EUROGRAPHICS 2010 / T. Akenine-Möller and M. Zwicker
(Guest Editors)

Volume 29 (2010), Number 2

Fast and Efficient Skinning of Animated Meshes

L. Kavan[1,2], P.-P. Sloan[1] and C. O'Sullivan[2]

[1]Disney Interactive Studios, USA
[2]Trinity College Dublin, Ireland

Abstract

Skinning is a simple yet popular deformation technique combining compact storage with efficient hardware accelerated rendering. While skinned meshes (such as virtual characters) are traditionally created by artists, previous work proposes algorithms to construct skinning automatically from a given vertex animation. However, these methods typically perform well only for a certain class of input sequences and often require long pre-processing times. We present an algorithm based on iterative coordinate descent optimization which handles arbitrary animations and produces more accurate approximations than previous techniques, while using only standard linear skinning without any modifications or extensions. To overcome the computational complexity associated with the iterative optimization, we work in a suitable linear subspace (obtained by quick approximate dimensionality reduction) and take advantage of the typically very sparse vertex weights. As a result, our method requires about one or two orders of magnitude less pre-processing time than previous methods.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Linear blend skinning (also known as matrix palette skinning) is a mesh deformation technique implemented in almost all modern 3D engines, most frequently used for virtual characters driven by skeletal animation. However, recent research suggests new possible applications, including animation of highly deformable objects, such as cloth.

Standard linear blend skinning works as follows: let us denote the i-th rest pose vertex as $\tilde{\mathbf{v}}_i \in R^{4\times1}$ (with the last coordinate equal to 1 as usual), the palette of P affine transformations for frame f as $\mathbf{M}_1^{(f)}, \ldots, \mathbf{M}_P^{(f)} \in R^{3\times4}$ and vertex weights as $w_{i,1}, \ldots, w_{i,P} \in R$. A skinned vertex position in frame f is then computed as:

$$\mathbf{v}_i^{(f)} = \sum_{j=1}^{P} w_{i,j}\mathbf{M}_j^{(f)}\tilde{\mathbf{v}}_i, \quad \mathbf{v}_i^{(f)} \in R^{3\times1} \tag{1}$$

Motivated by skeletal animation terminology, the transformations $\mathbf{M}_j^{(f)}$ are also sometimes referred to as *bones* (or *proxy-bones*). The vertex weights $w_{i,1}, \ldots, w_{i,P}$ are usually required to be convex (i.e. non-negative and summing to 1) and sparse—a maximum of 4 non-zero weights is a de facto standard in most interactive applications. The sparsity constraint is crucial for efficient evaluation of Formula (1), especially when implemented on graphics hardware.

An animated mesh with fixed connectivitycan be represented by a $3F \times N$ matrix $\mathbf{A}$, where F is the number of frames and N the number of vertices:

$$\mathbf{A} = \begin{pmatrix} \mathbf{v}_1^{(1)} & \cdots & \mathbf{v}_N^{(1)} \\ \vdots & \ddots & \vdots \\ \mathbf{v}_1^{(F)} & \cdots & \mathbf{v}_N^{(F)} \end{pmatrix}$$

The problem addressed in this paper is to approximate the animation matrix $\mathbf{A}$ by skinning with a small number of proxy-bones P. We can obtain useful insights about this problem by rewriting Formula (1) in matrix form. If we stack the skinning parameters in matrices $\mathbf{T} \in R^{3F\times4P}$ and $\mathbf{X} \in R^{4P\times N}$ as follows:

$$\mathbf{T} = \begin{pmatrix} \mathbf{M}_1^{(1)} & \cdots & \mathbf{M}_P^{(1)} \\ \vdots & \ddots & \vdots \\ \mathbf{M}_1^{(F)} & \cdots & \mathbf{M}_P^{(F)} \end{pmatrix}, \mathbf{X} = \begin{pmatrix} w_{1,1}\tilde{\mathbf{v}}_1 & \cdots & w_{N,1}\tilde{\mathbf{v}}_N \\ \vdots & \ddots & \vdots \\ w_{1,P}\tilde{\mathbf{v}}_1 & \cdots & w_{N,P}\tilde{\mathbf{v}}_N \end{pmatrix}$$

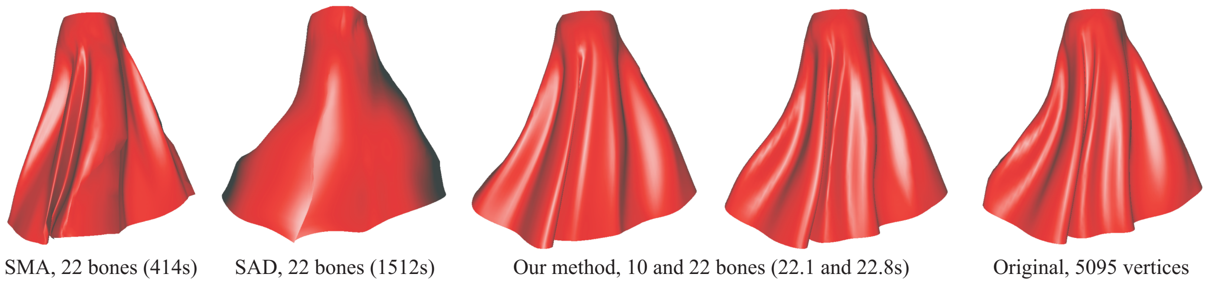

Figure 1: *Skinning approximation of a skirt animation computed using SMA (Skinning Mesh Animations) [JT05], SAD (Skinning Arbitrary Deformations) [KMD*07] and our proposed method (execution times in brackets). Our method produces more accurate skinning approximations in a fraction of the time required by previous methods (skirt dataset, see Table 2).*

then Formula (1) can be written in matrix form as:

$$\mathbf{TX} = \mathbf{A} \tag{2}$$

For small P, this equation typically cannot be satisfied exactly. Therefore, the task is to minimize the Frobenius norm $\|\mathbf{TX} - \mathbf{A}\|_F$ for a given number of proxy-bones P.

Formula (2) suggests that linear blend skinning can be also considered as a special kind of matrix decomposition (in the following, we will use the term *skinning decomposition*). The main difference to standard matrix decompositions, such as the popular Singular Value Decomposition (SVD), is the limit on the number of non-zero elements in each column of $\mathbf{X}$ (typically 16—four non-zero weights times four elements of $\tilde{\mathbf{v}}_i$). Compared to truncated SVD [AM00], skinning typically requires more per-frame data, but the amount of per-vertex data is small and fixed. The constant number of vertex attributes is probably one of the reasons for the popularity of skinning, as it simplifies many implementation issues and enables very efficient hardware processing.

Building on the results of previous work, we introduce a technique that produces quite accurate skinning decompositions with low bone counts (see Figure 1). Our method employs only standard linear blend skinning without any modifications or extensions and is thus suitable for many existing applications. However, we do not attempt to organize our proxy-bones into a hierarchy (see Figure 2), and we do not address the problem of motion post-processing.

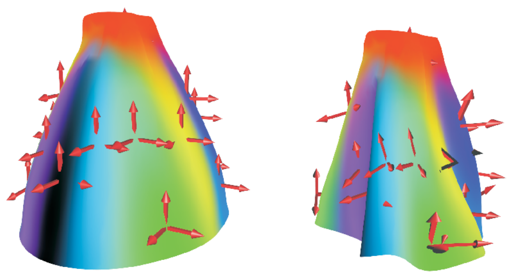

Figure 2: *Example of vertex weights and bone transformations produced by our method.*

2. Related Work

The problem of approximating mesh animation with skinning was first addressed by James and Twigg [JT05]. Their method, called Skinning Mesh Animations (SMA), works by applying mean-shift clustering to identify core bone triangles (and thereby bone transformations). Subsequently, vertex weights are determined by truncated SVD or non-negative least squares. The reconstruction quality can be optionally improved by a corrective technique similar to EigenSkin [KJP02], which works by compressing per-vertex residuals using truncated SVD. SMAs work best with models consisting of approximately rigid components (i.e., quasi-articulated), such as human or animal figures.

The follow-up work of Kavan and colleagues [KMD*07] aims to address the issues associated with highly deformable animations. Their method, called Skinning Arbitrary Deformations (SAD), uniformly distributes proxy-bones over the rest pose mesh and optimizes their transformations using least squares. While this provides more accurate skinning of highly deformable animations, it is slower than SMA and requires a lot of proxy-bones (typically 100) and EigenSkin corrections (typically 10) to achieve visually pleasing reconstructions. SAD also produces suboptimal results for quasi-articulated animations, because it does not attempt to identify coherent mesh components. Focusing on quasi-articulated animations, Schaefer and Yuksel [SY07] and deAguiar and colleagues [dATTS08] develop methods to extract not only the skinning parameters but also full hierarchical skeletons, thus enabling convenient editing. Both also propose faster clustering methods than mean-shift and alternative vertex weights optimization methods, but highly deformable animations (lacking any natural skeletal hierarchy) are not considered. Baran and Popović [BP07] show how to rig characters using only the rest pose and skeleton, without any training animation.

While standard linear blend skinning according to Formula (1) is the most popular method, several modifications are studied in the literature (we review only the most closely related work, please see [GB08] for a survey). Animation-Space [MMG06a] is a linear skinning technique which relaxes the constraints on matrix $\mathbf{X}$ from Formula (2) (similarly to its predecessor Multi-weight Enveloping [WP02]). The elements of $\mathbf{X}_{AnimationSpace}$ can be arbitrary and do not have to have the form $w_{i,j}\tilde{\mathbf{v}}_i$ as in standard skinning. However, the sparsity pattern of $\mathbf{X}_{AnimationSpace}$ remains unchanged, i.e., there is a maximum of 16 non-zero elements in each column. The fitting process described in [MMG06a] assumes the bone transformations matrix $\mathbf{T}$ to be known (e.g., specified by an animated skeleton).

Clustered PCA [SSK05] can also be considered as matrix decomposition according to Formula (2), but with a different sparsity pattern—in particular, $\mathbf{X}_{CPCA}$ is required to be a rectangular block diagonal matrix (while $\mathbf{T}$ can be an arbitrary dense matrix). On one hand, this allows for more flexibility within every single cluster, but on the other hand, every vertex belongs to one cluster only (i.e., the clusters are not allowed to overlap). Because of the latter property, the resulting reconstruction may be non-smooth, see Figure 7. This can be alleviated by reducing the number of clusters and increasing their dimension instead, which in the limit leads to truncated SVD (i.e., only one cluster) [AM00].

Nonlinear skinning techniques, such as dual-quaternion skinning [KCŽO08], have also been studied in the context of skinning decomposition [KMD*07, FKY08]. Dual-quaternion representation of bone transformations is more compact (8 scalars instead of 12) and avoids the sometimes undesirable non-rigid transformations. Unfortunately, even though the resulting optimization problem is still linear, it has different system matrix for every frame and therefore, its solution is rather time consuming. In this paper, we trade off the advantages of dual quaternions for faster pre-processing. This is not a major limitation, because nonlinear skinning can always be converted to linear skinning with extra proxy-bones [MG03, KCO09].

Formula (2) reveals that skinning decomposition is a special case of sparse matrix factorization—a more general problem with applications also in rendering [RK09] and statistics [WTH09]. While sharing certain common principles, each application considers different kinds of data and specific factorization constraints. Linear blend skinning is differentiated by the special structure of matrix $\mathbf{X}$ (see Section 1) and the requirement for convex vertex weights.

Another closely related field is general animation compression (see [Váš08]). While techniques such as those based on predictive coding [KG04] or progressive meshes [GK04, KG05] achieve high compression ratios, they typically require sequential decompression, which poses challenges for implementation on parallel graphics hardware.

Linear blend skinning finds its use not only in performance-critical interactive applications, but also in related research. Wang and colleagues [WPP07] employ skinning decomposition to obtain a very fast system for synthesizing non-linear deformations of quasi-articulated models. Feng and colleagues [FKY08] skin highly deformable animations (using an improved SAD procedure) to obtain an efficient posing system based on canonical correlation analysis. In general, we believe skinning decomposition itself deserves further investigation, as attempted in this paper.

2.1. Overview of Our Method

Our main tool for obtaining accurate skinning decompositions is alternating least squares, i.e., a variant of coordinate descent (proposed in a similar context already in [MG03]). In contrast to previous work, we optimize all of the skinning parameters respectively: bone transformations, vertex weights and rest pose positions. Since several iterations are required (we use 15, see Section 5), this results in a lot of least squares problems to be solved, suggesting high computational costs.

Therefore, we propose to formulate our optimization process in reduced coordinates, by decomposing the input animation matrix $\mathbf{A} \in R^{3F\times N}$ to $\mathbf{B} \in R^{3F\times d}$ and $\mathbf{C} \in R^{d\times N}$ such that $\mathbf{BC} \approx \mathbf{A}$, where $\mathbf{B}$ has orthonormal columns and $d << 3F$ (Section 3). Intuitively, this corresponds to eliminating vertex trajectories that can be expressed as linear combinations of representative vertex trajectories (columns of $\mathbf{B}$). This allows us to pose the problem as finding $\mathbf{T}_r \in R^{d\times 4P}$ and $\mathbf{X} \in R^{4P\times N}$ minimizing $\|\mathbf{BT}_r\mathbf{X} - \mathbf{BC}\|_F = \|\mathbf{T}_r\mathbf{X} - \mathbf{C}\|_F$ and not reconstruct the actual bone transformations matrix $\mathbf{T} = \mathbf{BT}_r$ until the whole optimization is finished. When combined with the sparse nature of $\mathbf{X}$ and our improved formulation of fitting vertex weights, we obtain a fast and efficient solver for each of the skinning parameters (Section 4).

3. Approximate Dense Dimensionality Reduction

Given the animation matrix $\mathbf{A} \in R^{3F\times N}$, the task is to find the dimension d and matrices $\mathbf{B} \in R^{3F\times d}$, $\mathbf{C} \in R^{d\times N}$, where $\mathbf{B}^T\mathbf{B} = \mathbf{I}$, such that:

$$\|\mathbf{BC} - \mathbf{A}\|_F \leq \varepsilon$$

We choose the error threshold ε conservatively to ensure that the error incurred by using $\mathbf{BC}$ instead of $\mathbf{A}$ is close to imperceptible (see Section 5).

The optimal d and the corresponding matrices $\mathbf{B}$ and $\mathbf{C}$ could be found using truncated SVD [GL96]. However, even the very efficient LAPACK [ABB*99] routines take an impractically long time to compute the SVD of our animation matrices. Therefore, we propose an alternative dimensionality reduction technique which requires slightly higher d, but runs much faster than SVD. One possibility would be to apply Gram-Schmidt orthonormalization [JF03], but this approach typically results in a too high d and not exactly orthogonal basis vectors (the modified Gram-Schmidt procedure [GL96] is more numerically stable, but still not perfect). We propose Algorithm 1 which addresses both of these issues.

Algorithm 1

```
Input: Matrix $\mathbf{A} \in R^{3F \times N}$, error threshold $\varepsilon \in R$
Output: Matrices $\mathbf{B} \in R^{3F \times d}$, $\mathbf{C} \in R^{d \times N}$ such that $\mathbf{B}^T\mathbf{B} = \mathbf{I}$ and $||\mathbf{BC} - \mathbf{A}||_F \le \varepsilon$
 1: $\mathbf{A}_1 = \mathbf{A}, \mathbf{B}_0 =$ empty matrix, $\mathbf{C}_0 =$ empty matrix, $i = 0$
 2: repeat
 3:     $i = i + 1$
 4:     $\mathbf{m}_i =$ maximum norm column of $\mathbf{A}_i$ $\{\mathbf{m}_i \in R^{3F \times 1}\}$
 5:     $\mathbf{m}_i = \mathbf{m}_i - \mathbf{B}_{i-1}\mathbf{B}_{i-1}^T\mathbf{m}_i$ {stabilization}
 6:     $\mathbf{m}_i = \mathbf{m}_i / ||\mathbf{m}_i||$ {normalization}
 7:     $\mathbf{A}_{i+1} = \mathbf{A}_i - \mathbf{m}_i\mathbf{m}_i^T\mathbf{A}_i$ {deflation}
 8:     $\mathbf{B}_i = (\mathbf{B}_{i-1} | \mathbf{m}_i)$ {append column}
 9:     $\mathbf{C}_i = \begin{pmatrix} \mathbf{C}_{i-1} \\ \mathbf{m}_i^T\mathbf{A}_i \end{pmatrix}$ {append row}
10: until $||\mathbf{B}_i\mathbf{C}_i - \mathbf{A}||_F \le \varepsilon$
11: $d = i, \mathbf{B} = \mathbf{B}_i, \mathbf{C} = \mathbf{C}_i$
```

The vector $\mathbf{m}_i$ defined on line 4 of Algorithm 1 is our approximation of the leading singular vector (principal component). Several alternatives are possible. For example, setting $\mathbf{m}_i$ to columns of $\mathbf{A}_i$ one by one corresponds to the Gram-Schmidt strategy. Another possibility we tried is to perform several iterations of the power method (i.e., $\mathbf{m}_i^{(k+1)} = \mathbf{A}_i\mathbf{A}_i^T\mathbf{m}_i^{(k)} / ||\mathbf{A}_i\mathbf{A}_i^T\mathbf{m}_i^{(k)}||$), which gives results close to exact SVD. The proposed strategy, i.e., selecting the maximum norm column of $\mathbf{A}_i$, presents a good compromise between fast computations and a small dimension d (see Table 1). Related techniques are also discussed in the literature, such as variations of the power method [PC98] or discrete principal components [Mar06], but we prefer the maximum norm strategy because of its simplicity and good performance.

Line 5 is the stabilization step and we can ignore it for a moment (in exact arithmetics it has no effect on $\mathbf{m}_i$ since $\mathbf{B}_{i-1}^T\mathbf{m}_i$ is a zero vector). Line 7 performs deflation, i.e., for every column of $\mathbf{A}_i$ it subtracts its component in the direction $\mathbf{m}_i$, see Figure 3. This also guarantees that columns of $\mathbf{A}_{i+1}$ will be orthogonal to $\mathbf{m}_i$ (and also to $\mathbf{m}_j$ for $j < i$). Since $\mathbf{m}_i$ is always selected from the column space of $\mathbf{A}_i$, this means that the columns of matrix $\mathbf{B}_i$ will be orthogonal. Line 9 incrementally updates $\mathbf{C}_i$ so that $\mathbf{C}_i = \mathbf{B}_i^T\mathbf{A}$. The termination condition on line 10 can be simplified as follows: $||\mathbf{B}_i\mathbf{C}_i - \mathbf{A}||_F^2 = ||\mathbf{A}||_F^2 - ||\mathbf{B}_i\mathbf{C}_i||_F^2$ and because $\mathbf{B}_i$ has orthonormal columns, $||\mathbf{B}_i\mathbf{C}_i||_F^2 = ||\mathbf{C}_i||_F^2$ (see supplemental materials for a detailed justification). Therefore, it is not necessary to explicitly compute the product $\mathbf{B}_i\mathbf{C}_i$ to determine the current approximation error.

Since $\mathbf{B}$ represents an orthonormal basis, it can be shown that even when working with reduced transformations $\mathbf{T}_r$ (and reconstructing $\mathbf{T} = \mathbf{BT}_r$ at the end as discussed in Section 2.1), we can get an upper bound of the total error of the non-reduced skinning decomposition:

$$\begin{aligned} ||\mathbf{TX} - \mathbf{A}||_F &= ||\mathbf{TX} - \mathbf{BC} + \mathbf{BC} - \mathbf{A}||_F \\ &\le ||\mathbf{TX} - \mathbf{BC}||_F + ||\mathbf{BC} - \mathbf{A}||_F \\ &= ||\mathbf{BT}_r\mathbf{X} - \mathbf{BC}||_F + ||\mathbf{BC} - \mathbf{A}||_F \\ &= ||\mathbf{T}_r\mathbf{X} - \mathbf{C}||_F + ||\mathbf{BC} - \mathbf{A}||_F \end{aligned}$$

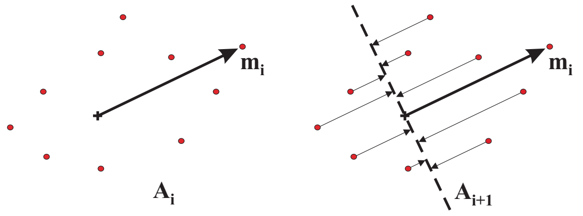

Figure 3: *Algorithm 1 selects* $\mathbf{m}_i$ *as the point farthest from the origin (line 4, left) and then subtracts projections in the* $\mathbf{m}_i$ *direction, obtaining* $\mathbf{A}_{i+1}$ *(line 7, right).*

Algorithm 1 provides $\mathbf{BC}$ such that $||\mathbf{BC} - \mathbf{A}||_F \le \varepsilon$, which is chosen to be very small, and therefore the error of the reduced skinning decomposition $||\mathbf{T}_r\mathbf{X} - \mathbf{C}||_F$ is the dominating factor (its minimization is the topic of Section 4). However, if $\mathbf{B}$ does not have exactly orthogonal columns, it is *not* the case that $||\mathbf{BT}_r\mathbf{X} - \mathbf{BC}||_F = ||\mathbf{T}_r\mathbf{X} - \mathbf{C}||_F$ and the error $||\mathbf{T}_r\mathbf{X} - \mathbf{C}||_F$ may get amplified through the skewed basis.

Fortunately, there is a simple and efficient way (line 5 in Algorithm 1) to prevent the accumulation of rounding errors that cause the loss of orthogonality. This is essentially a classical Gram-Schmidt step subtracting projected numerical inaccuracies $\mathbf{B}_{i-1}\mathbf{B}_{i-1}^T\mathbf{m}_i$, see [GLRvdE05] for a detailed theoretical analysis. Figure 4 compares the precision of the non-stabilized (without line 5) and stabilized (with line 5) version, computed with double precision. We can see that the non-stabilized version (left) exhibits an obvious loss of orthogonality (up to the 12th decimal place), whereas the stabilized version (right) is much more accurate.

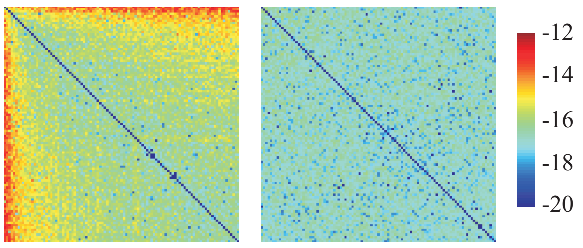

Figure 4: *Decadic logarithm of absolute values of* $(\mathbf{B}^T\mathbf{B} - \mathbf{I})$ *for the non-stabilized (left) and stabilized (right) versions of Algorithm 1 (flag dataset,* $d = 101$*).*

We also compared the performance of various strategies for selecting the vector $\mathbf{m}_i$ (line 4). While the asymptotical complexity of Algorithm 1 is always $O(FNd)$, the actual performance varies significantly. For illustration, Table 1 summarizes the resulting dimension d and the total runtime of Algorithm 1 on the flag dataset ($N = 6906, 3F = 600$, see Section 5) with the same ε in all cases.

	MAX	GS	PM1	PM2	SVD
d	101	278	80	72	69
time	2.78s	5.8s	6.3s	9.9s	27.9s

Table 1: *Results of Algorithm 1 for various* $\mathbf{m}_i$ *selection strategies: our method (MAX), Gram-Schmidt (GS), and 1 and 2 iterations of the power method (PM1, PM2). The last column (SVD) is the result of* `dgesdd` *[ABB*99].*

Note that while the Gram-Schmidt (GS) technique features the fastest update, its uninformed $\mathbf{m}_i$ selection strategy results in a high number of iterations d and thus the total runtime is longer than with our method (MAX). SVD is the theoretical minimum, i.e., it is not possible to obtain a smaller d for the given error threshold. We prefer our maximum norm strategy (MAX), because it is very fast and produces bases with dimensions not too far from the optimum.

4. Skinning Decomposition

In this section we present our method to find matrices $\mathbf{T}_r \in R^{d\times 4P}$ and $\mathbf{X} \in R^{4P\times N}$, where $\mathbf{X}$ is sparse and contains rest pose vertex positions multiplied by weights (as discussed in Section 1). For a given P, the task is to minimize $\|\mathbf{T}_r\mathbf{X} - \mathbf{C}\|_F$, where $\mathbf{C} \in R^{d\times N}$ was computed by Algorithm 1 from Section 3.

We start by establishing initial rigid skinning weights $\mathbf{X}_0$ (Section 4.1), followed by computing the corresponding $\mathbf{T}_{r,0}$ (Section 4.2). The reconstruction $\mathbf{T}_{r,0}\mathbf{X}_0$ is typically crude and non-smooth, but this is quickly improved after transformation (Section 4.2), rest pose (Section 4.3) and vertex weights optimization (Section 4.4), thereby producing a more accurate $\mathbf{T}_{r,k}\mathbf{X}_k$ in the k-th step. We experimentally found the order of optimization operations has little influence on the result.

4.1. Initialization

Since initialization of the rest pose is straightforward (for example, we can use the first frame of the animation), the main task is to cluster the vertices of the input animated mesh (note that rigid skinning corresponds to segmentation). This problem has been studied extensively in the literature [Sha08] and we tested several previously proposed approaches, such as mean-shift clustering [JT05], clustered PCA [SSK05] and hierarchical clustering [WPP07, SY07].

While each method has certain unique features, we found that the subsequent iterative optimization blends away the differences between different initializations of $\mathbf{X}_0$ and typically results in quite similar errors $\|\mathbf{T}_{r,k}\mathbf{X}_k - \mathbf{C}\|_F$ for higher k. Therefore, in our algorithm, we have decided to apply only a very simple segmentation based on multiple source region growing. We start by selecting P seed triangles distributed approximately uniformly over the rest pose mesh by solving the p-center problem as in [KMD*07] (only executed on triangles instead of vertices). Each cluster will be required to be contiguous and is initialized to the seed triangle vertices. We extract deformation gradients $\mathcal{D}_1, \ldots, \mathcal{D}_P$ of the seed triangles and for every cluster c we maintain a priority queue of immediately adjacent vertices, sorted by their $\mathcal{D}_c$ prediction error. Checking the fronts of all the priority queues, we find the vertex $\mathbf{v}$ with the minimal prediction error and append it to its cluster, updating the data structures accordingly. This process is repeated until all vertices are assigned a cluster.

Even though this approach is undoubtedly inferior to advanced clustering techniques, the subsequent vertex weights optimization (Section 4.4) attracts proxy-bone influences to their optimal locations (and also allows them to overlap). For example, the initial sketchy segmentation of the character's head (Figure 5 left) is considerably improved after 15 iterations of our optimization process (Figure 5 right)—in particular, note that the whole head becomes influenced by one bone only. While more careful initialization may lead to slightly lower approximation error, we prefer our region growing technique because it is simple to implement and very fast (e.g., the segmentation in Figure 5 was computed in 0.13s, which is several thousand times faster than mean-shift clustering, requiring 618s—see Section 5).

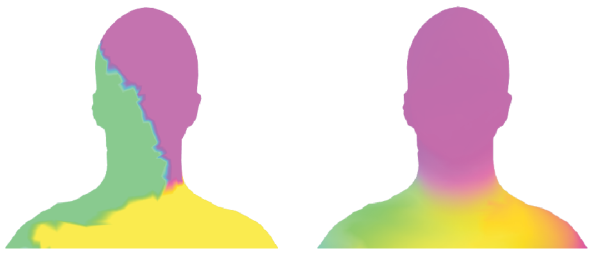
Figure 5: *Our clustering technique produces crude initial segmentation (left), but this is fixed in subsequent optimization (right) (samba dataset, see Table 2).*

4.2. Transformations Optimization

For any given $\mathbf{X} \in R^{4P\times N}$ and $\mathbf{C} \in R^{d\times N}$, the task is to compute the matrix $\mathbf{T}_r \in R^{d\times 4P}$ so that $\|\mathbf{T}_r\mathbf{X} - \mathbf{C}\|_F$ is minimized. This is a linear least squares problem and it can be solved efficiently by inverting the normal equations:

$$\mathbf{T}_r\mathbf{X}\mathbf{X}^T = \mathbf{C}\mathbf{X}^T$$

while exploiting the sparsity of $\mathbf{X}$ (i.e., the maximum of 16 non-zero elements in each column). With indexed storage, the product $\mathbf{X}\mathbf{X}^T$ can be computed in time $O(PN + P^2)$ instead of $O(P^2N)$. Similarly, the asymptotic cost of computing $\mathbf{C}\mathbf{X}^T$ can be reduced to $O(dP + dN)$.

We compute the inverse of $\mathbf{X}\mathbf{X}^T$ using Cholesky decomposition [GL96]. If the Cholesky algorithm detects singular $\mathbf{X}\mathbf{X}^T$, we fallback to a (slower) pseudoinverse computed using eigen-decomposition. However, this happens only in pathological cases (usually, the system is sufficiently overdetermined). We also experimented with sparse Cholesky factorization, but we found that $\mathbf{X}\mathbf{X}^T$ is typically not too sparse (and the Cholesky factor even less so) and therefore we opt for the dense LAPACK routines `dpotrf` and `dpotri` [ABB*99].

4.3. Rest Pose Optimization

Recall that $\mathbf{X} \in R^{4P\times N}$ combines both rest pose vertices and weights, i.e., $\mathbf{X}_{i,j} = w_{j,i}\tilde{\mathbf{v}}_j \in R^{4\times 1}$. For a given $\mathbf{T}_r \in R^{d\times 4P}$, $\mathbf{C} \in R^{d\times N}$ and vertex weights $w_{i,j} \in R$, the task is to compute the rest pose vertices $\tilde{\mathbf{v}}_1, \dots, \tilde{\mathbf{v}}_N \in R^{4\times 1}$ in $\mathbf{X}$ so that $\|\mathbf{T}_r\mathbf{X} - \mathbf{C}\|_F$ is minimized. Because of the special structure of $\mathbf{X}$, it is convenient to denote also the elements of $\mathbf{T}_r$ in vector form $(\mathbf{t}_{i,j})$ where $\mathbf{t}_{i,j} \in R^{1\times 4}$ for $i = 1,\dots,d$, $j = 1,\dots P$. Using this notation we can rewrite equations $\mathbf{T}_r\mathbf{X} \approx \mathbf{C}$ as:

$$\sum_{k=1}^{P} \mathbf{t}_{i,k} w_{j,k} \tilde{\mathbf{v}}_j \approx C_{i,j} \quad i = 1,\dots,d,\ j = 1,\dots,N$$

The optimization thus reduces to N linear least squares problems of the form:

$$\begin{pmatrix} \sum_{k=1}^{P} \mathbf{t}_{1,k} w_{j,k} \\ \vdots \\ \sum_{k=1}^{P} \mathbf{t}_{d,k} w_{j,k} \end{pmatrix} \tilde{\mathbf{v}}_j \approx \begin{pmatrix} C_{1,j} \\ \vdots \\ C_{d,j} \end{pmatrix}, \quad j = 1,\dots,N$$

Since the fourth coordinate of $\tilde{\mathbf{v}}_j$ is usually supposed to be 1 (homogeneous coordinates), we subtract the last column of the system matrix from the right hand side, obtaining equations $\Lambda_j \tilde{\mathbf{v}}'_j \approx \Gamma_j$, where $\tilde{\mathbf{v}}'_j \in R^{3\times 1}$, $\Lambda_j \in R^{d\times 3}$ and $\Gamma_j \in R^{d\times 1}$. The corresponding normal equations have especially compact form:

$$\Lambda_j^T \Lambda_j \tilde{\mathbf{v}}'_j = \Lambda_j^T \Gamma_j$$

where $\Lambda_j^T \Lambda_j$ is just a 3×3 matrix and thus can be inverted efficiently, e.g., using cofactors (with a fallback to pseudoinverse if $\det(\Lambda_j^T \Lambda_j)$ is close to zero. However, this is rarely the case.)

4.4. Vertex Weights Optimization

For a given $\mathbf{T}_r \in R^{d\times 4P}$, $\mathbf{C} \in R^{d\times N}$ and rest pose vertices $\tilde{\mathbf{v}}_1, \dots, \tilde{\mathbf{v}}_N \in R^{4\times 1}$, the task is to determine vertex weights $w_{i,j} \in R$ in $\mathbf{X}$ so that $\|\mathbf{T}_r\mathbf{X} - \mathbf{C}\|_F$ is minimized subject to the constraints that $w_{i,1}, \dots, w_{i,P}$ are convex and only 4 of them are non-zero for every $i = 1,\dots,N$.

As above, the weights can be optimized separately for every vertex $\tilde{\mathbf{v}}_i$. Using the notation of $\mathbf{T}_r$ elements introduced in Section 4.3, the problem for the i-th vertex can be stated in matrix form as follows:

$$\underbrace{\begin{pmatrix} \mathbf{t}_{1,1}\tilde{\mathbf{v}}_i & \cdots & \mathbf{t}_{1,P}\tilde{\mathbf{v}}_i \\ \vdots & \ddots & \vdots \\ \mathbf{t}_{d,1}\tilde{\mathbf{v}}_i & \cdots & \mathbf{t}_{d,P}\tilde{\mathbf{v}}_i \end{pmatrix}}_{\Theta \in R^{d\times P}} \begin{pmatrix} w_{i,1} \\ \vdots \\ w_{i,P} \end{pmatrix} \approx \underbrace{\begin{pmatrix} C_{1,i} \\ \vdots \\ C_{d,i} \end{pmatrix}}_{\mathbf{y} \in R^{d\times 1}}$$

We start by choosing the weights which will be allowed to be non-zero by selecting the four columns of Θ that are closest to the right hand side $\mathbf{y}$ (this corresponds to selecting the bones that individually best predict the deformed vertices, as in [JT05]).

If we denote the chosen columns as $\phi_1, \dots, \phi_4 \in R^{d\times 1}$ and the set of their convex combinations as:

$$L = \left\{ \sum_{k=1}^{4} \alpha_k \phi_k : \sum_{k=1}^{4} \alpha_k = 1, \alpha_k \geq 0 \right\} \subseteq R^{d\times 1}$$

we can formulate the problem as finding $\mathbf{m} \in L$ which minimizes $\|\mathbf{y} - \mathbf{m}\|$. While previous work typically solves this problem using general purpose Non-Negative Least Squares solver [LH74], we propose to take advantage of the small and fixed number of influencing bones per vertex. In this case, our approach is simpler, faster and more accurate (note that with NNLS, the affinity condition $\sum_{k=1}^{4} \alpha_k = 1$ is not enforced exactly, but is treated as a soft constraint [JT05]).

Geometrically, the set L is a tetrahedron in d-dimensional space and the task is to find its closest point to $\mathbf{y} \in R^{d\times 1}$. We will show that this problem can be reduced to three dimensions, which simplifies its solution considerably. To accomplish this, we shift both the tetrahedron L and the point $\mathbf{y}$, obtaining $L' = L - \phi_4$ and $\mathbf{y}' = \mathbf{y} - \phi_4$. The new tetrahedron L' thus has one of its vertices coincident with the origin and therefore, there is a 3D linear subspace $S \subseteq R^{d\times 1}$ such that $L' \subseteq S$. It can be shown that finding $\mathbf{m}' \in L'$ minimizing $\|\mathbf{y}' - \mathbf{m}'\|$ is equivalent to finding $\mathbf{m}' \in L'$ minimizing $\|\mathbf{y}'_S - \mathbf{m}'\|$, where $\mathbf{y}'_S$ is the orthogonal projection of $\mathbf{y}'$ on the subspace S (see Figure 6 for an intuitive justification and the Appendix for a formal proof).

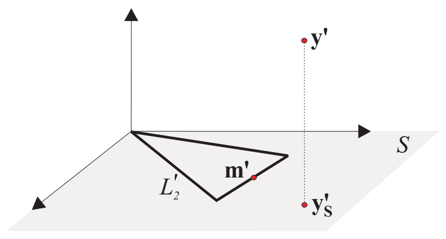

Figure 6: *Illustration of our problem for $d = 3$ and triangle L'_2 (instead of tetrahedron). We can see that if $\mathbf{m}' \in L'_2$ is as close as possible to $\mathbf{y}'_S$, it is also as close as possible to $\mathbf{y}'$.*

Therefore, if we pick an orthonormal basis $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \in R^{d\times 1}$ of S, we can express both the tetrahedron vertices $\phi_1 - \phi_4, \phi_2 - \phi_4, \phi_3 - \phi_4, 0$ and the point $\mathbf{y}'_S$ using three coordinates only, thereby reducing it to the 3D problem of finding the closest point on a tetrahedron to a point, which can be solved efficiently [Eri04] (since calculations in three dimensions are obviously much faster than in d dimensions). If the optimal $\mathbf{m}'$ (which always exists and is unique) is expressed in barycentric coordinates, then these coordinates represent exactly the desired vertex weights $\alpha_1, \dots, \alpha_4$.

The orthonormal basis $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \in R^{d\times 1}$ can be computed by executing Gram-Schmidt orthonormalization on vectors $\phi_1 - \phi_4, \phi_2 - \phi_4, \phi_3 - \phi_4 \in R^{d\times 1}$. Since there are only three vectors, this is very simple and numerically stable even with the most straightforward implementation. If the Gram-Schmidt process detects the vectors $\phi_1 - \phi_4, \phi_2 - \phi_4, \phi_3 - \phi_4$ to be close to linearly dependent, it means that one of the chosen proxy-bones is redundant and we replace it with another column of matrix Θ.

Note that while the above described procedure could be generalized to $h > 4$ influencing bones, the number of faces of a h-dimensional simplex grows exponentially with h and therefore, for higher h, a generic solver such as NNLS would be preferable. However, a high number of vertex weights is not desirable anyway and, if present, weight reduction techniques can be applied [LS09].

5. Runtime Implementation and Results

While the evaluation of Formula (1) is trivial, for a high performance implementation it is often necessary to consider particulars of the target graphics hardware [Lee07]. Most 3D engines contain skinning routines and/or shaders optimized for the supported platforms.

There are several strategies to handle vertex normals, the simplest being to transform them by $\sum_{j=1}^{P} w_{i,j} \left(\mathbf{M}_j^{(f)}\right)^{-T}$ [MG03]. This is only an approximation, but an accurate solution is possible at the cost of extra vertex attributes [MMG06b]. In our implementation we adopt the approach of Wang and colleagues [WPP07], who compute vertex normals by averaging normals of the incident triangles. This results in smooth normal fields and allows for efficient implementation on modern graphics hardware.

In previous work [JT05, KMD*07], approximation error is typically reported in terms of the following error metric, adopted from [KG04]:

$$100.0\% \cdot \|\mathbf{A} - \mathbf{A}_{approx}\|_F / \|\mathbf{A} - \mathbf{A}_{timeAverage}\|_F$$

where $\mathbf{A}_{timeAverage} \in R^{3F\times N}$ is a matrix representing a static mesh (average of all keyframes). Unfortunately, this metric is sensitive to global motion applied to the entire mesh. For example, adding linear motion increases the denominator $\|\mathbf{A} - \mathbf{A}_{timeAverage}\|_F$ but leaves the numerator $\|\mathbf{A} - \mathbf{A}_{approx}\|_F$ unchanged because skinning can trivially reproduce translation (by simply adding the translation to all bone transformations). For example, an on-spot walking sequence will have higher error than exactly the same animation where the character moves forward.

Therefore, we propose to simply use $\sqrt{3NF}$ as the normalizing factor, obtaining:

$$E_{RMS} = 1000 \frac{\|\mathbf{A} - \mathbf{A}_{approx}\|_F}{\sqrt{3NF}}$$

Since $3NF$ is the total number of elements of matrix $\mathbf{A}$, E_{RMS} is simply the average error per element (scaled by 1000 for convenience). To be able to compare E_{RMS} between objects with different proportions, we uniformly scale every animation so that its first frame is tightly enclosed by a unit ball. Therefore, roughly speaking, E_{RMS} can also be interpreted as the number of pixels on a 1000×1000 parallel-projection viewport displaying our mesh full screen. With our method, the error threshold ε in Algorithm 1 (Section 3) is always set to $0.0005\sqrt{3NF}$ to obtain approximately half-pixel accuracy of the initial dense dimensionality reduction. The results for various testing animations are summarized in Table 2. Please note the E_{RMS} reported for our method accounts for the final skinning error $\|\mathbf{TX} - \mathbf{A}\|_F$ (and not the error in the reduced coordinates $\|\mathbf{T}_r\mathbf{X} - \mathbf{C}\|_F$, which would be slightly lower).

With SMA, we used the (best possible) results from [JT05], where available. In the case of the skirt and cloak animations, we were unable to configure mean-shift clustering to produce enough clusters. This is probably an issue with the applied mean-shift software [GSM03], mentioned also in [KMD*07]. Our implementations of SMA and SAD use flexible (affine) bones and non-negative vertex weights, just like our method.

For comparison, we also report the results of two related methods: clustered PCA [SSK05] and truncated SVD [AM00]. The number of clusters for CPCA is set to the number of bones P and the number of dimensions per cluster is set to 4, to obtain the same amount of per-frame data as with skinning. While the approximation error of CPCA reported in Table 2 is attractive, the actual visual quality suffers from the fact that clusters are not allowed to overlap (see Figure 7). With truncated SVD, the number of dimensions is set to $4P$ (in order to obtain the same amount of per-frame data, as above). Even though the approximation error of SVD is very low, note that SVD requires much more per-vertex data (i.e., $4P$) than skinning (which is constant). In other words, each SVD "bone" is global and can affect the whole mesh (unlike bones in skinning which have localized influences). In practice, this implies not only higher memory consumption, but especially more complex vertex reconstruction, requiring $O(P)$ arithmetic operations and data fetches (while skinning requires only $O(1)$).

From Table 2 we can see that our method is never worse and is typically several times better than SMA, SAD and CPCA in terms of approximation error. In many cases, our method outperforms SMA and/or SAD even after rank-10 EigenSkin corrections (no corrective techniques were applied with our method). Note that while SMA performs better for quasi-articulated animations (Table 2 bottom) and SAD for highly deformable ones (Table 2 top), our method handles both cases equally well, as well as animations with both quasi-articulated and highly-deformable components (Table 2 middle).

We measured the execution time of all algorithms on an Intel Core 2 Duo (3.0GHz) processor with 2GB RAM. The

Input Data				Approximation Error E_{RMS}					Execution Time		
Model	Vertices	Frames	Bones	SMA	SAD	Our	CPCA	SVD	SMA	SAD	Our total (init, 1 step)
elasticCow	2904	204	18	17 (9.2)	16 (9.1)	3.6	7.3	1.3	2.3m	2.8m	5.0s (1.8s, 0.21s)
clothHorse	8431	53	6	48 (1.3)	26 (0.8)	8.3	12.4	1	2.7m	3.5m	3.5s (0.5s, 0.20s)
flag_{32}	6906	200	32	27 (7.6)	7 (2.6)	1.6	3.2	0.3	6.7m	22m	13.3s (4.0s, 0.62s)
flag_{100}	6906	200	100	1.9 (1.6)	1.6 (1)	0.7	1.4	0.01	10.5m	68m	23.9s (4.0s, 1.32s)
skirt_{22}	5095	360	22	34 (22)	6.6 (6.4)	2.5	4.6	1.2	6.9m	25.2m	22.8s (9.9s, 0.86s)
skirt_{50}	5095	360	50	-	3.4 (3)	1	2.6	0.2	-	32.1m	24.4s (9.9s, 0.97s)
cloak_{10}	3069	360	10	13 (12)	6.6 (5.8)	3	5.6	1.9	4.3m	9.8m	6.3s (3.2s, 0.21s)
cloak_{35}	3069	360	35	-	2.8 (2.5)	0.8	2.2	0.2	-	15m	7.0s (3.2s, 0.25s)
samba	9971	175	30	8.6 (3.6)	11.4 (6)	1.5	4	0.2	10.3m	20.2m	15.8s (4.2s, 0.77s)
crane	10002	175	40	6 (2.8)	5.9 (4.5)	1.4	3.6	0.2	12.6m	20.6m	21.7s (5.6s, 1.08s)
swing	9971	175	30	8.8 (3.5)	26 (16)	1.6	4.2	0.3	10.6m	18.7m	18.6s (5.0s, 0.91s)
horse	8431	48	30	2.3 (0.3)	4.9 (2.9)	1.3	2.4	2E-5	2.7m	6m	4.3s (0.4s, 0.26s)
camel	21887	48	23	3.1 (0.5)	4.7 (2.2)	1.4	2.8	2E-4	16.4m	7.5m	11.3s (1.0s, 0.69s)
elephant	42321	48	25	2.6 (0.5)	15 (6.5)	1.4	2.3	6E-5	26.3m	20.1m	21.9s (1.9s, 1.34s)

Table 2: *Results for Skinning Mesh Animations (SMA), Skinning Arbitrary Deformations (SAD), Our method, Clustered PCA and truncated SVD. With SMA and SAD we report also approximation error after rank-10 EigenSkin corrections (in brackets). Note that the time for SMA and SAD (without EigenSkin) is measured in minutes and the time for our method in seconds (with a breakdown to initialization and iteration time in brackets).*

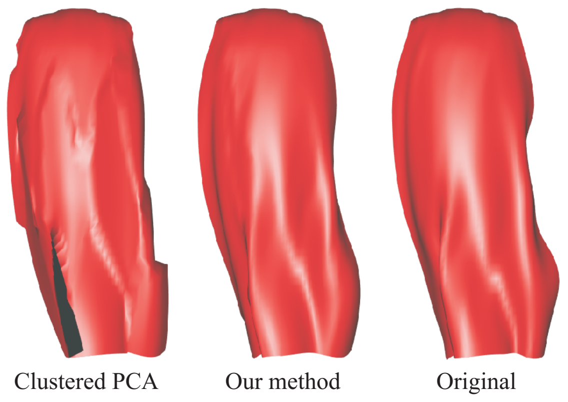

Figure 7: *In contrast to skinning computed with our method, clustered PCA may result in non-smooth reconstruction (cloak_{10} dataset).*

time reported for SMA is only the time of mean-shift clustering computed using the publicly available implementation [GSM03]. In the case of SAD, the execution time is dominated by the LSQR solver [PS82]. The total running time of our method is the sum of the initialization time (Section 3) plus the time for 15 iterations of coordinate descent (Section 4). From Table 2 we can see that our algorithm runs about one to two orders of magnitude faster than SMA and SAD.

To examine the convergence of our iterative optimization process (Section 4), we measured E_{RMS} after each iteration, see Figure 8. We can see that the error quickly approaches its optimal value regardless of the animation type—in our implementation we therefore terminate the optimization after 15 iterations. Each iteration takes typically less than 1 second (see Table 2), which enables almost interactive previewing of the optimization process. Note that the initialization needs to be done only once regardless of P, and therefore, manual selection of the number of bones is quite comfortable. If a fully automatic pipeline is required, we can find the optimal P for a given global error threshold (e.g., $E_{RMS} = 1.5$) using binary or interpolation search at the cost of running the optimization multiple times.

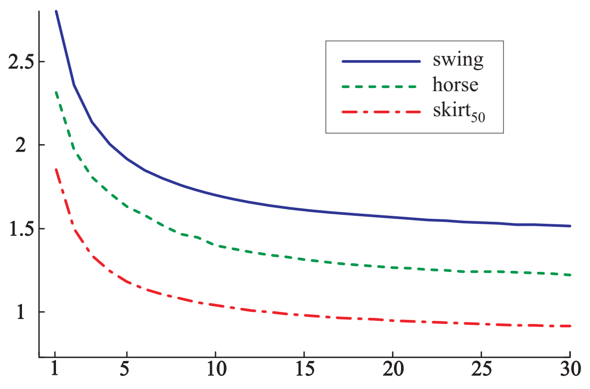

Figure 8: *Approximation error E_{RMS} (vertical axis) vs. number of iterations of our optimization process (horizontal axis).*

5.1. Applications

The main advantages of skinning are compact animation representation combined with efficient hardware accelerated rendering. For example, in applications featuring large numbers of characters with limited motion styles (such as pedestrian crowds), we can precompute physically based cloth animations for the given set of skeletal motions (perhaps organized in a motion graph) and use our method to create their skinned approximations, which can be trivially exported to any 3D engine that supports linear blend skinning. This enables us to enhance fidelity of virtual characters with minimal computational and implementation costs, see Figure 9. Many other applications of skinning decomposition were discussed in the literature, ranging from rest pose editing and efficient collision detection [JT05, KMD*07] (including the recently proposed technique for self-collision detection [SGO09]) to advanced non-linear deformation systems [WPP07] and animation interfaces [FKY08].

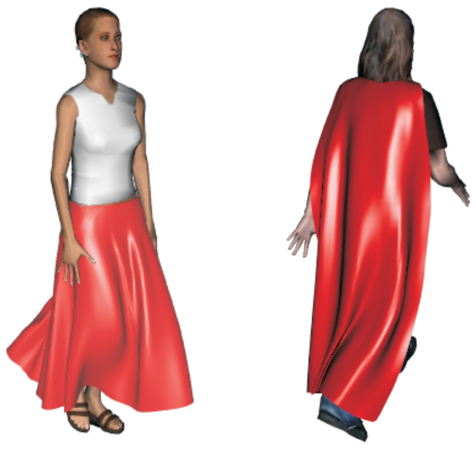

Figure 9: *Application of our method in practice: both the bodies and the cloth are rendered with linear blend skinning (skirt$_{50}$ and cloak$_{35}$ datasets).*

6. Conclusions and Future Work

We present an algorithm to quickly construct high quality skinned approximations of arbitrary mesh animations, demonstrating that even complex highly deformable animations can be skinned efficiently without any corrective techniques or excessive pre-processing. We believe that these features will make our method appealing for applications, such as in games development.

There are several possible avenues for future work. If higher pre-computation speeds were required, we could implement our algorithms on parallel computing architectures such as CUDA, taking advantage of easily parallelizable matrix computations. One limitation of skinning is that it only addresses spatial coherence. For longer (or finely sampled) animation sequences, it may be desirable to tackle temporal coherence also using methods such as in [Ari06]. From a broader perspective, skinning can be considered as a special kind of sparse matrix decomposition and does not have to be limited to mesh animation—one could, for example, "skin" ambient occlusion or precomputed radiance transfer.

7. Acknowledgements

We would like to thank the anonymous reviewers and our colleagues, Martin Pražák, Daniel Sýkora, Rachel McDonnell and Simon Dobbyn, for their insightful comments and suggestions. We are also indebted to Peter Lozsek for creating the characters and cloth animation (see Figure 9). We thank Doug James for the flag animation [JT05], Robert Sumner and Jovan Popović for the clothHorse, horse, camel and elephant datasets [SP04] and Daniel Vlasic and his colleagues for the samba, swing and crane mesh sequences [VBMP08]. The elasticCow animation was obtained from the geometry videos project [BSM*03]. This work was supported by Science Foundation Ireland (project Metropolis).

Appendix

Lemma: Let L' be a compact convex set and S a three dimensional linear subspace such that $L' \subseteq S \subseteq R^d$. Let $\mathbf{y}'$ be an arbitrary point from R^d and let $\mathbf{y}'_S \in R^d$ be its orthogonal projection on the subspace S. Then point $\mathbf{m}' \in L'$ minimizes $\|\mathbf{y}' - \mathbf{m}'\|$ if and only if $\mathbf{m}'$ minimizes $\|\mathbf{y}'_S - \mathbf{m}'\|$.

Proof: Let $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \in R^d$ be an orthonormal basis of S and let $\mathbf{b}_4, \ldots, \mathbf{b}_d \in R^d$ be its orthonormal complement to the basis of R^d. Therefore, $\mathbf{y}' = \sum_{i=1}^d \beta_i \mathbf{b}_i$ for $\beta_i \in R$ and its orthogonal projection on S is simply $\mathbf{y}'_S = \beta_1 \mathbf{b}_1 + \beta_2 \mathbf{b}_2 + \beta_3 \mathbf{b}_3$. Similarly, any $\mathbf{m}' \in L'$ can be written as $\mathbf{m}' = \mu_1 \mathbf{b}_1 + \mu_2 \mathbf{b}_2 + \mu_3 \mathbf{b}_3$ for some $\mu_i \in R$. Therefore,

$$\begin{aligned} \|\mathbf{y}' - \mathbf{m}'\|^2 &= \left\|\textstyle\sum_{i=1}^3 (\beta_i - \mu_i)\mathbf{b}_i + \sum_{i=4}^d \beta_i \mathbf{b}_i\right\|^2 \\ &= \left\|\textstyle\sum_{i=1}^3 (\beta_i - \mu_i)\mathbf{b}_i\right\|^2 + \left\|\textstyle\sum_{i=4}^d \beta_i \mathbf{b}_i\right\|^2 \\ &= \left\|\mathbf{y}'_S - \mathbf{m}'\right\|^2 + \textstyle\sum_{i=4}^d \beta_i^2 \end{aligned}$$

where we used the fact that $\mathbf{b}_1, \ldots, \mathbf{b}_d \in R^d$ are orthonormal. Since the term $\sum_{i=4}^d \beta_i^2$ is fixed (it is determined solely by $\mathbf{y}'$ and S), it follows that $\|\mathbf{y}' - \mathbf{m}'\|^2$ is minimized if and only if $\|\mathbf{y}'_S - \mathbf{m}'\|^2$ is minimized (and the same applies after taking their square roots). □

References

[ABB*99] ANDERSON E., BAI Z., BISCHOF C., BLACKFORD S., DEMMEL J., DONGARRA J., DU CROZ J., GREENBAUM A., HAMMARLING S., MCKENNEY A., SORENSEN D.: *LAPACK Users' Guide*, third ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.

[AM00] ALEXA M., MÜLLER W.: Representing animations by principal components. *Comput. Graph. Forum 19*, 3 (2000), 411–418.

[Ari06] ARIKAN O.: Compression of motion capture databases. *ACM Trans. Graph. 25*, 3 (2006), 890–897.

[BP07] BARAN I., POPOVIĆ J.: Automatic rigging and animation of 3D characters. *ACM Trans. Graph. 26*, 3 (2007), 72.

[BSM*03] BRICENO H. M., SANDER P. V., MCMILLAN L., GORTLER S., HOPPE H.: Geometry videos: a new representation for 3D animations. *SCA '03: Proceedings of the 2003*

ACM SIGGRAPH/Eurographics symposium on Computer animation (2003), 136–146.

[dATTS08] DE AGUIAR E., THEOBALT C., THRUN S., SEIDEL H.-P.: Automatic Conversion of Mesh Animations into Skeleton-based Animations. *Computer Graphics Forum (Proc. Eurographics EG'08) 27*, 2 (4 2008), 389–397.

[Eri04] ERICSON C.: *Real-Time Collision Detection*. Morgan Kaufmann Publishers Inc., 2004.

[FKY08] FENG W.-W., KIM B.-U., YU Y.: Real-time data driven deformation using kernel canonical correlation analysis. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (New York, NY, USA, 2008), ACM, pp. 91:1–91:9.

[GB08] GAIN J., BECHMANN D.: A survey of spatial deformation from a user-centered perspective. *ACM Trans. Graph. 27*, 4 (2008), 107:1–107:21.

[GK04] GUSKOV I., KHODAKOVSKY A.: Wavelet compression of parametrically coherent mesh sequences. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2004), ACM Press, pp. 183–192.

[GL96] GOLUB G., LOAN C. F. V.: *Matrix Computations*, third ed. John Hopkins University Press, Baltimore, 1996.

[GLRvdE05] GIRAUD L., LANGOU J., ROZLOŽNÍK M., VAN DEN ESHOF J.: Rounding error analysis of the classical Gram-Schmidt orthogonalization process. *Numerische Mathematik 101* (2005), 87–100.

[GSM03] GEORGESCU B., SHIMSHONI I., MEER P.: Mean shift based clustering in high dimensions: a texture classification example. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on* (2003), pp. 456–463 vol.1.

[JF03] JAMES D., FATAHALIAN K.: Precomputing interactive dynamic deformable scenes. Technical report CMU-RI-TR-03-33, Carnegie Mellon University, 2003.

[JT05] JAMES D. L., TWIGG C. D.: Skinning mesh animations. *ACM Trans. Graph. 24*, 3 (2005), 399–407.

[KCO09] KAVAN L., COLLINS S., O'SULLIVAN C.: Automatic linearization of nonlinear skinning. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (February/March 2009), ACM Press, pp. 49–56.

[KCŽO08] KAVAN L., COLLINS S., ŽÁRA J., O'SULLIVAN C.: Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph. 27*, 4 (2008), 105:1–105:23.

[KG04] KARNI Z., GOTSMAN C.: Compression of soft-body animation sequences. *Computers & Graphics 28*, 1 (2004), 25–34.

[KG05] KIRCHER S., GARLAND M.: Progressive multiresolution meshes for deforming surfaces. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2005), ACM Press, pp. 191–200.

[KJP02] KRY P. G., JAMES D. L., PAI D. K.: EigenSkin: real time large deformation character skinning in hardware. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2002), ACM Press, pp. 153–159.

[KMD*07] KAVAN L., MCDONNELL R., DOBBYN S., ŽÁRA J., O'SULLIVAN C.: Skinning arbitrary deformations. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (April/May 2007), ACM Press, pp. 53–60.

[Lee07] LEE M.: Seven ways to skin a mesh: Character skinning revisited for modern GPUs. In *Gamefest Unplugged (Europe)* (2007).

[LH74] LAWSON C. L., HANSON R. J.: *Solving Least Squares Problems*. Prentice Hall, Englewood Cliffs, NJ, 1974.

[LS09] LANDRENEAU E., SCHAEFER S.: Poisson-based weight reduction of animated meshes. *Computer Graphics Forum 28* (2009), to appear.

[Mar06] MARTIN S.: An approximate version of kernel PCA. In *ICMLA '06: Proceedings of the 5th International Conference on Machine Learning and Applications* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 239–244.

[MG03] MOHR A., GLEICHER M.: Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3 (2003), 562–568.

[MMG06a] MERRY B., MARAIS P., GAIN J.: Animation space: A truly linear framework for character animation. *ACM Trans. Graph. 25*, 4 (2006), 1400–1423.

[MMG06b] MERRY B., MARAIS P., GAIN J.: Normal transformations for articulated models. In *SIGGRAPH '06: ACM SIGGRAPH 2006 sketches* (New York, NY, USA, 2006), ACM, p. 134.

[PC98] PARTRIDGE M., CALVO R.: Fast dimensionality reduction and simple PCA. *Intelligent Data Analysis 2*, 1 (1998), 203–214.

[PS82] PAIGE C. C., SAUNDERS M. A.: Algorithm 583: LSQR: Sparse linear equations and least squares problems. *ACM Trans. Math. Softw. 8*, 2 (1982), 195–209.

[RK09] RUITERS R., KLEIN R.: BTF compression via sparse tensor decomposition. *Computer Graphics Forum 28*, 4 (July 2009), 1181–1188.

[SGO09] SCHVARTZMAN S. C., GASCÓN J., OTADUY M. A.: Bounded normal trees for reduced deformations of triangulated surfaces. In *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), ACM, pp. 75–82.

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. *Computer Graphics Forum 27*, 6 (2008), 1539–1556.

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3 (2004), 399–405.

[SSK05] SATTLER M., SARLETTE R., KLEIN R.: Simple and efficient compression of animation sequences. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2005), ACM Press, pp. 209–217.

[SY07] SCHAEFER S., YUKSEL C.: Example-based skeleton extraction. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 153–162.

[Váš08] VÁŠA L.: *Methods for size reduction of dynamic meshes*. PhD thesis, University of West Bohemia, 2008.

[VBMP08] VLASIC D., BARAN I., MATUSIK W., POPOVIĆ J.: Articulated mesh animation from multi-view silhouettes. *ACM Transactions on Graphics 27*, 3 (2008), 97:1–97:9.

[WP02] WANG X. C., PHILLIPS C.: Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2002), ACM Press, pp. 129–138.

[WPP07] WANG R. Y., PULLI K., POPOVIĆ J.: Real-time enveloping with rotational regression. *ACM Trans. Graph. 26*, 3 (2007), 73:1–73:9.

[WTH09] WITTEN D. M., TIBSHIRANI R., HASTIE T.: A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Biostat 10*, 3 (July 2009), 515–534.