Animation space: a truly linear framework for character animation

BRUCE MERRY, PATRICK MARAIS and JAMES GAIN
University of Cape Town

Skeletal subspace deformation (SSD), a simple method of character animation used in many applications, has several shortcomings; the best-known is that joints tend to collapse when bent. We present *animation space*, a generalization of SSD that greatly reduces these effects and effectively eliminates them for joints that do not have an unusually large range of motion.

While other, more expensive generalizations exist, ours is unique in expressing the animation process as a simple linear transformation of the input coordinates. We show that linearity can be used to derive a measure of average distance (across the space of poses), and apply this to improving parametrizations.

Linearity also makes it possible to fit a model to a set of examples using least-squares methods. The extra generality in animation space allows for a good fit to realistic data, and overfitting can be controlled to allow fitted models to generalize to new poses. Despite the extra vertex attributes, it is possible to render these animation-space models in hardware with no loss of performance relative to SSD.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Hierarchy and geometric transformations*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Character animation, skinning, parametrization

1. INTRODUCTION

Character animation is quite different from other forms of animation (even facial animation) because of the underlying skeletal structure — motions are characterized more by rotations than by linear motion. Animators typically animate the bones of a character, either directly, or indirectly using inverse kinematics. The skin follows the bones, deforming as necessary to produce smooth transitions at joints.

The challenge of character animation is to model the skin in a way that lets it deform realistically, without the animation framework becoming too unconstrained. If over-constrained, the modeler will find it impossible to express what he or she has conceived; if under-constrained, it may be possible to express what one wishes, but only with significant effort (for example, by manually setting every vertex in every frame of an animation — theoretically possible but practically infeasible). To further complicate matters, an animation system must consider the rendering requirements: expensive physical simulations may be suitable for offline rendering, but are too slow for real-time applications.

Animation of any kind also introduces new challenges when combined with other areas of computer graphics; particularly areas that have traditionally considered only static scenes. For example, parametrization is usually optimized for a single pose, and the quality of the parametrization in other poses is ignored.

Our contribution is *animation space*, a character animation framework that generalizes and improves skeletal subspace deformation (SSD), a simple technique popular for real-time rendering. Unlike other

Authors' address: Department of Computer Science, University of Cape Town, Private Bag, Rondebosch 7701, South Africa; email: {bmerry,patrick,jgain}@cs.uct.ac.za. The research was funded by the National Research Foundation of South Africa and the KW Johnston and Myer Levinson scholarships.

skeletal animation techniques, animation space is specified by a simple and elegant linear equation, $\mathbf{v} = G\mathbf{p}$. Here, $\mathbf{p}$ is a generalized set of coordinates, G is a non-square matrix that generalizes the usual 4×4 matrices used to animate rigid bodies, and $\mathbf{v}$ is the output position. This linearity makes it particularly easy to incorporate animation space into existing algorithms. Specifically, we show that the linearity of animation space has the following benefits:

— It is possible to determine the average distance between two points under animation. This is a useful tool in applications where we wish to minimize some energy function across all poses.

— It is possible to fit a model to a set of example poses using least-squares methods. This allows a modeler to easily generate animation-space models, even without any knowledge of the algorithm.

— It is possible to synthesize new vertices that are always a particular affine combination of existing vertices. This has important implications for subdivision, where every newly created vertex (and indeed, every point on the limit surface) can be expressed as an affine combination of the control vertices. This also makes it possible to apply subdivision during modeling as well as during rendering.

We start by reviewing some existing frameworks in section 2. In section 3 we examine SSD in more detail. The mathematical background for animation space is presented in section 4. Section 5 derives the average distance norm and shows how it can be incorporated into an existing parametrization algorithm. In section 6, we show how animation-space models may be computed from a set of examples. We finish with results (section 7) and conclusions (section 8).

2. BACKGROUND AND RELATED WORK

There are a number of ways to animate characters, including [Collins and Hilton 2001]:

—Skeletal (bone-driven) animation;

—Shape interpolation, popular for facial animation but not applicable to animation of limbs;

—Spatial deformation

—Physically-based deformation;

—Direct animation of vertices or spline control points.

Skeletal animation is an important subset, because it is particularly simple for animation and rendering. This has made it attractive for games and other virtual environments, where it is used almost exclusively. For these reasons, we will concentrate only on skeletal animation. The interested reader is referred to Collins and Hilton [2001] for a broader overview of character animation, including a discussion of data acquisition.

Early work in skeletal animation, such as that of Magnenat-Thalmann et al. [1988], is somewhat physically-based. The programmer writes an algorithm for each joint to control the skin around that joint. While this gives total control over each joint, it requires too much effort for all but the highest-quality animations.

Later research has focused on generic algorithms that allow the modeler to control the shape of each joint. One of the simplest frameworks, and the one most commonly used in games, is known by several names, including linear blend skinning, skeletal subspace deformation (SSD) and smooth skinning [Mohr and Gleicher 2003]. We will refer to it as SSD. It was not originally published in the literature, but is described in a number of papers that extend or improve it [Lewis et al. 2000; Sloan et al. 2001; Wang and Phillips 2002; Kry et al. 2002; Mohr et al. 2003; Mohr and Gleicher 2003]. A skeleton is embedded in the model, and vertices are assigned to bones of the skeleton. Each vertex can be assigned to multiple bones, with a set of weights to indicate the influence of each bone. Vertices near joints typically attach to the bones that meet at the joint, so that skin deforms smoothly around the joint. Implementation details of SSD are discussed in the next section.

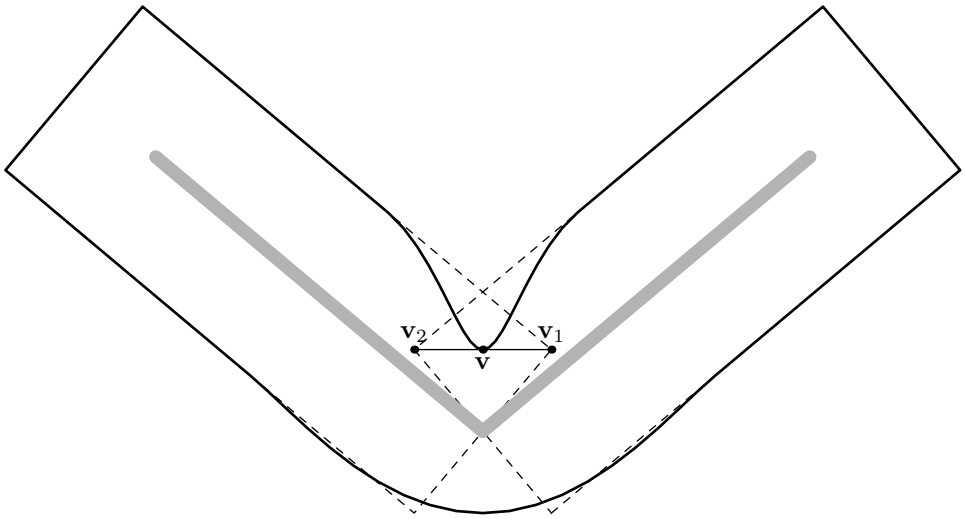


Fig. 1. Illustration of skeletal subspace deformation. Bones are shown in gray, and the rigid transformation of each half is shown by dashed lines. This produces $\mathbf{v}_1$ from the left bone and $\mathbf{v}_2$ from the right bone. The final position $\mathbf{v}$ is a linear combination of the two.

Because SSD uses linear interpolation, joints tend to collapse under animation (see Figure 1). Research into avoiding these flaws focuses on either example-based techniques, extra weights, extra bones, or non-linear interpolation:

Example-based techniques. Example-based techniques [Lewis et al. 2000; Sloan et al. 2001; Kry et al. 2002] use a database of sculpted examples, either manually produced by an artist, or generated from a physically-based modeling system (such as a finite element model, as used by Kry et al. [2002]). Each example has an associated pose, and the database should exercise the full range of each joint. The example meshes are compared to meshes generated with SSD in corresponding poses, and the difference vectors are stored. During rendering, scattered data interpolation generates a new difference vector that adjusts the result of SSD, giving a higher quality model.

These example-based techniques can produce high-quality models given a training set that spans that range of motion, but conversely the modeler must produce such a representative training set. Steps must also be taken to avoid over-fitting the model. The run-time efficiency is worse than that of SSD, as the results of the scattered interpolation must be applied in each frame.

Extra weights. Multi-weight enveloping (MWE) [Wang and Phillips 2002] replaces the scalar weights used in SSD with weight matrices. The extra degrees of freedom are sufficient to largely eliminate the flaws in SSD. A least-squares solver is used to compute weights that fit a number of examples. MWE is similar to our method, but has three times as many weights. This leads to potential problems with over-fitting and increases storage, bandwidth and computational costs.

Extra bones. Mohr and Gleicher [2003] introduce pseudo-bones into their models. For example, each joint is given a pseudo-bone that is rotated by half of the real joint angle. Vertices can be attached to this bone to receive this half-way rotation; since this is a spherical rather than linear interpolation, it avoids the collapsing effect seen in Figure 1. They also introduce pseudo-bones that scale rather than rotate, to support muscle bulging effects. The advantage of adding bones is that only the model, and not the framework, is altered; however the number of bone influences per vertex increases, which reduces rendering performance and may

make hardware acceleration infeasible (as a straight-forward implementation only allows a fixed number of attributes to be passed per vertex).

Non-linear interpolation. While we have described SSD as a linear blending of vertices, it can also be seen as a linear combination of the bone matrices, and the defects that occur arise because linear interpolation of rotation matrices does not correspond to an interpolation of their rotations. Magnenat-Thalmann et al. [2004] use the matrix blending operator of Alexa [2002] to perform the interpolation; this greatly improves the results, but is an expensive method. Spherical blend skinning [Kavan and Žára 2005] addresses the problem by computing linear interpolations of quaternions. This is less accurate than spherical interpolation and the system can only handle rotations, but the computational cost is reduced (it is still more expensive than SSD, however).

3. SKELETAL SUBSPACE DEFORMATION AND MULTI-WEIGHT ENVELOPING

Skeletal subspace deformation was briefly introduced in the previous section. Since animation space is an extension of SSD, we will describe it more fully here. We also review multi-weight enveloping, which shares some properties with animation space.

3.1 Notation and conventions

Except where otherwise noted, we will use the following notation:

—Scalars will be written in lowercase, normal font e.g., x.

—Vectors will be written in lowercase bold e.g., $\mathbf{x}$, with elements x_1, x_2, etc.

—Matrices will be written in uppercase, normal font e.g., M, with elements m_{11}, m_{12}, etc.

For modeling purposes, bones are represented as line segments, but for the purposes of analysis and rendering it is more useful to consider the *frames* (coordinates systems) that they define. Each bone defines a local coordinate system with its own origin (one end-point of the bone) and axes (relative to the direction of the bone). The joints define the transformations between parent and child frames.

The number of bones will be denoted by b, and the bones (and their frames) are numbered from 0 to $b-1$. The parent of bone j is denoted $\phi(j)$. We also make the following assumptions:

—There is a single root bone. Some modeling systems allow multiple root bones, but this can be resolved by adding a new bone to act as a "super-root".

—The root bone does not undergo character animation, and the root frame is thus equivalent to model space. This does not restrict the animator, as transformation of the root bone is equivalent to transformation of the model as a whole.

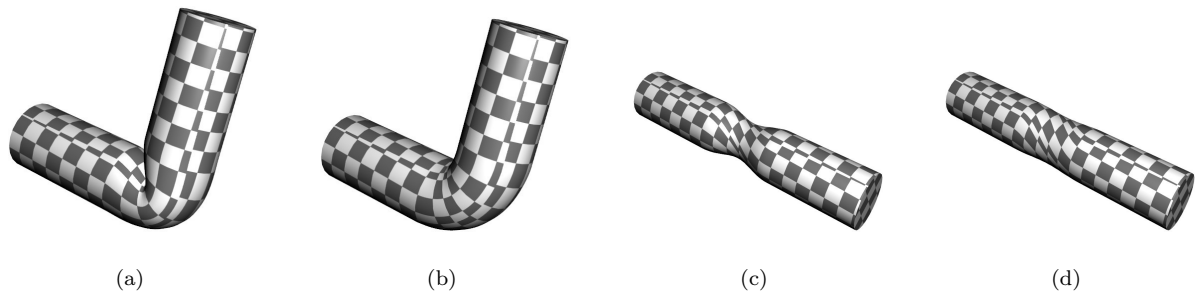
We will always label the root bone as bone 0. For each frame j, G_j is the matrix that transforms coordinates in that frame to model space (so $G_0 = I$, for example).

Modeling is done in a single pose, known as the rest pose (or dress pose). Values in this pose will be indicated with a hat (i.e., $\hat{\mathbf{v}}$, $\hat{G}$), while dynamic values will have no hat.

3.2 The SSD equation

We now explain the equation that underlies SSD. The formula is applied to each vertex independently, so we will consider only a single vertex $\mathbf{v}$ in homogeneous space. Let its position in the rest pose be $\hat{\mathbf{v}}$ and its influence from bone j be w_j. First, $\hat{\mathbf{v}}$ is transformed into the various local frames:

$$\hat{\mathbf{v}}_j = \hat{G}_j^{-1}\hat{\mathbf{v}}.$$



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

Fig. 2. (a) and (c): Tube animated with SSD, showing the collapsing elbow and candy wrapper wrapper effects. (b) and (d): Tube modeled in animation space, relatively free of these effects.

These local coordinates are then converted back into model-space coordinates, but this time using the dynamic positions of the bones (i.e., their positions in the current frame):

$$\mathbf{v}_j = G_j\hat{\mathbf{v}}_j = G_j\hat{G}_j^{-1}\hat{\mathbf{v}}.$$

Finally, these model-space positions are blended using the bone weights, as illustrated in Figure 1:

$$\mathbf{v} = \sum_{j=0}^{b-1} w_j G_j \hat{G}_j^{-1}\hat{\mathbf{v}} \quad \text{where} \sum_{j=0}^{b-1} w_j = 1. \tag{1}$$

3.3 Limitations of SSD

The "collapsing elbow" and "candy wrapper" effects, two well-known short-comings of SSD, are shown in Figures 2(a) and 2(c). They are caused by the linear blending of rotation matrices that have a large angle between them [Mohr and Gleicher 2003]. Figures 2(b) and 2(d) are animation-space models produced by fitting to a set of examples of a tube bending or twisting through 90°.

A less visible problem, but a more important one for algorithm design, is that one cannot synthesize new vertices as affine combinations of old ones. For example, suppose one wished to create a vertex that was the midpoint of two existing vertices. Averaging the rest positions and weights will not work, because they combine non-linearly in equation (1). This implies that a modeler needing extra vertices cannot simply subdivide a mesh; doing so will alter the animation, even if the new vertices are not displaced. As will be seen in section 5, it is also useful to be able to take the difference between two points; however, the behavior of the vector between two SSD vertices cannot be described within the SSD framework.

3.4 Multi-weight enveloping

To explain multi-weight enveloping (briefly mentioned in section 2), we first expand equation (1). For a bone j, let $N_j = G_j\hat{G}_j^{-1}$. This matrix defines how a bone moves relative to its rest position. Substituting N_j into equation (1) gives

$$\mathbf{v} = \sum w_j N_j \hat{\mathbf{v}} = \sum w_j \begin{pmatrix} n_{j,11} & n_{j,12} & n_{j,13} & n_{j,14} \\ n_{j,21} & n_{j,22} & n_{j,23} & n_{j,24} \\ n_{j,31} & n_{j,32} & n_{j,33} & n_{j,34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \hat{\mathbf{v}}. \tag{2}$$

Multi-weight enveloping assigns a weight to each entry of the matrix N_j, resulting in the equation:

$$\mathbf{v} = \sum \begin{pmatrix} w_{j,11}n_{j,11} & w_{j,12}n_{j,12} & w_{j,13}n_{j,13} & w_{j,14}n_{j,14} \\ w_{j,21}n_{j,21} & w_{j,22}n_{j,22} & w_{j,23}n_{j,23} & w_{j,24}n_{j,24} \\ w_{j,31}n_{j,31} & w_{j,32}n_{j,32} & w_{j,33}n_{j,33} & w_{j,34}n_{j,34} \\ 0 & 0 & 0 & \frac{1}{b} \end{pmatrix} \hat{\mathbf{v}}. \tag{3}$$

These extra weights allow for non-linear effects, and Wang and Phillips [2002] show that the extra degrees of freedom make it possible to avoid the collapsing elbow and candy-wrapper effects. They use a modified least-squares method to fit the weights to a set of examples. However, the rest pose configuration and vertex positions are determined by the user, presumably selected from the same database of examples.

The original paper uses 1 in place of $\frac{1}{b}$ in equation (3), but this gives $\mathbf{v}$ a weight of b rather than the more conventional 1. The difference is just a scale factor, so using this equation does not alter the method.

4. THE ANIMATION SPACE FRAMEWORK

Examining equation (1), we see that two vertex attributes (w_j and $\hat{\mathbf{v}}$) are multiplied together. Animation space is based on the idea of combining them into a single attribute.

4.1 Notation

In deriving some properties of the framework, we will make extensive use of homogeneous space, and will frequently need to extract parts of 4-vectors and affine 4×4 matrices (i.e., those whose last row is $\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$). We introduce the following notation to facilitate this:

$$\text{If } \mathbf{v} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \text{ then } \overline{\mathbf{v}} := \begin{pmatrix} x \\ y \\ z \end{pmatrix} \text{ and } \underline{\mathbf{v}} := w.$$

$$\text{If } A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ then } \overline{A} := \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \text{ and } \underline{A} := \begin{pmatrix} a_{14} \\ a_{24} \\ a_{34} \end{pmatrix}.$$

For matrices, we refer to $\overline{A}$ and $\underline{A}$ as the *linear* and *translation* components, respectively. We also note the following identities for affine matrices, which are easily verified by substitution:

$$\overline{A\mathbf{v}} = \overline{A}\overline{\mathbf{v}} + \underline{A}\,\underline{\mathbf{v}} \tag{4}$$

$$\underline{A\mathbf{v}} = \underline{\mathbf{v}} \tag{5}$$

$$\overline{AB} = \overline{A}\,\overline{B} \tag{6}$$

$$\underline{AB} = \underline{A} + \overline{A}\underline{B}. \tag{7}$$

4.2 Reformulating SSD

Let us re-examine equation (1). We can combine the weight, the inverse rest-pose matrix and the vertex into a single vector, and write

$$\mathbf{v} = \sum_{j=0}^{b-1} G_j \mathbf{p}_j, \tag{8}$$

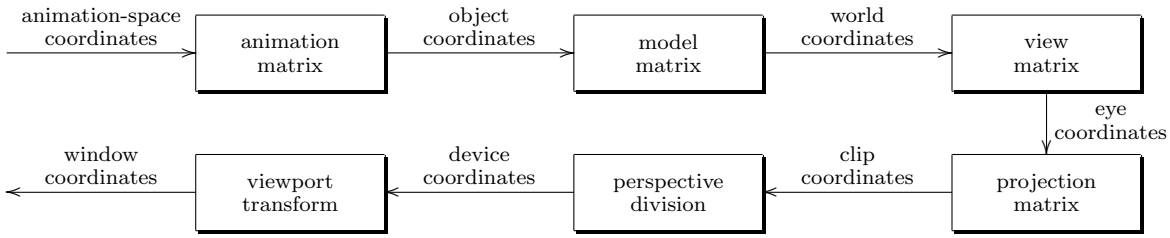


Fig. 3. Coordinate systems and the transformations between them. The animation matrix is our contribution, while the rest are standard (although OpenGL combines the model and view matrices into a "model-view" matrix). Note that animation of the model as a whole is done with the model matrix and not the animation matrix.

where $\mathbf{p}_j = w_j \hat{G}_j^{-1} \hat{\mathbf{v}}$. This sum can be rearranged as a matrix-vector product:

$$\mathbf{v} = \begin{pmatrix} G_0 & G_1 & \cdots & G_{b-1} \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_{b-1} \end{pmatrix} = G\mathbf{p}. \tag{9}$$

One can view the vector $\mathbf{p}$ as the coordinates of $\mathbf{v}$ in a multi-dimensional space, which we call *animation space*. This space is $4b$-dimensional[1]. The left hand matrix converts from animation space to model space; we label it G and refer to it as the *animation matrix* and its operation as *animation projection* (see Figure 3).

For a general element of the animation space, we find that

$$\underline{\mathbf{v}} = \underline{\mathbf{p}_0} + \cdots + \underline{\mathbf{p}_{b-1}}.$$

We refer to $\underline{\mathbf{v}}$ as the *weight* of $\mathbf{p}$, and also denote it $\underline{\mathbf{p}}$. Just as we usually work with homogeneous points in the 4D hyperplane $\underline{\mathbf{v}} = 1$, we will generally work with points in the animation-space hyperplane $\underline{\mathbf{p}} = 1$. In the context of equation (8), this simply says that the weights affecting a vertex sum to 1.

The restriction to this hyperplane still allows us $4b - 1$ degrees of freedom, while standard SSD has only $b + 2$ degrees of freedom ($b - 1$ independent weights plus the coordinates of $\mathbf{v}$).

4.3 Comparison to multi-weight enveloping

We can apply the same combining technique (that merged w_j and $\hat{\mathbf{v}}$) to multi-weight enveloping. Referring to equation (3), let $u_{j,rs} = w_{j,rs}\hat{v}_s$. Then

$$\mathbf{v} = \sum_{j=0}^{b-1} \begin{pmatrix} u_{j,11}n_{j,11} + u_{j,12}n_{j,12} + u_{j,13}n_{j,13} + u_{j,14}n_{j,14} \\ u_{j,21}n_{j,21} + u_{j,22}n_{j,22} + u_{j,23}n_{j,23} + u_{j,24}n_{j,24} \\ u_{j,31}n_{j,31} + u_{j,32}n_{j,32} + u_{j,33}n_{j,33} + u_{j,34}n_{j,34} \\ \frac{1}{b} \end{pmatrix} \tag{10}$$

Unlike SSD, MWE does not gain any generality from this transformation, as it can be reversed by setting $W_j = U_j$ and $\hat{\mathbf{v}} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}^T$. This also reveals that the so-called rest positions are in fact almost arbitrary, as weights can be found to match any rest position that does not have a zero component. From equation (10) it can be shown that animation space is a restriction of MWE, in which every row of U_j equals $(\hat{G}_j \mathbf{p}_j)^T$.

[1]It will later be seen that not all of these dimensions are used.

In practical terms, the extra generality of MWE allows a vertex to behave differently in different dimensions; for example, a vertex attached to a rotating joint may remain fixed in the x and y dimensions while moving sinusoidally in the z dimension. This has limited application, however, because these are global dimensions rather than the local dimensions of a bone frame, and hence any special effects obtained in this way will not be invariant under rotations of the model.

While MWE is more general than animation space, it is not necessarily better for all purposes. Most importantly, equation (10) does not have the elegant form of the animation-space equation $\mathbf{v} = G\mathbf{p}$ which makes the analysis of the following sections possible. The extra weights also require extra storage and processing, and do not necessarily contribute to the generality of the model: Wang and Phillips [2002] use Principal Component Analysis to reduce the dimension of the space.

5. DISTANCES IN ANIMATION SPACE

In this section, we will derive a measure for the average distance between two animated points, using parametrization as an example of its use in practice. A parametrization is a mapping between a surface and some part of the plane, or sometimes a simple surface such as a sphere or cube. Parametrizations are frequently used in texture mapping and other shading methods that use a lookup table (bump mapping, normal mapping, etc). They are also used in methods that re-sample a mesh [Eck et al. 1995].

The classical approach is to partition the mesh into a fairly small number of *charts* with disc-like topology, then flatten these charts onto the plane. The goal is to achieve a reasonably uniform sampling rate over the mesh, without producing too many discontinuities[2]. But sampling rate is a geometric property — so it will be affected by animation, and the goal must be to obtain uniform sampling both across the mesh and across the space of poses. In this section we will show how a particular flattening scheme can be adapted to use the animation space framework. An overview of other flattening schemes can be found in the survey by Floater and Hormann [2004].

The flattening stage of Least Squares Conformal Maps (LSCM) [Lévy et al. 2002] aims to maximize the conformality (angle preservation) of the map. While conformality does not imply area preservation, and in fact can produce very large variation in sampling rate, in practice it produces good results as long as the charts are roughly planar.

LSCM initially creates a local isometric parametrization for each triangle. The conformality of the triangle is expressed as a relationship between this local parametrization and the overall parametrization that is being computed, and the loss of conformality is measured by the deviation from this expression. This deviation is measured in such a way that the total loss of conformality is a quadratic function of the global parametric coordinates. A small number of vertices (usually two) are initially pinned, and a sparse least squares solver is then used to optimize the function (later work by Ray and Lévy [2003] improves efficiency using a multi-resolution solver).

By using a norm that measures average geometric distance across all poses, rather than the distance in only a single pose, we can adapt LSCM to produce a parametrization that maximizes conformality over a range of poses, rather than for just a single pose.

The modified norm (and an associated inner product) are derived in the following subsections. Given this norm, we need only derive a local isometric parametrization for each triangle (see Lévy et al. [2002] for details of how these local parametrizations are used to compute the global parametrization). Let the vertices have coordinates $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$ in animation space, and $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$ in the local parametric space. We aim to choose the parametric coordinates so that

$$\|\mathbf{u}_i - \mathbf{u}_j\| = \|\mathbf{p}_i - \mathbf{p}_j\|_{2,2} \tag{11}$$

[2]Some schemes go as far as to disallow any discontinuities, at the expense of a uniform sampling rate.

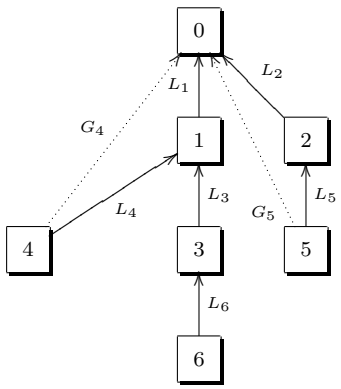


Fig. 4. Relationships between frames. Solid connectors represent joint transformations, while dashed connectors represent other transformations. The matrix L_i transforms from frame i to its parent, while G_i transforms from frame i to the root frame (frame 0). G_1 and G_2 are the same as L_1 and L_2, while G_3 and G_6 are omitted to prevent clutter.

for each i and j, where $\|\cdot\|_{2,2}$ is our modified norm. This is a fairly routine exercise in vector algebra, the details of which can be found in Appendix B.

5.1 Statistical analysis

So far we have used the term "average" distance, which is a misnomer: in fact we will derive the root-mean-squared distance. We term this the $L_{2,2}$ metric, indicating that it is spatially a 2-norm (Euclidean norm), and is also a 2-norm across the space of poses. In this section we will derive the $L_{2,2}$ metric; readers who are only interested in traditional skinning applications can proceed directly to section 6.

We will borrow the statistical notation $E[\cdot]$ to denote the expected (mean) value of an expression. We define our metric between points $\mathbf{p}$ and $\mathbf{q}$ in animation space in terms of a norm on the difference vector $\mathbf{s} = \mathbf{q} - \mathbf{p}$. We stress that what follows is valid *only* if $\mathbf{s}$ is a vector ($\underline{\mathbf{s}} = 0$) rather than a point.

We define the $L_{2,2}$ norm to be

$$\|\mathbf{s}\|_{2,2} = \sqrt{E\left[\|G\mathbf{s}\|^2\right]}. \tag{12}$$

Here, G is the random variable. Let us expand this formula:

$$\begin{aligned}
\|\mathbf{s}\|_{2,2}^2 &= E\left[\mathbf{s}^T G^T G \mathbf{s}\right] \\
&= \mathbf{s}^T E\left[G^T G\right] \mathbf{s} \\
&= \mathbf{s}^T E\left[\begin{pmatrix} G_0^T G_0 & \cdots & G_0^T G_{b-1} \\ \vdots & \ddots & \vdots \\ G_{b-1}^T G_0 & \cdots & G_{b-1}^T G_{b-1} \end{pmatrix}\right] \mathbf{s}.
\end{aligned} \tag{13}$$

This is not simply a distance in model space for a particular pose: the expectation operator gives the mean of its argument over all values for G, i.e., over all poses.

So far we have treated each frame as a separate entity, with no relationships between them, but for any meaningful statistical analysis we need to consider how the bones connect to each other. Each frame is either animated relative to a parent frame, or else is frame 0 (model space). Rather than work with the global transformations G_j (which transform directly to model space), we define L_j to be the local transformation which maps frame j to its parent (see Figure 4). In particular, $G_j = G_{\phi(j)} L_j$ if $j \neq 0$ (recall that $\phi(j)$ is the parent of j). We call these local matrices *joint* matrices, since they define the action of the joints connecting bones.

Let P be $E[G^T G]$; P is an expectation and hence independent of any particular pose of the model. To compute the expectation, we will need further information about how the skeleton will be animated. One

option is to take samples of the pose space (such as from a recorded animation), and average the values of $G^T G$ together. However, in such a high-dimensional space, it is impractical to obtain a representative sampling. In order to make sampling practical, we make the simplifying assumption that different parts of the body are not correlated e.g., arms can move independently of legs. This allows us to sample positions for each joint separately, rather than having to sample across all possible combinations of positions.

We describe next a number of statistical models, together with the prior information required to compute P given the assumptions of the model. The details of the computations can be found in Appendix A. These computations take $O(b^2)$ time for b bones (which is optimal given that P has $O(b^2)$ elements). The statistical models are listed from the most general (fewest assumptions, but most prior information) to the most specific (more restricted, but with less prior information required).

Independent joints. We assume that the joint matrices L_j are independent of the other, but may themselves have any distribution. We require $E[L_j]$ and $E[G_j^T G_j]$ for each joint.

Independent components. We assume that as well as the joints being independent, the linear and translation components $\overline{L_j}$ and $\underline{L_j}$ are independent for each joint. We require:

(1) $E[L_j]$
(2) $E[\overline{G_j}^T \overline{G_j}]$ and
(3) $E[\|\overline{G_{\phi(j)}}\underline{L_j}\|^2]$ for each joint.

The second expectation is a measure of the scaling effects of the transformation (and is the identity if no scaling occurs), while the third is related to the lengths of the bones.

This model is unlikely to be used in practice, but is useful in the derivation of the subsequent models.

Fixed translations. Usually, the bones in a model are rigid and thus cannot change their lengths. Within our framework, that corresponds to a fixed translation component for each joint matrix. Since this constant factorizes out of the expectations above, we require only $E[\overline{L_j}]$, $E[\overline{G_j}^T \overline{G_j}]$ and the constant value of $\underline{L_j}$ (i.e., the bones themselves).

Rotations and fixed translations. While our framework is general enough to support scaling matrices, they are not often used in practice (with the exception of muscle bulging). If we assume that only rotations and translations occur, then $\overline{G_j}^T \overline{G_j}$ is always the identity and so we only require $E[\overline{L_j}]$ and $\underline{L_j}$. The former can be computed given information about the range of motion of each joint: either by sampling the range, or analytically from a distribution function. As an example, consider the case where the joint can rotate only about the Z axis by angles between ϕ_0 and ϕ_1, with a uniform distribution. Then

$$\begin{aligned} E[\overline{L_j}] &= \frac{1}{\phi_1 - \phi_0} \int_{\phi_0}^{\phi_1} \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} d\phi \\ &= \frac{1}{\phi_1 - \phi_0} \begin{pmatrix} \sin\phi_1 - \sin\phi_0 & \cos\phi_1 - \cos\phi_0 & 0 \\ \cos\phi_0 - \cos\phi_1 & \sin\phi_1 - \sin\phi_0 & 0 \\ 0 & 0 & \phi_1 - \phi_0 \end{pmatrix}. \end{aligned}$$

This can be generalized to arbitrary axes of rotation and to multiple degrees of freedom (with Euler angles), although the formulae become more cumbersome.

Uniformly distributed rotations, fixed translations. In the absence of any prior knowledge (other than the skeleton itself), we can assume that all rotations are equally likely, giving $E[\overline{L_j}] = 0$. This is a dubious assumption (since it implies total freedom for every joint), but for some applications we have found the results to be better than just using an arbitrary pose and ignoring animation altogether.

Hybrid models. It is possible to combine several statistical models within a single skeleton. For example, if two joints are known to be correlated with each other but neither is correlated with any other, then only a subset of the expectations computed in Appendix A need to be modified to account for the relationship between the joints.

So far we have only used the two models with rotations and fixed translations, which are suitable for simple models where bones have fixed lengths. Where pseudo-bones are used (for example, to model muscle bulging), one of the more general models would be required.

5.2 Inner products

In the previous section we showed how an average value for squared distance can be found. This can be extended to finding the average value for the dot product. We use this in Appendix B to simplify the computation of a local parametrization for a triangle. In general, information about dot products may be useful because they are closely related to angles, and hence can indicate properties such as orthogonality.

We define $\langle \mathbf{s}, \mathbf{t} \rangle_{2,2}$ to be the expected dot product of model-space vectors $G\mathbf{s}$ and $G\mathbf{t}$:

$$\begin{aligned} \langle \mathbf{s}, \mathbf{t} \rangle_{2,2} &= E[G\mathbf{s} \cdot G\mathbf{t}] \\ &= E[\mathbf{s}^T G^T G \mathbf{t}] \\ &= \mathbf{s}^T E[G^T G] \mathbf{t} \\ &= \mathbf{s}^T P \mathbf{t}. \end{aligned} \tag{14}$$

Since $\langle \mathbf{s}, \mathbf{s} \rangle_{2,2}$ is the expectation of a squared length, it cannot be negative. The other properties of inner products are easily verified (since $G^T G$ is symmetric for all G), with the exception of $\langle \mathbf{s}, \mathbf{s} \rangle_{2,2} = 0 \implies \mathbf{s} = \mathbf{0}$. This will fail for any vector $\mathbf{s}$ in the null-space of P. Such a vector always exists in the *rotations and fixed translations* statistical model, because the skeleton has only three degrees of freedom per joint while animation space has four dimensions per joint. In spite of this short-coming of the $L_{2,2}$ norm and the inner product, we will continue to refer to them as a "norm" and an "inner product". Formally, they can be considered to be a norm and an inner product on the quotient space $\mathcal{M}/\mathcal{N}$, where

$$\mathcal{M} = \{\mathbf{x} \in \mathbb{R}^{4b} : \underline{\mathbf{x}} = 0\}$$

and $\mathcal{N}$ is the null-space of P; in this space they satisfy all the requirements of a norm and an inner product.

In practice, the issue of rank deficiency in P only becomes important when optimizing locations in animation space (for example, this makes the $L_{2,2}$ norm unsuitable for the edge constraints described in section 6.2). We do not project onto $\mathcal{N}^\perp$, as that would destroy sparsity in animation-space coordinates.

6. FITTING A MODEL TO EXAMPLES

SSD models tend to be modeled by the user in a modeling package. The rest pose of the model is built, then weights are "painted" onto the surface. This is a tedious and frustrating process, as setting weights only indirectly controls vertex positions. Mohr et al. [2003] have developed an authoring tool that allows vertices to be directly manipulated in posed positions, but the modeling is still all done by hand. In animation space, it is not practical to directly assign a $\mathbf{p}$ vector to each vertex, so some other modeling technique is necessary to take full advantage of the extra degrees of freedom that are available.

In the last few years there has been interest in fitting a model to a set of example poses. Example poses can come from a variety of sources, such as laser scans, high-end animation (e.g., a physically-based model), or an artist. Wang and Phillips [2002] use a modified linear solver to fit a multi-weighted enveloping model. Mohr and Gleicher [2003] use a bi-linear solver to fit an SSD model, while James and Twigg [2005] take the rest positions as given and solve a non-negative least squares problem to compute the weights.

All of these fitting schemes have two initial requirements: the example meshes must be in correspondence (i.e., each vertex in one mesh is associated with a particular vertex in the other meshes), and a skeleton, or at least a set of bone coordinate frames, must be known along with the associated pose for each of the examples. Establishing correspondence is a separate problem that we do not address (see for example Anguelov et al. [2004] for an automated solution). We discuss skeleton extraction in section 6.4.

The linearity of animation space makes fitting a relatively simple least-squares problem. Consider a single vertex, and a set of examples with poses $G^1, G^2, \ldots, G^n$ and corresponding vertex positions $\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^n$ (we use superscripts to distinguish G^1, an example of G, from G_1, a component of G). If the vertex has position $\mathbf{p}$ in animation space, then we aim to satisfy

$$\begin{pmatrix} G^1 \\ G^2 \\ \vdots \\ G^n \end{pmatrix} \mathbf{p} = \begin{pmatrix} \mathbf{v}^1 \\ \mathbf{v}^2 \\ \vdots \\ \mathbf{v}^n \end{pmatrix}. \tag{15}$$

In general this may be an over-constrained problem, so instead we minimize the objective function

$$E(\mathbf{p}) = \sum_{k=1}^{n} \left\| G^k \mathbf{p} - \mathbf{v}^k \right\|_2^2. \tag{16}$$

In practice, however, this will lead to over-fitting, and the resulting model will not generalize to new poses. Furthermore, it is likely that every bone will influence every vertex, which will make it unusable for real-time rendering. In the following sub-sections we address these problems.

6.1 Influence sets

Over-fitting tends to create spurious influences, where moving a bone in one part of the body has an effect in a completely separate part of the body, simply because this produces a better fit. This is usually avoided by limiting the bones influencing a vertex to a predefined set. Wang and Phillips [2002] require the user to paint an influence map onto the surface to establish where influences may exist for each bone; Mohr and Gleicher [2003], Kry et al. [2002] and James and Twigg [2005] automatically determine influences by looking for correlations between joint movement and vertex movement in the examples. Either of these approaches can be used in animation space; we have used manually designated influence sets in this paper.

6.2 Regularization

Regularization is a common method to prevent over-fitting. Equation 16 is modified to take the form:

$$E'(\mathbf{p}) = \sum_{k=1}^{n} \left\| G^k \mathbf{p} - \mathbf{v}^k \right\|_2^2 + \lambda \left\| \mathbf{p} \right\|_2^2. \tag{17}$$

The regularization parameter λ is usually chosen to be much smaller than the other coefficients, so that it has little effect when $\mathbf{p}$ is properly determined, but prevents $\mathbf{p}$ from becoming unnaturally large when it is under-determined.

The use of influence sets introduces a characteristic artefact that is not fixed by this form of regularization. Vertices at the boundary of an influence set may receive a large influence from the associated bone in order to obtain a good fit, while neighboring vertices outside the influence set receive no influence at all. When the model is placed into new poses, this often manifests as a discontinuity, such as a shearing effect in a limb. Wang and Phillips [2002] address this by adjusting the relative weights of the fitting and regularization terms to penalize large influences near the boundary (this requires a smooth user-specified influence map).

We address discontinuities by penalizing large jumps between neighboring vertices. Our final objective function is thus

$$E'' = \sum_{i=1}^{V} \sum_{k=1}^{n} \left\| G^k \mathbf{p}_i - \mathbf{v}_i^k \right\|_2^2 + \delta \sum_{j=1}^{E} \frac{1}{\ell_j} \left\| \mathbf{p}_{j,1} - \mathbf{p}_{j,2} \right\|_2^2, \tag{18}$$

where i runs over the vertices and j runs over the edges of the model, and k runs over the examples as before. $\mathbf{p}_{j,1}$ and $\mathbf{p}_{j,2}$ are the animation-space positions at the end-points of edge j, and ℓ_j is an estimate of the length of edge j (computed as the average length in the example meshes). The user specifies a value for δ, which weights the relative importance of continuity and closeness of fit.

Since we have used a Euclidean measure throughout, the function is not scale-invariant. In particular, w components in animation space are scale-invariant while the other components are not. To address this, we initially transform the models so that an axis aligned bounding box around the reference model has maximum side length of 1 and is centered at the origin. It is tempting to use the $L_{2,2}$ norm (which elegantly handles the issue of scale invariance), but, unfortunately, the directions which are under-determined are precisely those to which the $L_{2,2}$ norm assigns a low weight, because they are unlikely to contribute to the animated position.

The primary disadvantage of using these edge constraints is that the objective function is now defined globally over all vertices, rather than separately for each vertex. We have used LSQR [Paige and Saunders 1982], as implemented by the Meschach library [Stewart and Leyk 1994] to efficiently solve this large but sparse system. Meschach allows the matrix to be represented implicitly by providing callbacks to compute $A\mathbf{x}$ and $A^T\mathbf{x}$ given $\mathbf{x}$, and we have exploited this to reduce the memory required. The implementation uses $\mathrm{O}(Vn + I)$ memory to solve for V vertices, n example meshes and I variables; we show in the results that, in practice, the time and memory requirements are quite reasonable.

6.3 Homogeneous weight

The fitting process does not provide any guarantee that the resulting position $\mathbf{p}$ will satisfy $\underline{\mathbf{p}} = 1$; in fact, the regularization terms work against this. To guarantee $\underline{\mathbf{p}} = 1$, we choose some bone k in the influence set and substitute

$$\underline{\mathbf{p}_k} = 1 - \sum_{j \neq k} \underline{\mathbf{p}_j}$$

into the linear equations before eliminating $\underline{\mathbf{p}_k}$. The value of $\underline{\mathbf{p}_k}$ is determined from this equation after solving for the rest of $\mathbf{p}$. Note that here $\mathbf{p}_j$ is the 4-element sub-vector corresponding to bone j in position $\mathbf{p}$, whereas in equation (18) $\mathbf{p}_i$ is the animation-space position of vertex i.

6.4 Skeleton fitting

So far we have assumed that the skeleton (i.e., G) is known for all the example poses, but this may not be the case. For ease of implementation, we have used a method that is partially user-guided to determine a skeleton and associated poses. There are several fully-automated techniques available; see for example Anguelov et al. [2004] or James and Twigg [2005].

In our system, the user creates a skeleton, and marks a set of core vertices associated with each bone. The user-drawn skeleton is used only to provide a hierarchy and bone-vertex associations — the geometry is ignored. From here the approach is similar to that of James and Twigg [2005]. For each bone and each frame, a least-squares optimization is performed to find the affine matrix that best transforms the core vertices to fit the positions in this pose. For the models we have dealt with, scaling and shearing effects seem to result from over-fitting rather than properties of the models, and we eliminate them by taking only the rotation component of the polar decomposition [Golub and Van Loan 1996]. We then perform another optimization

to find the best translation for the chosen rotation.

We initially tried to preserve the bone lengths marked by the user and modify only the rotation; however, we found that it was difficult to manually determine the proper lengths, and incorrect lengths would over-constrain the problem and lead to poor fits.

7. RESULTS AND DISCUSSION

We now proceed to demonstrate the advantages of animation space, namely, that it:

—can be used to express practical models that cannot be expressed with SSD, in particular avoiding the problem of collapsing joints;

—allows parametrization to consider the full range of an animated character rather than a single pose, without being prohibitively expensive; and

—is suitable for real-time rendering, with performance equivalent to SSD.

The results were generated on an Athlon XP 2800+ processor, with 512MB of memory and an NVIDIA GeForce 5700LE graphics card.

In the tables that follow, the columns labeled *bones* and *influences* are respectively the total number of bones in the model (excluding any artificial root bone), and the average number of bones that influence each vertex. Our implementation of animation space exploits sparsity, since for many practical models (such as the horse shown in Figure 5), the average number of influences is small, even if the total number of bones is large.

In a number of places we have made comparisons to static versions of the models; these are processed with the same code, so the run-times are not as good as they might be with code designed specifically for static models.

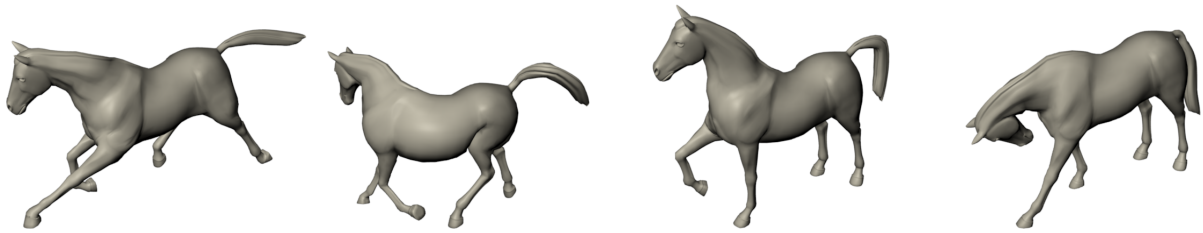
7.1 Fitting models to examples

Figures 2(b) and 2(d) show two models created with our fitting technique. In each case, we procedurally generated 40 example poses of the tube bending or twisting from a neutral pose to the poses shown. Since the skeleton was particularly simple, we allowed the bones to influence all the vertices. The process took 1.6 seconds per model. While this is a synthetic example, it clearly illustrates that automatic fitting can generate more accurate models than hand-designed SSD models (see the SSD models in Figure 2 for comparison).

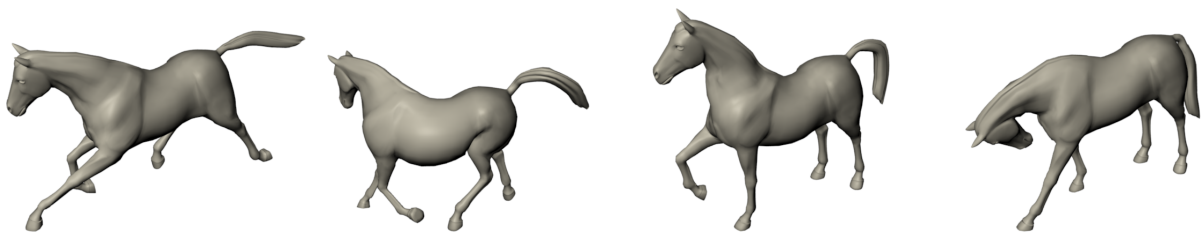
Where the range of motion is greater, the results are less satisfying, particularly for twisting. The behavior will depend on various trade-offs, such as which poses are considered more important and the balance between expansion and contraction. As an example, a joint which twists in a 180° range can be modeled such that the radius reduces by 12% at both extreme poses and expands by 10% in the central pose, a great improvement on the 100% collapse that SSD models undergo when twisted 180°. Where such a large range of motion exists, it may be necessary to introduce pseudo-bones [Mohr and Gleicher 2003] to minimize the collapsing effect.

A more realistic example is the horse shown in Figure 5. The available data consists of 11 example poses (four of which are shown in Figure 5(a)), and a 48-frame animation of the horse galloping. The example poses are not simply a subset of the gallop animation, but cover a range of head positions and standing, walking and running poses.

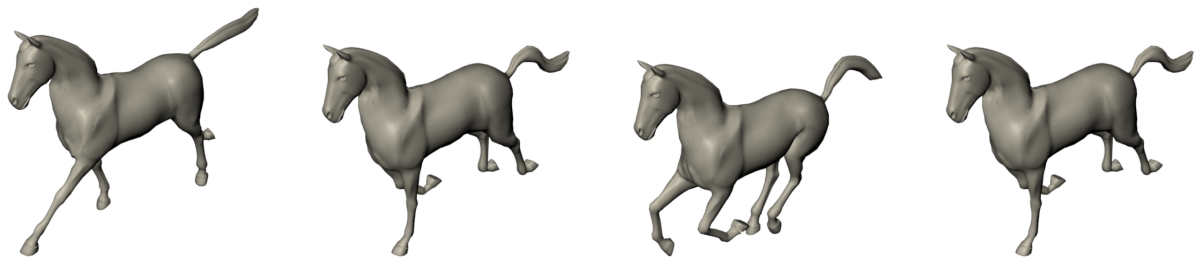
We used the 11 example poses to fit a model, then tested the model by attempting to match the gallop sequence. Since no skeleton was available, we manually inserted a skeleton with 24 bones into the model, and marked out influence sets. To adjust the skeleton to each pose, we used the vertices that were affected by exactly one bone as the core vertex sets to provide a reference frame; after doing so, we grew the influence sets slightly to allow for a better fit (we did not use the larger influence sets initially, as that would have



(a) 4 of the 11 examples meshes used to create the model



(b) the fitted model in the same four poses



(c) new poses generated from the fitted model

Fig. 5. Horse model created from 11 example meshes

reduced the core vertex sets, in some cases to nothing). Figure 5(b) shows the corresponding poses for the fitted model.

The fitting was reasonably well determined, and even with no regularization we found none of the gross artefacts usually associated with over-fitting. However, some regularization was required to prevent small artefacts at the boundaries of influence sets. We found that using $\delta = 0.005$ in equation (18) produced good results. The iterative least-squares solver converged in 4185 iterations, and the whole fitting process took 524 seconds (mostly spent in the least-squares solver). In fitting the 8431 vertices to the 11 models, slightly less than 40MB of resident memory was used.

In order to match the gallop sequence, we solved for the skeleton for each frame. This was done essentially

Table I. Parametrization speeds in seconds for various models. The difference column indicates the increase in processing time due to processing the skeletal information.

Model	Figure	Vertices	Bones	Influences	Static	Animated	Difference
Cylinder	2(b)	2426	2	2.0	14.4	16.6	15%
Tube	7	898	2	1.7	5.1	7.3	43%
Arm	6	1600	3	1.7	8.2	12.0	46%
Horse	5	8431	24	2.6	43	93	116%
Gingerbread man	—	12674	23	6.5	123	444	261%

as described in section 6.4, except that we used all the vertices of the model and solved for all the bones simultaneously to give an optimal fit. We did not, however, modify the animation-space positions. Figure 5(c) shows a few frames from this animation.

The arm model shown in Figure 6 posed more of a challenge. Only four poses were available, which is not enough to fully define the motion of a joint as complex as the shoulder. The solution was found to be particularly sensitive to the quality of the fitted skeleton. We manually marked the influence sets (Figure 6(e)), and separately marked the sets of core vertices used to fit the skeleton. We used $\delta = 0.02$. The choice of δ was quite important (far more so than for the horse). Smaller values tend to allow discontinuities, while larger values cause volume loss in convex areas (such as the muscles) as the edges seek to become as short as possible in order to minimize the penalty term. The iterative solver converged in 554 iterations, and the entire fitting process took 8 seconds.

Figure 6(a) shows the original data and Figure 6(b) shows the fitted model in these same poses. The third figure shows that the fit is not perfect: the outside of the shoulder is more pronounced in the original, and the highlights below the shoulder are slightly different. Nevertheless, the fitted model is a good qualitative fit and is free of discontinuity artefacts. Figure 6(c) shows the model in some new poses, indicating that it has not over-fitted the given data. Note that the slight collapse of the elbow in the third figure is due to joint collapse in the original data: the bent-elbow pose has significant volume loss, although it is not visible because the mesh folds over itself. Although the model generalizes best when interpolating the given poses, large extrapolations such as letting the arm drop to the side (Figure 6(d)) are still reasonable and do not suffer from discontinuities or gross volume loss.

Note that Lewis et al. [2000] have shown that SSD produces extremely poor results for the same arm model; in particular, the shoulder muscle tends to collapse (refer to the paper for images). Animation space thus provides a clear advantage here.

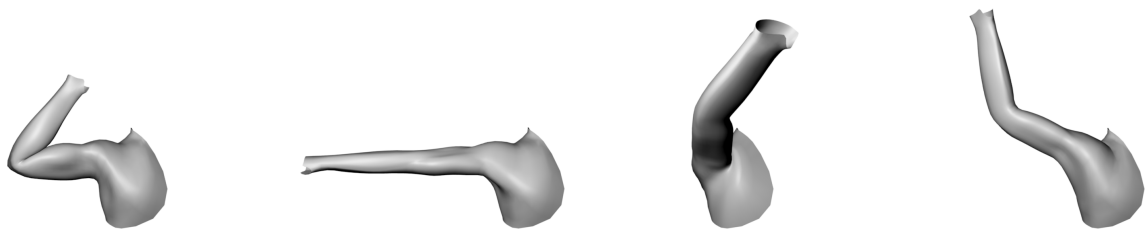
7.2 Parametrization

We have applied the modified parametrization technique described in section 5 to several models. For comparison, we performed a parametrization in model space without the skeleton (in the fully extended pose), and a parametrization in animation space. The relative speeds are shown in Table I. Note that while the gingerbread man is much slower to parametrize due to the large average number of influences, the horse (with a similar number of bones) has a smaller penalty. The technique is thus scalable, provided the number of influences is kept reasonable[3]. Since parametrization is performed once off-line, the extra processing is justified by the reduced distortion.

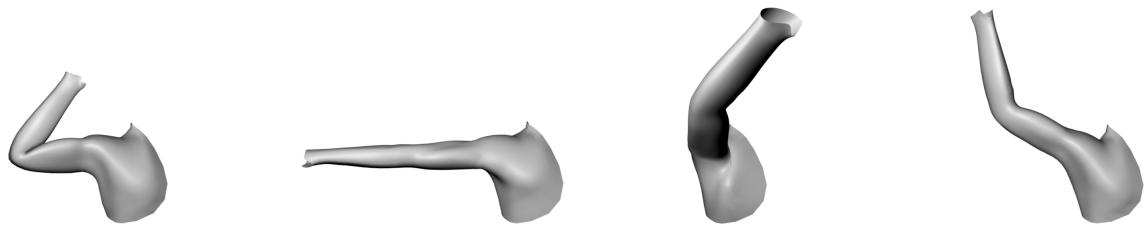
Figure 7 shows the results of parametrizing the tube model using three different statistical models. The numbers under the models give the stretch metric developed by Sander et al. [2001][4]. It measures the number of extra samples required to compensate for non-uniform sampling rates (0% indicating an isometric

[3]The inner product and $L_{2,2}$ norm take quadratic time in the number of influences, but for the small numbers of influences used here, this term does not dominate the overall running time.

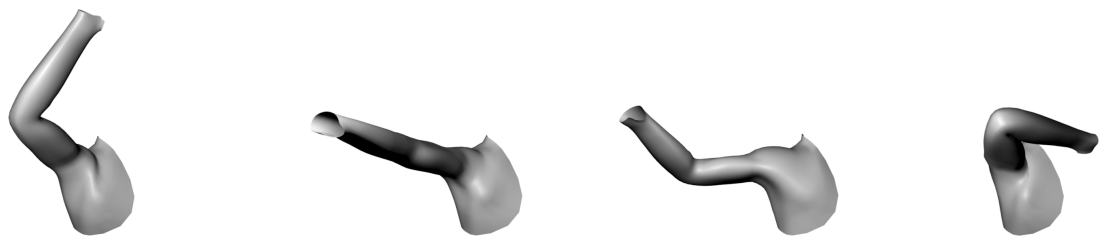
[4]We use their L_2 stretch metric, but we will just refer to it as the "stretch metric" to avoid confusion with our $L_{2,2}$ metric.



(a) example meshes used to create the model



(b) the fitted model in the same poses

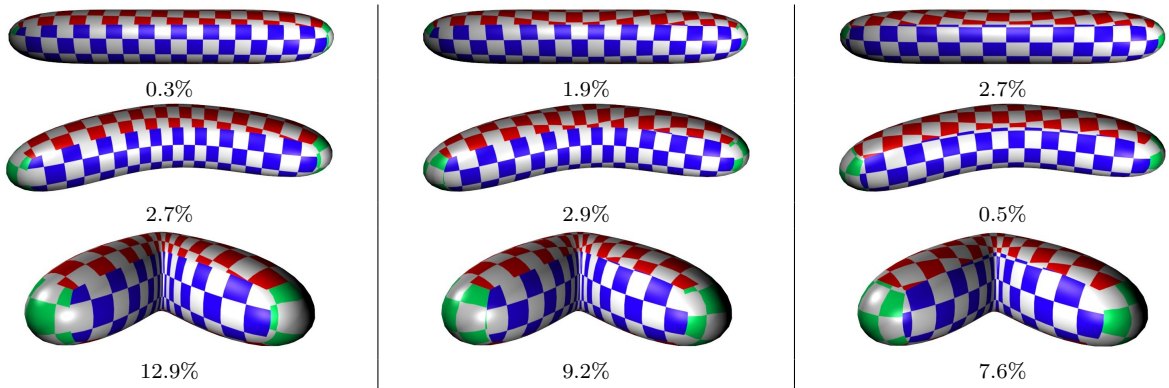


(c) new poses generated from the fitted model

(d) large extrapolation

(e) influence sets used

Fig. 6. Arm model, generated from the four poses in (a)



(a) No dynamic information (b) Skeleton only (c) Skeleton and animation

Fig. 7. Three parametrizations of a tube, in three different poses. (a) Only the top pose is considered. (b) The skeleton is used, but no information about the animation is considered. (c) Statistics about the animation are used. The numbers below each figure indicate the texture distortion. The naïve implementation (left) does well in the rest pose, but much worse than the others in the bent case.

Table II. Parametrization stretch. Each model is parametrized using the three methods illustrated in Figure 7. The stretch is averaged across all frames of the animation. In every case our skeleton-and-animation parametrization has less stretch than a naïve parametrization using only a single pose.

Model	**Figure**	**Static**	**Skeleton only**	**Skeleton and animation**
Cylinder	2(b)	1.5%	3.6%	0.8%
Arm	6(a)	15.5%	7.3%	3.7%
Horse	5	19.8%	51.1%	13.4%
Tube	7	4.1%	3.8%	2.2%
Gingerbread man	—	1.8%	17.8%	1.2%

parametrization). The left-hand column shows the results of parametrization in model space, without the skeleton. This produces an excellent parametrization in the rest pose (top), but when bent the texture is distorted in the region of the joint (pinched on the blue chart and curved on the red chart). The second parametrization (center column) uses the *uniformly distributed rotations, fixed translations* statistical model, with no information about how the model will be animated. Since there is no information about how the tube will be bent, it is not able to make a significant improvement beyond reducing the angle distortions on the red patch. The final column is created with the *independent rotations, fixed translations* statistical model. The required expectations were estimated from a 100-frame animation of the tube bending from the top to the bottom position. Notice that in the half-bent pose, the parametrization is close to isometric, with small distortions in both of the extreme poses. It could be argued that the static parametrization should have been done in this half-bent pose: in this case this would clearly have been beneficial, but for a more complex model it is not clear what pose would generate an optimal parametrization (especially where joints have multiple degrees of freedom), or even that such a single pose exists.

For each model, we have taken a sample animation (in the case of the arm model, we used just the four example poses) and parametrized the model using the three methods described above. For each static

Table III. Comparison of rendering performance for skeletal subspace deformation (SSD), animation space (AS) and multi-weight enveloping (MWE).

Model	Vertices	Bones	Average Influences	FPS		
				SSD	AS	MWE
Cylinder	38786	2	2.0	290	458	177
Tube	57346	2	1.8	200	327	94
Arm	100160	3	1.8	112	186	63
Hand	102274	29	2.5	109	187	–
Horse	137634	24	3.0	101	144	–

parametrization, the algorithm considers only a single rest frame, generally one with all limbs fully extended. Table II shows the average value of the stretch metric across all frames of the animation. The "skeleton-only" parametrization is often worse than a static parametrization. This occurs because it assumes that joint rotations are uniformly distributed (that is, any rotation is equally likely, including for example 180° rotations), while in most of these animations the animation deviates only slightly from the rest pose. The tube model, however, has greater variation, and the skeleton-only parametrization pays off.

The skeleton-and-animation parametrization takes the range of motion into account. Because of this, it produces the best parametrizations in all cases. In the case of the horse, we computed the expectations from the 11 reference poses, but measured the resulting error on the 48-frame gallop animation. This shows that the method is robust to new poses and does not simply optimize for known poses.

7.3 Hardware implementation

To compare the relative speeds of SSD, animation space, and multi-weighted enveloping, we have implemented a simple renderer that uses a vertex shader to implement animation on the GPU. We split each mesh into subsets according to the number of vertex influences, and used a separate vertex program for each piece. This allowed the performance to be determined by the average rather than the maximum number of influences.

In order to measure only the vertex transformation performance, we performed no shading, did no buffer clears and placed the model outside of the view frustum — thus generating no demands on the pixel pipeline. We also subdivided the models to create extra vertices, so as to minimize the effect of constant-time overheads (particularly switching between the vertex programs). A side effect is that the average number of influences increases slightly due to smoothing.

Surprisingly, the vertex programs for the three animation methods are very similar in length and structure. The programs for SSD are actually one instruction longer, because an extra multiplication is required to apply the weights at run-time whereas for animation space and multi-weighted enveloping the weights are pre-multiplied (we have used equation 10 for MWE). Of course, SSD can be implemented using the animation space algorithm, at the expense of passing extra attributes per vertex. NVIDIA [Dominé 2003] have presented this method, except that they fail to gain the extra instruction because they do not pre-multiply the weight.

The results are shown in Table III. The animation-space models were converted to SSD models simply by using the w coordinate from each bone as the associated weight. The result is not necessarily a good approximation, but since the models lie outside the view frustum, the final vertex positions have no effect on rendering performance.

Animation space clearly benefits from requiring one instruction fewer, in spite of the extra attributes. Due to the vector architecture of the hardware vertex engine, the animation space program only requires these extra attributes for vertices with three or more influences. Multi-weight enveloping, on the other hand, suffers from the much larger demand for attributes, despite using almost identical vertex programs to animation space. It also cannot handle more than four influences per vertex with our hardware, which is why we were unable to produce results for the more complex models. Animation space can handle up to 12

influences per vertex[5]. We have examined a number of the character models used in the game Doom III and found that the maximum number of influences is typically in the range 5–9 i.e., well within the capabilities of animation space but not of multi-weighted enveloping (models with fewer influences were either limbless or mechanical).

It should be emphasized that the results shown are exaggerated as they only consider the performance of the animation algorithm. In practical use, the animation algorithm is only one part of the rendering system and so the differences would be less pronounced.

8. CONCLUSIONS AND FUTURE WORK

Since animation space is a generalization of SSD, the improved parametrization technique discussed in section 5 can be readily applied to the large body of existing SSD models and modeling packages. However, real benefits will be seen when using the more general animation space: collapsing joints can be largely avoided, yet the animation-space models are no more costly to render and introduce a relatively small storage overhead[6].

We have also demonstrated that it is practical to create these models, using a small set of example poses. Nevertheless, some problems with overfitting remain. These could potentially be alleviated by starting with an SSD approximation, possibly created by hand, and doing the final fit by solving for corrections to this initial model. Regularization would then cause this initial approximation to be used where there is insufficient information in the examples.

Finding a way to comprehensively evaluate skinning techniques is not easy. Simple measures such as mean-square errors or Hausdorff distances are poor models of the human visual system, and may hide errors that are small yet highly visible — such as the discontinuities discussed in section 6. A detailed comparison must necessarily also involve the techniques used to establish all the necessary parameters, including skeleton fitting, influence sets, pseudo-bones and regularization parameters.

Several of the extensions to SSD discussed in section 2 (in particular, example-based methods and pseudo-bones) are largely independent of the underlying animation framework, and could benefit from using animation space rather than SSD. This would lead to even higher fidelity models than animation space alone can produce.

There are other areas of computer graphics that can potentially be adapted to animation space. We are currently investigating ways to incorporate animation space into progressive meshes, using the $L_{2,2}$ metric to measure the deviation between high- and low-resolution models. We have also developed a $L_{2,\infty}$ metric that measures maximum Euclidean distance across poses, and are investigating its application to visibility culling and collision detection.

APPENDICES

A. DERIVATION OF STATISTICAL MODELS

A.1 Independent joints

We start with knowledge of $E[L_j]$ and $E[G_j^T G_j]$ for each bone j.

In computing $E[G_j^T G_k]$, we start with the assumptions that $j \neq k$ (since otherwise we already have the value) and that k is not an ancestor frame of j (if it is, we can just swap the roles of j and k). These

[5]The gingerbread man was deliberately created to have large numbers of influences, and since it has 13 influences for some vertices we could not produce results for it.

[6]Although animation space has four times as many weights, it need not store a rest position. Thus, a model with an average of three influences per vertex requires only double the amount of storage.

assumptions guarantee that L_k is independent of both $G_{\phi(k)}$ and of G_j, so that

$$E[G_j^T G_k] = E[G_j^T G_{\phi(k)} L_k] = E[G_j^T G_{\phi(k)}] E[L_k]. \tag{19}$$

We can apply this formula recursively to the first factor on the right, until we obtain a factor of the form $E[G_l^T G_l]$.

A.2 Independent components

We start with the following prior information for each frame:

(1) $E[L_j]$

(2) $E[\overline{G_j}^T \overline{G_j}]$;

(3) $E[\|\overline{G_{\phi(j)}}\underline{L_j}\|^2]$ for $j \neq 0$, which we rewrite as $E[\underline{L_j}^T \overline{G_{\phi(j)}}^T \overline{G_{\phi(j)}} \underline{L_j}]$.

Since we showed above how to compute $P = E[G^T G]$ from $E[L_j]$ and $E[G_j^T G_j]$, we will now show only how to derive $E[G_j^T G_j]$. We first separate it into its components.

$$\begin{aligned} E[G_j^T G_j] &= E\left[\begin{pmatrix} \overline{G_j}^T & \mathbf{0} \\ \underline{G_j}^T & 1 \end{pmatrix} \begin{pmatrix} \overline{G_j} & \underline{G_j} \\ \mathbf{0}^T & 1 \end{pmatrix}\right] \\ &= E\left[\begin{pmatrix} \overline{G_j}^T \overline{G_j} & \overline{G_j}^T \underline{G_j} \\ \underline{G_j}^T \overline{G_j} & \underline{G_j}^T \underline{G_j} + 1 \end{pmatrix}\right] \\ &= \begin{pmatrix} E[\overline{G_j}^T \overline{G_j}] & E[\overline{G_j}^T \underline{G_j}] \\ E[\underline{G_j}^T \overline{G_j}] & E[\underline{G_j}^T \underline{G_j}] + 1 \end{pmatrix}. \end{aligned} \tag{20}$$

Since G_0 is the identity (in which case the problem is trivial), we can ignore the case $j = 0$ and assume that j has a parent. From equation (7), we have $\underline{G_j} = \underline{G_{\phi(j)} L_j} = \underline{G_{\phi(j)}} + \overline{G_{\phi(j)}} \underline{L_j}$. Substituting this into the bottom-right element of the matrix (20) gives

$$\begin{aligned} E[\underline{G_j}^T \underline{G_j}] &= E\left[\left(\underline{G_{\phi(j)}}^T + \underline{L_j}^T \overline{G_{\phi(j)}}^T\right)\left(\underline{G_{\phi(j)}} + \overline{G_{\phi(j)}} \underline{L_j}\right)\right] \\ &= E[\underline{G_{\phi(j)}}^T \underline{G_{\phi(j)}}] + 2E[\underline{G_{\phi(j)}}^T \overline{G_{\phi(j)}} \underline{L_j}] + E[\underline{L_j}^T \overline{G_{\phi(j)}}^T \overline{G_{\phi(j)}} \underline{L_j}] \\ &= E[\underline{G_{\phi(j)}}^T \underline{G_{\phi(j)}}] + 2E[\underline{G_{\phi(j)}}^T \overline{G_{\phi(j)}}] E[\underline{L_j}] + E[\underline{L_j}^T \overline{G_{\phi(j)}}^T \overline{G_{\phi(j)}} \underline{L_j}]. \end{aligned} \tag{21}$$

The expectations on the right hand side are either assumed to be given, or else form part of $E[G_{\phi(j)}^T G_{\phi(j)}]$. Similarly,

$$\begin{aligned} E[\overline{G_j}^T \underline{G_j}] &= E[\overline{L_j}^T \overline{G_{\phi(j)}}^T (\underline{G_{\phi(j)}} + \overline{G_{\phi(j)}} \underline{L_j})] \\ &= E[\overline{L_j}^T \overline{G_{\phi(j)}}^T \underline{G_{\phi(j)}} + \overline{L_j}^T \overline{G_{\phi(j)}}^T \overline{G_{\phi(j)}} \underline{L_j}] \\ &= E[\overline{L_j}]^T E[\overline{G_{\phi(j)}}^T \underline{G_{\phi(j)}}] + E[\overline{L_j}]^T E[\overline{G_{\phi(j)}}^T \overline{G_{\phi(j)}}] E[\underline{L_j}]. \end{aligned} \tag{22}$$

Again, the expectations on the right hand side are either given or part of $E[G_{\phi(j)}^T G_{\phi(j)}]$. Since $G_0 = I$, we can process the tree top-down to produce all the necessary values.

B. DERIVATION OF LOCAL ISOMETRIC PARAMETRIZATIONS

Recall that at the end of section 5, we needed to compute a local parametrization for a triangle in animation space that is isometric with respect to the $L_{2,2}$ norm. The details given below.

We exploit the fact that the inner product derived in section 5.2 corresponds to the $L_{2,2}$ norm, and thus the desired isometry will map it to the dot product of the parametric space (keeping in mind that it can only be applied to vectors, not positions).

The positions in animation and parametric space are $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ and $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ respectively. The absolute positions are not relevant, so we redefine the problem in terms of vectors:

$$\begin{aligned} \mathbf{q}_2 &= \mathbf{p}_2 - \mathbf{p}_1, \qquad & \mathbf{q}_3 &= \mathbf{p}_3 - \mathbf{p}_1 \\ \mathbf{d}_2 &= \mathbf{u}_2 - \mathbf{u}_1, \qquad & \mathbf{d}_3 &= \mathbf{u}_3 - \mathbf{u}_1 \\ \mathbf{u}_1 &= \begin{pmatrix} 0 & 0 \end{pmatrix}^T \end{aligned}$$

The orientation of the triangle in the parametric space is also irrelevant, so without loss of generality we can let $\mathbf{d}_2 = \begin{pmatrix} \|\mathbf{q}_2\|_{2,2} & 0 \end{pmatrix}^T$. This leaves only $\mathbf{d}_3$ to be computed; let $\mathbf{d}_3 = \begin{pmatrix} x & y \end{pmatrix}^T$. Then

$$\begin{aligned} x\|\mathbf{q}_2\|_{2,2} &= \mathbf{d}_2 \cdot \mathbf{d}_3 \\ &= \langle \mathbf{q}_2, \mathbf{q}_3 \rangle_{2,2} \end{aligned} \tag{23}$$

$$\begin{aligned} y^2\|\mathbf{q}_2\|_{2,2}^2 &= (2 \cdot \text{Area})^2 \\ &= \|\mathbf{d}_2\|^2\|\mathbf{d}_3\|^2 - (\mathbf{d}_2 \cdot \mathbf{d}_3)^2 \\ &= \|\mathbf{q}_2\|_{2,2}^2\|\mathbf{q}_3\|_{2,2}^2 - \langle \mathbf{q}_2, \mathbf{q}_3 \rangle_{2,2}^2. \end{aligned} \tag{24}$$

From these equations it is a simple matter to compute x and y. There will be two choices for y, corresponding to a reflection about the x axis; we choose y so that the triangle is wound counter-clockwise.

ACKNOWLEDGMENTS

We would like to thank J. P. Lewis, Robert Sumner and Jovan Popovic for making their data available for download.

We would also like to thank the anonymous reviewers, whose comments and suggestions helped to greatly improve this paper.

REFERENCES

Alexa, M. 2002. Linear combination of transformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press, New York, NY, USA, 380–387.

Anguelov, D., Koller, D., Pang, H.-C., Srinivasan, P., and Thrun, S. 2004. Recovering articulated object models from 3D range data. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, Arlington, Virginia, United States, 18–26.

Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Pang, H., and Davis, J. 2004. The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces. In *Proceedings of the Neural Information Processing Systems (NIPS) Conference*.

Collins, G. and Hilton, A. 2001. Modelling for character animation. *Software Focus 2,* 2, 44–51.

Dominé, S. 2003. Mesh skinning. NVIDIA presentation. `http://developer.nvidia.com/object/skinning.html`.

Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle, W. 1995. Multiresolution analysis of arbitrary meshes. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM Press, 173–182.

Floater, M. S. and Hormann, K. 2004. Surface parameterization: a tutorial and survey. In *Multiresolution in Geometric Modelling*, N. A. Dodgson, M. S. Floater, and M. A. Sabin, Eds. Springer.

Golub, G. H. and Van Loan, C. F. 1996. *Matrix computations*, 3rd ed. Johns Hopkins University Press.

James, D. L. and Twigg, C. D. 2005. Skinning mesh animations. *ACM Trans. Graph. 24,* 3, 399–407.

Kavan, L. and Žára, J. 2005. Spherical blend skinning: a real-time deformation of articulated models. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*. ACM Press, New York, NY, USA, 9–16.

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.

Lévy, B., Petitjean, S., Ray, N., and Maillot, J. 2002. Least squares conformal maps for automatic texture atlas generation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press, 362–371.

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., Cordier, F., Seo, H., and Papagianakis, G. 2004. Modeling of bodies and clothes for virtual environments. In *Third International Conference on Cyberworlds (CW'04)*. 201–208.

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.

Mohr, A. and Gleicher, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graphics 22,* 3, 562–568.

Mohr, A., Tokheim, L., and Gleicher, M. 2003. Direct manipulation of interactive character skins. In *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM Press, 27–30.

Paige, C. C. and Saunders, M. A. 1982. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw. 8,* 1, 43–71.

Ray, N. and Lévy, B. 2003. Hierarchical least squares conformal map. *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, 263–270.

Sander, P. V., Snyder, J., Gortler, S. J., and Hoppe, H. 2001. Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press, 409–416.

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.

Stewart, D. and Leyk, Z. 1994. Meschach library version 1.2b. `http://www.math.uiowa.edu/~dstewart/meschach/README`.

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.

Received Month Year; revised Month Year; accepted Month Year