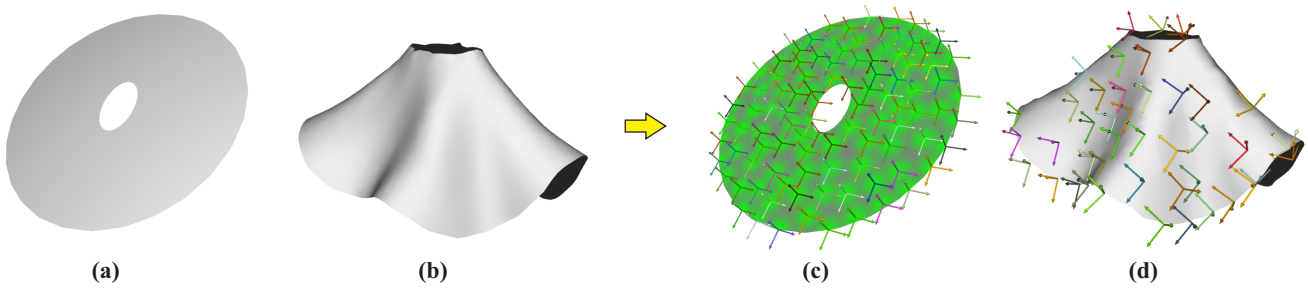
Skinning Arbitrary Deformations

Ladislav Kavan* [1,2] Rachel McDonnell[1] Simon Dobbyn[1] Jiří Žára[2] Carol O'Sullivan[1]

[1]Trinity College Dublin, [2]Czech Technical University in Prague



(a) (b) (c) (d)

Figure 1: Overview of Skinning Arbitrary Deformations: as input we have the rest-pose model (a), and a deformed one (b). Our algorithm first determines the proxy-joints and their influences (c) and then computes the joint transformations, whose application in matrix palette skinning (d) gives a good approximation of the input deformation. Even though (b) and (d) appear to be almost identical, (d) needs about 17 times less memory than (b) and can be rendered efficiently using the popular skinning algorithms.

Abstract

Matrix palette skinning (also known as skeletal subspace deformation) is a very popular real-time animation technique. So far, it has only been applied to the class of quasi-articulated objects, such as moving human or animal figures. In this paper, we demonstrate how to automatically construct skinning approximations of arbitrary pre-computed animations, such as those of cloth or elastic materials. In contrast to previous approaches, our method is particularly well suited to input animations without rigid components. Our transformation fitting algorithm finds optimal skinning transformations (in a least-squares sense) and therefore achieves considerably higher accuracy for non-quasi-articulated objects than previous methods. This allows the advantages of skinned animations (e.g., efficient rendering, rest-pose editing and fast collision detection) to be exploited for arbitrary deformations.

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

Keywords: skinning, approximation, animation compression, dual quaternions

1 Introduction

The main problem with processing and re-using pre-computed animations is the amount of data to be handled. Current graphics hardware is capable of rendering millions of triangles in real-time. Therefore, pre-computed animations that are stored keyframe by keyframe are likely to consume a lot of memory, so data reduction becomes imperative. A number of animation compression algorithms have been discussed in the literature, most of them naturally focusing on maximal data reduction. However, as pointed out by James and Twigg [2005], low bitrate is not the only criterion. In practical applications, other factors become important as well: e.g., efficient rendering and additional operations possible directly with the representation (as playback is not always sufficient – sometimes we need to perform other tasks as well, such as collision detection).

An established way of representing the animation of quasi-articulated objects, such as virtual characters, is known as skinning (or matrix palette skinning). It is based on the observation that an animation of a virtual character can be compactly described by its skeleton. Typically, a skeleton has far fewer degrees of freedom than a character's skin, which therefore makes character animation much easier. Nonetheless, this requires the skeleton to be designed, and its relationship (binding) to the skin to be established.

James and Twigg's Skinned Mesh Animations (SMAs) [2005] construct a skinning approximation automatically from the input animation. Their algorithm works by identifying quasi-rigid components (such as individual body parts of a virtual character) and computing their transformations. Note that the resulting transformations do not have any hierarchical structure as in a skeleton. SMAs produce excellent results for quasi-articulated animations: significant data reduction coupled with a hardware-accelerated rendering of unprecedented efficiency.

Unfortunately, SMAs do not perform as well for highly deformable animations, such as those of cloth or elastic materials. This is understandable, because in such animations, no quasi-rigid components can be found. Does it mean that no efficient skinning approximation exists in this case? In fact, experiences with animating dressed virtual humans indicate the contrary: some videogames, as well as the system presented by Oh et al. [2005], successfully use matrix palette skinning to animate cloth.

This paper presents an algorithm to find skinning approximations for arbitrary animations. Because we do not make any assumptions about the input animation, we distribute the control transformations (proxy-joints) uniformly over the animated 3D model. This has proven to be sufficient for our testing animations, though it might not be satisfactory for animations with highly non-uniformly distributed deformations. Subsequently, for each frame of the animation, our algorithm finds the set of transformations whose application in matrix palette skinning approximates the current shape of the model as closely as possible (see Figure 1). This way, we ensure that the overall deformation is approximated reasonably, without relying on quasi-rigid components. As a result, we obtain good

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

visual fidelity even for highly deformable animations.

Of course, this does not come for free: our pre-computation times are slower than those of SMAs and our algorithm does not find the smallest reasonable number of proxy-joints automatically (as a result, we typically use more proxy-joints than necessary). On the other hand, according to our experiments, our algorithm constructs much more accurate skinning approximations of highly deformable 3D models than SMAs. This enables us to exploit the advantages of skinned animations, such as efficient hardware-accelerated rendering, rest-pose editing and fast collision detection, even outside the realm of quasi-articulated models.

2 Background and Related Work

A lot of literature is devoted to matrix palette skinning and its variations. The most common matrix palette skinning algorithm is linear blend skinning [Lindholm et al. 2001], even though it suffers from artifacts, such as the candy-wrapper problem. Wang and Phillips [2002] addressed this problem by assigning different weights to all coefficients of the matrix, while Mohr and Gleicher [2003] added auxiliary joints. Recently, we suggested implementing skinning with blending of dual quaternions instead of matrices [Kavan et al. 2007].

However, all these methods assume that the model to be deformed is already equipped with an animated skeleton, i.e., that the transformations determining the shape of the deformed model are already given. In contrast, the technique proposed in this paper automatically computes the skinning transformations. Another method that extends matrix palette skinning is EigenSkin [Kry et al. 2002]. This technique is aimed at correcting small displacements for already computed skinning transformations. In effect, EigenSkin is complementary to our approach, and both methods can be advantageously used in conjunction (see Section 5).

Our research suggests that matrix palette skinning could be compared to other general deformation methods, such as the popular Free Form Deformation [Sederberg and Parry 1986]. In our approach, we also embed a control structure into the rest-pose (undeformed) model. However, our control structure is just a set of points, instead of a lattice as in FFD. In our method, the deformation is given by rigid transformations in the control points, rather than by displacement of the lattice vertices, as in FFD.

Various animation compression algorithms have been described since the pioneering work of Lengyel [1999]. Alexa proposes applying Principal Components Analysis (PCA) [Alexa and Müller 2000]. PCA can also be advantageously augmented by another common compression technique, Linear Prediction Coding, as shown in [Karni and Gotsman 2004]. Decorrelation based on wavelets also has been successfully applied to animation data [Guskov and Khodakovsky 2004]. Sattler et al. [2005] show the advantages of clustered PCA and present fast GPU-based decompression. Another interesting method is to encode the mesh as *geometry images* [Gu et al. 2002] and apply established 2D animation compression methods [Briceno et al. 2003]. Alternatively, deformable geometry can be stored using progressive multiresolution meshes [Kircher and Garland 2005], which represent the animation at multiple levels of detail. This allows the correct level of detail to be selected at runtime, either in a static or view-dependent way.

While our algorithm offers significant data reduction, animation compression is not our only goal. Another major goal is to achieve fast, hardware accelerated rendering using the simple and popular matrix palette skinning method. This is advantageous, because matrix palette skinning is already implemented in most real-time animation systems and enables further operations to be performed directly. For example, skinned approximations can be exploited to speed up collision detection and facilitate rest-pose editing (see Section 6). Also, it should be noted that skinning is not a linear representation (even though the term "linear blend skinning" might suggest the opposite). In fact, skinning can easily outperform even the best linear representation (PCA). This is because skinning allows objects to rotate and/or bend. For example, PCA does not work very well on a rotating rigid object (unless the rotation is small and the trajectories can be approximated well by straight lines). However, rotating a rigid object presents no problem for skinning, which needs just one proxy-joint to represent such an animation exactly.

Skinned Mesh Animations (SMAs) [James and Twigg 2005] is not the only technique for approximating animations by skinning. Collins and Hilton [2005] describe a method based on a rigid transformation basis. Their algorithm delivers efficient data reduction, but the reconstructed animation suffers from discontinuities between individual clusters. The authors suggest correcting this by adding another post-processing step to smooth out the geometry. Mamou et al. [2006] present a variation of SMAs with emphasis on animation compression. However, all of the above-mentioned papers focus only on quasi-articulated animations.

3 Preliminaries

As input we have a sequence of polygonal meshes with constant connectivity. Furthermore, we assume that a rest-pose mesh is given. This can be as simple as the first mesh of the animation. If available, we can also use the rest-pose of the mesh used to produce the animation, e.g., an unfolded piece of cloth. In order to support more accurate shading, vertex normals are usually stored for each keyframe also. If the model is already rigged, i.e., equipped with joints and their influences, we can skip the following steps and proceed directly to transformation fitting (Section 4). However, in general, we do not assume that those structures are present (as is the case for our experimental data). Therefore, an algorithm for their automatic generation is described below.

Proxy joints. The proxy-joints are distributed over the rest-pose mesh so that each proxy-joint influences approximately the same amount of geometry. Let us assume that we want to use p proxy-joints (samples). We need to position the proxy joints in space so that the maximal distance of a rest-pose vertex to the nearest proxy-joint is minimized. This ensures that each proxy-joint controls approximately the same amount of geometry, even if the vertices of the rest-pose mesh are non-uniformly distributed. This problem is known in computational geometry as the p-center problem [Agarwal and Sharir 1998]. The simple greedy algorithm has been shown to produce good results [Gonzales 1985]. This algorithm places the centers (in our case, the proxy-joints) so that they coincide with the mesh vertices. At each step of the algorithm, a new proxy-joint is created at the vertex that has maximal distance from all proxy-joints created so far. The resulting sampling is illustrated in Figure 2. Note that in our method, every proxy-joint is independent, and there is no hierarchical structure as in a skeleton.

Vertex weights. The joint influences (vertex weights) are also easily computed. Let r be the maximal distance from a rest-pose mesh vertex to the nearest joint (i.e., the value that the p-center solution minimizes). The influence of each joint is limited to the ball centered in the joint with radius $P \cdot r$, where $P \geq 1$ is a user-defined parameter controlling the area of the joint's influence. We found the value $P = 1.5$ to perform well in practice. The weight decays linearly from 1 at the joint's center to 0 on the ball's boundary. If more than one joint influences the vertex, as is usually the case, we normalize the weights of all joints so that they sum to 1. Since

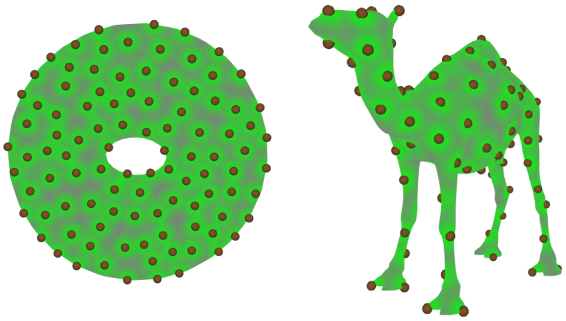


Figure 2: One hundred proxy-joints generated by the greedy algorithm for the rest-pose skirt and camel model. Joint influences are also depicted.

weights are obviously non-negative, this guarantees convex vertex weights. We also experimented with a more sophisticated weight assignment, such as one based on least-squares optimization [Mohr and Gleicher 2003], but we found the improvement to be almost negligible.

Linear blend skinning. Let us assume that the object's proxy-joints and vertex weights are already given. The model can then be animated by matrix palette skinning, which requires specifying the transformations for all proxy-joints. The simplest and most widely used matrix palette skinning algorithm is linear blend skinning, which works as follows: let m be the number of vertices in our mesh, and p the total number of proxy-joints, whose transformation matrices we denote as $C_1,\ldots,C_p$. Now, if a rest-pose vertex $\mathbf{v}_k$, $k \in \{1,\ldots,m\}$ is influenced by n_k joints with indices $j_{k,1},\ldots,j_{k,n_k} \in \{1,\ldots,p\}$ and convex weights $w_{k,1},\ldots,w_{k,n_k} \in \langle 0,1\rangle$, then the position of the vertex in the deformed mesh is computed as

$$\mathbf{v}_k^{def} = \left(\sum_{i=1}^{n_k} w_{k,i} C_{j_{k,i}}\right) \mathbf{v}_k \qquad (1)$$

Dual quaternion skinning. In advanced skinning methods, the linear blending of matrices (i.e., the term $\sum_{i=1}^{n_k} w_{k,i} C_{j_{k,i}}$) is replaced by a more sophisticated method. For example, when transformations $C_1,\ldots,C_p$ are rigid, we can apply dual quaternion linear blending [Kavan et al. 2007]. We will now provide a brief overview of dual quaternions. Please refer to [McCarthy 1990] or [Kavan et al. 2007] for more details.

Dual quaternions are similar to Hamilton's classical quaternions, but besides the classical quaternion units $1,i,j,k$, there is an additional *dual* unit, usually denoted as ε. The dual unit ε satisfies $\varepsilon^2 = 0$ and commutes with i,j,k (e.g., $i\varepsilon = \varepsilon i$), which defines dual quaternion multiplication. In the text, we distinguish dual quantities from non-dual ones by a caret. The overline denotes dual conjugation, i.e., replacement of ε by $-\varepsilon$, for example $\overline{1+2\varepsilon-\varepsilon i+3k} = 1-2\varepsilon+\varepsilon i+3k$. Dual quaternions are important for computer graphics, because they represent general rigid transformations, not just rotations as with regular quaternions.

Conversion from a regular quaternion $\mathbf{r}$ and translation vector (t_x,t_y,t_z) to a dual quaternion is simple: the corresponding dual quaternion is $(1+(t_x i+t_y j+t_z k)\varepsilon/2)\mathbf{r}$. The opposite conversion, from a dual quaternion $\hat{\mathbf{q}}$ to a regular quaternion and translation, is equally easy: let us assume that $\hat{\mathbf{q}}$ has a non-dual part $\mathbf{q}_0$ and a dual part $\mathbf{q}_\varepsilon$. Then the rotation is just $\mathbf{q}_0$ and the translation vector is given by $2\mathbf{q}_\varepsilon\mathbf{q}_0^*$, where the star symbol denotes classical quaternion conjugation.

Dual quaternion skinning then works as follows. First, the rigid joint transformations $C_1,\ldots,C_p$ are converted to dual quaternions $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_p$. Every vertex $\mathbf{v}_k$, with components $\mathbf{v}_k = (v_{k,x},v_{k,y},v_{k,z})$, is represented by a dual quaternion $\hat{\mathbf{v}}_k = 1+\varepsilon(v_{k,x}i+v_{k,y}j+v_{k,z}k)$. Using this convention, we compute

$$\hat{\mathbf{v}}_k^{def} = \left(\sum_{i=1}^{n_k} w_{k,i}\hat{\mathbf{q}}_{j_{k,i}}\right)\hat{\mathbf{v}}_k\left(\overline{\sum_{i=1}^{n_k} w_{k,i}\hat{\mathbf{q}}_{j_{k,i}}}\right)^{-1} \qquad (2)$$

which has also the form $\hat{\mathbf{v}}_k^{def} = 1+\varepsilon(v_{k,x}^{def}i+v_{k,y}^{def}j+v_{k,z}^{def}k)$, from which the coordinates of the deformed vertex $\mathbf{v}_k^{def} = (v_{k,x}^{def},v_{k,y}^{def},v_{k,x}^{def})$ can be immediately extracted. Note that Formula (2) is correct even if $\sum_{i=1}^{n_k} w_{k,i}\hat{\mathbf{q}}_{j_{k,i}}$ does not have unit length (it just needs to be invertible). Dual quaternion skinning is a little bit slower than linear blend skinning, but in turn has better properties – there are no artifacts such as the candy-wrapper [Kavan et al. 2007].

From a practical point of view, dual quaternion skinning is just a more sophisticated version of matrix palette skinning. Both linear as well as dual quaternion blending can be easily implemented in a vertex shader. Irrespective of the skinning method, the animation is controlled only by the transformations $C_1,\ldots,C_p$. Their automatic computation is the main contribution of this paper and is described in the next section. We will see that usage of the dual quaternion representation of $C_1,\ldots,C_p$ offers some advantages for our purposes. The number of joints p is typically a small number when compared to the number of vertices of the input mesh. This is the reason for the efficiency of matrix palette skinning: each keyframe needs to store or send only p transformations to the graphics card. This is a significant improvement over processing m vertices, because m can be greater than p by several orders of magnitude.

4 Joint Transformation Fitting

Let us assume that the (proxy-)joints are already given, either designed by animators or generated automatically according to Section 3. The problem now is to find the joint transformations for each keyframe of our input animation. This is done independently for every keyframe; in the following, we therefore describe transformation fitting for one fixed frame. We cannot simply apply the transformation fitting method from [James and Twigg 2005], because it derives the joint transformations from previously identified quasi-rigid components (while we do not assume the existence of any quasi-rigid components).

We denote the vertex positions in the current frame of the input animation as $\mathbf{v}'_k$, $k \in \{1,\ldots,m\}$ (the number of vertices m does not change between frames). The task now is to find the transformations $C_1,\ldots,C_p$, so that the skinning according to Formula (1) produces vertices $\mathbf{v}_k^{def}$ as close as possible to $\mathbf{v}'_k$. All other quantities, i.e., the number of influencing joints n_k, the influencing joint indices $j_{k,1},\ldots,j_{k,n_k}$ and the weights $w_{k,1},\ldots,w_{k,n_k}$, are known and fixed. The question is: how general should the class of transformations that we consider be? The simplest option is to consider general affine transformations, because in this case, we can optimize each element in every matrix C_i independently. Let us study this method first.

4.1 Affine Transformation Fitting

The problem can be stated as minimization of

$$\sum_{k=1}^{m} \left\| \mathbf{v}'_k - \mathbf{v}_k^{def} \right\|^2$$

over the joint transformation matrices $C_1,\ldots,C_p$. This is equivalent to the least-squares solution of the linear system

$$\mathbf{v}'_k = \sum_{i=1}^{n_k} w_{k,i} C_{j_{k,i}} \mathbf{v}_k,\ k \in \{1,\ldots,m\} \tag{3}$$

with $3m$ equations and $12p$ unknowns (the elements of 3×4 matrices $C_1,\ldots,C_p$). The system from Formula (3) can be rewritten as

$$A\mathbf{x} = \mathbf{b}$$

where $\mathbf{x}$ is a $12p$-dimensional unknown vector, A is a $3m\times 12p$ known matrix, and $\mathbf{b}$ is the $3m$-dimensional right-hand side. The matrix A is constructed from vertex weights, rest-pose vertex positions and influencing joints. The vector $\mathbf{b}$ is formed by stacking vertices $\mathbf{v}'_1,\ldots,\mathbf{v}'_m$. Note that since each vertex is influenced only by a small number of joints (typically no more than 4), the matrix A will be quite sparse. We obtain the least-squares solution $\mathbf{x}$ with the LSQR algorithm which exploits the sparsity of the matrix A [Paige and Saunders 1982]. The transformation matrices $C_1,\ldots,C_p$ are then extracted from vector $\mathbf{x}$.

Fitting of affine transformations works very well for vertex positions. However, because of realistic shading, we also need to handle vertex normals. For accurate normals, it is not sufficient to simply transform them by the 3×3 submatrices of $C_1,\ldots,C_p$, as observed already in [Mohr and Gleicher 2003]. Instead, it is necessary to incorporate the normals into our fitting process, which can be done in two ways. Either, we can combine the vertex and normal equations together, and find transformations that would be useful for skinning both vertex positions and normals. Alternatively, we can solve two independent systems, one for vertex positions, one for normals, thus computing two sets of transformations. Even though the latter method needs more memory, we found it more advantageous in practice – it also avoids problems with different magnitudes of vertex positions and normals. Henceforth, we will therefore consider a separate set of transformations for normals, $C'_1,\ldots,C'_p$.

Vertex normals are transformed in linear blend skinning according to an equation similar to Formula (1):

$$\mathbf{n}_k^{def} = \left(\sum_{i=1}^{n_k} w_{k,i} C'_{j_{k,i}}\right)^{-T} \mathbf{n}_k \tag{4}$$

where $C'_1,\ldots,C'_p$ are the normal transformation matrices. Besides the inverse transposition, the problem is that this equation does not produce unit normals $\mathbf{n}_k^{def}$, even if the input normals have unit length [Mohr and Gleicher 2003]. This means that we would actually have to minimize

$$\sum_{k=1}^{m} \left\| \mathbf{n}'_k - \frac{\mathbf{n}_k^{def}}{\|\mathbf{n}_k^{def}\|} \right\|^2 \tag{5}$$

which leads to a system of non-linear equations. A non-linear optimization would make our pre-processing times impractical – not to mention the associated numerical issues. Luckily, by constraining our transformations to rigid ones, we are able to obtain a linear least squares problem.

4.2 Rigid Transformation Fitting

Rigid transformations are a subset of affine transformations, because they consist of rotation and translation only (i.e., no scale or shear). If we restrict our transformations to rigid ones, and employ a blending method which preserves rigidity, we obtain useful simplifications. Namely, the inverse transposition from Formula (4) as well as the normalization in Formula (5) disappear, thus linearizing the problem. Moreover, rigid transformations have only 6 degrees of freedom, which means that we need only half the memory required for affine transformations.

On the other hand, fitting rigid transformations is more complex than fitting affine transformations because, in the former case, we must constrain the transformations to be rigid. The straightforward way to do this would be to require that the 3×3 submatrix of each C_i is orthogonal. Unfortunately, the orthogonality condition is quadratic, and therefore we would again obtain a non-linear optimization problem. In addition to this, linear blend skinning cannot be applied, because we need a blending method that preserves the rigidity of the input transformations (otherwise the transformations of normals would not preserve their unit length).

Fortunately, all of the above-mentioned problems can be elegantly solved by switching to dual quaternion skinning. Dual quaternions are automatically restricted to rigid transformations, and their blending also naturally preserves rigidity, so the lengths of normal vectors naturally stay unit. Dual quaternion skinning is given by Formula (2) in Section 3. One technical difficulty with this equation is that it is only valid if the dual quaternion $\left(\overline{\sum_{i=1}^{n_k} w_{k,i}\hat{\mathbf{q}}_{j_{k,i}}}\right)$ is invertible. We must thus ensure that our fitting method will produce invertible dual quaternions, i.e., those with a non-zero non-dual part – at least one coefficient of 1, i, j or k must be non-zero. We can enforce this easily by setting the first (real) component of each dual quaternion $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_p$ to one. This does not restrict our set of rigid transformations, because any dual quaternion $\hat{\mathbf{p}}$ represents the same rigid transformation as its real multiple $\alpha\hat{\mathbf{p}}$, for any real number α [McCarthy 1990].

Equation (2) can be further simplified: if we multiply both sides by $\left(\overline{\sum_{i=1}^{n_k} w_{k,i}\hat{\mathbf{q}}_{j_{k,i}}}\right)$ from the right and replace $\hat{\mathbf{v}}_k^{def}$ by the desired vertex position $\hat{\mathbf{v}}'_k$, we obtain:

$$\hat{\mathbf{v}}'_k \left(\overline{\sum_{i=1}^{n_k} w_{k,i}\hat{\mathbf{q}}_{j_{k,i}}}\right) - \left(\sum_{i=1}^{n_k} w_{k,i}\hat{\mathbf{q}}_{j_{k,i}}\right)\hat{\mathbf{v}}_k = 0 \tag{6}$$

for $k=1,\ldots,m$. This is a linear system, because multiplication of a dual quaternion by a constant dual quaternion is a linear transformation. Therefore, the above system of equations can be written as

$$A'\mathbf{x}' = \mathbf{b}' \tag{7}$$

where $\mathbf{x}'$ is a $7p$-dimensional unknown vector, A' is a $3m\times 7p$ known matrix, and $\mathbf{b}'$ is the $3m$ dimensional right-hand side (which is non-zero due to the substitution of 1 for the real component of dual quaternions $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_p$). Construction of the matrix A' is just a technical matter of rewriting Equation (6). The vector $\mathbf{b}'$ is formed from the rest-pose vertices $\mathbf{v}_k$ and the target ones, $\mathbf{v}'_k$. The matrix A' is sparse and thus Formula (7) can be efficiently solved in the least-squares sense using LSQR [Paige and Saunders 1982]. This is even faster than affine transformation fitting, because here we have only $7p$ unknowns instead of $12p$. The dual quaternions $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_p$ leading to an optimal fit are then constructed easily: their first component is 1, and the last 7 components are extracted from the vector $\mathbf{x}'$.

The fitting of normals can be done in essentially the same way. The only difference is that, in the case of normals, we ignore the translation component, i.e., set the dual parts of all dual quaternions $\hat{\mathbf{q}}_1,\ldots,\hat{\mathbf{q}}_p$ to zero. This leads to a linear system with only $3p$ unknowns, which we obtain from Equation (7) by simply substituting zeros for dual components.

4.3 Discussion

Is it more advantageous to use affine or rigid transformation fitting? Intuitively, it would seem that for highly deformable animations, affine transformations should be more approprite: in particular, they cannot give a worse fit, because rigid transformations are a subset of affine ones. However, our experiments (see Figure 5) show that the fitting accuracy of both rigid and affine transformations is actually quite similar. We therefore argue in favor of using rigid transformations, because they need only half as much memory as affine ones, simplify the treatment of normals and also offer faster pre-processing. Theoretically, it would be possible to consider a hybrid system that uses affine transformations for vertex positions and quaternions for vertex normals. However, we find it much more convenient to treat both vertex positions and normals in a unified way. Our final implementation therefore uses dual quaternion-based rigid transformation fitting. Note that James and Twigg [2005] arrived at the same conclusions, stating that rigid transformations are more favorable for highly deformable animations than affine ones.

5 Adding Fine Details

In some cases, the transformation fitting process described in Section 4 (either affine or rigid) has the side effect of smoothing out deformations. This is understandable, because matrix palette skinning simply has insufficient degrees of freedom to reproduce all the fine details of the deformation field. We can increase the accuracy of fitting by adding more proxy-joints, i.e., choosing a higher p. However, some subtle effects, such as delicate wrinkles on the cloth, would require a very high p, thus defeating the purpose of our approach. In order to support such effects, we propose an alternative method, based on skinning corrections similar to EigenSkin [Kry et al. 2002]. This method is most suitable for fine, low-amplitude deformations, and thus presents a perfect complement to our joint transformation fitting, which is most advantageous for low-resolution global shape approximation.

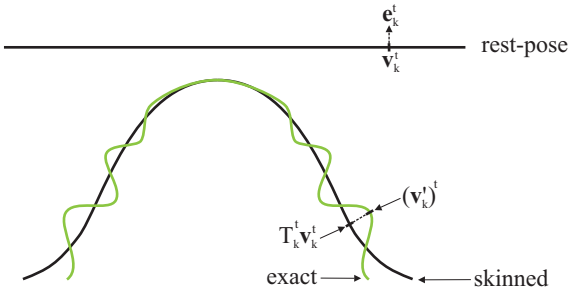


Figure 3: Differences between the exact mesh position, $(\mathbf{v}_k')^t$, and its skinning approximation, $T_k^t\mathbf{v}_k^t$, are mapped to the rest-pose, and denoted as $\mathbf{e}_k^t$.

In the rest of this section, we will need to work with the animation as a whole, as opposed to the per-frame approach used in Section 4. We denote the quantities in keyframe (time) t by a superscript ^{t}, specifically, $\mathbf{v}_k^t, (\mathbf{v}_k')^t$ and C_i^t or $\hat{\mathbf{q}}_i^t$. Let us assume that we have already computed the joint transformations for each frame according to Section 4 (irrespective of whether rigid or affine fitting was used). We denote the final transformation of vertex $\mathbf{v}_k^t$ as T_k^t, e.g., in linear blend skinning, $T_k^t = \sum_{i=1}^{n_k} w_{k,i} C_{j_{k,i}}^t$. The trick of EigenSkin is to transform the approximation error, i.e., the vector $(\mathbf{v}_k')^t - T_k^t\mathbf{v}_k^t$, to the rest-pose by multiplying it by $(T_k^t)^{-1}$ from the left. We denote the rest-pose error as $\mathbf{e}_k^t$,

$$\mathbf{e}_k^t = (T_k^t)^{-1}(\mathbf{v}_k')^t - \mathbf{v}_k^t$$

The vector $\mathbf{e}_k^t$ is the displacement which, if added to $\mathbf{v}_k^t$ before transformation, corrects the skin to match the input exactly (see Figure 3).

We stack all vectors $\mathbf{e}_k^t$ in a $3m \times n$ matrix E, where n is the total number of keyframes. Then we apply Singular Value Decomposition to decompose the matrix E to $E = DK$, where D is a $3m \times n$ matrix whose columns are so-called eigen-displacement vectors and K is an $n \times n$ matrix of eigen-displacement coefficients. The reason for this decomposition is that only the first few eigen-displacements are necessary for a good approximation of matrix E (this is a consequence of the high correlation between the columns of matrix E). This means that instead of storing the matrix E, which is as big as the original animation, we store only the first f columns of matrix D and the first f rows of matrix K. An approximation of matrix E, which we denote as E', is then given as a matrix multiplication $E' = D'K'$, where D' and K' have the same size as D and K, but the last $n-f$ columns of matrix D' and last $n-f$ rows of matrix K' are zero. In a practical implementation, the matrix E' is actually never explicitly evaluated. Instead, the elements of E' are computed as needed, which can be efficiently implemented in a vertex shader. When computing the matrix palette skinning, the elements $(\mathbf{e}_k')^t$ of matrix E' are used as corrections to rest-pose vertex positions, i.e., the corrected deformed vertex positions are now computed as

$$\mathbf{v}_k^{defcorr} = T_k^t(\mathbf{v}_k^t + (\mathbf{e}_k')^t)$$

Typically, even a value of $f = 1$ adds most of the fine details, see Figure 4. When using rigid transformation fitting (Section 4.2), the normals can be corrected in the same way as vertex positions, giving other correction matrices D_n', K_n' for normals. If allowing non-rigid transformations, we can also use the same scheme, but we must take into account that the normals will be transformed by the inverse transposition of matrix T_k^t in this case.

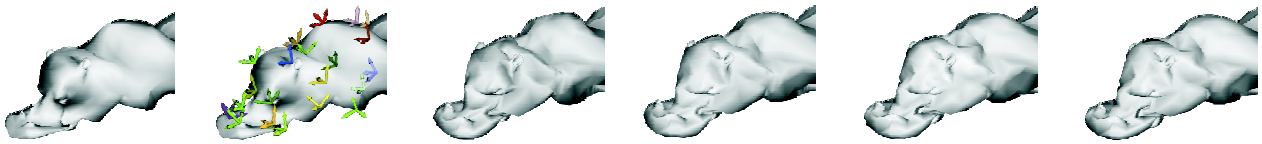
6 Experiments and Comparison

In order to obtain comparable results, we use the same error metric for measuring the fitting accuracy as in [James and Twigg 2005]. This metric is expressed as the percentage of distortion:

$$\%Error = 100\frac{\|P_{exact} - P_{approx}\|_F}{\|P_{exact} - P_{average}\|_F}$$

where P_{exact} is the $3m \times n$ matrix storing the original animation, P_{approx} is the animation reconstructed using our method and $P_{average}$ is a matrix where every column is the same and is equal to the average of P_{exact} over all columns (keyframes). The symbol $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. Our testing animations (except the elastic hippopotamus and shark) were created in 3D Studio Max using physically based cloth simulation. The elastic hippopotamus and shark are animated in the same software but as FFD soft bodies. For the visual results please see Figures 1, 7, 9 and the accompanying video. The results of approximating these animations using 100 proxy-joints are reported in Table 1. This relatively high number of proxy-joints is a conservative estimate of how many transformations are really needed. However, 100 rigid transformations per frame are unlikely to present an issue in terms of memory budget or a performance bottleneck (which is much more likely to occur in the vertex or fragment shader, because of the 3D models' sizes).

An important issue is how our method compares to Skinned Mesh Animations (SMAs) [James and Twigg 2005]. Unfortunately, we



no corrections | 1 eigen-disp. | 2 eigen-disp. | 3 eigen-disp. | original

Figure 4: EigenSkin corrections [Kry et al. 2002], adapted to our settings. Matrix palette skinning sometimes smoothes out the deformations, because of lack of samples (transformations). However, fine details can be recovered by adding only few eigen-displacements.

Animation	Vertices	Triangles	Keyframes	Pre-processing	% Error	Compression
Falling Skirt 1	1468	2811	30	2.2 min	0.69	5.9
Falling Skirt 2	4969	9706	60	10.7 min	0.86	17.6
Skirt Walk 1	2963	5747	54	8.2 min	0.84	11.6
Skirt Walk 2	7526	14766	60	19.9 min	1.33	23.1
Curtain	6409	12528	200	48.7 min	0.16	27.6
Elastic Hippopotamus	6442	11542	100	151.8 min	0.11	24.4
Elastic Shark	10070	19301	100	205.7 min	0.04	33.5
Cloth Hippopotamus	6442	11542	100	33.5 min	0.86	24.4
Cloth Camel	20330	28332	100	79.5 min	0.8	50.4

Table 1: Results of Skinning Arbitrary Deformations for our testing animations. Conditions: rigid transformation fitting, separate handling of vertex positions and normals, 100 proxy-joints, no corrections.

Animation	Vertices	Triangles	Keyframes	Joints	SAD Pre-process	SAD % Error	SMA Pre-process	SMA % Error
Cloth Horse	8431	16843	53	6	3.0 min	7.67 (0.17)	7.7 min	41.7 (0.88)
$\text{Flag}_{(32\,joints)}$	6906	13436	200	32	40.7 min	1.17 (0.44)	9.8 min	21.2 (5.93)
$\text{Flag}_{(100\,joints)}$	6906	13436	200	100	142.2 min	0.46 (0.17)	16.4 min	2.26 (1.25)
Elastic Cow	2904	5804	204	18	5.3 min	2.48 (1.2)	3.1 min	2.82 (1.54)

Table 2: Performance of our algorithm (SAD) executed on the highly deformable animations from the Skinning Mesh Animations (SMA) paper [James and Twigg 2005]. In both algorithms, we use rigid transformations and the same number of proxy-joints. The numbers in brackets denote the error after correction by 10 eigen-displacements. Compression ratio is not reported because is the same for both SAD and SMA.

could not simply run SMAs on our testing animations, because their implementation is not publicly available. However, we executed our algorithm on the highly deformable animations from the SMA paper. We compare the algorithms in the same setting (both use rigid transformations and the same number of proxy-joints). Note that in this case, the numbers of proxy-joints has not been set by the user (as before), but selected by James and Twigg's algorithm. The results are summarized in Table 2. We see that, with the sole exception of the elastic cow animation, our algorithm fits with more than five times higher accuracy. This is true for both uncorrected skinned animations as well as for corrections with 10 eigendisplacements (in brackets). In the case of the elastic cow, our algorithm achieves only a slightly better fit than SMA. We presume this is because the elastic cow is still quite similar to a quasi-articulated object, unlike the remaining cloth models.

Experiments on SMAs with about 100 proxy-joints for extra animations (e.g., the cloth horse animation) were carried out by James [2006]. Unfortunately, issues arose with the mean shift implementation (an algorithm used by SMAs), and it was found that it is not robust enough to handle such cases (i.e., with a lot of little clusters). We did not encounter any such robustness issues with our algorithm.

However, it should be noted that while our algorithm is also applicable to quasi-articulated animations, SMAs are more advantageous in this case. They automatically determine the lowest suitable number of proxy-joints, and also have shorter pre-processing times than our algorithm. Our algorithm spends the vast majority of pre-processing time in the optimization process (i.e., the LSQR algorithm [Paige and Saunders 1982]). This is because in our algorithm, each transformation in each keyframe undergoes optimization. Note that the pre-processing times are highly influenced by the related numerical issues, e.g., the condition number of the matrix (compare the pre-processing times for the Elastic and Cloth Hippopotamus).

An interesting question is how fast the error decreases for increasing numbers of proxy-joints. We computed the error for 1 to 100 proxy-joints, using both rigid and affine transformation fitting (see Figure 5). We observe two things: first, the rigid transformations quickly become almost as accurate as the affine ones. Therefore, rigid transformations (Section 4.2) are preferred, because they lead to twice as good compression as the affine ones. Second, after about 60 proxy-joints, the approximation error decreases very slowly. This suggests that the mesh corrections described in Section 5 are more suitable for adding fine details than increasing the number of proxy-joints.

The performance of skin corrections according to Section 5 is re-

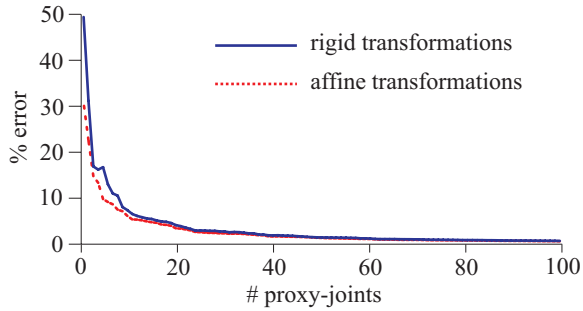


Figure 5: Falling Skirt 1 animation: error of fitting for increasing number of proxy-joints. Even for low numbers of proxy-joints, rigid fitting is almost as accurate as affine.

ported in Figure 6. We see that eigen-displacements are indeed a good alternative to increasing the number of proxy-joints, as the fitting error quickly drops to an unobservable level after applying the few first eigen-displacements. This is in accordance with the experiments of Kry et al. [2002].

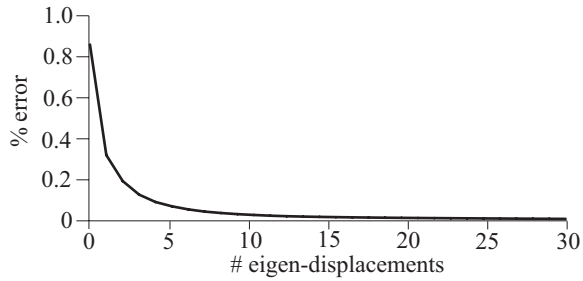


Figure 6: The cloth hippopotamus animation (with 100 proxy-joints and rigid transformations) is improved by adding eigen-displacements (Section 5). Even one eigen-displacement adds most fine details.

Three frames of the collapsing cloth hippopotamus animation are shown in Figure 7. We see that even the rigid transformation fitting without any corrections (top row) approximates the overall shape very well. However, some fine details are smoothed out, e.g., the crease under the hippopotamus' eye – see Figures 4 and 7. This is improved by adding 5 corrective eigen-displacements (Figure 7 middle row), which makes the reconstruction visually indistinguishable from the original animation (Figure 7 bottom). The error of the uncorrected skinning is 0.86% and decreases to 0.07% after correction with 5 eigen-displacements.

Matrix palette skinning approximations reduce the degrees of freedom of the animation, which is useful in a number of applications, as discussed already by James and Twigg [2005]. Our algorithm facilitates efficient hardware accelerated rendering, collision detection and rest-pose editing for non-quasi articulated models. Our performance testing scenario involves one thousand unsimplified elastic shark models (see Figure 8). The skinned approximations avoid the bottleneck of sending large amounts of data down the graphics pipeline, and therefore allows us to achieve 2.42 FPS instead of 0.64 FPS as in the classic approach (i.e., sending all vertex and normal data for each keyframe). Our implementation also benefits from the rigid transformation fitting, by sending dual quaternions instead of matrices (which is more efficient because a dual quaternion needs only 8 floats, instead of 12 for a rigid transformation matrix).

The deformable collision detection in [James and Twigg 2005] is based on a Bounded Deformation Tree [James and Pai 2004], constructed for each rigid component. Obviously, this cannot work for our method, because we do not assume the existence of any rigid components. However, the skinned approximation computed by our algorithm can be used for efficient collision detection as described in [Kavan and Zara 2005]. This deformable collision detection algorithm builds a sphere-tree for the rest-pose of the model, and refits the spheres on-demand, as required by each particular collision detection query. This refitting is based on the joint transformations, and thus is much faster than refitting for the original animation, which has to work directly with the vertex positions.

Finally, we demonstrate that our skinning approximations are robust enough to propagate small changes of the rest-pose geometry over the rest of the animation. This is a convenient tool for animation editing and/or simplification. Imagine, for example, that we want to change an original plain skirt into a pleated one. Thanks to our skinning approximation, this can be done simply by editing the rest-pose – see Figure 9. Note also that thanks to our simple weighting scheme (Section 3), it is also possible to change the mesh connectivity and, for example, perform mesh simplification.

7 Conclusions and Future Work

This paper presents a method to automatically generate skinned approximations of arbitrary deformations. To our knowledge, this is the first attempt to extend the popular matrix palette skinning

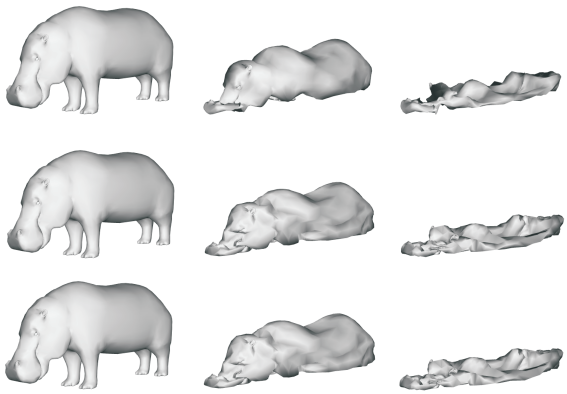


Figure 7: The animation of a collapsing cloth hippopotamus. **Top:** Uncorrected skinning with 100 proxy-joints and rigid transformations. **Middle:** Skinning corrected by adding 5 eigen-displacements. **Bottom:** The original animation.

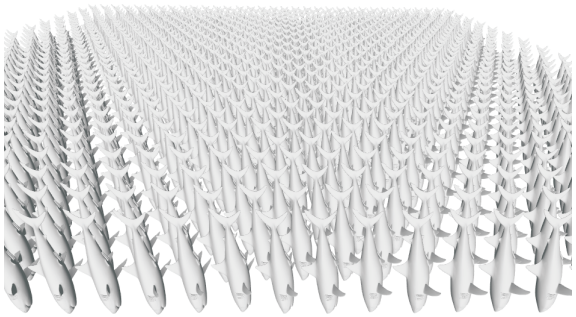


Figure 8: One thousand unsimplified shark models, animated using our method on a GeForce 6600 GT at 2.42 FPS.

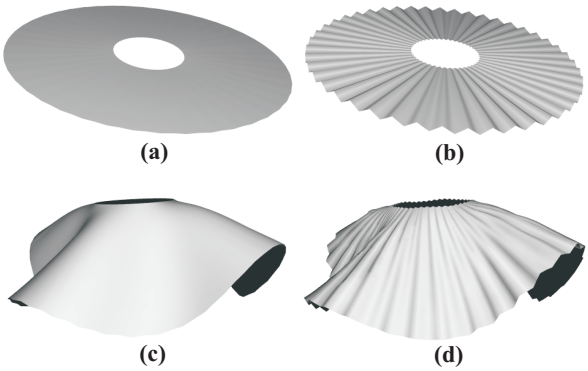


Figure 9: Rest-pose editing: the original plain skirt in the rest-pose (a), is changed to a pleated skirt (b). Our method then automatically propagates the pleats over the whole animation. The image (c) shows one frame of the original animation, and (d) the resulting one.

method to a more general class of deformations, such as those of cloth and elastic materials. Our algorithm can be easily incorporated into most existing real-time animation systems, yielding the benefits of memory saving and efficient rendering. There is also an interesting theoretical aspect to our approach: the description of a deformation field by sampling at discrete points, analogous to 2D digital image processing.

There are many possible avenues of future work. In our method, we combat only spatial correlation, but not temporal. The coherency between frames could be exploited to achieve more efficient data reduction. Other interesting future work would be to improve proxy-joint positioning, e.g., using relaxation [Turk 1991], or adaptive (and perhaps hierarchical) approaches. This would enable the algorithm to focus on the important parts of the animation and assign them more proxy-joints (even though the transformation fitting would probably be the same as described in this paper). Finally, the theoretical aspects of deformation field sampling could be studied, along the lines of digital image processing. For example, it might be possible to find an analogy in deformations fields to the sampling theorem, Fourier transform and band filtering.

8 Acknowledgements

We are indebted to Doug L. James and Christopher D. Twigg for their extensive support. We are grateful to Doug L. James also for providing the flag animation and for performing further tests with Skinned Mesh Animations. We also wish to thank Matthias Muller for the cow dataset, Robert Sumner for the collapsing horse and the anonymous reviewers for their helpful comments. We would like to acknowledge the support of the Higher Education Authority of Ireland. This work has been partly supported by the Ministry of Education of the Czech Republic under the research programs LC-06008 (Center for Computer Graphics) and MSM 6840770014.

References

AGARWAL, P. K., AND SHARIR, M. 1998. Efficient algorithms for geometric optimization. *ACM Comput. Surv. 30*, 4, 412–458.

ALEXA, M., AND MÜLLER, W. 2000. Representing animations by principal components. *Comput. Graph. Forum 19*, 3, 411–418.

BRICENO, H. M., SANDER, P. V., MCMILLAN, L., GORTLER, S., AND HOPPE, H. 2003. Geometry videos: a new representation for 3d animations. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 136–146.

COLLINS, G., AND HILTON, A. 2005. A rigid transform basis for animation compression and level of detail. In *Vision, Video, and Graphics*, 1–7.

GONZALES, T. 1985. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci. 38*, 22, 293–306.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 355–361.

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

JAMES, D. L., AND PAI, D. K. 2004. BD-Tree: output-sensitive collision detection for reduced deformable models. *ACM Trans. Graph. 23*, 3, 393–398.

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

JAMES, D. L., 2006. Personal communication.

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

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

KAVAN, L., COLLINS, S., O'SULLIVAN, C., AND ZARA, J. 2007. Skinning with dual quaternions. In *SI3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM Press, this issue.

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

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.

LENGYEL, J. E. 1999. Compression of time-dependent geometry. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 89–95.

LINDHOLM, E., KLIGARD, M. J., AND MORETON, H. 2001. A user-programmable vertex engine. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 149–158.

MAMOU, K., ZAHARIA, T., AND PRETEUX, F. 2006. A skinning approach for dynamic 3D mesh compression: Research articles. *Comput. Animat. Virtual Worlds 17*, 3-4, 337–346.

MCCARTHY, J. M. 1990. *Introduction to theoretical kinematics*. MIT Press, Cambridge, MA, USA.

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

OH, S., KIM, H., MAGNENAT-THALMANN, N., AND WOHN, K. 2005. Generating unified model for dressed virtual humans. *The Visual Computer 21*, 8-10, 522–531.

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

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

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 151–160.

TURK, G. 1991. Generating textures on arbitrary surfaces using reaction-diffusion. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 289–298.

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.