Automatic Linearization of Nonlinear Skinning

Ladislav Kavan* Steven Collins Carol O'Sullivan

Trinity College Dublin

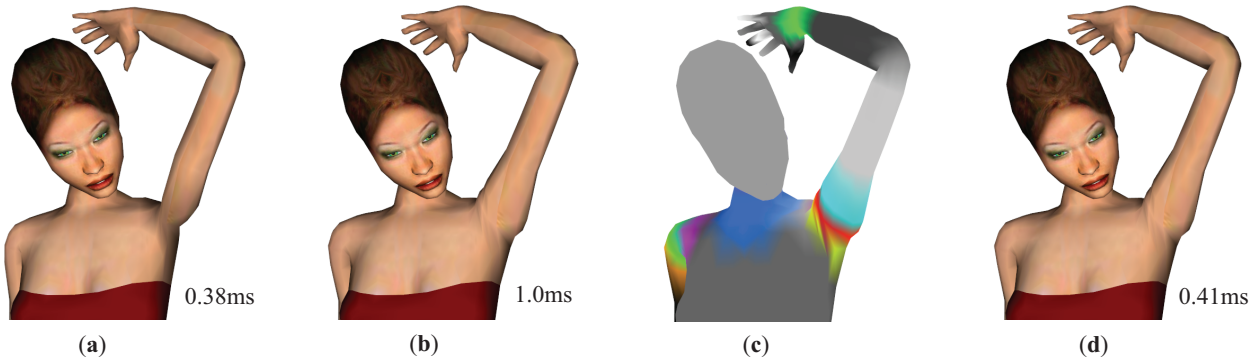

Figure 1: Linear blending is one of the fastest skinning methods, but suffers from volume-collapsing artifacts (a). These artifacts can be removed by a nonlinear blending method (such as dual quaternions), but at the cost of more complex vertex processing (b). Our method automatically places virtual bones in critical parts of the model (c), so that linear blending using these virtual bones efficiently approximates the nonlinear skinning technique (d).

Abstract

Linear blending is a very popular skinning technique for virtual characters, even though it does not always generate realistic deformations. Recently, nonlinear blending techniques (such as dual quaternions) have been proposed in order to improve upon the deformation quality of linear skinning. The trade-off consists of the increased vertex deformation time and the necessity to redesign parts of the 3D engine. In this paper, we demonstrate that any nonlinear skinning technique can be approximated to an arbitrary degree of accuracy by linear skinning, using just a few samples of the nonlinear blending function (virtual bones). We propose an algorithm to compute this linear approximation in an automatic fashion, requiring little or no interaction with the user. This enables us to retain linear skinning at the core of our 3D engine without compromising the visual quality or character setup costs.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: skinning, approximation, linearization

1 Introduction

Linear blending is the most popular skin deformation technique for virtual characters, being used in numerous 3D game engines and digital content creation software. Unfortunately, larger joint rotations result in joint collapsing artifacts (also known as *candy-wrapper* effects). One approach to eliminate these artifacts is dual quaternion skinning [Kavan et al. 2008], which employs a nonlinear transformation blending method. Even though dual quaternion blending is still a relatively simple procedure, it is slower than linear blending (especially when scaling bones have to be supported) and requires modifications of time-critical components of the 3D engine, which can be painstaking (especially if the routines and/or shaders are optimized at the lowest level). Moreover, nonlinear skinning is typically only necessary in certain parts of the model (such as shoulders and wrists), while linear blending can be safely applied elsewhere.

A technique that eliminates the artifacts of linear skinning without introducing the overheads of nonlinear blending has been described previously [Weber 2000]. The main idea is to use virtual bones to split large joint rotations into smaller ones which can be blended linearly without introducing any noticeable artifacts. The drawback is the extra manual effort needed to setup the auxiliary bones and reweight skin vertices accordingly. This needs to be repeated for every character model.

In this paper, we propose a method to create virtual bones and update vertex weights in an automatic fashion, using only the original character model and a training skeletal animation (to estimate which parts of the body are more flexible). During pre-processing, our algorithm computes optimal placement of the virtual bones and their vertex weights so that the artifacts of linear blending are compensated for. At run-time, the nonlinear blending is executed only once per virtual bone and vertices are deformed by simple linear blending. At a higher level, our approach can be viewed as constructing piecewise linear approximations of nonlinear skinning operators, with virtual bones corresponding to samples of the nonlinear operator (see Figure 2 for an illustrative example).

While skinning methods that employ virtual bones have been proposed previously [James and Twigg 2005; Wang et al. 2007], our approach is different in that it produces explicit closed-form formulas for computing virtual bone transformations as a function of the skeletal posture. In our case, the input animation is used only to determine the critical regions of our 3D model (and not to learn any particular deformation style, as in previous work). Therefore, our

*e-mail: kavanl@cs.tcd.ie

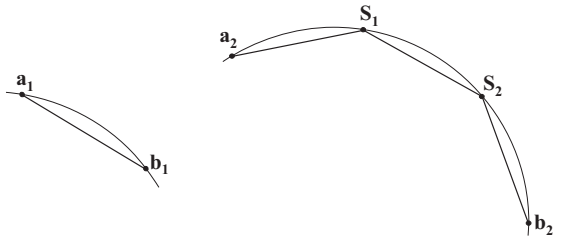

Figure 2: Spherical arc $\mathbf{a}_1, \mathbf{b}_1$ can be approximated linearly with sufficient accuracy. The same level of accuracy can be obtained even for arc $\mathbf{a}_2, \mathbf{b}_2$ by using piecewise linear approximation with two samples (S_1, S_2) of the original curve.

virtual bones can be applied easily on top of an arbitrary skeletal animation (e.g., a mo-cap clip) and produce plausible deformations even for animations far different from the training one.

From a practical point of view, the main contribution of our work is that the linear skin deformation routines and/or shaders can be retained at the core of a 3D engine, without compromising visual quality. This also enables us to directly apply algorithms that assume linear vertex blending [Lewis et al. 2000; Kavan and Žára 2005; Merry et al. 2006b]. Moreover, our method makes it possible to employ more advanced (and more complex) nonlinear skinning operators, including those that were previously considered too slow for practical purposes.

2 Related Work

A number of algorithms improve upon linear blend skinning in terms of deformation quality. Linear blending [Magnenat-Thalmann et al. 1988] is considered to be the simplest and fastest method to produce smooth skinning. The advanced methods can therefore be classified in terms of time and memory overheads over linear blending, see Table 1.

Method	Extra time	Extra memory
Linear blending	0	0
Nonlinear blending	$O(N)$	$O(1)$
Multilinear blending	0	$O(N)$
Virtual bones with stitching	$O(\beta^2)$	$O(\beta^2)$
Direct virtual bones (our method)	$O(\beta)$	$O(\beta)$

Table 1: Real-time skinning techniques classified by their overheads when compared to linear blending, with N denoting the number of mesh vertices and β the number of virtual bones (note that typically $\beta << N$).

Nonlinear blending. The idea of nonlinear skinning methods is that curved vertex trajectories generate more natural deformations than straight lines. In particular, quaternion-based methods [Hejl 2004; Kavan et al. 2008] operate along spherical (or helical) arcs. Another possibility is to apply customizable curves [Yang et al. 2006; Forstmann et al. 2007], useful especially when users require detailed control over the deformation style. Although these methods require zero or constant memory overhead, the nonlinear deformation function is executed once per vertex. Optimization techniques are therefore often employed in order to improve run-time performance. Note that while [Hejl 2004] reports a faster algorithm than linear blending, it only works correctly for a restricted class of models (in particular, vertices influenced by non-neighboring bones are not supported).

Multilinear blending. Methods employing blending in a higher-dimensional linear space were pioneered by Wang and Philips [2002] and further developed by Merry et al. [2006a] into a concept called *Animation space*. While Animation space offers the same performance as linear blending, the overhead consists of extra per-vertex attributes, thereby increasing the memory footprint of a skinned model. These additional attributes are learned from an input mesh animation, where care must be taken in order to avoid overfitting. Even though Animation space can reduce candy wrapper artifacts, it cannot remove them completely, as the number of vertex attributes is fixed. Our proposed technique converges to artifact-free skinning as the number of virtual bones increases and does not suffer from overfitting.

Virtual bones with stitching. An interesting alternative to direct control of vertex positions is to manipulate the triangle deformation gradients [Sumner and Popović 2004; Anguelov et al. 2005; Wang et al. 2007; Weber et al. 2007]. Given the deformation gradients, vertex positions can be reconstructed by solving a linear least squares problem (process sometimes called *Poisson stitching*). This can be time consuming, but Wang et al. [2007] showed that the time and memory complexity can be reduced by up to $O(\beta^2)$ (where β is the number of virtual bones) by precomputing the inverse of the normal matrix. Skin deformation is then computed using a three-pass GPU technique consisting of stitching (with the precomputed matrix stored in a texture), linear skinning and normal computation steps. The advantage is that advanced effects such as muscle bulging can be captured efficiently. On the other hand, our method is simpler to implement and has a much smaller memory overhead.

Direct virtual bones. This class of methods employs virtual bones to directly influence skin vertices [Weber 2000; Mohr and Gleicher 2003]. The deformation algorithm consists of computing the virtual bone transformations followed by linear skinning. The first pass has linear time and memory complexity with respect to the number of virtual bones β. In practice this overhead is negligible compared to linear skinning (as the number of virtual bones is normally orders of magnitude smaller than the number of vertices). Weber [2000] proposes to compute virtual bone transformations using user-defined B-splines. Mohr and Gleicher [2003] apply one rotation-interpolating virtual bone per joint, computed using Spherical Linear Interpolation [Shoemake 1985] and therefore limited to blending between two bones. Our proposed algorithm can be seen as a generalization of Mohr and Gleicher's approach, allowing us to add an arbitrary number of virtual bones and blend them using a general transformation blending technique. Moreover, our algorithm automatically determines the optimal placement of virtual bones for a training animation.

Skinning mesh animations. Several techniques to approximate general mesh animations by skinning were discussed recently, such as [James and Twigg 2005; Kavan et al. 2007], producing a (non-hierarchical) sequence of virtual bone transformations. While these methods are useful, e.g., for animation compression and local editing, they do not allow a new skeletal motion (such as a mo-cap clip) to be applied easily because of the lack of a bone hierarchy. Subsequent work therefore proposed methods for automatic extraction of hierarchical skeletons [Schaefer and Yuksel 2007; de Aguiar et al. 2008]. In this paper, however, we assume that a hierarchical skeleton is already given (either automatically extracted or authored by an artist) and the goal is to improve on skin deformation. Our virtual bones are not physically present in the skeletal hierarchy and their transformations are computed by blending the original bone transformations for a given posture (therefore, in the following we will use the term *blend bones*). Note that embedding virtual bones in the skeletal hierarchy would be undesirable as this would produce non-physiological skeletons, thereby complicating character animation.

3 Problem Statement

As input, we assume we have a skeletally animated model, consisting of a mesh with N vertices $\mathbf{v}_1,\ldots,\mathbf{v}_N$ (with fixed connectivity) and a skeletal animation containing F frames. For every frame $f = 1,\ldots,F$ we are given bone transformation matrices $C_1^{(f)},\ldots,C_B^{(f)}$, where B is the total number of original bones in the input skeleton (which we will call *master bones*). For every vertex $\mathbf{v}_i$ we are given scalar weights $w_{i,1},\ldots,w_{i,B}$ describing the amount of influence exerted on $\mathbf{v}_i$ by every bone. We require that the weights are convex, i.e., $w_{i,j} \geq 0,\ w_{i,1}+\ldots+w_{i,B} = 1$. Note that most applications assume that a maximum of 4 of the vertex weights $w_{i,1},\ldots,w_{i,B}$ are non-zero (due to hardware considerations and the observation that more than 4 influences are seldom required). Our algorithm is also designed to meet this constraint, which has important implications for the optimization routines (see Section 4).

An arbitrary (both linear or nonlinear) blending technique can be specified by matrix operator $\Phi(\mathbf{w}_i;C_1^{(f)},\ldots,C_B^{(f)})$, where $\mathbf{w}_i = (w_{i,1},\ldots,w_{i,B})$ is a vector containing the vertex weights. The deformed vertex positions for frame f are then computed as follows:

$$\mathbf{v}_i^{(f)} = \Phi(\mathbf{w}_i;C_1^{(f)},\ldots,C_B^{(f)})\mathbf{v}_i$$

A similar formula applies for vertex normals, just using the inverse transpose of Φ. The simplest blending operator is linear:

$$\Phi_{LBS}(\mathbf{w}_i;C_1^{(f)},\ldots,C_B^{(f)}) = \sum_{j=1}^{B} w_{i,j}C_j^{(f)}$$

corresponding to linear blend skinning. We are mostly interested in nonlinear operators Φ (such as Φ_{DQS} defined in Formula (7)), and we want to approximate them by a piece-wise linear operator using β blend bones. In particular, the k-th blend bone is given by weights $\mathbf{u}_k = (u_{k,1},\ldots,u_{k,B})$, which are assumed to be convex (just like vertex weights).

In the output model, every vertex $\mathbf{v}_i$ can be influenced both by master and blend bones. The new weights of vertex $\mathbf{v}_i$ will be denoted as $w'_{i,1},\ldots,w'_{i,B},w'_{i,B+1},w'_{i,B+\beta}$. In order to avoid confusion, we introduce the term *u-weights* for blend bone weights (e.g., $u_{k,1},\ldots,u_{k,B}$) and *w′-weights* for vertex weights in the output model (e.g., $w'_{i,1},\ldots,w'_{i,B},w'_{i,B+1},w'_{i,B+\beta}$). The vertices of our output model will therefore deform as follows:

$$\mathbf{v'}_i^{(f)} = \sum_{j=1}^{B} w'_{i,j}C_j^{(f)}\mathbf{v}_i + \sum_{k=1}^{\beta} w'_{i,B+k}\Phi(\mathbf{u}_k;C_1^{(f)},\ldots,C_B^{(f)})\mathbf{v}_i \qquad (1)$$

which is a linear blending model, because Φ does not depend on w'-weights. Linear skinning with blend bones will therefore firstly precompute $\Phi(\mathbf{u}_k;C_1^{(f)},\ldots,C_B^{(f)})$ for $k = 1,\ldots,\beta$ and secondly evaluate Formula (1) for $i = 1,\ldots,N$. This results in faster execution as β is typically orders of magnitude smaller than N (note that our algorithm ensures that a maximum of 4 of the weights $w'_{i,1},\ldots,w'_{i,B+\beta}$ are non-zero). However, skinning according to Formula (1) introduces a certain error, which can be quantified as:

$$E(\mathbf{W'},\mathbf{U}) = \sum_{f=1}^{F}\sum_{i=1}^{N} \|\mathbf{v}_i^{(f)} - \mathbf{v'}_i^{(f)}\|^2 \qquad (2)$$

where $\mathbf{W'}$ is the set of vectors $\{\mathbf{w'}_1,\ldots,\mathbf{w'}_N\}$ and $\mathbf{U} = \{\mathbf{u}_1,\ldots,\mathbf{u}_\beta\}$. Our optimization procedure to find the values for $\mathbf{W'}$ and $\mathbf{U}$ that minimize $E(\mathbf{W'},\mathbf{U})$ is presented in Section 4.

Figure 3 shows an example featuring a very simple skinned model consisting of two master bones ($B = 2$) and only one animation frame ($F = 1$). Influences of master bones are visualized in gray, while influences of blend bones are colored. In this frame, transformation $C_1^{(1)}$ is the identity, while $C_2^{(1)}$ is an 180 degrees twist along the main axis. The original vertex weights of three chosen vertices $\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3$ are summarized in Table 2 (left). Linear blend skinning leads to skin collapsing artifacts, which can be removed by applying dual quaternion skinning. Let us approximate this nonlinear skinning using one blend bone (i.e., $\beta = 1$). Its optimal u-weights are $u_{1,1} = 0.5, u_{1,2} = 0.5$, i.e., interpolating half-way between $C_1^{(1)}$ and $C_2^{(1)}$ (note that this is also the case considered by Mohr and Gleicher [2003]). The optimal w'-weights are summarized in Table 2 (right), with the influences of the blend bone in the rightmost column ($j = 3$).

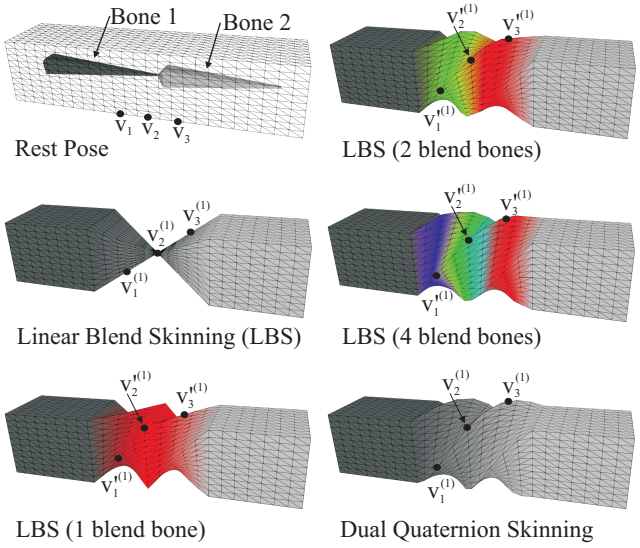

Figure 3: Simple skinned model in the rest pose (top left) and after twisting Bone 2 by 180 degrees (using several different methods).

$w_{i,j}$	$j=1$	$j=2$
$i=1$	0.75	0.25
$i=2$	0.5	0.5
$i=3$	0.25	0.75

$w'_{i,j}$	$j=1$	$j=2$	$j=3$
$i=1$	0.5	0	0.5
$i=2$	0	0	1
$i=3$	0	0.5	0.5

Table 2: Original bone influence weights (left) and resulting ones (after adding one blend bone, right).

Adding one blend bone improves the deformation quality to some extent. When adding further blend bones, our linear approximation converges to nonlinear skinning – see the results for $\beta = 2$ and $\beta = 4$ in Figure 3.

4 Blend Bones Optimization

In this section we discuss how to solve the optimization problem

$$min\ E(\mathbf{W'},\mathbf{U}) \qquad (3)$$

subject to

$$w'_{i,j} \geq 0,\ u_{k,l} \geq 0,\ \sum_{j=1}^{B+\beta} w'_{i,j} = 1,\ \sum_{l=1}^{B} u_{k,l} = 1$$

where the error term $E(\mathbf{W}', \mathbf{U})$ is defined as in Formula (2). This is a constrained least squares problem with $N \cdot F$ nonlinear equations (as the unknowns $\mathbf{u}_1, \ldots, \mathbf{u}_\beta$ are arguments of the nonlinear Φ). An additional constraint is that only 4 weights out of $w'_{i,1}, \ldots, w'_{i,B}, w'_{i,B+1}, w'_{i,B+\beta}$ can be non-zero. We assume that the original vertex weights $w_{i,1}, \ldots, w_{i,B}$ already satisfy this constraint (if not, it is possible to simply discard the smallest influences and re-normalize so that $w_{i,1} + \ldots + w_{i,B} = 1$).

Our optimization algorithm will add blend bones one by one, terminating either when $E(\mathbf{W}', \mathbf{U})$ drops below a given threshold or when we exhaust our blend bones budget. For a new blend bone, we first compute an initial estimate of its u-weights (Section 4.1) and secondly, iteratively optimize over all w'- and u-weights (Sections 4.2 and 4.3). We apply coordinate descent, i.e., we first fix the u-weights and compute the optimal w'-weights, then, we fix the w'-weights and optimize over the u-weights. Our optimization procedure is summarized in Algorithm 1, with details provided in the subsequent sections.

Algorithm 1

Input: Rest pose vertices $\mathbf{v}_1, \ldots, \mathbf{v}_N$
Vertex weights $w_{i,j}$ $(i = 1, \ldots, N,\ j = 1, \ldots, B)$
Skeletal animation $C_b^{(f)} (b = 1, \ldots, B,\ f = 1, \ldots, F)$
Maximum number of blend bones β_{max}, threshold τ

Output: Blend bone positions $u_{k,g}$ $(k = 1, \ldots, \beta,\ g = 1, \ldots, B)$
New vertex weights $w'_{i,j}$ $(i = 1, \ldots, N,\ j = 1, \ldots, B + \beta)$

```
β = 0; W' = W; U = ∅;
while β < β_max and E(W', U) > τ do
    β = β + 1;
    u_β = initialize new blend bone (Section 4.1)
    repeat
        W' = optimize w'-weights (Section 4.2)
        U = optimize u-weights (Section 4.3)
    until convergence
end while
```

4.1 Initialization of New Blend Bone

Initialization of the u-weights of a new blend bone consists of two sub-problems: first, it is necessary to decide which components out of $u_{k,1}, \ldots, u_{k,B}$ will be non-zero (i.e., which master bones will drive the new blend bone) and next, initial values of these non-zero components must be chosen.

Intuitively, it is not necessary to consider all possible combinations of master bones (for example, the forearm bone does not need to blend with the thigh bone). Therefore, we first create a list of potential candidates, i.e., list L of bone sets overlapping in the original model (a set of bones $\mathscr{B}$ is overlapping if there exists vertex v_i such that $w_{i,b} > 0$ for every $b \in \mathscr{B}$). Every element in this list consists of at most 4 bones. In a typical character model, this list will consist of bone sets such as {upper arm, clavicle, spine}, {upper arm, forearm}.

For every bone set $\mathscr{B} = \{b_1, \ldots, b_n\}$ from list L, we choose several samples of the vector $(u_{k,b_1}, \ldots, u_{k,b_n})$, for each of which, we execute w'-weights optimization over all vertices (Section 4.2). Subsequently, we compute the error term $E(\mathbf{W}', \mathbf{U})$, which tells us how much this u-weights sample decreases the overall error (obviously, this is a conservative estimate, as the error would decrease further by iterative optimization). We repeat this process for every $\mathscr{B} \in L$ and select the u-weights that lead to the maximal reduction of the error term. We experimented with several strategies for sampling the values of $(u_{k,b_1}, \ldots, u_{k,b_n})$, namely random, uniform and stratified, concluding that uniform sampling with just 3 samples per degree of freedom is sufficient.

This strategy results in placing blend bones in those parts of the model that exhibit the most severe volume collapsing artifacts. According to our experience, while the choice of initial values of $(u_{k,b_1}, \ldots, u_{k,b_n})$ is not critical, a suitable starting point can speed up convergence of the subsequent optimization.

4.2 Optimization of w'-weights

During the w'-weights optimization process we assume that the u-weights are fixed. Therefore, the minimization problem (3) becomes a linear least squares problem. The weights $w'_{i,1}, \ldots, w'_{i,B+\beta}$ can be determined independently for every vertex $\mathbf{v}_i$. Therefore, for every $i = 1, \ldots, N$, we have one optimization problem of the form:

$$min\ \|A_i \mathbf{w}'_i - \mathbf{b}_i\|^2 \tag{4}$$

subject to

$$w'_{i,j} \geq 0,\ \sum_{j=1}^{B+\beta} w'_{i,j} = 1$$

This optimization problem can be solved by off the shelf quadratic programming routines. However, a simpler method is to employ a Non-Negative Least Squares solver (NNLS) [Lawson and Hanson 1974], which is easier to implement than full quadratic programming (featuring arbitrary linear constraints). This approach, advocated by James and Twigg [2005], has the additional benefit of generating sparse solution vectors (i.e., where most of the components are exactly zero), which helps us to satisfy the additional constraint of at most 4 non-zero weights per vertex. A drawback is that the equality constraint $\sum_{j=1}^{B+\beta} w'_{i,j} = 1$ has to be treated as a soft constraint, i.e., by optimizing the extended system

$$\begin{pmatrix} cA_i \\ 1 \ldots 1 \end{pmatrix} \mathbf{w}'_i = \begin{pmatrix} c\mathbf{b}_i \\ 1 \end{pmatrix} \tag{5}$$

where $c = 1/\|b_i\|$ is a scaling coefficient.

In the case where NNLS returns more than 4 non-zero weights, we discard the smallest ones and re-optimize with respect to the 4 remaining bones only. Finally, we re-normalize the remaining weights so that their sum is exactly (and not only approximately) one.

The solver can be improved by considering only those of the $B + \beta$ master and blend bones that can be expected to influence $\mathbf{v}_i$. In particular, we take into account only the master bones that influenced vertex $\mathbf{v}_i$ in the original model, i.e., bones $b_1, \ldots, b_n \in \{1, \ldots, B\}$ such that $w_{i,b_j} > 0$. Regarding blend bones, we consider only those driven by at least one of the master bones $b_1, \ldots, b_n$ (i.e., $u_{k,b_j} > 0$ for at least one $j \in \{1, \ldots, n\}$).

4.3 Optimization of u-weights

The optimization of u-weights is a bit more involved, because the minimization problem (3) is nonlinear in $\mathbf{u}$ (and the w'-weights are assumed to be fixed). A common approach to tackle this kind of problem is to employ local linearization and iterate towards a local minimum [Pighin and Lewis 2007]. It is advantageous to have a reasonable starting point; in our case, we use the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{\beta-1}$ (computed in the previous iteration of the while loop in Algorithm 1) and $\mathbf{u}_\beta$ (given by the initialization step, see Section 4.1). Our goal is to compute $\tilde{\mathbf{U}} = \{\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_\beta\}$ so that $E(\mathbf{W}', \tilde{\mathbf{U}})$ is as small as possible.

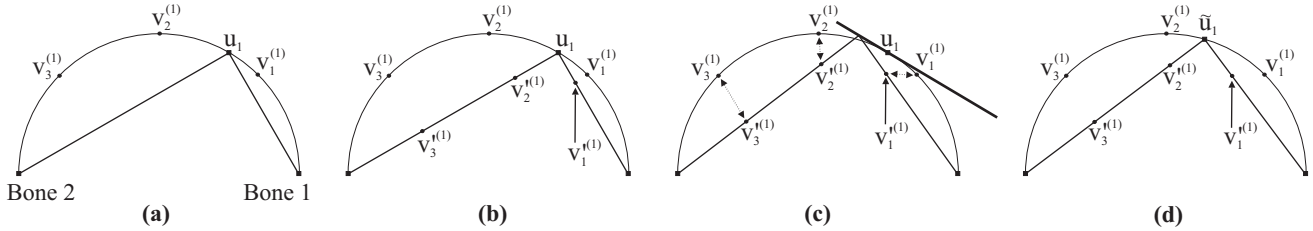

Figure 4: Illustration of one iteration of the w'-weights solver followed by an iteration of the u-weights solver.

Note that we will be optimizing over multiple u-weights simultaneously, thereby allowing the solver to reposition also previously placed blend bones (and not just the current one). However, to simplify the process, we will only be modifying the non-zero components of every $\mathbf{u}_k$, $k = 1,\ldots,\beta$ (henceforth we will drop index k in order to simplify notation). In particular, if the master bones have indices $\mathscr{B} = \{b_1,\ldots,b_n\}$ then $\tilde{u}_j = 0$ for all $j \notin \mathscr{B}$. This means that we do not allow our optimization routines to reassign the master bones, i.e., we trust the initial choice of the driving bones (see Section 4.1).

The components of $\tilde{\mathbf{u}}$ are subject to convexity constraints, i.e., $\tilde{u}_{b_j} \geq 0$ and $\tilde{u}_{b_1} + \ldots + \tilde{u}_{b_n} = 1$. We can use the latter constraint to express $\tilde{u}_{b_n}$ and substitute this into Φ. Thereby, we obtain an expression of Φ in terms of independent variables alone:

$$\Phi\left((\tilde{u}_{b_1},\ldots,\tilde{u}_{b_{n-1}},\tilde{u}_{b_n});C^{(f)}_{b_1},\ldots,C^{(f)}_{b_n}\right) = \Phi\left((\tilde{u}_{b_1},\ldots,\tilde{u}_{b_{n-1}},1-\tilde{u}_{b_1}-\ldots-\tilde{u}_{b_{n-1}});C^{(f)}_{b_1},\ldots,C^{(f)}_{b_n}\right)$$

Let us consider the first order Taylor expansion of this function at $\mathbf{u} = (u_{b_1},\ldots,u_{b_{n-1}})$. Assuming that $\tilde{\mathbf{u}}$ is in a small neighborhood of $\mathbf{u}$, we can write:

$$\Phi(\tilde{\mathbf{u}};\mathscr{C}^{(f)}) \approx \Phi(\mathbf{u};\mathscr{C}^{(f)}) + \sum_{j=1}^{n-1}(\tilde{u}_{b_j} - u_{b_j})\frac{\partial}{\partial u_{b_j}}\Phi(\mathbf{u};\mathscr{C}^{(f)}) \quad (6)$$

where $\mathscr{C}^{(f)}$ is a shorthand for the list of transformations $\left[C^{(f)}_{b_1},\ldots,C^{(f)}_{b_n}\right]$. Note that the partial derivatives of Φ corresponding to dual quaternion skinning (and its variants) can be expressed in a closed form (see Appendix A).

By substituting Formula (6) into Formula (2) for every blend bone, we obtain a linear least squares problem whose solution yields $\tilde{\mathbf{U}} = \{\tilde{\mathbf{u}}_1,\ldots,\tilde{\mathbf{u}}_\beta\}$ closer to a local minimum. To find the solution, we apply the same procedure as in Section 4.2. However, we actually keep the redundant variables $\tilde{u}_{b_n}$ among the unknowns, along with the constraints $\tilde{u}_{b_1} + \ldots + \tilde{u}_{b_n} = 1$. This ensures that $\tilde{u}_{b_j} \geq 0$ for all $j = 1,\ldots,n$. The linear system will be augmented by β equality constraints (one for each blend bone) and not just one as in the case of Formula (5).

The whole process would normally iterate until convergence. However, we found that it is sufficient to execute only one iteration and then hand over to the w'-weights solver (see the repeat-until loop in Algorithm 1). This intuitively makes sense, as there is no point in perfecting u-weights while fixing imperfect w'-weights.

A similar speed-up mechanism to that described in Section 4.2 is possible even in the case of u-weights optimization. In particular, not all of the previous blend bones $\mathbf{u}_1,\ldots,\mathbf{u}_{\beta-1}$ have to be updated, but only those sharing a common region with $\mathbf{u}_\beta$. Therefore, in the optimization, we will only consider such $\mathbf{u}_i$ for which j exists so that both $u_{i,j} > 0$ and $u_{\beta,j} > 0$, i.e., $\mathbf{u}_i$ shares at least one master bone with the most recently added blend bone $\mathbf{u}_\beta$.

Figure 4(a) presents an example with our simple model from Figure 3, with u-weights of the (only) blend bone $\mathbf{u}_1$ set to sub-optimal values $u_{1,1} = 0.66$ and $u_{1,2} = 0.34$. Our w'-weights solver finds such w'-weights so that the vertices $\mathbf{v}'^{(1)}_i$ (given by our linear approximation) are as close as possible to $\mathbf{v}^{(1)}_i$ (given by the nonlinear Φ), see Figure 4(b). Next, the u-weights solver establishes the local linear approximation of Φ at $\mathbf{u}_1$ and uses it to reduce the residuals further, see Figure 4(c). The new u-weights $\tilde{\mathbf{u}}_1$ are closer to the optimal solution $(0.5, 0.5)$, see Figure 4(d). After several iterations, both the w'- and u-weights converge to the values stated in Section 3.

4.4 Local Refitting

In Sections 4.2 and 4.3 we discussed how to speed up our solver by considering only a subset of all bones. Similarly, it is possible to also restrict the set of active vertices, based on the observation that every blend bone typically influences only a small part of the mesh. Intuitively, only the vertices in proximity to the most recently added blend bone need to be taken into account (e.g., when correcting artifacts in the shoulder joint it is not necessary to consider vertices in the legs). To formalize this principle, we define the set of potentially influenced vertices for every bone set $\mathscr{B}$:

$$V(\mathscr{B}) = \left\{\mathbf{v}_i : i \in \{1,\ldots,N\} \text{ and } \exists\, b \in \mathscr{B} \text{ such that } w_{i,b} > 0\right\}$$

i.e., all the vertices influenced by at least one of the master bones in $\mathscr{B}$. This results in a conservative but still reasonably small set of influenced vertices. Note that $V(\mathscr{B})$ can be precomputed for all active bone sets $\mathscr{B}$ in our input model. All steps of our solver (as described in Sections 4.1, 4.2 and 4.3) will then work only with the potentially influenced vertices, thereby avoiding optimization over unrelated regions.

5 Results

To compare fitting accuracy among different models, we employ the Enveloping Error (EE) metric introduced in [Wang et al. 2007]. This normalizes the total error $E(\mathbf{W}',\mathbf{U})$ from Formula (2) using the optimal articulated rigid body prediction:

$$EE = 100\sqrt{\frac{\sum_{f=1}^{F}\sum_{i=1}^{N}\|\mathbf{v}^{(f)}_i - \mathbf{v}'^{(f)}_i\|^2}{\sum_{f=1}^{F}\sum_{i=1}^{N}\|\mathbf{v}^{(f)}_i - C^{(f)}_{r_i}\mathbf{v}_i\|^2}}$$

where r_i is the master bone that most accurately predicts $\mathbf{v}^{(f)}_i$ over all frames (typically, r_i is the bone with the highest weight, i.e., $r_i = \text{argmax}_j\, w_{i,j}$). The enveloping error therefore measures only local deformations (and not the global rigid motion of individual body parts).

Input data					Pre-processing				Run-time performance						
Model	Vertices	Bones	Infl.	Frames	Blend bones	EE	Time	Infl.'	LBS	DQS	DQS'	SDQS	SDQS'	DIB	DIB'
Melissa	3036	66	1.32	12	10	3.34	10.5s	1.42	0.38ms	1ms	0.41ms	1.28ms	0.42ms	5.7ms	0.45ms
Soldier	8245	27	1.2	12	10	2.98	15.9s	1.33	0.97ms	2.46ms	1.03ms	3ms	1.03ms	14.9ms	1.08ms
Masha	7992	56	1.84	12	13	3.7	20.5s	1.89	1.23ms	3.14ms	1.3ms	4.1ms	1.3ms	18.8ms	1.36ms
Kevin	1938	65	1.95	12	10	3.31	6.1s	2.03	0.32ms	0.76ms	0.34ms	0.99ms	0.34ms	4.82ms	0.39ms
Eric	7927	75	1.56	12	15	4.66	40s	1.84	1.1ms	2.9ms	1.26ms	3.8ms	1.27ms	16.9ms	1.31ms
Box	5166	2	1.3	4	4	3.6	4.1s	1.3	0.64ms	1.85ms	0.65ms	2.39ms	0.65ms	9.34ms	0.66ms
Tube	1380	4	3.86	6	12	4.6	21.9s	3.34	0.37ms	0.64ms	0.35ms	0.93ms	0.35ms	3.77ms	0.37ms

Table 3: Experimental results for various 3D models. *Infl.* denotes the average number of influencing bones per vertex in the input model, *Infl.'* is the same in the output model (computed by Algorithm 1). *LBS* stands for Linear Blend Skinning, *DQS* is Dual Quaternion Skinning, *SDQS* is Scale supporting Dual Quaternion Skinning and *DIB* is Dual quaternion Iterative Blending. *DQS', SDQS'* and *DIB'* stand for our linear approximations of the said nonlinear methods.

We tested our blend bones optimization (Algorithm 1) on several human characters (see Figure 5) and two synthetic examples (the box from Section 3 and the tube shown in Figure 6). The tube model has vertices that are almost all influenced by all four master bones and was so designed to test the robustness of our method. Table 3 summarizes our results. The run-time performance is that of a CPU implementation running on a 2.4GHz Intel Core 2 (single thread). We only measured the time for actual skin deformation, and not the rendering time (which is the same for every skinning method). Note that the running time of our linear approximation is only a few percent slower than that of linear blending, where the overhead consists of evaluating the blend bone transformations and the usual slightly increased average number of vertex influences. In the case of the tube model, we obtained even slightly faster runtime performance than original linear blending (in this case our blend bones reduced the average number of vertex influences).

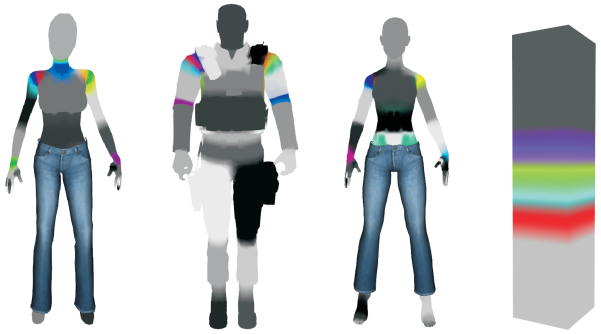

Figure 5: Some of our test models (Melissa, Soldier, Masha, Box). The jeans worn by the female models constitute a separate mesh (not considered by our algorithm).

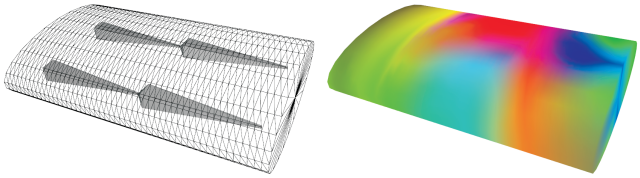

Figure 6: Model of a highly elastic tube controlled by four master bones (left). Note that blend bones control almost the entire mesh (right).

Figure 7 presents a comparison of our technique with that of Mohr and Gleicher [2003] (without scaling bones). Their method improves considerably upon linear blending, but it does not support blending between non-neighboring bones. Our blend bones do not have this restriction and can therefore achieve higher accuracy. Note that the system presented by Wang et al. [2007] also solves this problem, but using a more complex algorithm with a higher memory footprint.

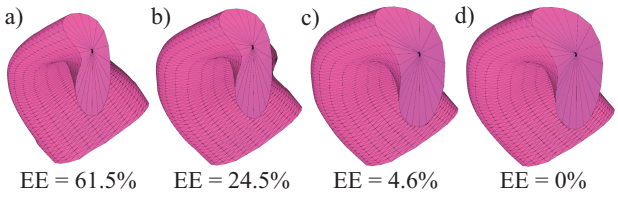

Figure 7: Simultaneous bending and twisting of an elastic tube, deformed by linear blending (a), Mohr and Gleicher [2003] (b), our technique (c) and dual quaternion blending (d). Note that the latter two are practically indistinguishable.

The training animation for all characters was identical and consisted of a motion captured exercise sequence, downsampled to $F = 12$ frames (designed to exercise critical regions such as shoulders). To test the generalizability of the linear approximation computed by Algorithm 1, we measured the enveloping error also on 1) all 385 frames of the input sequence and 2) an unseen mo-cap clip (jumping jacks, 61 frames). The results are reported in Table 4 (all characters use the same animation that is automatically retargeted to their skeletons). The measured enveloping errors confirm that overfitting is not an issue with our method.

Model	EE input	EE input full	EE unseen
Melissa	3.34	3.6	5.04
Soldier	2.98	3	4.78
Masha	3.7	3.95	8.42
Kevin	3.31	3.58	5.4
Eric	4.66	5.12	6.14

Table 4: Generalizability test of our linear approximation – enveloping error measured on three animations (original input, full-frames input and an unseen one).

To obtain an insight into how quickly the error decreases with increasing numbers of blend bones, we plot its graph for three of our models, see Figure 8. We can see that the tube model is indeed the most challenging one (because of its large overlapping influences), and the box is the simplest one (i.e., its error decreases most quickly). The character model (Melissa) is in between the two, featuring a large drop after adding the first two blend bones (corresponding to left and right shoulder regions).

Our linear approximation makes it feasible to consider more com-

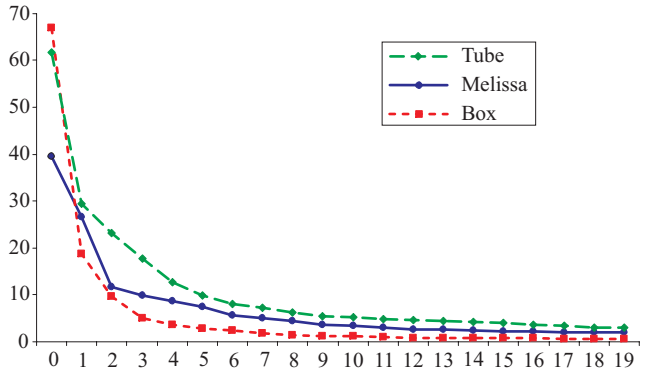

Figure 8: Enveloping error (vertical axis) vs. number of blend bones (horizontal axis).

plex nonlinear operators, such as Dual quaternion Iterative Blending (DIB) [Kavan et al. 2008], for use in real-time applications. The DIB operator produces correct intrinsic averages and is therefore more accurate than standard dual quaternion skinning (DQS) (computing only approximate averages). While the difference between DIB and DQS is normally quite small, for rotation angles approaching 360 degrees DQS produces serious artifacts, see Figure 9 (and note that automatic antipodality resolution has to be switched off in order to allow angles higher than 180 degrees). DIB guarantees a correct solution, but at a substantial run-time cost (with the box model, DIB is more than 5 times slower than DQS). Using our linear approximation, we can achieve comparable skin deformation quality in a time almost equivalent to that of linear blending, see Figure 9.

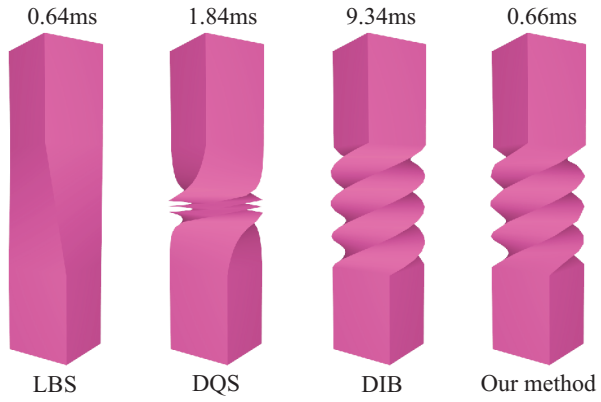

Figure 9: Bending box rotated by nearly 360 degrees. Our method provides an efficient linear approximation of the correct solution (DIB).

6 Conclusions and Future Work

This paper proposes a method to automatically generate piece-wise linear approximations of nonlinear skinning operators. Based on an input 3D model and its training animation, our technique automatically identifies the most critical regions of the mesh and places virtual bones (samples of the nonlinear skinning operator) accordingly. The resulting skinned model can be deformed by off the shelf linear blending routines, with results almost indistinguishable from nonlinear skinning. This approach also enables us to directly apply algorithms that assume linear vertex blending, such as corrective enveloping [Lewis et al. 2000; Sloan et al. 2001; Kry et al. 2002], collision detection [Kavan and Žára 2005] or accurate normal transformation [Merry et al. 2006b].

We believe our algorithm fits naturally into existing character production pipelines, such as those employed in the games industry. Artists design character rigs in their favourite digital content creation software, taking advantage of nonlinear transformation blending operators. Part of this process is testing the rig on a training animation, which can be advantageously used as an input to our algorithm. We envisage that our algorithm could be implemented as part of an export tool, converting the model to linear skinning before it is loaded by the 3D engine.

Our method also makes it possible to employ more sophisticated nonlinear operators, such as Dual quaternion Iterative Blending, which would otherwise be too slow for real-time applications. An interesting line of future work is to explore other nonlinear skin deformation operators, such as those based on user-defined curves [Forstmann et al. 2007; Gregory and Weston 2008]. Taking this one step further, it should be possible to explore a whole family of nonlinear skinning operators (generalizing both dual quaternions as well as curve-based deformations), exploiting the fact that complex operators can be approximated efficiently using our method.

7 Acknowledgements

We wish to thank Martin Pražák and Rachel McDonnell for assistance with motion capture and Daniel Sýkora, Simon Dobbyn, Cormac O'Brien and the anonymous reviewers for their valuable comments. This work has been supported by the Higher Education Authority of Ireland and Science Foundation Ireland.

A Derivatives of Dual Quaternion Blending

In this appendix we describe how to compute the partial derivatives required by Formula (6) in the case where operator Φ is dual quaternion blending (or its variants). Recall that the arguments of Φ are blending weights (with the last weight removed as discussed in Section 4.3) and the result is a 3×4 matrix. We will assume that the input transformations are rigid and were already converted to dual quaternions $\mathcal{Q} = [\hat{\mathbf{q}}_1, \ldots, \hat{\mathbf{q}}_n]$. Therefore, the Φ corresponding to dual quaternion blending can be expressed as follows:

$$\Phi_{DQS}(x_1,\ldots,x_{n-1};\mathcal{Q}) = M\left(\frac{\Psi(x_1,\ldots,x_{n-1};\mathcal{Q})}{\|\Psi(x_1,\ldots,x_{n-1};\mathcal{Q})\|}\right) \quad (7)$$

where M is the operator that converts a dual quaternion to a matrix [Kavan et al. 2008] and Ψ denotes linear blending of dual quaternions, i.e.,

$$\Psi(x_1,\ldots,x_{n-1};\mathcal{Q}) = \sum_{i=1}^{n-1} x_i\hat{\mathbf{q}}_i + \left(1-\sum_{i=1}^{n-1} x_i\right)\hat{\mathbf{q}}_n$$

Because of the properties of dual quaternion to matrix conversion, we can simplify Formula (7) to:

$$\Phi_{DQS}(x_1,\ldots,x_{n-1};\mathcal{Q}) = \frac{M(\Psi(x_1,\ldots,x_{n-1};\mathcal{Q}))}{\|\Psi_0(x_1,\ldots,x_{n-1};\mathcal{Q})\|^2}$$

where Ψ_0 denotes the non-dual part of Ψ. This form simplifies computation of the partial derivatives:

$$\frac{\partial\Phi_{DQS}}{\partial x_i} = \frac{\partial(\|\Psi_0\|^{-2})}{\partial x_i}M(\Psi) + \|\Psi_0\|^{-2}\frac{\partial M(\Psi)}{\partial x_i}$$

where we omitted function arguments for better readability. The first term can be written as:

$$\frac{\partial(\|\Psi_0\|^{-2})}{\partial x_i} = \frac{-2}{\|\Psi_0\|^4}\left(\Psi_0\cdot\frac{\partial\Psi_0}{\partial x_i}\right) = \frac{-2}{\|\Psi_0\|^4}(\Psi_0\cdot(\mathbf{q}_i-\mathbf{q}_n))$$

where $\cdot$ denotes the standard dot product in 4-dimensional Euclidean space. Using the chain rule, the second term expands to:

$$\frac{\partial M(\Psi)}{\partial x_i} = \sum_{j=1}^{8} \frac{\partial M}{\partial \Psi_{s_j}} \frac{\partial \Psi_{s_j}}{\partial x_i}$$

where Ψ_{s_j} denotes dual quaternion elements (i.e., $\Psi = \Psi_{s_1} + \Psi_{s_2} i + \Psi_{s_3} j + \Psi_{s_4} k + \Psi_{s_5} \varepsilon + \Psi_{s_6} \varepsilon i + \Psi_{s_7} \varepsilon j + \Psi_{s_8} \varepsilon k$). Note that $\partial \Psi / \partial x_i = \hat{\mathbf{q}}_i - \hat{\mathbf{q}}_n$ and the partial derivatives of operator M can also be computed easily.

In situations where non-rigid bone transformations are required, we need to employ two-phase skinning, i.e., linear blending followed by dual quaternion blending [Kavan et al. 2008]. The Φ corresponding to two-phase skinning has the form:

$$\Phi_{SDQS} = \Phi_{DQS} \Phi_{LBS}$$

and therefore its partial derivatives are given as:

$$\frac{\partial \Phi_{SDQS}}{\partial x_i} = \frac{\partial \Phi_{DQS}}{\partial x_i} \Phi_{LBS} + \Phi_{DQS} \frac{\partial \Phi_{LBS}}{\partial x_i}$$

Dual quaternion Iterative Blending [Kavan et al. 2008] has no closed-form formula and therefore its partial derivatives would be hard to compute explicitly. It is possible to either employ approximation using finite difference, or (as we did) to simply use the training with Φ_{DLB}, relying on the extrapolation capability of blend bones and the fact that, for rotations below 180 degrees, $\Phi_{DLB} \approx \Phi_{DIB}$.

References

ANGUELOV, D., SRINIVASAN, P., KOLLER, D., THRUN, S., RODGERS, J., AND DAVIS, J. 2005. SCAPE: shape completion and animation of people. *ACM Trans. Graph. 24*, 3, 408–416.

DE AGUIAR, E., THEOBALT, C., THRUN, S., AND SEIDEL, H.-P. 2008. Automatic conversion of mesh animations into skeleton-based animations. *Computer Graphics Forum (Proc. Eurographics EG'08) 27*, 2 (4), 389–397.

FORSTMANN, S., OHYA, J., KROHN-GRIMBERGHE, A., AND MCDOUGALL, R. 2007. Deformation styles for spline-based skeletal animation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, 141–150.

GREGORY, A., AND WESTON, D. 2008. Offset curve deformation from skeletal animation. In *SIGGRAPH '08: ACM SIGGRAPH 2008 talks*, ACM, New York, NY, USA.

HEJL, J., 2004. Hardware skinning with quaternions. *Game Programming Gems 4*, Charles River Media, 487–495.

JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Trans. Graph. 24*, 3, 399–407.

KAVAN, L., AND ŽÁRA, J. 2005. Fast collision detection for skeletally deformable models. *Computer Graphics Forum 24*, 3, 363–372.

KAVAN, L., MCDONNELL, R., DOBBYN, S., ŽÁRA, J., AND O'SULLIVAN, C. 2007. Skinning arbitrary deformations. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM Press, 53–60.

KAVAN, L., COLLINS, S., ŽÁRA, J., AND O'SULLIVAN, C. 2008. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph. 27*, 4, 105.

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

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

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 165–172.

MAGNENAT-THALMANN, N., LAPERRIÈRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, Canadian Information Processing Society, 26–33.

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

MERRY, B., MARAIS, P., AND GAIN, J. 2006. Normal transformations for articulated models. In *SIGGRAPH '06: ACM SIGGRAPH 2006 sketches*, ACM, New York, NY, USA, 134.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3, 562–568.

PIGHIN, F., AND LEWIS, J. P. 2007. Practical least-squares for computer graphics. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, ACM, New York, NY, USA, 1–57.

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

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM Press, 245–254.

SLOAN, P.-P. J., ROSE, III, C. F., AND COHEN, M. F. 2001. Shape by example. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, 135–143.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3, 399–405.

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

WANG, R. Y., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. Graph. 26*, 3, 73.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3.

WEBER, J. 2000. Run-time skin deformation. In *Proceedings of Game Developers Conference*.

YANG, X., SOMASEKHARAN, A., AND ZHANG, J. J. 2006. Curve skeleton skinning for human and creature characters: Research articles. *Comput. Animat. Virtual Worlds 17*, 3-4, 281–292.