Free-Form Motion Processing

Scott Kircher
University of Illinois at Urbana-Champaign
and
Michael Garland
NVIDIA

Motion is the center of attention in many applications of computer graphics. Skeletal motion for articulated characters can be processed and altered in a great variety of ways to increase the versatility of each motion clip. However, analogous techniques have not yet been developed for free-form deforming surfaces like cloth and faces. Given the time consuming nature of producing each free-form motion clip, the ability to alter and reuse free-form motion would be very desirable. We present a novel method for processing free-form motion that opens up a broad range of possible motion alterations, including motion blending, keyframe insertion, and temporal signal processing. Our method is based on a simple, yet powerful, differential surface representation that is invariant under rotation and translation, and which is well suited for surface editing in both space and time.

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

General Terms: Algorithms

Additional Key Words and Phrases: deforming surfaces, motion editing, motion blending, temporal signal processing, rotation-invariant surface representation, mesh editing

1. INTRODUCTION

Animators currently have at their disposal a wealth of techniques for editing and processing the motion of articulated characters. Characters can be easily reposed and their motion warped to interpolate the desired keyframe. Motion clips can be blended together to produce composite motions, and skeleton parameters can be manipulated to enhance or suppress features of the motion. Because generating original motion can be very expensive and time consuming, these motion processing techniques and others like them, are essential production tools.

Our goal is to make similarly powerful processing of free-form motion possible. Such motions can be free-form in both space and time: the object may be described by an arbitrary unstructured triangle mesh, and in each frame every vertex may have an arbitrary position. Datasets of this kind are typical of several common motion sources, including physical simulations of non-rigid objects (e.g., cloth) and direct high-resolution motion capture of surfaces (e.g., faces).

The foundation of our motion processing framework is a new differential mesh representation—where the position and orientation of each vertex and face is expressed relative to those of its neighbors. Because our representation

e-mail: kircher@engineering.uiuc.edu, mgarland@nvidia.com

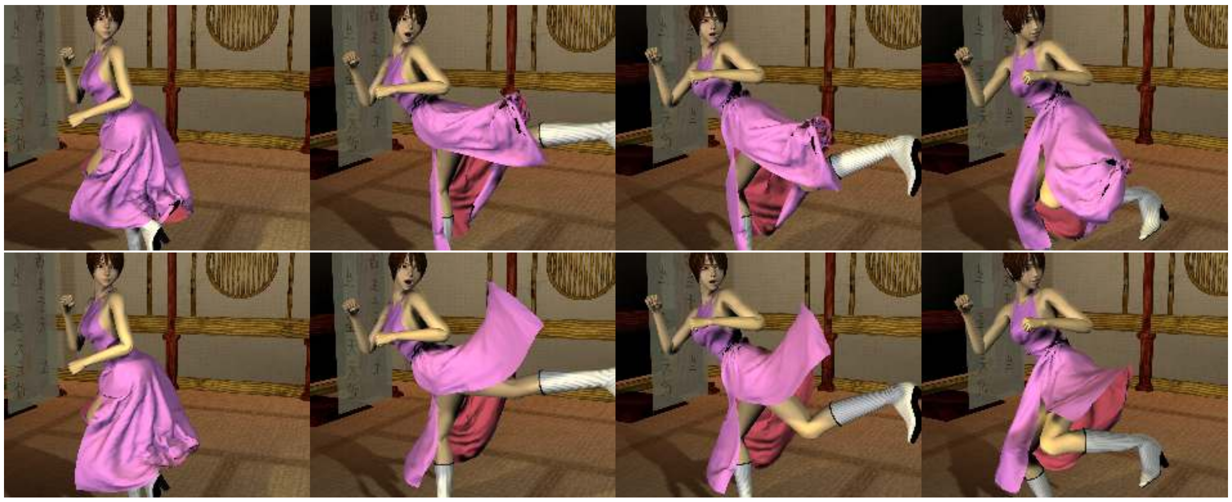


Fig. 1. We develop powerful tools for manipulating free-form deforming surfaces, like this simulated cloth (top). Here the user has applied fold mollification to fix simulation errors and inserted 4 keyframes to provide a more dramatic effect. Our system has automatically warped the motion to smoothly interpolate these constraints (bottom).

is constructed from relative transformations of triangles, rather than using the conventional vertex-based approach, it avoids the need to invent vertex tangent planes that lie outside the surface. This leads to a simpler formulation and allows us to accommodate any triangulated model, including non-manifold and non-orientable surfaces, with no additional effort. Existing vertex-based approaches would require, at least, careful formulation of special cases for non-manifold regions.

Our differential representation provides the basis for both spatial geometry processing and temporal motion processing. We use it to provide a keyframe editor; but, other geometric operations, such as smoothing, are also possible.

More importantly, we show how this differential representation can be used to provide a collection of powerful motion processing tools. Given one or more keyframes, we can insert them into a motion clip, warping the motion to smoothly interpolate them. We also derive techniques for blending clips together and for performing temporal signal processing on the motion itself, both of which are clearly superior to linear blending based on vertex positions.

2. BACKGROUND

Providing powerful tools for processing free-form motion, where an unstructured triangle mesh is allowed to move in an arbitrary manner, is a largely unaddressed problem. Our prior work [Kircher and Garland 2006] was the first to consider this problem. In that work, we developed a method for transporting multiresolution geometry edits made in one frame to all other frames in a temporally coherent manner. Although this provides several useful editing tools, it is rather limited in scope and cannot perform rotational edits. The approach we propose here essentially subsumes the functionality our previous method can provide, while providing much more powerful tools including keyframe insertion, temporal signal processing, and motion blending.

Like our proposed system, the method presented by Xu *et al.* [2007] also uses a differential representation for editing deforming surfaces, which they developed concurrently to our own work. Unlike our representation, however, theirs is still vertex based and thus requires the construction of local tangent planes. Also, our method for interpolating between keyframes can be used with any method for specifying those keyframes, whereas their's cannot.

Sumner *et al.* [2007] also present a deforming surface editing method developed concurrently to our work. Their method deforms space, rather than only the surface of a mesh, and so can be used with other types of animations, such as particle simulations. However, it is more difficult to make detailed edits with their coarse-graph structure. Furthermore, general motion processing operations such as blending and signal processing are not supported by their method.

In contrast to the general absence of methods for free-form motion processing, a broad spectrum of techniques have been developed to support the editing and processing of articulated motion. Direct manipulation of the skeleton for generating keyframes is usually performed via inverse kinematics or similar constraint-based methods [Gleicher 2001]. Motion warping [Witkin and Popović 1995], motion signal processing [Bruderlin and Williams 1995], and motion blending [Lamouret and van de Panne 1996; Kovar et al. 2002] are only a few examples of the kind of motion processing that can be done by algebraically manipulating the skeleton parameters.

James and Twigg [2005] showed how to fit a set of non-rigid bones to a deforming surface. For surfaces undergoing nearly articulated motion, this technique could be used to extend traditional skeletal motion processing techniques to a more general setting. Indeed, they demonstrate that their system can be used to perform simple edits of the animation. However, this approach does not generalize to our setting, where surfaces are allowed to deform in an arbitrary fashion. MESHIK [Sumner et al. 2005] is another system that aims to map traditional articulated motion techniques onto unstructured surfaces. It provides an inverse kinematics engine for unstructured meshes, where the kinematic behavior of the surface is automatically inferred from a set of example poses. This would be an appealing interface for creating keyframes that can be used in a system such as ours.

There are also many approaches available for editing and processing geometry represented by unstructured triangle meshes. Our work is inspired by recent developments in differential representations [Sorkine 2006], which represent local shape information in relative coordinate systems. The main challenge with such methods is to make the differential representation invariant under global transformations of the surface. Sorkine et al. [2004] use a formulation based on the Laplace equation and use a local linear approximation of rotations to ameliorate artifacts caused by rotational edits. The formulation used by Yu et al. [2004] is based on the more general Poisson equation, and they attempt to deal with rotations by explicitly propagating handle transformations to alter the coordinate gradient fields before reconstructing. Sheffer and Kraevoy [2004] developed the first rotationally invariant representation; however, their reconstruction process is non-linear and so relatively expensive. Lipman et al. [2005] developed a rotationally invariant representation whose reconstruction process is linear. Our approach continues developments in this area by providing a rotation-invariant representation with a linear reconstruction process that is simple to state and implement, and which can easily handle non-manifolds and non-orientable surfaces. Our representation is especially well suited to free-form motion processing. In particular, we do not orthonormalize our local frames, and can thereby easily capture the local stretch/skew motion of the surface.

3. DIFFERENTIAL REPRESENTATION

All editing operations in our system—in space and in time—are built on a differential representation of the surface. Assuming that we are given a simplicial 2-complex with n vertices and m faces, we aim to represent the position of each vertex in terms of differences from its neighbors, without reference to a global coordinate system. Our representation is built in two stages: first and second order differences. The first order differences are invariant under translations of the original surface. The second order differences are invariant under rigid motions (rotation and translation) of the original surface.

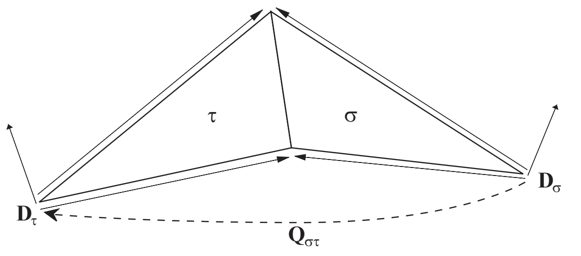


Fig. 2. Local trihedrons $\mathbf{D}_\sigma$ and $\mathbf{D}_\tau$ provide non-orthonormal frames for triangles σ and τ. Connection map $\mathbf{Q}_{\sigma\tau}$ encodes the transformation from one frame to the other.

3.1 First-Order Differences

For each triangle σ, with corners (i, j, k), we choose two edge vectors $(\mathbf{u}_\sigma, \mathbf{v}_\sigma)$ to provide a (non-orthonormal) basis for its tangent plane. We represent this tangential frame by the 3×2 matrix

$$\mathbf{F}_\sigma = \begin{bmatrix}\mathbf{u}_\sigma & \mathbf{v}_\sigma\end{bmatrix} = \begin{bmatrix}(\mathbf{x}_j - \mathbf{x}_i) & (\mathbf{x}_k - \mathbf{x}_i)\end{bmatrix} \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^3$ is the position of vertex i. This matrix provides the linear mapping of tangent vectors in the local coordinate system to world coordinates. It also has the important property of being a linear function of the positions of the triangle corners. Thus, if we collect all tangential frames into a single $3\times 2m$ matrix $\mathbf{F}$, there is a $2m\times n$ matrix $\mathbf{G}$, dependent only on the mesh connectivity, relating the tangential frames to the triangle positions

$$\mathbf{G}\mathbf{X} = \begin{bmatrix}\mathbf{F}_1 & \cdots & \mathbf{F}_m\end{bmatrix}^\mathsf{T} = \mathbf{F}^\mathsf{T} \tag{2}$$

where $\mathbf{X} = [\mathbf{x}_1 \ \cdots \ \mathbf{x}_n]^\mathsf{T}$ is the $n\times 3$ matrix of vertex positions.

These tangential frames are obviously invariant under translations of the original surface, but not under rotation.

3.2 Second-Order Differences

Given the unit normal $\mathbf{n}_\sigma$, we can extend our existing tangential frame to define a complete local trihedron for each triangle

$$\mathbf{D}_\sigma = \begin{bmatrix}\mathbf{u}_\sigma & \mathbf{v}_\sigma & \mathbf{n}_\sigma\end{bmatrix}. \tag{3}$$

Again, this matrix is the linear transform from the local coordinate system defined for the triangle to the corresponding world coordinates. As with the tangential frames, we define a $3\times 3m$ matrix $\mathbf{D}$ collecting all trihedrons together.

For a pair of adjacent triangles σ and τ, we can define a *connection map* $\mathbf{Q}_{\sigma\tau}$ mapping from the local coordinate frame of σ to that of τ as

$$\mathbf{Q}_{\sigma\tau} = \mathbf{D}_\tau^{-1}\mathbf{D}_\sigma. \tag{4}$$

This is depicted in Figure 2. Note that this is well-defined so long as τ is not degenerate.

The collection of all connection maps—one for each pair of adjacent triangles—provides a completely rotation-invariant representation of the surface geometry. It is easy to see that this is true since $(\mathbf{R}\mathbf{D}_\tau)^{-1}(\mathbf{R}\mathbf{D}_\sigma) = \mathbf{D}_\tau^{-1}(\mathbf{R}^{-1}\mathbf{R})\mathbf{D}_\sigma = \mathbf{D}_\tau^{-1}\mathbf{D}_\sigma$ for any orthonormal matrix $\mathbf{R}$ (rotation matrix).

Note, however, that the connection maps are *not* invariant under a general affine transformation $\mathbf{A}$ of the original surface. The above proof works for rotational matrices, but not for a general affine transformation because $\mathbf{AD}_\tau$ is not necessarily the trihedron corresponding to triangle τ after being transformed by $\mathbf{A}$. This is because the normal component of $\mathbf{D}_\tau$ should transform by $(\mathbf{A}^{-1})^\mathsf{T}$ not $\mathbf{A}$, in order to ensure that it remains normal to the surface.

3.2.1 *Analysis of Connection Equations.*

Our connection maps encode the local structure of the surface in a purely relative way, much like a Frenet representation of curves. In particular, we can think of them as encoding how the local trihedrons change between adjacent triangles.

Suppose that we wish to represent the change in the local trihedron going from σ to τ in the local coordinate frame of σ. This would give us the following *discrete connection equations*

$$\begin{aligned}\mathbf{u}_\tau - \mathbf{u}_\sigma &= \Gamma_{11}\mathbf{u}_\sigma + \Gamma_{12}\mathbf{v}_\sigma + \Gamma_{13}\mathbf{n}_\sigma\\ \mathbf{v}_\tau - \mathbf{v}_\sigma &= \Gamma_{21}\mathbf{u}_\sigma + \Gamma_{22}\mathbf{v}_\sigma + \Gamma_{23}\mathbf{n}_\sigma\\ \mathbf{n}_\tau - \mathbf{n}_\sigma &= \Gamma_{31}\mathbf{u}_\sigma + \Gamma_{32}\mathbf{v}_\sigma + \Gamma_{33}\mathbf{n}_\sigma\end{aligned}$$

which are analogous to the classical Gauss–Weingarten equations. Collecting the Γ_{ij} coefficients into a single matrix, we can rewrite the connection equations as $\mathbf{D}_\tau - \mathbf{D}_\sigma = \mathbf{D}_\sigma\boldsymbol{\Gamma}_{\sigma\tau}$, from which we can easily derive the formula for the connection coefficients

$$\boldsymbol{\Gamma}_{\sigma\tau} = \mathbf{Q}_{\tau\sigma} - \mathbf{I}. \tag{5}$$

Thus, we can see that the connection maps completely encode the variation of the local trihedron without reference to the global coordinate system.

Our connection equations bear a strong resemblance to the discrete surface equations defined by Lipman et al. [2005], and indeed they serve a similar purpose. However, our formulation describes connections between adjacent faces rather than neighboring vertices. This is a crucial distinction, because it avoids the need to invent tangent planes at vertices. Furthermore, we require no additional apparatus (e.g., first and second fundamental forms) since all of this information is contained within our connection maps. Also, our use of non-orthonormal tangential frames makes our representation better suited to motion processing, since the stretch skew motion of the surface is automatically captured in the frames themselves. These frames can then be easily factored into rotational and stretch/skew components via standard polar decomposition. In contrast, in the approach taken by Lipman et al. [2005], the local frames at each vertex are orthonormal. Thus, the stretch/skew information would be contained in their 1st and 2nd forms and would be mixed in with other information (such as the local heights of the vertices in each one-ring), making it harder to extract the stretch/skew component for motion processing.

3.3 Deformation and Reconstruction

At any instant in time, the surface geometry is completely captured by the set of trihedrons $\mathbf{D}$ and one or more *anchor* points, which specify the world coordinates for corresponding vertices. Given these, the linear system (2) uniquely determines the positions of all vertices, and guarantees reproduction of the original geometry up to the numerical precision of the solver.

Rather than simply reintegrating the surface, we are of course much more interested in deforming the surface in a natural way. We would like any deformation to preserve the basic structure of the surface, and it is this structure that is described by the set of connection maps. When deforming the surface, there is rarely an exact solution for reintegrating the shape, thus requiring least-squares approximation of the geometry.

3.3.1 *Vertex Fitting.* Suppose we are given a target tangential frame (or trihedron) for each triangle. We wish to find the vertex positions $\mathbf{X}$ that best fit these frames, in the least squares sense. We compute these positions by solving the system of normal equations

$$\mathbf{G}^{\mathsf{T}}\mathbf{G}\mathbf{X} = \mathbf{G}^{\mathsf{T}}\mathbf{F}^{\mathsf{T}} \tag{6}$$

with the position of at least one *anchor vertex* held fixed, to guarantee uniqueness. Solving this system is relatively inexpensive. The matrix $\mathbf{G}^{\mathsf{T}}\mathbf{G}$ is quite sparse and can be constructed directly from the mesh connectivity. Furthermore, we can factor it exactly once and reuse that factorization for all computations, since the matrix is strictly a function of the mesh connectivity. For the 23K polygon dress shown in Figure 1, construction, factorization, and back-substitution required 0.27, 0.20, and 0.06 seconds, respectively. These timings reflect using SuperLU [Li 2005] and unoptimized CBLAS on a 3 GHz Intel Xeon machine.

For a closed orientable manifold, this least-squares reconstruction is a Poisson system and is in essence equivalent to Poisson coordinate gradient fitting [Yu et al. 2004] and deformation gradient fitting [Sumner et al. 2005]. One can, for instance, interpret $\mathbf{F}_\sigma$ as the deformation gradient for the mapping of the canonical triangle $\langle(0,0),(1,0),(0,1)\rangle$ to σ. Botsch et al. [2006] provide a good analysis of the connections between these schemes.

3.3.2 *Trihedron Fitting.* When editing the surface, it is our connection maps that remain constant and from which we derive the set of trihedrons used in reconstructing the vertex positions.

In order to maintain accurate normal components of the local trihedrons during the reconstruction process, it would be natural to use a non-linear rotational norm (for example, performing polar decomposition on the trihedrons and using the matrix logarithm of the rotational part as a measure of error). However, this would obviously lead to a (potentially costly) non-linear optimization problem. Instead, we approximate a true rotational norm with a skewing norm— in particular the Frobenius matrix norm— which allows us to use linear least squares for reconstruction. This is analogous to the linear approach taken by Lipman et al. [2005] for finding local frames using their discrete surface equations.

From the definition of $\mathbf{Q}_{\sigma\tau}$, we note that

$$\mathbf{Q}_{\sigma\tau}^{\mathsf{T}}\mathbf{D}_{\tau}^{\mathsf{T}} - \mathbf{D}_{\sigma}^{\mathsf{T}} = 0. \tag{7}$$

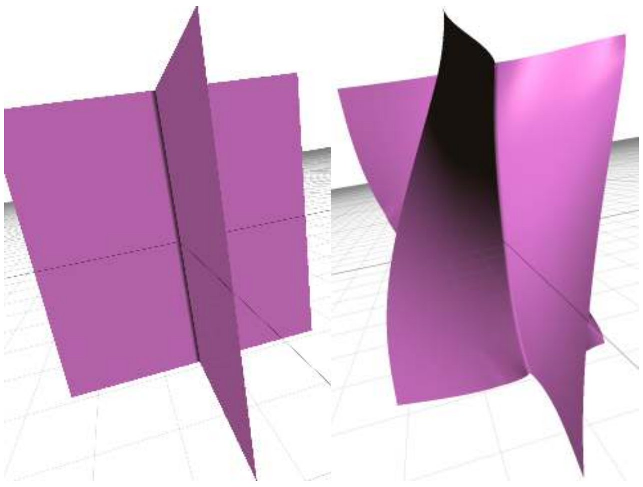
Consequently, we can define a $3p \times 3m$ matrix $\mathbf{H}$, where p is the number of triangle pairs, such that

$$\mathbf{H}\mathbf{D}^{\mathsf{T}} = \begin{bmatrix} & & & & & & \\ \cdots & -\mathbf{I} & \cdots & \mathbf{Q}_{\sigma\tau}^{\mathsf{T}} & \cdots \\ & & & & & & \end{bmatrix} \begin{bmatrix} \vdots \\ \mathbf{D}_{\sigma}^{\mathsf{T}} \\ \vdots \\ \mathbf{D}_{\tau}^{\mathsf{T}} \\ \vdots \end{bmatrix} = 0. \tag{8}$$

Given one or more target trihedron constraints, we can compute the least squares optimal collection of trihedrons by solving the system of normal equations

$$\mathbf{H}^{\mathsf{T}}\mathbf{H}\mathbf{D}^{\mathsf{T}} = 0 \tag{9}$$

with at least one *anchor trihedron* held fixed, to guarantee uniqueness. As with the vertex least squares system, the matrix $\mathbf{H}^{\mathsf{T}}\mathbf{H}$ is quite sparse and can easily be constructed directly. Our empirical results also indicate that the matrix $\mathbf{H}^{\mathsf{T}}\mathbf{H}$ is well-conditioned. However, since it is dependent on the connection maps and not just the mesh connectivity, it must be factored once per frame rather than once for all time. Construction and factorization of $\mathbf{H}^{\mathsf{T}}\mathbf{H}$ and back-substitution to solve for $\mathbf{D}$ required 0.51, 0.88, and 0.19 seconds, respectively, for the dress shown in Figure 1.


(a) Twisting a non-manifold

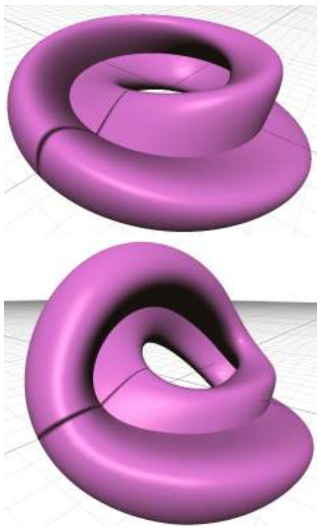

(b) Bending a Klein bottle

Fig. 3. Our approach naturally accommodates rotational edits, even on non-manifold and non-orientable surfaces.

Since we are using a linear approximation of a rotational norm, rather than an actual rotational norm, the normal components of the reconstructed local trihedrons are not necessarily the correct normals of the surface. However, we can easily recompute the correct normals from the two tangential components. This re-normalization has no effect on the subsequent vertex reconstruction— as the normal components are ignored in solving Equation (6). It is needed only for motion processing, where the full trihedron is used.

The fact that the reconstructed normal components may be incorrect is not, itself, a problem (since they can be recomputed). However, the degree to which they are incorrect is an indication of the degree to which the linear Frobenius norm failed in approximating the true rotational error. The main effect of this approximation is to cause the surface to shrink or skew slightly when the connection maps are inconsistent with each-other or with the imposed constraints. As with other linear approximations, we can produce better results in the presence of highly inconsistent connection maps or constraints by repeated linearizations. For example, an extreme bending edit (170° or more) can be broken into a couple of smaller steps— as is also done by Lipman et al. [2005]. Indeed, this repeated linear solving is essentially a simple form of nonlinear solver. As such, it is entirely possible to formulate the trihedron fitting problem as a nonlinear system, and apply an appropriate nonlinear solution method. However, this would be less efficient, and in many cases is not necessary.

Having fit the set of trihedrons, we find the final vertex positions by solving (6) above. At a high level, this two-phase reconstruction is similar to the scheme used by Lipman et al. [2005] where local orthonormal frames must be reconstructed before reconstructing the vertex positions. Like their approach, ours also has the limitation that rotational motion of the positional constraints does not automatically produce rotation of the surface. Rotational constraints must be supplied separately from the translational constraints.

Figure 3 shows two examples of deforming surfaces by this technique. It also highlights an added benefit of the fact that we do not need to construct coordinate frames at vertices: we are able to process both non-manifold and non-orientable surfaces. Every vertex where the two sheets of the surface shown in Figure 3a meet is a non-manifold vertex, yet our system reconstructs the intended twisted surface without difficulty. The surface in Figure 3b is a figureeight Klein bottle where the dark band indicates the orientation discontinuity; however, it can be bent just as easily as an orientable manifold. Such results are not possible using prior methods, which use vertex-based coordinate frames.

3.4 Temporal Representation

The representation we have just outlined captures the geometry of a static surface. An f-frame sequence is determined by the sequence $\mathcal{D} = (\mathbf{D}^1, \mathbf{D}^2, \ldots, \mathbf{D}^f)$ of per-frame trihedrons, plus one or more anchor vertices per frame. By default, we select the vertex with the least positional variance, although the user is allowed to select anchors directly. For animations with specific positional constraints, such as cloth simulations with points constraining the cloth to the character's body, it is preferable to use these constraint points as anchors instead. The motion of the anchor vertices is processed using linear vertex methods, i.e., anchor vertex keyframes are interpolated with a standard positional spline. During keyframe insertion, positioning the anchor vertex is analogous to positioning the root of a skeletal hierarchy.

Prior to performing any motion processing, we factor each local trihedron of each frame into a rotational and a stretch/skew part via polar decomposition [Shoemake and Duff 1992]. Because the trihedrons are constructed with unit normals, we know that all stretch/skew components are purely tangential. Therefore, we can represent the polar decomposition of the trihedron $\mathbf{D}^t_\tau$ of triangle τ at frame t with (1) a quaternion $\mathbf{q}^t_\tau$ to encode global rotation and (2) a symmetric 2×2 matrix $\mathbf{S}^t_\tau$ to encode tangential stretch/skew. For notational purposes, we also introduce the 3×3 matrices $\mathbf{R}^t_\tau$ and $\hat{\mathbf{S}}^t_\tau$ such that:

$$(\mathbf{q}^t_\tau, \mathbf{S}^t_\tau) = \mathsf{matrix}(\mathbf{q}^t_\tau) \begin{bmatrix} \mathbf{S}^t_\tau & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{R}^t_\tau \hat{\mathbf{S}}^t_\tau = \mathbf{D}^t_\tau. \quad (10)$$

Throughout the rest of this paper, we assume that all operations on pairs of quaternions occur after they have been oriented such that the spherical linear interpolation distance between the two quaternions is minimal.

The only pre-processing our method requires is to compute the polar decomposition of each local trihedron for each frame of the animation. This is a very fast process; the 23K triangle dress (Figure 1) requires only 0.03 seconds per frame. This is in stark contrast to the extensive pre-processing required by our previous work. On the same hardware, building the time-varying transform required over 9 seconds per frame to pre-process a model of only 16K triangles [Kircher and Garland 2006].

Any particular editing operation will generally involve specifying new constraints on the trihedron field. The connection maps allow us to reconstruct the deformed trihedron field for any particular frame, which subsequently determine updated vertex positions. Since construction time is small, we compute connection maps for frames as needed, rather than persistently storing them.

3.4.1 *Numerical Stability.* If a surface, such as a piece of cloth, deforms considerably over time, it is virtually guaranteed that a number of thin triangles will occur. Since we use non-orthonormal local frames in our representation, numerical stability in the presence of thin triangles is a concern.

There are two possible sources of numerical failure in our computations. First, the connection map computation involves the inversion of local trihedrons. Second, local trihedrons are factored into rotation and stretch/skew via polar decomposition. One or both of these operations might fail if the triangle is nearly degenerate. Our empirical tests indicate that a triangle with two sides of unit length may have an angle as small as 2×10^{-6} degrees before these operations fail. In the case where they fail— when the triangle is for all practical purposes merely a line segment— the solver must simply be careful to avoid attempting to reconstruct the triangle in question.

We have experimented with our method extensively during development. The examples shown in this paper have angles as small as 0.0036 degrees. As long as very basic numerical precautions are taken (i.e., rescaling models whose

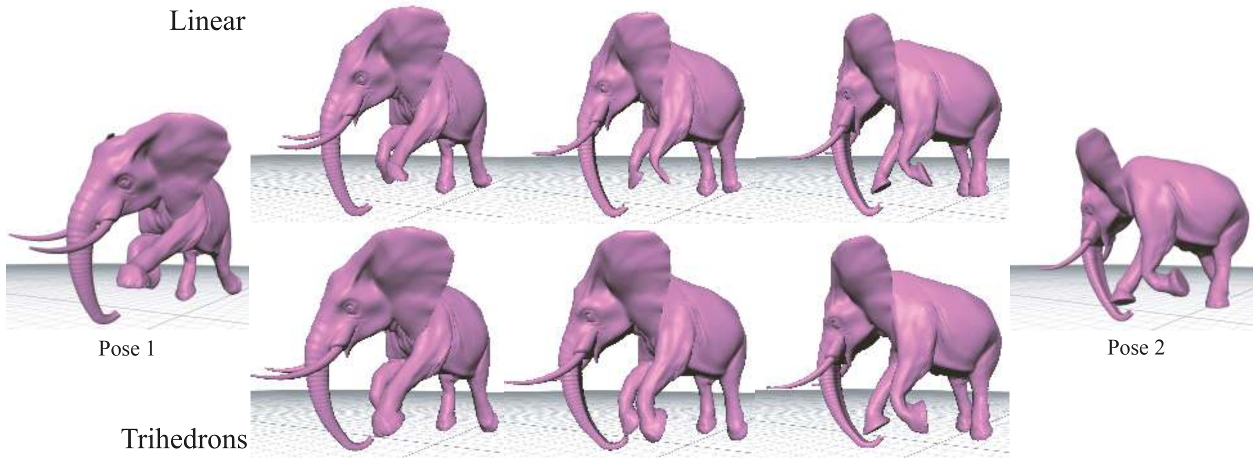


Fig. 4. Shape blending. Linear interpolation of vertices causes artifacts, such as a collapsing foot, that absolute blending prevents.

coordinate ranges are excessively small) we have not observed any artifacts from even very thin triangles.

Also, if the original mesh triangles are of highly different sizes, the least-squares solution does not take this into account. Thus, very small triangles will have more influence then they should, and very large triangles will have not enough influence. This can be corrected by weighting each row of equation (2) by the corresponding triangle areas.

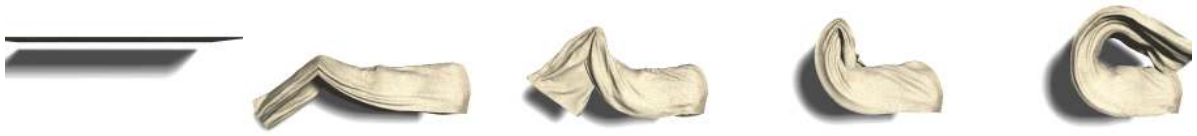
4. MOTION PROCESSING

We aim to provide the ability to perform general processing of free-form motion in ways analogous to existing techniques for skeletal motion. The deforming surface animations we seek to process may originate in a number of ways, including physical simulation, shape blending/morphing, or surface motion capture. Powerful editing tools would allow such motions to be manipulated, combined, and reused in new and different ways.

Propagating geometric edits across a clip from a single frame can provide some limited tools for these kinds of applications [Kircher and Garland 2006]. However, using the differential representation developed in the previous section, we can provide a much broader range of processing tools. In this section, we demonstrate how to perform signal processing in the temporal domain, blend different motion sequences together, and interpolate new keyframes into the animation sequence. Because an arbitrary number of constraints can be supplied to the reconstruction systems (§3.3) all of this processing can be performed subject to external constraints. This is analogous to constraint-based skeletal motion editing methods [Gleicher 2001], which have enjoyed great success in graphics research and the industry.

4.1 Frame Blending Operators

One of the most fundamental tools that we need is the ability to combine and to compute differences between corresponding frames. These algebraic operations on frames will allow us to perform a number of operations on frame sequences, including blending and signal processing.


(a) Absolute blending takes the shortest path in global rotation space, producing an undesirable result

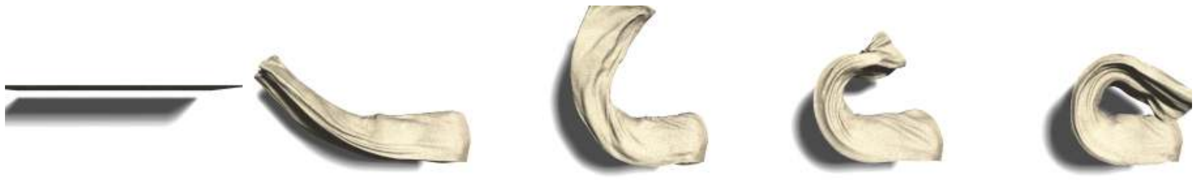

(b) Relative blending resolves this problem

Fig. 5. Blending an initially flat cloth animation (left) with a curved copy of itself (right).

4.1.1 *Absolute Blending.* Given the local trihedron of a triangle τ in frames s and t, we can directly blend the trihedrons in each of the frames. We define the generalized absolute subtraction operator $\ominus$ as

$$(\mathbf{q}^t_\tau, \mathbf{S}^t_\tau) \ominus (\mathbf{q}^s_\tau, \mathbf{S}^s_\tau) = (\mathbf{q}^t_\tau (\mathbf{q}^s_\tau)^{-1}, \mathbf{S}^t_\tau - \mathbf{S}^s_\tau). \tag{11}$$

Similarly, we define the generalized absolute weighted addition operator $\oplus$, with scalar weight w as

$$w(\mathbf{q}^t_\tau, \mathbf{S}^t_\tau) \oplus (\mathbf{q}^s_\tau, \mathbf{S}^s_\tau) = (\mathsf{pow}(\mathbf{q}^t_\tau, w)\mathbf{q}^s_\tau, w\mathbf{S}^t_\tau + \mathbf{S}^s_\tau). \tag{12}$$

We refer to these operators as *absolute* because we are combining orientations in the absolute world coordinates in which the surface is embedded.

Using these operators, we can easily define the operator for blending the trihedron fields of frames s and t as

$$\mathsf{blend}(\mathbf{D}^s, \mathbf{D}^t, w) = w(\mathbf{D}^t \ominus \mathbf{D}^s) \oplus \mathbf{D}^s, \quad w \in [0,1] \tag{13}$$

where $\oplus$ and $\ominus$ applied to the entire frame simply indicates pair-wise application to each pair of corresponding trihedrons. This amounts to spherical linear interpolation on $\mathbf{q}$ and linear interpolation on $\mathbf{S}$. Having constructed the blended trihedron field, the interpolated vertex positions are recovered using Equation (6). This is similar to the Poisson shape interpolation method [Xu et al. 2005] and deformation gradient based shape blending [Sumner et al. 2005] in that the orientation and stretch/skew of each triangle is interpolated separately and then recombined with a least-squares system. However, our method is based on a more general formulation (for example we do not require a separate reference mesh in order to define our local trihedrons) on which we build more powerful machinery, such as keyframe interpolation and temporal signal processing. Figure 4 shows an example of blending frames in this manner. As we would expect, it is clearly superior to linear interpolation of vertex positions.

4.1.2 *Relative Blending.* While absolute blending is obviously superior to simple linear interpolation on the vertices, it can produce quite undesirable behavior in certain situations. For example, if a surface folds over itself and is blended with a surface that lies flat, blending in absolute rotation space will cause the surface to pass through itself (see Figure 5a).

Absolute blending fails in such cases because it relies on the absolute global orientation of triangles. We eliminate this problem by introducing a relative blending operator that operates on the connection maps between triangles. Given the polar decompositions $(\mathbf{q}^t_\sigma, \mathbf{S}^t_\sigma)$ and $(\mathbf{q}^t_\tau, \mathbf{S}^t_\tau)$ for the trihedrons of two neighboring triangles σ and τ, we see that

$$\begin{aligned}\mathbf{Q}^t_{\sigma\tau} &= \left(\mathbf{R}^t_\tau \hat{\mathbf{S}}^t_\tau\right)^{-1} \mathbf{R}^t_\sigma \hat{\mathbf{S}}^t_\sigma \\ &= (\hat{\mathbf{S}}^t_\tau)^{-1} \left(\mathsf{matrix}((\mathbf{q}^t_\tau)^{-1}\mathbf{q}^t_\sigma)\right) \hat{\mathbf{S}}^t_\sigma. \end{aligned} \tag{14}$$

Therefore, to blend the connection maps $\mathbf{Q}^t_{\sigma\tau}$ and $\mathbf{Q}^s_{\sigma\tau}$ in frames s and t, we perform linear interpolation on $\mathbf{S}_\sigma$ and $\mathbf{S}_\tau$ and spherical linear interpolation on $(\mathbf{q}_\tau)^{-1}\mathbf{q}_\sigma$. Having blended the connection maps, the local trihedrons and vertex positions are recovered through the linear systems (9) and (6), respectively. The trihedrons of the anchor faces, which provide the constraints in (9), must be combined via absolute blending. As mentioned in §3.3.2, due to the fact that (9) minimizes the Frobenius norm of the local trihedron difference, rather than some nonlinear rotational norm, the surface can exhibit shrinkage when the source and destination frames are very different. We fix this problem by simply ignoring the stretch/skew components of the reconstructed local trihedrons, and instead using the stretch/skew matrices that are directly interpolated in construction Equation (14). This ensures that surface area is locally maintained.

By blending connection maps, which are entirely relative to the surface, rather than local trihedrons, which contain an absolute rotational component, we resolve the problem with absolute blending (see Figure 5b). This is very much analogous to performing skeletal motion blending by blending joint angles, rather than absolute bone orientations. However, it comes with the caveat that relative blending can be substantially slower than absolute blending. Relative blending requires refactoring (9) in each frame of the blend region, since the connection maps are changing, whereas absolute blending requires only back substitution. Relative blending may also produce visually distracting surface motion when one or more of the source frames contains highly crumpled geometry. For example, consider a long piece of cloth which is extremely crumpled in the middle, and which is being blended with a flattened piece of cloth. Suppose also that the trihedron anchors are all on one end of the cloth (all on the same side of the crumple). Now, the other end of the cloth will be affected by all of the extra movement that the crumpled region must go through to "uncrumple" itself. In such cases, the "incorrect" motion produced by absolute blending may be preferred, or it may behove the artist to add more trihedron anchors at the unconstrained end of the cloth to stabilize its global motion. More generally, relative and absolute blending can easily be combined. Absolute blending *must* be used for trihedron anchors, and in practice, we find it best to use absolute blending over an entire region around the anchor, since this more reliably maintains the global orientation of the blended shape.

4.2 Motion Blending

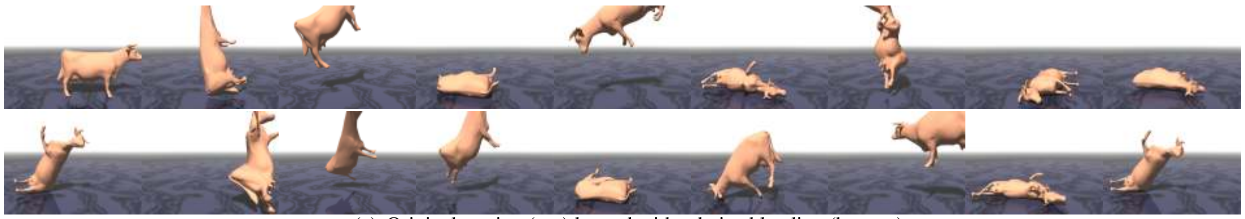
Having defined per-frame blending operators, we can now easily blend entire motion clips together. Given two clips $\mathcal{D} = (\mathbf{D}^1, \mathbf{D}^2, \dots, \mathbf{D}^n)$ and $\mathcal{E} = (\mathbf{E}^1, \mathbf{E}^2, \dots, \mathbf{E}^m)$, we must simply choose an appropriate alignment of the frames and a suitable weighting function to be able to blend corresponding pairs of frames.

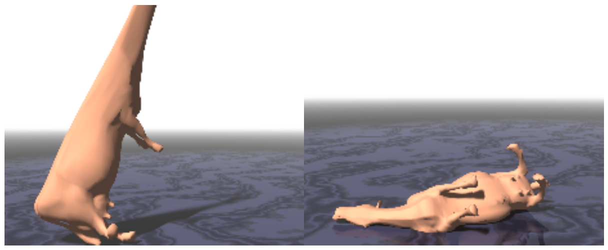
For the examples shown in this paper, we have used a simple linear blend weight function over a region of b frames. Schematically, the parameters of blend() will look like this:

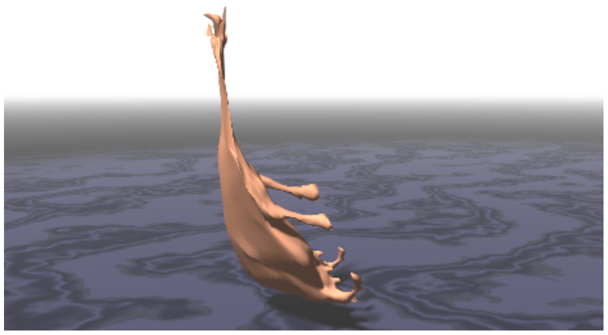
$\mathbf{D}^1$	$\cdots$	$\mathbf{D}^{n-b+1}$	$\mathbf{D}^{n-b+2}$	$\cdots$	$\mathbf{D}^n$		
		$\mathbf{E}^1$	$\mathbf{E}^2$	$\cdots$	$\mathbf{E}^b$	$\cdots$	$\mathbf{E}^m$
		$1/b$	$2/b$	$\cdots$	b/b		

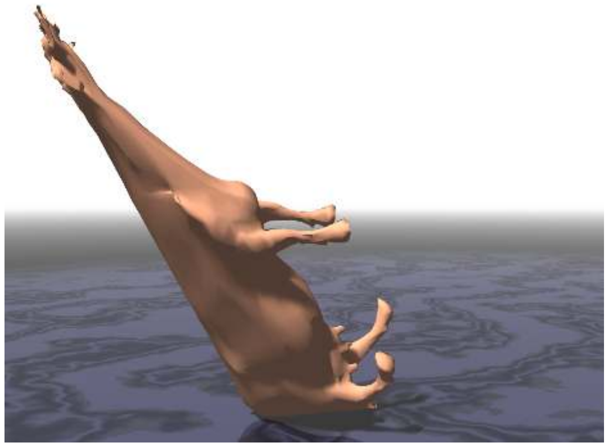
This basic construct can obviously be generalized in any number of ways, including use of higher order weighting functions and allowing separate weighting functions to be specified on a per-triangle basis.

We have already seen an example of blending two cloth animations together (Figure 5). By blending a bent copy of the original with itself, we can make the cloth appear to curl up over time. Another example of what motion blending can achieve is shown in Figure 6. Here we take an initial animation where a complex surface, namely a cow, undergoes a


(a) Original motion (top) looped with relative blending (bottom)


(b) Two source frames for blending


(c) Linear blending


(d) Absolute blending

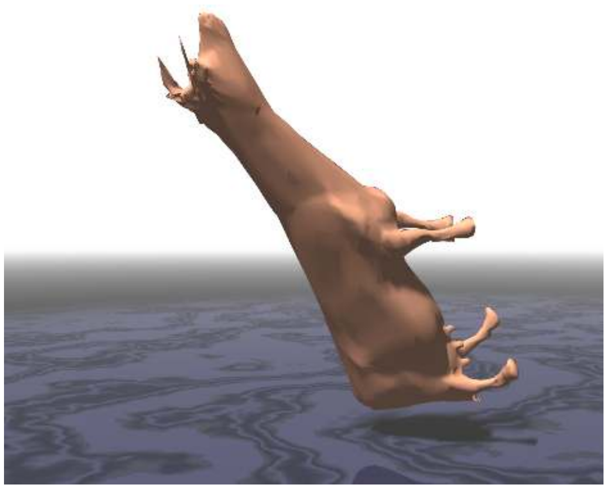

(e) Relative blending

Fig. 6. Making a clip loop seamlessly by blending first and last 60 frames. Relative blending maintains the shape quite well.

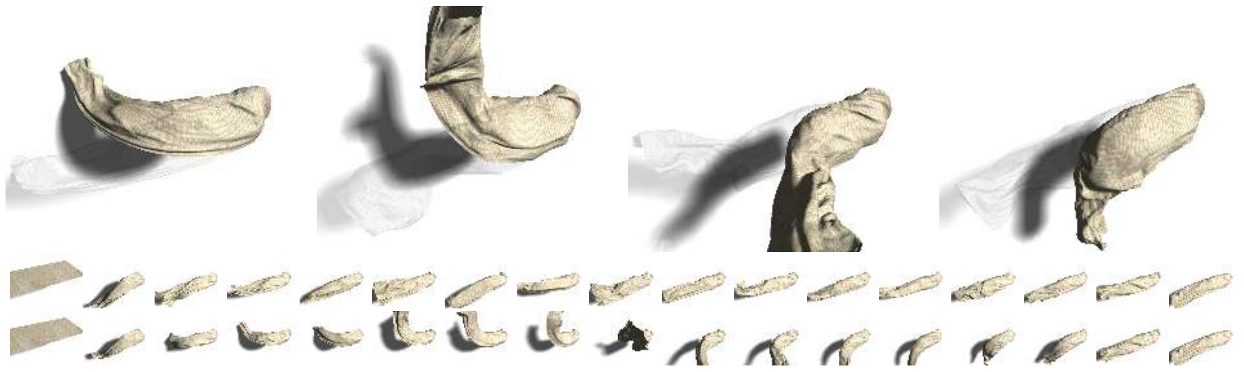


Fig. 7. By inserting new keyframes (top) into an existing cloth animation (middle) we can easily make significant alterations to the motion while preserving its character (bottom).

series of fairly radical non-rigid motions. By using relative blending to combine two clips formed from the first and last 60 frames, we can produce a new clip that loops seamlessly. This would be an obviously useful tool in building motion databases for later playback. Also notice that the shape of the cow is preserved quite nicely during blending, despite the rather extreme motion. Even absolute blending is not able to satisfactorily handle this case (though it is still significantly better than linear blending).

4.3 Key-frame Interpolation

One of the most important animation editing techniques is the ability to insert key-frames into existing animation sequences. Our previous work [Kircher and Garland 2006] allows edits of a single frame to be blended in over time, but that method provides no mechanism for interpolating between multiple keyframes overlaid on an existing deforming surface animation. Our new framework makes this easy.

Suppose that, given the animation sequence $\mathcal{D} = (\mathbf{D}^1, \mathbf{D}^2, \ldots, \mathbf{D}^f)$, the user is free to edit the geometry of frame t in any way they desire. This new geometry will determine a new trihedron field $\mathbf{E}$, and we can compute the keyframe difference $\mathbf{K} = \mathbf{E} \ominus \mathbf{D}^t$ that encapsulates the changes the user has made. It is then a simple matter to propagate these changes to all other frames of the animation by computing

$$\mathcal{D}' = (w_1\mathbf{K} \oplus \mathbf{D}^1, w_2\mathbf{K} \oplus \mathbf{D}^2, \ldots, w_f\mathbf{K} \oplus \mathbf{D}^f). \tag{15}$$

This provides functionality very similar to what we proposed in our earlier work [Kircher and Garland 2006], albeit without the need for much of the machinery that method requires (e.g., basis smoothing). Our new, differential approach can also easily support a much more general operation.

Now suppose that the user is free to edit any number of frames, producing several new keyframe differences $\mathbf{K}^s$, $\mathbf{K}^t$, $\mathbf{K}^u$, $\mathbf{K}^v$, etc. By using linear splining for $\mathbf{S}$ and quaternion splining [Eberly 2002] for $\mathbf{q}$—there is a separate spline for each triangle— we can directly produce the interpolated difference sequence $\mathcal{K} = (\mathbf{K}^1, \mathbf{K}^2, \ldots, \mathbf{K}^f)$. We then simply apply this difference sequence to the original

$$\mathcal{K} \oplus \mathcal{D} = (\mathbf{K}^1 \oplus \mathbf{D}^1, \mathbf{K}^2 \oplus \mathbf{D}^2, \ldots, \mathbf{K}^f \oplus \mathbf{D}^f) \tag{16}$$

to produce the final animation. We use interpolatory splines; therefore, the new animation sequence passes exactly through the user-defined keyframes. It also contains all the motion of the original animation. This is analogous to Motion Warping for skeletal animations [Witkin and Popović 1995].

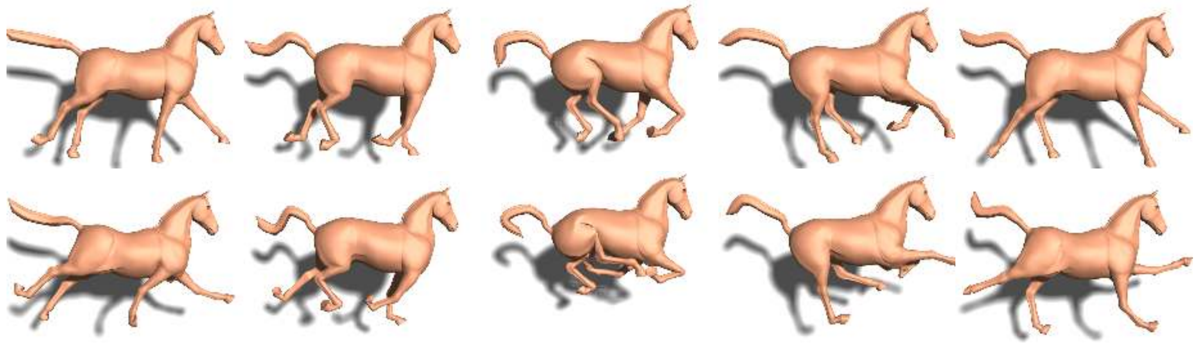


Fig. 8. Temporal signal processing. Enhancing one angular frequency band of the original animation (top) by a factor of five (bottom) produces exaggerated leg movement.

Figure 7 shows an example of inserting four keyframes into an existing cloth animation. Each of the new keyframes represents a significant bending of the cloth away from its initial configuration, which is shown in light grey. The keyframes were produced using our rotation-invariant surface editing technique (§5.1) and inserted into the animation using the method described above. Notice how the final animation passes exactly through the keyframes while preserving the basic structure of its original motion. This effect would be very difficult to achieve with a simple edit-propagation based approach [Kircher and Garland 2006].

4.4 Temporal Signal Processing

The blending operations we have defined (§4.1) also allow us to manipulate motion via temporal signal processing. Note that this is distinct from the temporally coherent *spatial* signal processing we presented previously [Kircher and Garland 2006], which provides no means of amplifying or dampening actual *motion* frequencies.

Consider the "triangle path" formed by the sequence of local trihedrons for a single triangle over time. It consists of two signals: the rotational component $\mathbf{q}$ and the tangential stretch/skew component $\mathbf{S}$. We use a standard pyramid scheme, where down-sampling and detail differencing are used to compute a time-domain multiresolution pyramid, and up-sampling with detail addition are used to reconstruct the signal-processed path. Each signal is processed separately for each triangle, and the resulting trihedrons allow us to reconstruct new vertex positions.

To down-sample a triangle path we pre-smooth—making each $\mathbf{q}$ and $\mathbf{S}$ the average of its time-domain neighbors and itself—and then discard every other frame. Up-sampling simply fills in the missing frames with the average of their two time-domain neighbors.

As usual, detail differences are computed between successive pyramid levels by up-sampling the coarser level and subtracting this smooth approximation from the finer level via our $\ominus$ operator. During reconstruction, details are added back via our $\oplus$ operator and user defined weights for each level. The detail differences between each level thus act like frequency bands, with coarse-scale details containing low-frequency motion and fine-scale details containing high-frequency motion. This is largely analogous to the skeletal motion signal processing methods proposed by Bruderlin and Williams [1995].

Figure 8 shows an example of enhancing one of the medium angular frequency bands of a galloping horse. As expected, this exaggerates the basic motion of the horse. Temporal signal processing can also be used to damp out certain motion frequencies (Figure 9b).

4.4.1 *Motion Detail Transfer.*

We can also use the signal pyramid to transfer motion frequency bands from one animation to another. We simply copy the desired motion detail band from the source animation and combine it with the corresponding detail band of the target animation using the $\oplus$ operator. While this requires that the two sequences have the same number of frames, animations can always be resampled to make the frame counts match if necessary. Figure 9 shows an example of transferring the motion of wind-driven ripples from one cloth animation to another animation. The anchor vertices are those points originally constrained by the cloth simulator to conform to the character. This helps prevent the close-fitting portions of the cloth from intersecting with the body during signal processing and motion transfer. We easily removed the remaining cloth/object intersections in the loose part of the dress via keyframe editing.

5. GEOMETRY PROCESSING

Although our primary purpose is to support motion processing, our differential representation also provides a convenient basis for manipulating the geometry of individual frames.

5.1 Rotation-Invariant Surface Editing

Prior work on differential surface representations has focused primarily on surface editing. Our representation can also be used for this task. Our method handles translation, rotation, shearing, and anisotropic stretching of the surface. All these edits can be performed interactively even on models with tens of thousands of triangles (i.e., Figures 1 and 7). Furthermore, keyframe splining (§4.3) and reconstruction of vertex positions takes place in the background, so the user does not need to wait for their changes to propagate before continuing with the editing of a particular frame.

Following the conventions used by others [Lipman et al. 2005; Yu et al. 2004], the user selects a *region of interest* (ROI) that will be deformed and a *handle* region that can be rotated, translated, scaled, or skewed. The ROI is deformed by solving the least squares reconstruction systems. The non-ROI regions provide the constraints necessary to guarantee uniqueness of the reconstruction. The linear systems are built from the original surface and are not changed during editing; only the constraints supplied by the user vary. The fact that connection maps are not changed is actually a powerful guard against unintended numerical drift.

It is often desirable to preserve surface area when performing rotational edits, especially when editing cloth. We can provide this option to the user by simply discarding interpolated stretch/skew components in favor of the original. This helps to counteract some of the skewing that can be introduced by our linear approximation of a rotational norm.

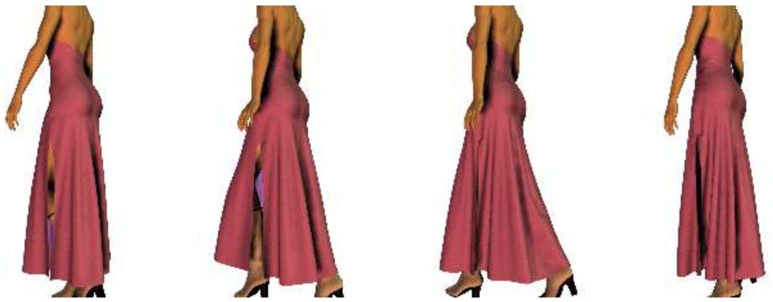
This editing method was used to specify all keyframe constraints used in this paper. Figure 10 shows some further examples that of detail-preserving edits made using our connection map representation.

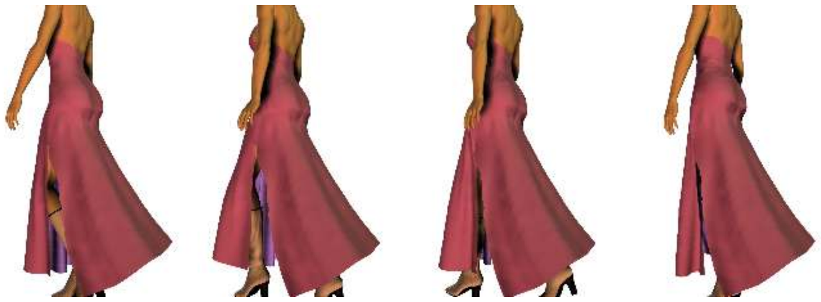
5.1.1 *Material Modulation.*

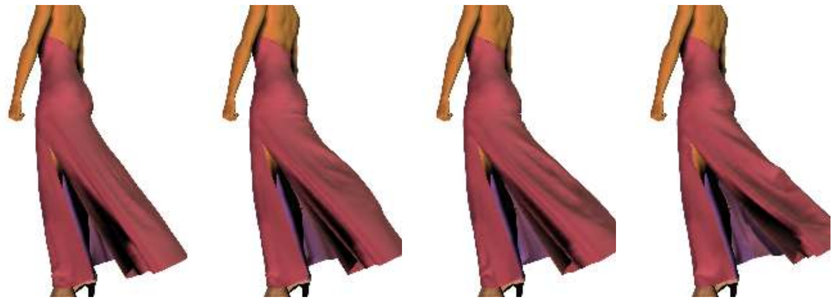
When editing naturally articulated models, such as a human or animal, a user intuitively expects the surface to bend primarily at joints. We can easily provide this behavior by adding per-edge weights to Equation (8) and per-triangle weights to Equation (2). Higher edge weights increase resistance to bending at that edge, and higher triangle weights increase resistance to stretching/skewing that element. We automatically compute weights from the original motion sequence following Popa et al. [2006], with every animation frame serving as an example pose. The effect of material modulation on the galloping horse is shown in Figure 11.

5.1.2 *Surface Smoothing.*

We can easily smooth a given shape by diffusing differences in extrinsic rotation (rotation of the normal) between neighboring triangles with Jacobi-like iterations and then reconstructing a smoothed surface with Eq. (6) By smoothing only the rotational components of the trihedron, and leaving the stretch/skew components unchanged, we can diffuse local variations in shape without parametric smoothing or significant surface area loss. This


(a) Cloth on walking character.


(b) High frequencies removed from back; lift added by keyframing.


(c) Cloth on character standing in strong wind.

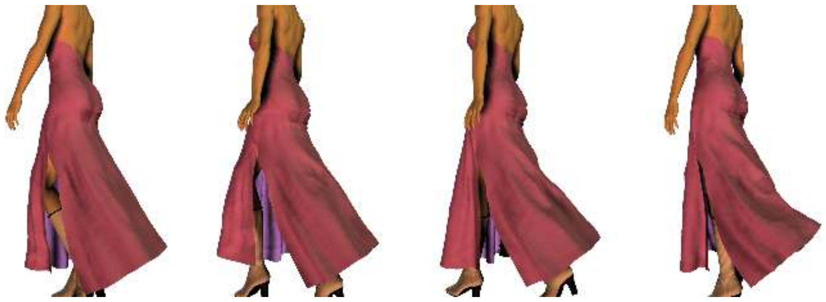

(d) High frequency motion from (c) transferred to (b).

Fig. 9. Temporal signal processing can be used not only to enhance or damp out motion (b), but also to transfer selected frequencies of motion from one animation to another (d).

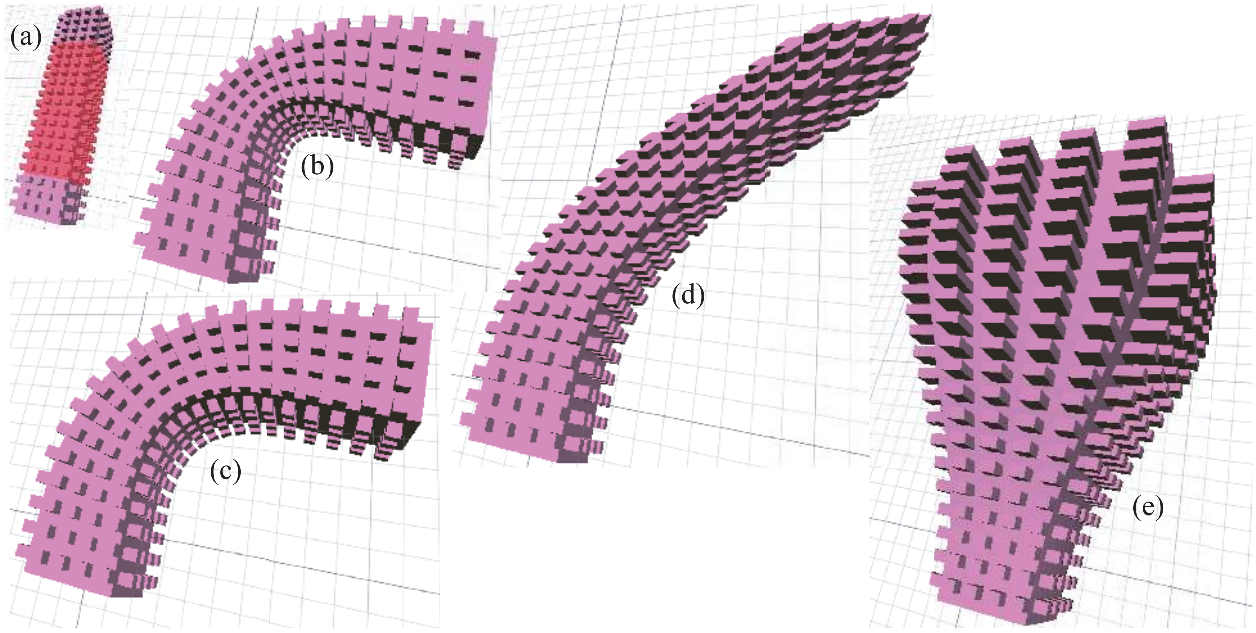


Fig. 10. Detail-preserving edits of a surface (a) with ROI shown in red. Our method supports (b) rotation, (c) rotation with area preservation, (d) shear, and (e) anisotropic scaling.

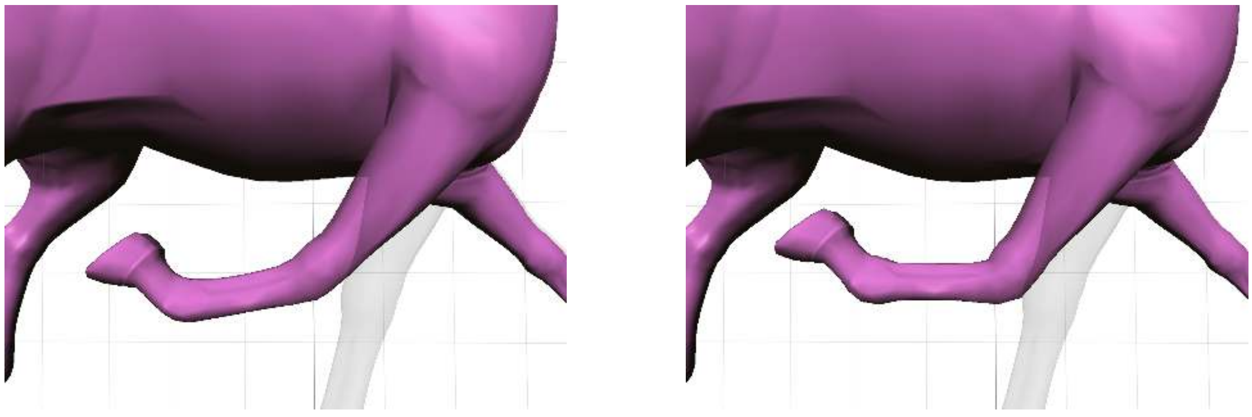


Fig. 11. Rotating a foot by 135°. Automatically inferred edge/triangle weights produce noticeably more natural behavior.

is closely related to the smoothing process described by Yu et al. [2004]. Figure 12 shows an example of smoothing the spiny box with 500 Jacobi iterations at a step size of 0.2. The shape is now very smooth, yet has lost almost none of its surface area.

5.1.3 *Fold Mollification.* The local trihedron based surface smoothing method is good at removing high-frequency details from a surface, but like all Jacobi iteration based methods it converges very slowly. Thus, it cannot be used for

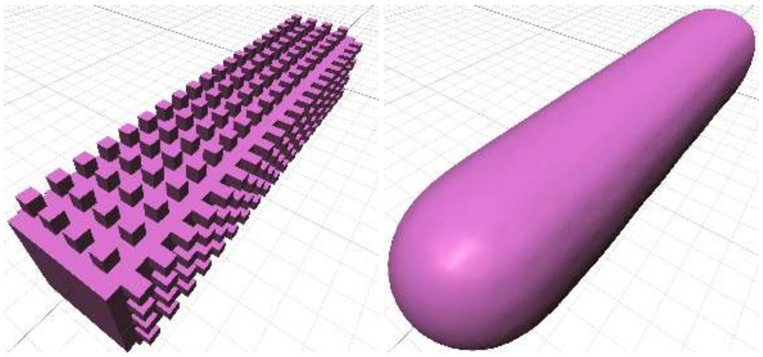


Fig. 12. The polar decomposition of local trihedrons allows us to smooth orientations while locally preserving surface area.

removing lower-frequency folds or tangles without an enormous number of iterations. However, using our connection map representation we can directly solve for a surface configuration that has all extrinsic rotation removed (or as close as possible in the least-squares Frobenius norm sense). This is done by removing all extrinsic rotation from the each $\mathbf{Q}_{\tau\sigma}$ and then solving Equation (6) for the new trihedron field. This method was used to untangle the cloth in Figure 1. Fold mollification works best on open, developable surfaces, where a flat configuration is possible. It tends to fail on closed surfaces, since the "flattened" connection maps describe an impossible configuration.

6. CONCLUSION

We have described a new approach to free-form motion processing that provides much greater power and flexibility than existing methods [Kircher and Garland 2006; Xu et al. 2007; Sumner et al. 2007]. To demonstrate its flexibility, we have shown that it makes motion blending, keyframe insertion, temporal signal processing, and motion transfer possible. Using the tools we have developed here, an artist can alter and combine existing deforming surface animations without having to re-simulate or re-capture them, thus speeding the iterative design cycle. Our framework also provides a conceptual basis for developing further motion processing tools. Our motion processing framework is based on a simple differential surface representation that is invariant under rotation and translation, does not fabricate tangent planes outside the surface itself, and which is capable of handling both non-manifold and non-orientable surfaces with ease.

There are a number of interesting ways in which this work could be extended. We have explored only a small sample of the possible motion processing tools that could be built upon our framework; there are certainly others worthy of attention. We have also made the convenient simplifying assumption that the mesh connectivity is fixed throughout time. This assumption could be relaxed by constructing inter-frame correspondences and providing appropriately generalized blending operators. We believe that good results can almost certainly be achieved without the need for bijective correspondences, using techniques like those proposed by Sumner and Popović [2004]. Also, since our system treats rotational and translation constraints separately, artifacts can occur from inappropriate use of constraints (i.e., attempting to use translational constraints to induce a rotation). It may be possible, and would be highly beneficial, to find a way to infer rotational constraints from a set of positional constraints. Finally, GPUs are currently powerful enough to support run-time solution of our least squares systems for modestly sized meshes, indicating that our approach could potentially be used for online motion blending.

Acknowledgements

We wish to thank the following people for generously providing input data for some of the examples in this paper: Robert Sumner (galloping horse and elephant poses) and Matthias Müller (the "heavy-cow").

REFERENCES

BOTSCH, M., SUMNER, R., PAULY, M., AND GROSS, M. 2006. Deformation transfer for detail-preserving surface editing. In *Vision, Modeling & Visualization*. 357–364.

BRUDERLIN, A. AND WILLIAMS, L. 1995. Motion signal processing. In *SIGGRAPH '95: ACM SIGGRAPH 1995 Papers*. ACM Press, New York, NY, USA, 97–104.

EBERLY, D. 2002. Quaternion algebra and calculus. *Geometric Tools*. Geometric Tools, Inc. 4 January 2007. `http://www.geometrictools.com/Documentation/Quaternions.pdf`.

GLEICHER, M. 2001. Comparing constraint-based motion editing methods. *Graph. Models 63,* 2, 107–134.

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

KIRCHER, S. AND GARLAND, M. 2006. Editing arbitrarily deforming surface animations. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*. ACM Press, New York, NY, USA, 1098–1107.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *SIGGRAPH '02: ACM SIGGRAPH 2002 Papers*. ACM Press, New York, NY, USA.

LAMOURET, A. AND VAN DE PANNE, M. 1996. Motion synthesis by example. In *Computer Animation and Simulation '96*. 199–212.

LI, X. S. 2005. An overview of superlu: Algorithms, implementation, and user interface. *ACM Trans. Math. Softw. 31,* 3, 302–325.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24,* 3, 479–487.

POPA, T., JULIUS, D., AND SHEFFER, A. 2006. Material-aware mesh deformations. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*. IEEE Computer Society, Washington, DC, USA, 22.

SHEFFER, A. AND KRAEVOY, V. 2004. Pyramid coordinates for morphing and deformation. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium on (3DPVT'04)*. IEEE Computer Society, Washington, DC, USA, 68–75.

SHOEMAKE, K. AND DUFF, T. 1992. Matrix animation and polar decomposition. In *Proceedings of the conference on Graphics interface '92*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 258–264.

SORKINE, O. 2006. Differential representations for mesh processing. *Computer Graphics Forum 25,* 4, 789–807.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM Press, New York, NY, USA, 175–184.

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

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph. 26,* 3, 80.

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Trans. Graph. 24,* 3, 488–495.

WITKIN, A. AND POPOVIĆ, Z. 1995. Motion warping. In *SIGGRAPH '95: ACM SIGGRAPH 1995 Papers*. ACM Press, New York, NY, USA, 105–108.

XU, D., ZHANG, H., WANG, Q., AND BAO, H. 2005. Poisson shape interpolation. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*. ACM Press, New York, NY, USA, 267–274.

XU, W., ZHOU, K., YU, Y., TAN, Q., PENG, Q., AND GUO, B. 2007. Gradient domain editing of deforming mesh sequences. *ACM Trans. Graph. 26,* 3, 84.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*. ACM Press, New York, NY, USA, 644–651.

Received Month Year; revised Month Year; accepted Month Year