min	0.6 min	6	0.13	41.7 (15.3)	44.1 (0.88)	21.9	$\underline{28.1}$ (–)	30.3 ($\underline{0.74}$)	21.3
flag$_{(\varepsilon=.05)}$	200	6906	13436	8.2 min	1.6 min	32	0.42	$\underline{21.2}$ (7.12)	21.5 ($\underline{5.93}$)	44.5	26.0 (147)	26.6 (38.9)	33.1
flag$_{(\varepsilon=.10)}$	200	6906	13436	14.0 min	2.4 min	100	(0.92)	2.26 (1.26)	2.39 ($\underline{1.25}$)	22.0	$\underline{1.49}$ (1.67)	1.58 (1.78)	14.4

Table 1: **SMA statistics** *for models sorted by near-rigid fraction, η. Includes precomputation time for mean shift ($t_{meanShift}$) and model skinning (t_{skin}) with $\beta = \min(4, B)$. We report approximation errors in terms of percent distortion, $E = 100\% * \|\mathbf{P} - \mathbf{P}_{approx}\|_F / \|\mathbf{P} - \mathbf{P}_{timeAverage}\|_F$ (as in [Karni and Gotsman 2004]) for the four possible combinations of Rigid/Flexible bones with TSVD/NNLS weights. In addition to uncorrected skin error, we also report error for a rank-10 displacement correction model (in brackets); we underline the optimal model for the uncorrected and rank-10 cases. The general trend is that for models with a lot of near-rigid structure (high η), flexible bones are always better, with TSVD weights typically better when no corrections are used (rank 0), but NNLS weights better with corrections–a sign of TSVD over-fitting. For highly deformable models ($\eta < 0.5$), rigid bones become more favorable than flexible bones, probably because the latter over-fit at low η values. Finally, overall compression (Compress) of uncorrected skins tends to favor 7-float rigid bones (quaternion + translation) instead of 12-float flexible bones for sequences with larger SB values.*

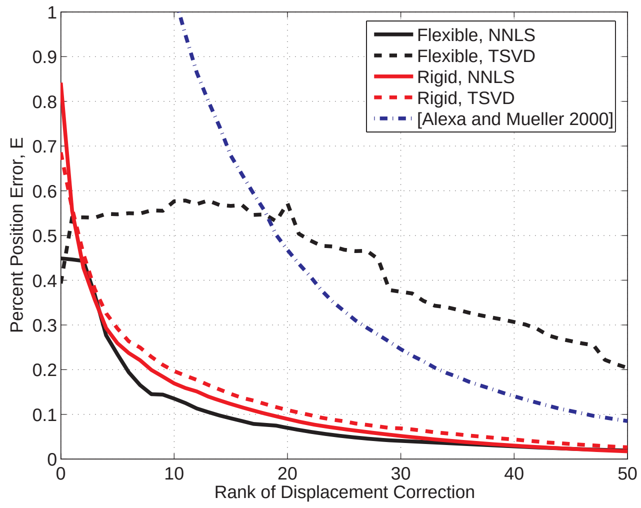


Figure 13: **Progressive approximations:** *"Chicken Crossing" animation ($B = 22$ bones at $\varepsilon = 0.05$) shown for increasing displacement correction rank. Flexible bones with NNLS weights provide the best overall SMA approximation, with passable low-rank (even zero-rank) approximations that make them suitable for efficient hardware rendering. Their stable displacement correction behavior is in contrast to flexible bones with* TSVD *weights, which has irregular convergence (also see Figure 11). For comparison, we plot (blue curve) the PCA approximation of Alexa and Müller [2000] ($B = 1$ flexible bone) which requires much higher rank to achieve similar accuracy, e.g., rank 37 ($B = 1$) is comparable to the rank 7 ($B = 22$), NNLS, flexible bone approximation shown in Figure 11. Note that the $B = 1$ approach can provide better overall compression for long low-rank animations than the frame-heavy $B = 22$ case, but it is inconvenient for hardware rendering.*

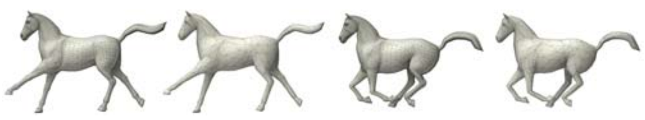


Figure 14: **SMA Simplification:** *(Left) Rest pose horse (16843 triangles) and simplification (2100 triangles); (Right) horses in skinned pose (flexible bones, NNLS weights, $\beta = 4$).*

Rest pose editing: SMAs allow displacement edits applied to the reference mesh to be automatically applied to all animation meshes. The displacement edits are defined in the local frame of reference so that they orient correctly as the mesh is animated. See Figure 15 and our video for an example. We use weights computed with NNLS solves to avoid over-fitting that would make rest pose editing less intuitive.

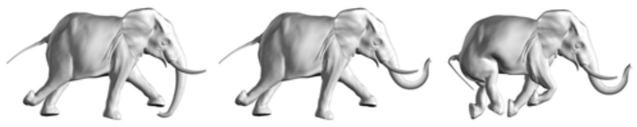


Figure 15: **Rest Pose Editing:** *(Left) Reference mesh before edit; (Middle) Displacement edited reference mesh; (Right) Animated pose using skinned mesh with displacement edit applied (flexible bones, NNLS weights, $\beta = 4$).*

Highly deformable models: We observe that the near-rigid triangle fraction, η, ($\varepsilon = 0.05$) provided by mean shift reliably indicates models with sufficient near-rigid structure for skinning to be successful. Datasets with low η values tend to indicate highly deformable models for which skinned approximations tend to be inefficient. Examples include a collapsing cloth horse ($\eta = 0.12$; see Figure 16), or a flag flapping in the wind ($\eta = 0.42$; see Figure 16). One characteristic problem, is that highly deformable regions, such as the flapping flag edge, lack sufficient near-rigid structure for skinning (see Figure 17) unless extremely many bones are used. It appears that datasets with $\eta(\varepsilon = 0.05)$ values below 0.5 are non-robust candidates for skinning, and that progressive skin corrections can be expected to perform no better (and often much worse) than direct data reduction methods.

Deformable collision detection can also be accelerated for SMAs by exploiting the articulated mesh parameterization. By partitioning mesh triangles into regions most strongly associated with each rigid bone, we can build rigid local frames of reference for each bone's mesh region. By computing the displacement correction model $\mathsf{U}^{(b)}\mathsf{q}^{(b)}$ for bone b's triangle's vertices, we can construct a Bounded Deformation Tree [James and Pai 2004] on each region (see Figure 18). The method is effective because building BD-Trees on local mesh regions can exploit the local frame of reference and simpler resulting deformation models. Such approaches

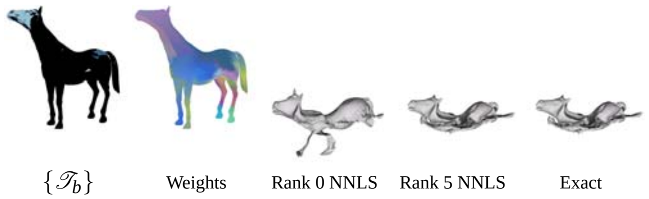


Figure 16: **Skinning highly deformable animations**, *such as a collapsing cloth horse, are particularly hard since the distribution of estimated core bone triangles (far left) can be too sparse (small* $\eta = 0.12$*). In general, highly deformable animations yield unpredictable SMA approximations often no better than direct data reduction.*

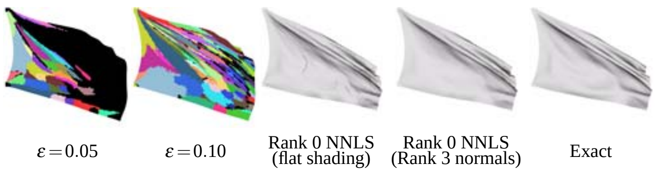


Figure 17: **A flapping flag** *has deficient bone structure at* $\varepsilon = 0.05$*, but is reasonably approximated by 100 bones (*$\varepsilon = 0.10$*). Non-smooth SMA reconstructions are typical of highly deformable animations, but can be partly hidden using normal corrections, or data-driven appearance models.*

can reduce bounding volume hierarchy updating costs, e.g., in physical simulations (see Figure 18).

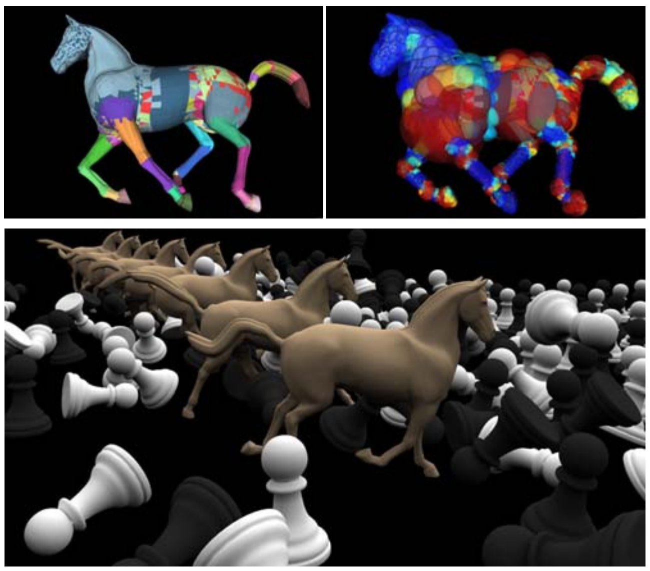


Figure 18: **SMA Collision Processing:** *(Top) Bounded Deformation Trees can be precomputed on near-rigid components (or any desirable mesh partitioning) to provide output-sensitive collision detection for SMAs; (Bottom) SMA collisions driving a physical simulation.*

8 Summary

We have presented an automatic method for constructing skinned mesh approximations to parametrically coherent mesh sequences. Our experiments indicate that (a) mean shift clustering of triangle rotation sequences is a robust estimator of near-rigid mesh structure, and (b) nonrigid affine bone transformations combined with positive vertex weights (computed using nonnegative least squares (NNLS)) lead to accurate approximations for examples with sufficient near-rigid structure (e.g., $\eta > 0.7$). The resulting skinned models tend to avoid over-fitting, making their reconstructions stable under small displacement modifications introduced by progressive displacement corrections, SMA simplification, and rest pose editing. Finally, the method is well-suited to real-time hardware rendering of mesh animations of quasi-articulated phenomena, such as those shown in Figure 1, and skinned animations can be used with output-sensitive collision detection methods for deformable models.

Acknowledgments: We would like to thank the NSF (CAREER-0430528), Pixar, and NVIDIA for generous support; the anonymous reviewers for helpful suggestions; the Rutgers Robust Image Understanding Laboratory for providing their mean shift implementation; Bob Sumner, Jernej Barbič, Hector M. Briceño, Matthias Müller, and co-authors for mesh data. "The chicken character was created by Andrew Glassner, Tom McClure, Scott Benza, and Mark Van Langeveld. This short sequence of connectivity and vertex position data is distributed solely for the purpose of comparison of geometry compression techniques."

Appendix

A Flexible Bones using Least Squares

For completeness, we provide equations to estimate bone b's affine transform, $\bar{\mathsf{T}}_b = (\mathsf{F},\mathsf{v})$, for a given mesh sequence instance. We define $\bar{\mathsf{T}}$ as the transform that minimizes the triangle area-weighted sum over all core bone triangles, $\mathscr{T}_b$, of the squared position errors between transformed reference centroids, $\bar{\mathsf{T}}_b\tilde{\mathsf{c}}$, and the deformed centroids, c. Specifically, we minimize

$$\Phi = \sum_{i\in\mathscr{T}_b} a_i\|\mathsf{c}_i - \mathsf{F}\tilde{\mathsf{c}}_i - \mathsf{v}\|_2^2 = \sum_{i,x} a_i\Big(\mathsf{c}_{ix} - \sum_{y}\mathsf{F}_{xy}\tilde{\mathsf{c}}_{iy} - \mathsf{v}_x\Big)^2 \qquad (14)$$

where $x,y = 1,2,3$. Taking the derivatives of Φ w.r.t the 12 unknowns, F_{xy} and v_x, and setting each to zero, leads to 12 sparse equations in 12 unknowns,

$$\begin{aligned}\sum_{iz} a_i\tilde{\mathsf{c}}_{iz}\tilde{\mathsf{c}}_{iy}\mathsf{F}_{xz} + \Big(\sum_i a_i\tilde{\mathsf{c}}_{iy}\Big)\mathsf{v}_x &= \sum_i a_i\mathsf{c}_{ix}\tilde{\mathsf{c}}_{iy}, \quad x,y = 1,2,3;\\ \sum_{iz} a_i\tilde{\mathsf{c}}_{iz}\mathsf{F}_{xz} + \Big(\sum_i a_i\Big)\mathsf{v}_x &= \sum_i a_i\mathsf{c}_{ix}, \quad x = 1,2,3,\end{aligned}$$

where z is summed over the three components. The (F,v) solution is the optimal flexible bone in an area-weighted least-squares sense.

References

ALEXA, M., AND MÜLLER, W. 2000. Representing Animations by Principal Components. *Computer Graphics Forum 19*, 3 (Aug.), 411–418.

ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2002. Articulated Body Deformation From Range Scan Data. *ACM Transactions on Graphics 21*, 3 (July), 612–619.

BRICEÑO, H. M., SANDER, P. V., MCMILLAN, L., GORTLER, S., AND HOPPE, H. 2003. Geometry Videos: A New Representation for 3D Animations. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 136–146.

BUSS, S. R., AND FILLMORE, J. P. 2001. Spherical Averages and Applications to Spherical Splines and Interpolation. *ACM Transactions on Graphics 20*, 2 (Apr.), 95–126.

CHENG, Y. 1995. Mean Shift, Mode Seeking, and Clustering. *IEEE Trans. Pattern Anal. Mach. Intell. 17*, 8, 790–799.

COMANICIU, D., AND MEER, P. 2002. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 5, 603–619.

ETZMUSS, O., KECKEISEN, M., AND STRASSER, W. 2003. A fast finite element solution for cloth modelling. In *Proceedings of Pacific Graphics*, 244–251.

GARLAND, M., AND HECKBERT, P. S. 1998. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In *IEEE Visualization '98*, 263–270.

GEORGESCU, B., SHIMSHONI, I., AND MEER, P. 2003. Mean Shift Based Clustering in High Dimensions: A Texture Classification Example. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV 2003)*. Adaptive mean shift code available at http://www.caip.rutgers.edu/riul/research/code/AMS.

GOLUB, G. H., AND LOAN, C. F. V. 1996. *Matrix Computations*, third ed. Johns Hopkins University Press, Baltimore.

GUPTA, S., SENGUPTA, K., AND KASSIM, A. A. 2002. Compression of dynamic 3D geometry data using iterative closest point algorithm. *Comput. Vis. Image Underst. 87*, 1-3, 116–130.

GUSKOV, I., AND KHODAKOVSKY, A. 2004. Wavelet Compression of Parametrically Coherent Mesh Sequences. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM Press, 183–192.

HAKURA, Z. S., LENGYEL, J. E., AND SNYDER, J. M. 2000. Parameterized Animation Compression. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, 101–112.

IBARRIA, L., AND ROSSIGNAC, J. 2003. Dynapack: Space-Time compression of the 3D animations of triangle meshes with fixed connectivity. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, Eurographics Association, 126–135.

JAMES, D. L., AND FATAHALIAN, K. 2003. Precomputing Interactive Dynamic Deformable Scenes. *ACM Transactions on Graphics 22*, 3 (July), 879–887.

JAMES, D. L., AND PAI, D. K. 2004. BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models. *ACM Transactions on Graphics (SIGGRAPH 2004) 23*, 3 (Aug.).

JONES, T. R., DURAND, F., AND DESBRUN, M. 2003. Non-Iterative, Feature-Preserving Mesh Smoothing. *ACM Transactions on Graphics 22*, 3 (July), 943–949.

KARNI, Z., AND GOTSMAN, C. 2004. Compression of soft-body animation sequences. *Computers & Graphics 28*, 1, 25–34.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. EigenSkin: Real Time Large Deformation Character Skinning in Hardware. In *ACM SIGGRAPH Symposium on Computer Animation*, 153–160.

LAWSON, C. L., AND HANSON, R. J. 1974. *Solving Least Square Problems*. Prentice Hall, Englewood Cliffs, NJ.

LEE, D. D., AND SEUNG, H. S. 2000. Algorithms for non-negative matrix factorization. In *NIPS*, 556–562.

LENGYEL, J. E. 1999. Compression of Time-dependent Geometry. In *SI3D '99: Proceedings of the 1999 Symposium on Interactive 3D Graphics*, ACM Press, 89–95.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose Space Deformations: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 165–172.

LINDHOLM, E., KILGARD, M. J., AND MORETON, H. 2001. A User-Programmable Vertex Engine. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 149–158.

MOAKHER, M. 2002. Means and Averaging in the Group of Rotations. *SIAM J. Matrix Anal. Appl. 24*, 1–16.

MOHR, A., AND GLEICHER, M. 2003. Building Efficient, Accurate Character Skins From Examples. *ACM Transactions on Graphics 22*, 3 (July), 562–568.

MÜLLER, M., AND GROSS, M. 2004. Interactive Virtual Materials. In *Proceedings of Graphics Interface*.

SHAMIR, A., BAJAJ, C., AND PASCUCCI, V. 2000. Multi-resolution dynamic meshes with arbitrary deformations. In *Proc. of Visualization '00*, IEEE Computer Society Press, 423–430.

SHOEMAKE, K., AND DUFF, T. 1992. Matrix Animation and Polar Decomposition. In *Graphics Interface '92*, 258–264.

SLOAN, P.-P. J., III, C. F. R., AND COHEN, M. F. 2001. Shape by Example. In *ACM Symp. on Interactive 3D Graphics*, 135–144.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Transactions on Graphics 21*, 3 (July), 527–536.

WANG, X. C., AND PHILLIPS, C. 2002. Multi-Weight Enveloping: Least-Squares Approximation Techniques for Skin Animation. In *ACM SIGGRAPH Symp. on Computer Animation*, 129–138.