Editing Arbitrarily Deforming Surface Animations

Scott Kircher* Michael Garland†

University of Illinois at Urbana–Champaign

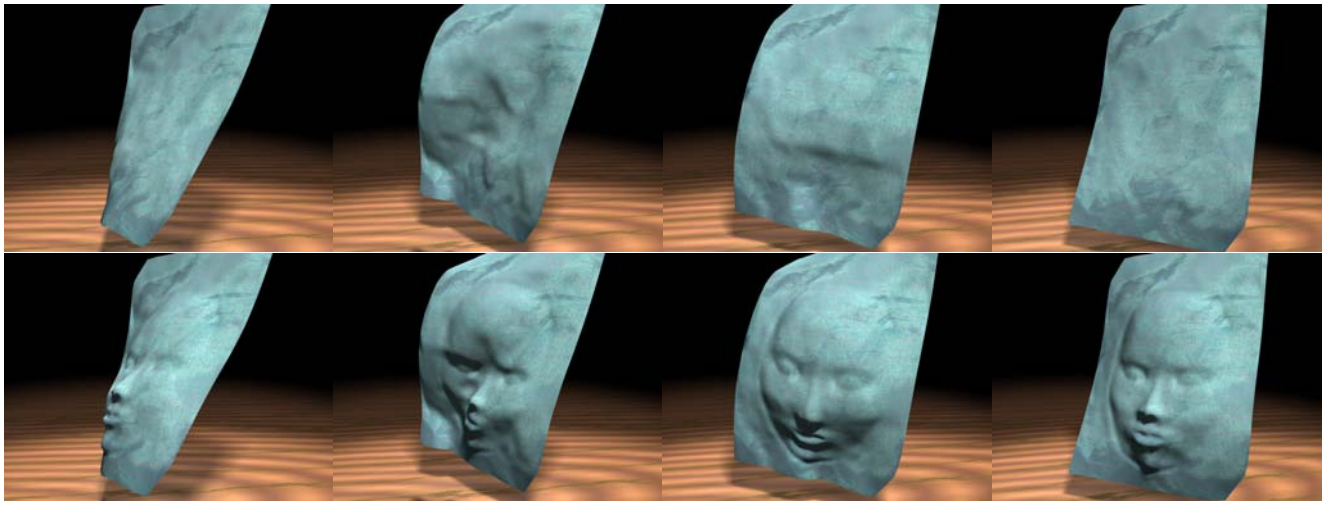


Figure 1: Taking a piece of deforming, motion-captured cloth (top), we can easily perform a broad range of edits, including multiresolution embossing of a person's face (bottom). Note how nicely the face bends with the deformations present in the original animation.

Abstract

Deforming surfaces, such as cloth, can be generated through physical simulation, morphing, and even video capture. Up to this point, such data has been very difficult to alter after the generation process is complete. Data generated for one purpose cannot, for the most part, be adapted to other uses. Such adaption, however, would be extremely useful. Being able to take cloth captured from a flapping flag and attach it to a character to make a cape, or enhance the wrinkles on a simulated garment, would greatly enhance the usability and re-usability of deforming surface data. In addition, it is often necessary to cleanup or "tweak" simulation results. Doing this by editing each frame individually is a very time consuming and tedious process. Extensive research has investigated how to edit and re-use skeletal motion capture data, but very little has addressed completely non-rigid deforming surfaces. We have developed a novel method that now makes it easy to edit such arbitrary deforming surfaces. Our system enables global signal processing, direct manipulation, multiresolution embossing, and constraint editing on arbitrarily deforming surfaces, such as simulated cloth, motion-captured cloth, morphs, and other animations. The foundation of our method is a novel time-varying multiresolution transform, which adapts to the changing geometry of the surface in a temporally coherent manner.

*e-mail: kircher@uiuc.edu

†e-mail: garland@uiuc.edu

1 Introduction

Movie and digital effects production involves heavy use of deforming surfaces. For example, the loose fitting portions of a character's garment are generally simulated cloth. Such physically simulated deforming surfaces are quite expensive to generate. Moreover, the design process of tweaking initial conditions to get the results "just right" is notoriously time consuming and labor intensive. This process would be greatly improved by a method for editing a deforming surface *after* it has been generated.

Recent advances in cloth motion capture [Guskov et al. 2003; Scholz et al. 2005; White et al. 2005] point toward a future where cloth motion data is routinely captured to supplement simulations. Human skeletal motion capture has become ubiquitous in production environments. Being able to alter and re-use such motion capture data is an important and interesting problem that has become the focus of a great deal of research. As with skeletal motion capture data, there will be a pressing need for good ways to edit and "tweak" cloth motion capture data to adapt it to different purposes. For example, cloth that was captured flapping in the breeze could be adapted to look as if it was hit by something at a certain point in time, even though the original data contained no such event. Or perhaps the captured cloth data has the desired general motion, but not quite the right material. Filtering out or enhancing certain geometric frequencies can alter the appearance and "feel" of cloth in subtle ways, making the data more versatile. Again, this is a situation where pre-deforming surface editing and signal processing would be extremely useful.

Editing of arbitrarily deforming surfaces is a largely un-addressed problem, but one which is likely to become very important as simulation and motion capture technology advances. Methods developed for other purposes can sometimes be used to perform a limited set of editing operations on deforming surface data. However, to the best of our knowledge, our work is the first to provide a unified

framework for a very general set of editing operations.

Our goal is to provide a system allowing animators to easily edit and manipulate deforming surface data. The deforming surfaces we consider are represented by a sequence of triangle meshes with static connectivity, where the surface motion is specified by a complete set of new vertex positions for each frame. The mesh connectivity should be manifold or manifold with boundaries.

To achieve our goal, we have developed a system that can perform global geometric signal processing, direct manipulation of the surface, multiresolution embossing, and constraint-based editing on an arbitrarily deforming mesh. Our approach is entirely geometric. We do not consider the dynamics of cloth, or the material properties of the original surface. This makes our technique versatile enough to be applied to many different kinds of data, such as morphs and other animations.

We present a multiresolution editing and geometric signal approach for deforming surfaces that is based, in part, on techniques that have been shown to work well in the static case [Zorin et al. 1997; Guskov et al. 1999]. This multiresolution approach not only supports signal processing, but also yields more intuitive results when transporting edits to other frames of an animation. Underlying our system is a time-varying multiresolution transform that, through a technique we call *basis smoothing*, yields temporally coherent signal processing and editing results even in the presence of a changing multiresolution hierarchy. Other contributions of our paper include a *vertex teleportation* extension to Kircher & Garland's hierarchy improvement algorithm [2005] and an improved irregular mesh wavelet compression scheme, using our time-varying transform, based on Guskov & Khodakovsky's wavelet compression method [2004].

2 Related Work

Dynamic Surface Editing. One of the few existing methods that can be used to edit an already deforming surface, with no pre-existing controls, is due to James & Twigg [2005]. They fit a set of nonrigid bones to an existing deforming surface, thereby encoding the animation as a static mesh and a sequence of bone transformations. In addition, they encode deviations of the true surface from the skinned result. This allows limited editing, either by modifying the bone transformations, or by adding rest-pose displacement edits. However, their focus was not on editing, but rather on making deforming surface animations bandwidth friendly for fast GPU rendering, and their method assumes the animation consists mainly of articulated motion. As they point out, this makes their method less suitable for cloth and other non-articulated deformations.

Sumner *et al.* [2005] have developed a mesh-based inverse kinematics method that could be extended to provide limited editing of an already deforming surface. By placing time-varying constraints, the user could produce novel animations from existing animations. However, this method provides no facility for more detailed edits or applying new deformations to the object that are completely unlike any in the original data.

Motion Capture Editing. A great deal of research has investigated how to edit and reuse human skeletal motion capture data. A few examples include path editing [Gleicher 2001], motion signal processing in the time domain [Bruderlin and Williams 1995], and constraint based editing [Gleicher 1997]. While enlightening as to the importance of the problem we address, these methods all rely on the animation data being skeletal, and thus are not applicable to arbitrary deforming surface editing.

Static Multiresolution Editing. Zorin *et al.* [1997] developed the first multiresolution editing method for static surfaces, based on Loop subdivision [1987]. This allowed large-scale edits to be performed while preserving fine details. Subdivision based analysis, however, is limited to meshes with semi-regular connectivity.

In contrast to subdivision schemes, mesh decimation methods start with an irregular fine mesh and produce coarser approximating meshes, all with irregular connectivity. Modern mesh decimation methods are predominantly based on iterative edge contraction. Examples include the methods developed by Hoppe [1996] and Garland & Heckbert [1997]. Kobbelt *et al.* [1998] presented a multiresolution editing and signal processing method for irregular static meshes, using mesh decimation to build the multiresolution pyramid. Guskov *et al.*[1999] improved upon this, with the definition of the 2nd-order divided differences relaxation operator for arbitrary meshes. However, since the pyramid construction in both these methods depends heavily on the geometry, it is suitable only for analyzing similar geometry (i.e., nearby frames in an animation).

Dynamic Multiresolution Meshes. To produce a sequence of mesh pyramids for a deforming surface, one could simply run a standard mesh decimation algorithm on each frame. However, a sequence of hierarchies produced in this manner would have little or no temporal coherence. Shamir *et al.*[2000] address this problem by taking history into account when decimating the next frame in an animation. On the other hand, Kircher & Garland [2005] compute the transformation of the previous frame's hierarchy to one better suited to the current geometry. This has the added benefit of yielding a compact representation of the difference between the two hierarchies. One shortcoming of this method is that it can fairly easily get stuck in suboptimal local minima. We rectify this problem by adding vertex teleportation.

Mesh Animation Compression. Sequences of animated mesh geometry can be potentially extremely costly to store. Lengyel [1999] first addressed this problem by fitting the animation to a low-parameter motion model, and then encoding the residual. Alexa and Müller [2000] compress the difference from the mean shape using PCA techniques. Briceño *et al.*[2003] encode the geometry as an image, and then apply video compression. Guskov and Khodakovsky [2004] perform a wavelet transform, based on a multiresolution mesh pyramid. Our time-varying transform can be used with this method to improve its performance on animations with more extreme deformations and less parametric coherence.

3 Time-Varying Transform

Utilizing a multiresolution approach to editing enables important signal processing operations, such as smoothing of motion capture noise and the enhancement of subtle details. In addition, it makes edits applied on multiple animation frames behave sensibly in the presence of highly deforming geometry (see §4.2). Our approach is based on an irregular mesh analog of the Burt-Adelson pyramid [1983]. Our *mesh pyramid* will represent the mesh at decreasing levels of detail, and we will encode the detail differences between levels.

Figure 2: The vertex $r(v)$ represents v in the next coarser level and has the same geometric position as $r^*(r(v))$.

3.1 Pyramid Scheme

We construct our mesh pyramid in a manner similar to that proposed by Guskov *et al.* [1999]. However, our pyramid is different in that each removed vertex does not count as a new level. Instead, we repeatedly remove a fixed percentage (usually 50% or 75%) of vertices, to produce a pyramid structure stratified into levels. This is the *multilevel mesh* defined by Kircher & Garland [2005]. The decimation metric we use is a weighted average of the planar quadric error metric [Garland and Heckbert 1997] and vertex quadrics [Garland and Zhou 2005] to regularize edge lengths (important for achieving meaningful signal processing results). We use subset placement to find decimated vertex positions. Each level is a mesh in its own right, and can be processed as such. The levels will be denoted $M_0, M_1, ..., M_n$, where M_0 is the original input mesh, and M_n is the coarsest level constructed. Each vertex $u \in M_{k+1}$ was produced by contracting together some number of vertices at level M_k. For each such vertex v, we call u the *representative* of v, and denote it $u = r(v)$. In addition, there is one vertex $w \in M_k$ that is really the same vertex as u. We shall denote this vertex $w = r^*(u)$. Note that for a general vertex x, $r(r^*(x)) = x$, but $r^*(r(x)) \neq x$ necessarily. In fact, for each cluster of vertices with the same representative, exactly one will have the property that $r^*(r(x)) = x$. See Figure 2.

Up-sampling proceeds in a manner reminiscent of the interpolate and relax phase of a multigrid solver. To produce a smooth *subdivided* mesh from a coarse level M_k, we take the connectivity specified by level M_{k-1} and set each vertex $v \in M_{k-1}$ to the position of $r(v)$ in M_k. We then generate interpolated positions for the vertices of M_{k-1} by performing a fixed number of Gauss-Seidel iterations using the 2nd-order divided differences relaxation operator [Guskov et al. 1999], while holding fixed every vertex $x \in M_{k-1}$ satisfying $r^*(r(x)) = x$. Following that, we perform a small number of unconstrained Jacobi iterations, with the same relaxation operator, to relax the mesh into a smooth approximation of M_k, with the connectivity of M_{k-1}. We shall denote the subdivision of M_k by this process as $\sigma(M_k)$. We use Gauss-Seidel iterations during interpolation for faster convergence, and Jacobi iterations during smoothing to avoid bias. In our experiments, we have found 15 Gauss-Seidel iterations with a step size of 1.0 to be sufficient for the interpolation phase. The user can vary the number of Jacobi iterations for the smoothing phase, but we generally use 2, with a step size of 0.5, which we have found to work best in our experiments.

Having computed $\sigma(M_k)$, we compute *detail vectors* by taking the difference between each vertex in M_{k-1} and the corresponding vertex in $\sigma(M_k)$. The detail vectors are represented relative to a local frame computed for each vertex of $\sigma(M_k)$. These local frames are very important in our editing application, and we will denote the local frame of a vertex u by $\mathbf{F}_u$, a 3x3 rotation matrix taking local vectors to global ones. Local frames for M_{k-1} are computed from $\sigma(M_k)$. Also, for editing applications we need to keep track of, at all hierarchy levels, the local frames of the unedited geometry. This *canonical frame* for vertex u will be denoted $\mathbf{F}^c_u$.

The original surface can be reconstructed from the coarsest level vertex positions and all the detail vectors by again using the mesh pyramid. Starting from M_n, we produce $\sigma(M_n)$ and add the detail vectors for level $n-1$, to obtain M_{n-1}. This process repeats until we obtain M_0.

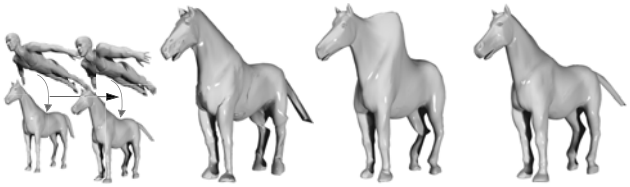


Figure 3: (a) A nonlinear morph. (b)-(d) Middle frequencies enhanced. Multiresolution analysis from first frame breaks down on later frames (b). Even updating relaxation coefficients (c) doesn't fix the problem. (d) Our result.

To deal with boundaries, we add boundary constraint quadrics [Garland and Heckbert 1997] to the decimation metric, ensuring that boundaries are well represented on all levels of the mesh pyramid. Then, during the interpolate phase of subdivision, we use an inverse edge-length weighted Laplacian operator, restricted to the boundary, to make sure that interpolated vertex positions lie on the boundary. The smoothing phase is unaltered. We have also found that triangles with very bad aspect ratio may appear on mesh boundaries, causing numerical instabilities. To fix this, we simply ignore edges attached to such triangles when computing relaxation coefficients, which is the same method Guskov *et al.* used to deal with non-manifold edges [1999].

It is worth noting that, as stated, this pyramid scheme is not a true wavelet transform, due to the presence of oversampling. The oversampling is important for editing and signal processing applications, but if desired, it can be eliminated easily by not performing any smoothing iterations, and then not computing detail vectors for the vertices satisfying $r^*(r(x)) = x$ (since those detail vectors would be zero).

3.2 Adaptive Transform

The mesh pyramid described above can be used for signal processing and multiresolution editing on the original input mesh, just as in Guskov *et al.*[1999]. However, our goal is to perform such operations on a highly deforming surface, such as a cloth animation. Since the pyramid was constructed from a particular input geometry, it is useful only for signal processing of surfaces whose geometry closely resembles that of the original input. An example of how this can fail is shown on a man to horse morph in Figure 3. In that figure, we are doubling the length of the detail vectors corresponding to the middle level of the hierarchy. However, the frequency content of the man is very different from that of the horse. A static frequency analysis (Figure 3b) thus yields obviously incorrect results. Even updating the relaxation coefficients alone (Figure 3c) does not give the right behavior. Only through updating both the relaxation coefficients and the hierarchy (Figure 3d) do we achieve the desired result. Note that this situation doesn't arise in the regular image setting, because the signal being processed (pixel value) does not affect the construction of the image pyramid. In the irregular mesh setting, however, the geometry of the surface has a heavy influence on the constructed mesh pyramid.

To apply multiresolution techniques to a deforming surface we can

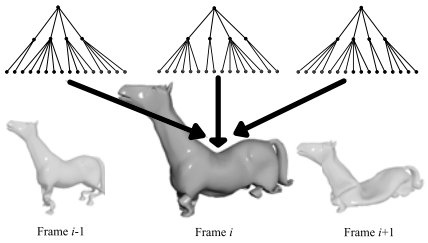


Figure 4: Basis smoothing processes a single frame i with all nearby mesh pyramids, the results are then combined to produce temporally coherent output.

simply build a new hierarchy for each frame. However, since mesh decimation is a chaotic process, changing the input geometry slightly produces a potentially drastic change in the output hierarchy. Moreover, a pyramid scheme is only an approximation of frequency decomposition, so it is important to maintain consistency of the approximation from one frame to the next. Thus, we make use of the hierarchy adaptation method presented by Kircher and Garland [2005]. We have carefully constructed our mesh pyramid scheme to be compatible with this method. This yields a sequence of multilevel meshes with a high degree of temporal coherence. If, instead, we were to simply build a completely new pyramid on each frame, without regard to previous frames, the signal processing and editing results would be extremely noisy.

Since the relaxation operator we use performs better when the surface is manifold, we utilize topology preservation rules both during initial pyramid construction and during subsequent hierarchy updating.

3.3 Basis Smoothing

Performing hierarchy adaptation, rather than building a new hierarchy from scratch, greatly increases the temporal coherence of the mesh pyramid sequence. However, any change to the hierarchy has the potential to cause a pop in the signal processed results. This is because changes to the hierarchy are combinatorial in nature. For example, an edge flip in one of the coarser levels may potentially cause an abrupt change in the relaxed position of either involved vertex. Such changes in the relaxed vertex positions cause corresponding changes in the direction and magnitude of the detail vectors. If one is reconstructing the original surface, then such changes are immaterial. However, when performing signal processing, we may scale the detail vectors, and if their directions have changed, their endpoints will shift. Visually, this manifests itself as a small "pop" in the surface.

To alleviate these abrupt position corrections, we would like to "smooth out" the hierarchy changes, so that they occur over some time interval, rather than suddenly. However, since the changes to the hierarchy are combinatorial, this is problematic. Our solution is essentially to let the various mesh pyramids nearby the current frame "vote" on the result of a particular signal processing operation. When processing frame i, we take all the pyramids from frames j within a certain window around frame i, and signal process the geometry of frame i with each of them. We then average the resulting vertex positions (See Figure 4). It is vital to note, however, that we are *not* averaging the result of applying pyramid j to geometry j, for $j \neq i$, and averaging with the other frames in the window. Doing that would smooth out the original geometry motion, which is quite undesirable. Instead, we are only diffusing the effect of hierarchy changes, on geometry from a *single* frame i. In essence, the space we are smoothing over is the space of possible mesh pyramids (hence the name *basis smoothing*). We are *not* smoothing over time. So, if you were to apply basis smoothing on a sequence where your signal processing filter was the identity function, you would get back exactly the original sequence (up to numerical noise).

There is, of course, a performance impact from using basis smoothing. If you have to combine 10 different bases, the transform will run approximately 10 times slower. However, we greatly improve the speed of basis smoothing by breaking the sequence into blocks (§3.4). In addition, during interactive editing, basis smoothing is not performed, to give the user the quickest feedback. The user can request a basis smoothed preview at any time.

3.4 Blockification

Although we can compute a new multiresolution pyramid for each frame of the animation, this is by no means necessary. For smoothly deforming meshes, the same pyramid is likely to be well suited for an entire contiguous subsequence of the animation. Thus, we can delay changes to the hierarchy until they exceed some threshold. This breaks the sequence into blocks, each of which is represented by a different mesh pyramid. We pick blocks based on the total approximation error of the hierarchy. If the improved hierarchy is better suited to the geometry than the current one by more than some small threshold we create a new block.

Blockification greatly accelerates basis smoothing, since smoothing need only be performed near block boundaries. For example, on the collapsing horse sequence (Figure 10), blockified basis smoothing with a window consisting of 19 frames was performed using an average of only 3.5 bases. On sequences with less drastic deformation, the speedup is even greater. Blockified basis smoothing with a 19 frame window on the motion captured cloth sequence (Figure 1) used only 1.7 bases per frame (more than 11 times faster than regular basis smoothing). We typically use a threshold of nearly zero (1×10^{-6}), and block sizes are typically on the order of 10 frames per block.

3.5 Vertex Teleportation

Kircher & Garland's hierarchy improvement algorithm is useful for generating a sequence of mesh pyramids, as well as for encoding the transformation from one hierarchy to the next. However, using only local changes (i.e., their *swaps*), without any backtracking (as would be present in a full Kernighan-Lin algorithm [1970]), means that the algorithm can fairly easily get stuck in local minima.

We solve this by introducing a vertex teleportation operation, which is built from a special sequence of swap operations (each swap in the special sequence may not decrease the total error, but the sequence as a whole will). Essentially, we merge two clusters together into one, and simultaneously split a single cluster somewhere else into two clusters. Thus the total number of clusters (i.e., *approximation vertices*) remains the same. The effect is the same as removing one vertex in the approximation, at one location, and introducing a new vertex somewhere else in the approximation. This is essentially the same strategy that has successfully been used to improve other iterative minimization schemes [Cohen-Steiner et al. 2004] To reduce the number of splits that must be considered, we examine only clusters with *pinch vertices* in them [Kircher and Garland 2005]. To preserve topology during vertex teleportation, we use

both the homeomorphic swap validity rule [Kircher and Garland 2005] and the link condition for edge contraction [Dey et al. 1998].

4 Multiresolution Editing

Constructing the time-varying multiresolution transform for a given animation sequence is an entirely automatic preprocessing step. Once that is complete, the user can efficiently perform interactive signal processing and editing of that sequence. See Figure 5 for examples of interactive editing sessions. We view both signal processing and editing as the application of *filters* to the detail vector representation of the surface. User edits are stored in a special filter, which can be applied to any frame of the animation to produce a corresponding edit. Edits can also be weighted by a *temporal scaling function*, which is an arbitrary real valued function of time that governs the strength of an edit on each frame of the animation.

4.1 Direct Manipulation

The most basic edit that can be performed with our system is dragging a vertex to a new position. Naturally, coarser level vertices effect larger areas. An example of direct manipulation is shown in the top left portion of Figure 5. The user lengthens the ear of the collapsing horse by dragging it.

A common approach in static multiresolution editing is to associate edits with some coarse scale vertex. However, in our case the mesh pyramid may change from frame to frame. Thus, coarse-scale vertex position edits cannot be associated with a coarse vertex, since there is no single corresponding coarse vertex for all frames. Instead, all edits that would normally affect the detail vectors of coarse level vertices are stored at the finest level (whose connectivity does not change).

When the user moves a vertex u at some level k of the current hierarchy, the edit vector is "replicated" and associated with all vertices of M_0 that are represented by u. When the resulting set of *edit replicators* are applied to some other hierarchy, they are propagated up the hierarchy to level k and averaged together. By the nature of the multiresolution transform, this gives us how the coarse vertices should move. Since the hierarchy is not necessarily the same as the original, more than one vertex of level k may now be affected by the edit. See Figure 6 for a schematic depiction of how edit replicators work.

We take care to make edit replicators from different edits be independent of each other. This makes applying the edit filter simpler, since edits need not be applied in the order the user specified them. It also allows edits from different frames to be combined in a sensible way. Given an edit vector e represented in global coordinates and associated with a vertex $u \in M_k$, replication proceeds as follows. Let $e' = \mathbf{F}_u^c \mathbf{F}_u^{-1} e$ be the global representation of e independent of all other edits. Now, let $V = (r^{-1})^k(\{u\})$ be the set of vertices in M_0 represented by u. For each $v \in V$, let $e_v = (\mathbf{F}_v^c)^{-1} e'$. This set $\{e_v | v \in V\}$ is the set of edit replicators associated with this edit, represented in the local canonical frames of the finest level geometry. These edit replicators, along with the level number k are sufficient to reconstruct the edit, and apply the edit to other frames, even if they have different hierarchies.

Given a set of base vertices V, a set of edit replicators $\{e_v | v \in V\}$, and a level number k, edit application proceeds as follows. Let $U = r^k(V)$ be the set of vertices at level k representing those in V. For each $u \in U$, let $W = (r^{-1})^k(\{u\})$ and compute the global average edit vector $a = (\sum_{w \in W \cap V} \mathbf{F}_w^c e_w)/|W|$. Finally, for each $u \in U$, add $(\mathbf{F}_u^c)^{-1} a$ to the detail vector associated with u.

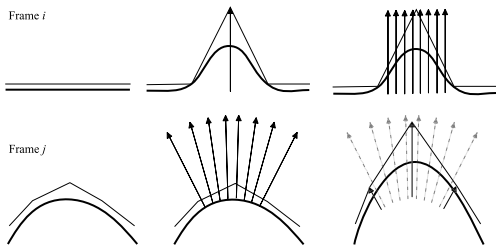


Figure 6: Thin polyline represents coarse surface manipulated by user. Thick curved line represents finest level surface. Top, left to right: Original surface. User drags a vertex. Edit is replicated and stored at finest level. Bottom, left to right: Original surface on a new frame. Edit replicators move with finest level surface. Vertices at coarse level are moved according to edit replicators.

As described, different edits have *cumulative semantics* (meaning that, if they affect the same vertex, their effects add up). We can also make edits from the same animation frame have *averaging semantics* by treating all the edit replicators affecting level k of the same animation frame as one big set of edit replicators. Thus, if two spatially nearby edits originally affected different coarse vertices, but on a new animation frame they affect the same vertex, then the total effect on that vertex will be a weighted average of the two edits. We have found that this gives more intuitive user control. Edits from different frames still have cumulative semantics.

Edit replicators are versatile, and need not be used only for transporting single-vertex style edits. By specifying the edit replicators more directly, we can provide a variety of useful tools. For example, we can provide a direct manipulation *brush* by laying a brush texture over a portion of the base surface, and computing edit replicators for each vertex of M_0 that lies within the textured region. The edit replicator magnitudes are determined by texture lookup. Then, the edit replicators can be applied to any level desired. This allows the user to "pull" multiple vertices at once.

4.2 Multiresolution Embossing

Our edit replicators can also be used to perform multiresolution embossing, which is much more effective than single resolution embossing. This is a multi-stage process. First, edit replicators are computed exactly as in the direct manipulation brush case, and are applied to a very coarse level of the mesh pyramid. The whole multiresolution transform is then computed, to determine the affect of the low-res embossing on the finest level geometry. Then, this new geometry is compared against the desired embossing shape, and difference vectors are computed. These are converted to edit replicators, and applied to the next finer level of the mesh pyramid. Again the transform is computed to obtain the refined finest level geometry. This process repeats until the finest level is reached. The result is a multiresolution set of edit replicators that, when applied to any frame of the animation, produces a nice embossing of the surface, deformed appropriately for the new geometry. Embossing effects can be scaled or thresholded by temporal scaling functions, and can even have an animated source texture.

Figure 7 shows the difference between multiresolution and single resolution embossing. In that figure, a piece of motion-captured cloth data has been embossed with the word "HI" while the cloth is

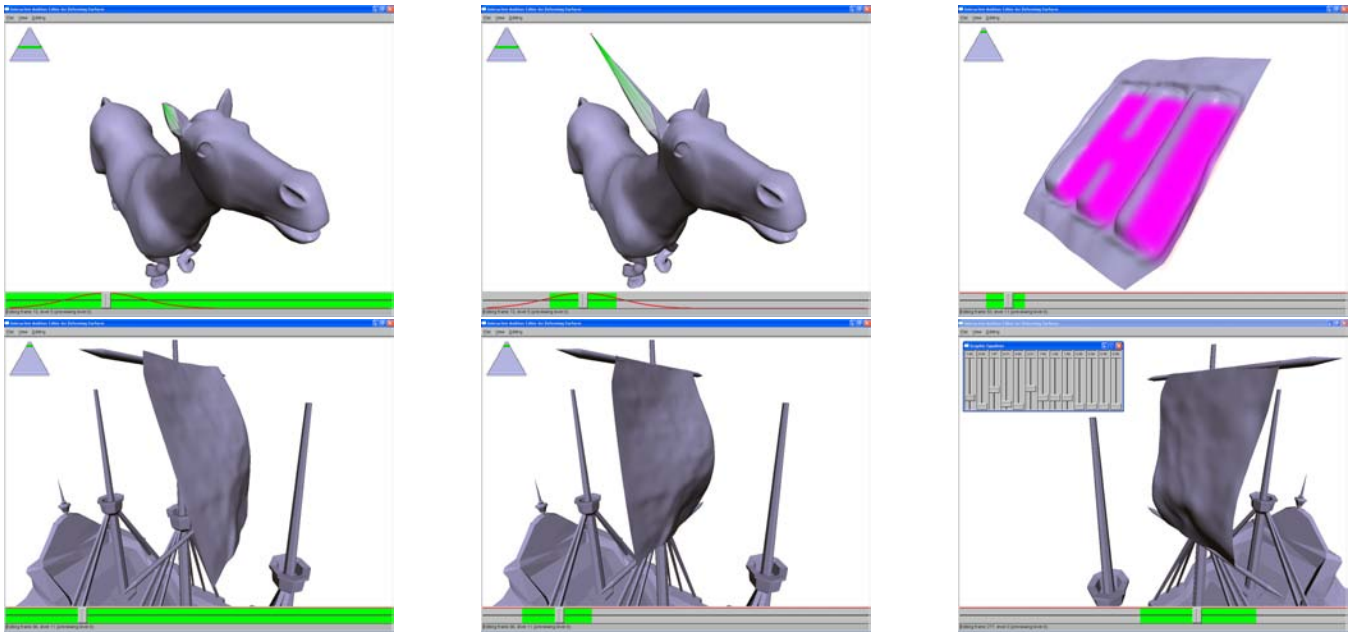


Figure 5: Our editing interface. Top: Direct manipulation (left, middle) and multiresolution embossing (right). Bottom: Attaching a sail to a ship (left, middle) and signal processing (right).

flat. When the cloth bends, the single resolution embossing (applied at the finest level) breaks down, whereas the multiresolution result behaves in an aesthetically pleasing manner.

4.3 Constraints

Constraint edits allow the user to specify that a particular part of the surface should be in exactly a certain location at a certain animation frame (or multiple frames). This is useful for fixing up "constraint skate" in motion captured cloth (similar to the "foot skate" problem of skeletal motion capture), or attaching cloth to objects it wasn't attached to in the original simulation or motion capture environment.

Constraints are represented in global absolute coordinates. Each constraint c is associated with a single vertex v of the finest level mesh, and can be either hard (vertex moves exactly to the specified location) or soft (vertex is a weighted average of constrained and unconstrained locations). Also associated with c is the coarsest level k at which the constraint will have any affect. All vertices up the hierarchy from v to $r^k(v)$, inclusive, are affected by c. If two or more constraints end up affecting the same coarse level vertex, their affects are averaged at that level. Choosing different k will change how localized the effect of the constraint is. Choosing the coarsest level will give the most "global" motion of the surface to satisfy the constraint. Choosing the finest level will move only v to the constraint location, creating a spike effect (see top of Figure 8).

Unlike global signal processing and edit replicators, constraints do not affect detail vectors directly. Instead, they are applied during the reconstruction phase. A hard constraint affecting vertex u of level k will cause that vertex to be moved exactly to the constraint position, ignoring the detail vector associated with that vertex. Soft constraints are the same, except they first compute the unconstrained location, using the detail vector, and then combine it with the constraint location.

Figure 8 shows an example of using constraints to modify a motion captured cloth sequence. In the original sequence, only the top two corners were pinned, allowing the bottom to flap freely. In our edited result, all four corners are pinned, yet the surface still looks natural.

4.4 Geometric Signal Processing

Motion captured cloth can have a great deal of noise, which may need to be removed. In addition, an artist may wish to enhance, or downplay, certain folds and wrinkles in the cloth. We provide signal processing tools as a way to accomplish these tasks. We apply frequency band filters, where each mesh pyramid level is a separate band, by scaling the detail vectors and then reconstructing the surface. As with all pyramid methods, the levels of the hierarchy are not exactly frequency bands, but they behave in a similar manner. By scaling the detail vectors, we can achieve all the usual signal processing effects, such as smoothing, band-pass filtering, and enhancement.

5 Adaptive Wavelet Compression

It may seem that constructing our time-varying multiresolution transform will seriously inflate the size of the animation data. However, with a few modifications, the transform can actually help us compress the animation.

The wavelet based compression method due to Guskov and Khodakovsky [2004] requires that every frame of the animation sequence be well represented by a single multiresolution hierarchy. For highly deforming surfaces, such as morphs and cloth, this requirement is not met. By using our time-varying transform, we can achieve better normal reconstruction at low bit-rates, even though the total geometric RMS error does not necessarily decrease. See Figure 9 for an example.

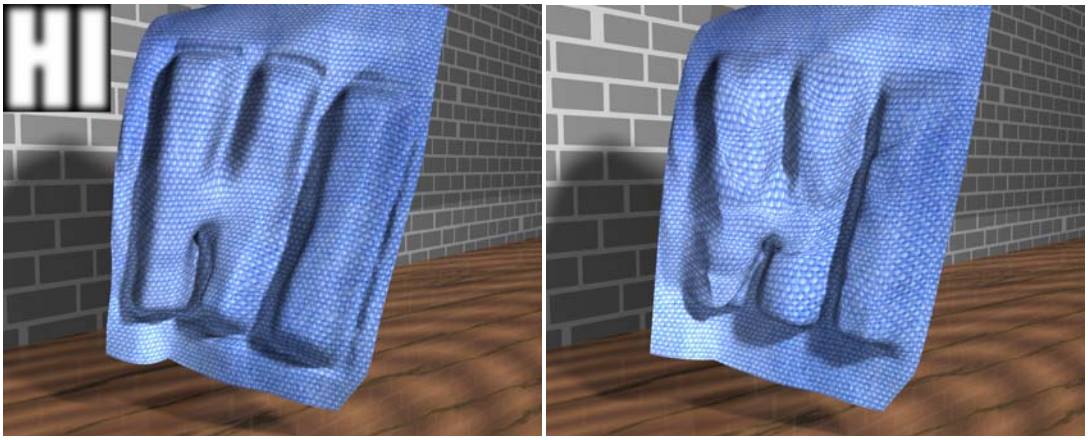


Figure 7: Embossing motion captured cloth. Notice that the multiresolution result (left) is free from the "crumpling" artifacts present in the single resolution result (right).

First, we segment the sequence into blocks (§3.4). Each block will have its own wavelet basis, which requires both a multiresolution hierarchy and a set of relaxation operator coefficients. The hierarchy and relaxer coefficients for the first block are both implicitly defined by the geometry of the first frame, which can be compressed by a static mesh compression scheme, or left uncompressed.

Hierarchies for subsequent blocks are encoded using a more compact version of the swap representation described in [Kircher and Garland 2005]. The relaxation operator coefficients for each block will be computed from the geometry of the first frame of the block. However, to avoid quality degradation, the first frame of each block needs to be encoded at a much higher bitrate (as much as an order of magnitude higher, for very low bitrate encodings) than the rest of the block.

In order to have a true wavelet transform, there must be no oversampling. Thus when using our transform for compression we do not perform the smoothing phase of the subdivision (only the interpolation phase), as described in §3.1. Also, to achieve better rates, we use the anisotropic version of the 2nd-order divided differences relaxation operator [Guskov and Khodakovsky 2004].

Figure 9 shows a comparison between our implementation of Guskov & Khodakovsky's wavelet compression method and our improved version using adaptive bases with blockification. The sequence being compressed is the man to horse morph shown in Figure 3. Both methods use an average of only 1 bit per vertex per frame (decreasing the animation from 41MB to 1.2MB), and use P-frames for all possible frames (the beginning of blocks in our adaptive method must be I-frames). The single, non-adaptive wavelet basis is not as suitable for representing the last frame, resulting in noticeable noise in the reconstruction. Our result is of higher visual quality, despite having the detail vectors actually encoded at only 0.7 bits per vertex per frame, to compensate for the extra swap information and geometry that must be encoded for the basis transitions. The overall RMS errors for the two methods are essentially the same.

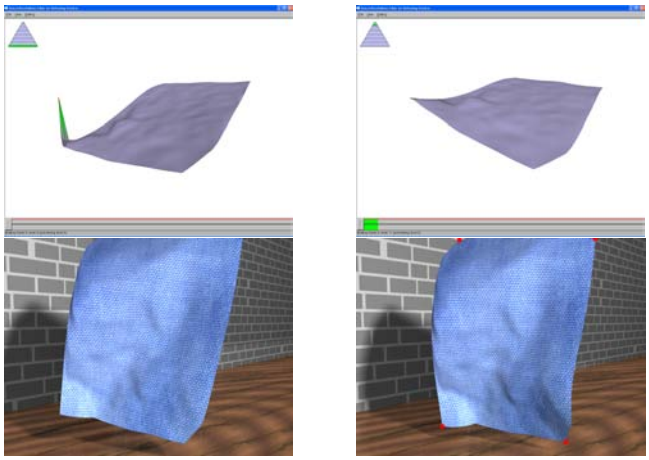
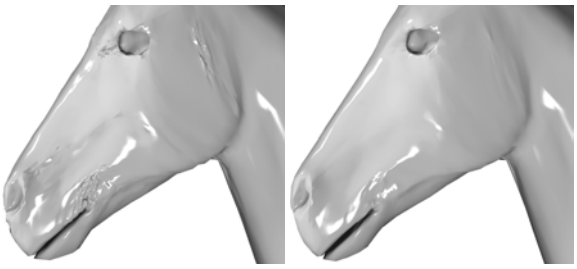


Figure 8: Top: Constraints at fine levels (left) have very local effects. Higher level constraints (right) have more global effects. Bottom: Before (a) and after (b) adding constraints at all four cloth corners.



(a) Single wavelet basis (b) Adaptive wavelet bases

Figure 9: Compressing the man-to-horse morph shown in Figure 3 at an average rate of 1.0 bits per vertex per frame. (a) Static wavelet basis. Note the noise on the side of the head. (b) Adaptive wavelet basis.

6 Results

We will now demonstrate some of the results that can be achieved with our dynamic surface editing system. We begin with a few embossing examples on a piece of cloth obtained through motion

capture and then subdivided with two levels of Loop subdivision. Figure 7 shows both multiresolution and single resolution embossing of the word "HI" into the cloth. Figure 1 shows a multiresolution embossing of a human face into the cloth. Note how the embossed result bends naturally with the bending of the cloth. These examples highlight a useful aspect of dynamic surface editing: the ability to create non-physical cloth results, from physical data or physically-based simulations, that still look natural. Moreover, these simple results can be created with only a few mouse clicks on the part of the user.

A more complicated edit is shown in Figure 10. The original sequence is of a horse collapsing as if it were made of rubber. In just a few minutes, the user has edited the sequence so that the horse becomes a camel-horse hybrid, complete with hump, and so that its face shows an expression of extreme surprise when it realizes it has no skeleton. Notice the natural way that the hump folds over when it collapses. Remember, the hump was never part of the geometry during the original simulation run. Preprocessing time for the 16K triangle, 52 frame collapsing horse was about 8 minutes on a 3Ghz Intel Xeon processor.

Figure 11 shows another complex example. We have taken the subdivided, motion captured cloth data (Figure 1) and attached it to a ship to make a sail. Signal processing was used to make the cloth appear heavier, and less mobile, and a brushed direct manipulation edit was applied to make the sail bow out as if in a strong wind. Preprocessing time for the 16K triangle, 450 frame subdivided cloth was about 70 minutes. Remember, preprocessing time is a one-time cost for each sequence. What matters most is total user editing time, which was less than six minutes.

An even more complicated example is shown in Figure 12. Here, a cape that was simulated on a cow is edited to fit a galloping horse. Before editing, the cape intersects the horse in multiple locations, and it doesn't move with the galloping motion of the horse. By attaching the cape to the shoulders of the horse, using time-varying constraints, the cape now fits the horse snugly, and moves with the galloping motion. The inter-penetration of the tail with the cape was fixed with a few embossing and direct manipulation edits, with a temporal scaling function synched with the rise and fall of the horse's tail. Our first, rough edit is shown in the upper right of the figure. We refined the edit by making the cloth above the horse's back follow the horse's shape more closely, and move with the rise and fall of the horse's back. Our system makes it easy to perform this kind of iterative design process.

7 Conclusion and Future Work

We have presented a novel method for editing arbitrarily deforming surfaces. This method allows a user to take an existing animation sequence and modify it so that it can be used for a different purpose or so that it better fulfills its original purpose. The user can directly manipulate the surface, emboss it, and attach it to moving objects not present in the original environment. Through our novel time-varying multiresolution transform, the edits can be propagated naturally to any other frame of the animation. A system like this could go a long way toward easing the burden currently placed on artists and designers who must repeatedly "tweak" simulation conditions to get the result they desire, or go in by hand and manually apply edits to multiple frames. Deforming surface editing is also likely to become even more important as cloth motion capture technology develops. While many of our best examples are cloth data, we stress that our method is not limited to cloth.

While our system enables a wide variety of edits, one interesting possibility to extend it even further is signal processing in the temporal domain (being able to scale different frequencies of the motion itself). Also, though multiple translational edits can be combined to produce a rotation, it would be useful to have an explicitly rotational kind of edit. In fact, using our system as a foundation, a great variety of editing tools, like procedural edit filters and other "plugins," could be created easily. Also, if a purely cloth-oriented system is desired, it may be fruitful to incorporate actual knowledge of cloth dynamics.

References

ALEXA, M., AND MÜLLER, W. 2000. Representing animations by principal components. In *Eurographics*, The Eurographics Association/Blackwell Publishers.

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

BURT, P. J., AND ADELSON, E. H. 1983. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications COM-31,4*, 532–540.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Trans. Graph. 23*, 3, 905–914.

DEY, T., EDELSBRUNNER, H., GUHA, S., AND NEKHAYEV, D., 1998. Topology preserving edge contraction. Technical Report RGI-Tech-98-018, Raindrop Geomagic Inc., Research Triangle Park, North Carolina.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *SIGGRAPH 1997*, ACM Press/Addison-Wesley Publishing Co., 209–216.

GARLAND, M., AND ZHOU, Y. 2005. Quadric-based simplification in any dimension. *ACM Trans. Graph. 24*, 2, 209–239.

GLEICHER, M. 1997. Motion editing with spacetime constraints. In *SI3D 1997*, ACM Press, New York, NY, USA, 139–ff.

GLEICHER, M. 2001. Motion path editing. In *SI3D 2001*, ACM Press, New York, NY, USA, 195–202.

GUSKOV, I., AND KHODAKOVSKY, A. 2004. Wavelet compression of parametrically coherent mesh sequences. In *SCA 2004*, ACM Press, New York, NY, USA, 183–192.

GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution signal processing for meshes. In *SIGGRAPH 1999*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 325–334.

GUSKOV, I., KLIBANOV, S., AND BRYANT, B. 2003. Trackable surfaces. In *SCA 2003*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 251–257.

HOPPE, H. 1996. Progressive meshes. In *SIGGRAPH 1996*, ACM Press, 99–108.

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

KERNIGHAN, B. W., AND LIN, S. 1970. An efficient heuristic for partitioning graphs. *Bell Systems Tech. J. 49* (Feb.), 291–308.

KIRCHER, S., AND GARLAND, M. 2005. Progressive multiresolution meshes for deforming surfaces. In *SCA 2005*, ACM Press, New York, NY, USA, 191–200.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH 1998*, ACM Press, New York, NY, USA, 105–114.

LENGYEL, J. E. 1999. Compression of time-dependent geometry. In *SI3D 1999*, ACM Press, New York, NY, USA, 89–95.

LOOP, C. 1987. *Smooth Subdivision Smfaces Based on Triangles*. Master's thesis, Department of Mathematics, University of Utah.

NO, H. M. B., SANDER, P. V., MCMILLAN, L., GORTLER, S., AND HOPPE, H. 2003. Geometry videos: a new representation for 3d animations. In *SCA 2003*, Eurographics Association, 136–146.

SCHOLZ, V., STICH, T., KECKEISEN, M., WACKER, M., AND MAGNOR, M. 2005. Garment motion capture using color-coded patterns. In *Eurographics*, The Eurographics Association/Blackwell Publishers.

SHAMIR, A., PASCUCCI, V., AND BAJAJ, C. 2000. Multi-resolution dynamic meshes with arbitrary deformations. In *Proc. Visualization '00*, IEEE Computer Society Press, 423–430.

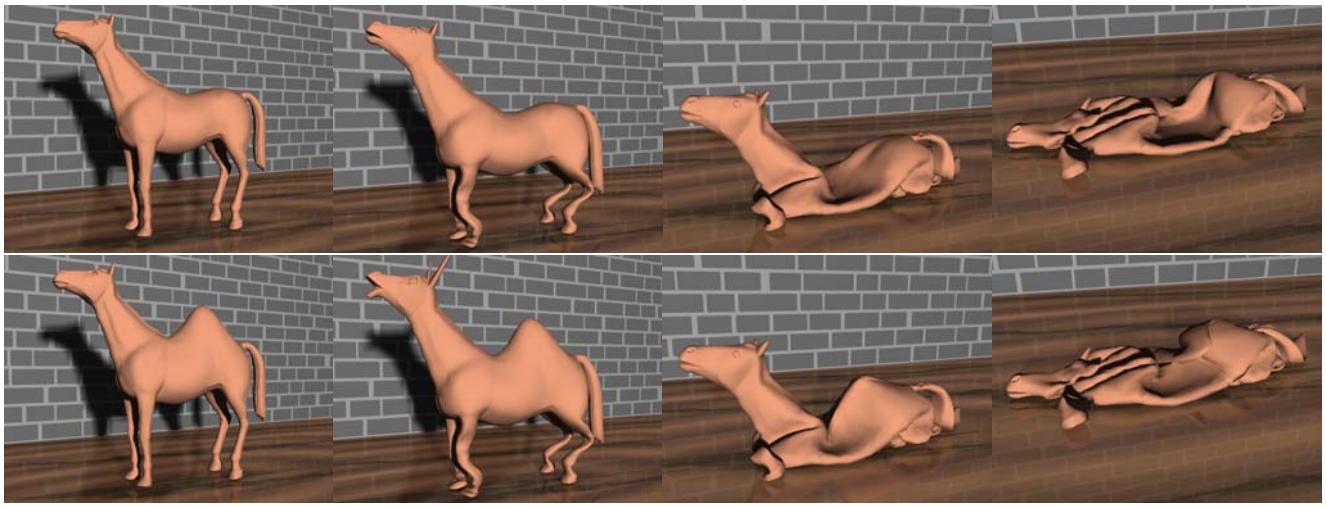


Figure 10: Top: The original collapsing horse. Bottom: A very surprised collapsing camel-horse hybrid. Note the natural way in which the hump folds over as the geometry collapses.

Figure 11: Editing a piece of motion captured cloth (Figure 1) to make a ship's sail. Our system makes it easy to reuse deforming surface data.

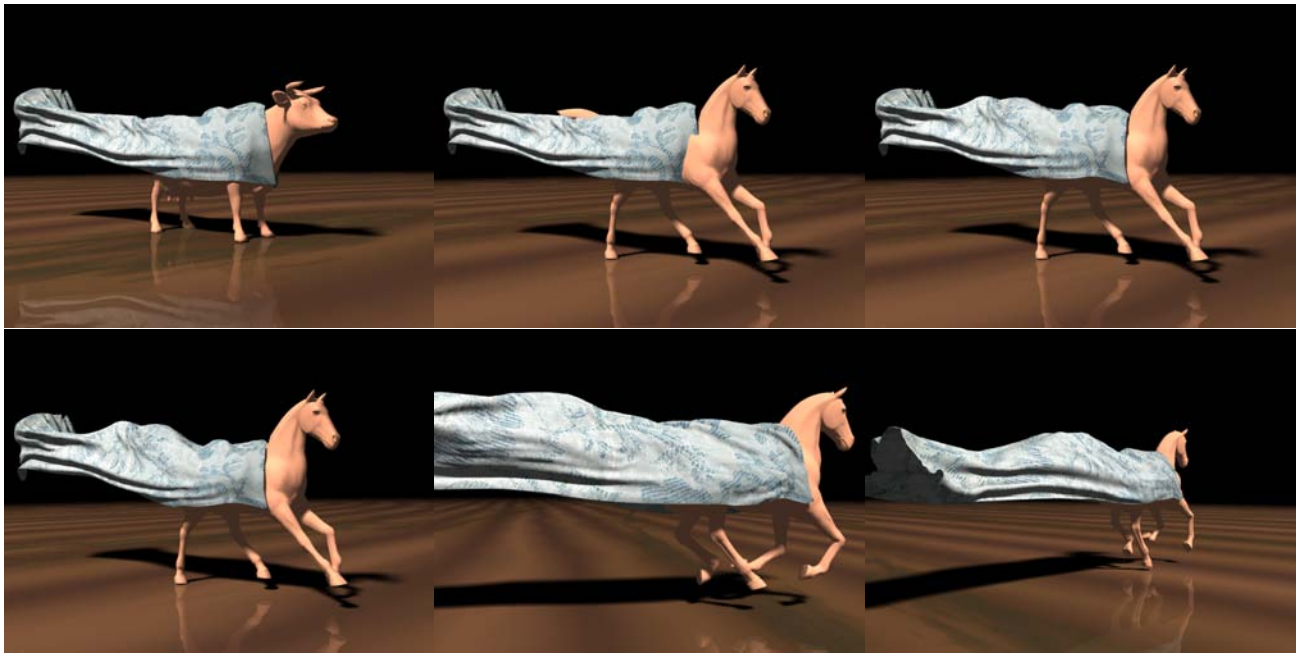


Figure 12: Top Left: A simulated cape on a cow. Top Middle: The cape doesn't fit the horse, but after some quick editing (Top Right) it does. Not satisfied with the fit, we make a few more edits. Bottom: Three frames of the final editing results.

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

WHITE, R., LOBAY, A., AND FORSYTH, D., 2005. Cloth capture. Technical Report No UCB/CSD-5-1387, EECS Department, U. of California, 2005. http://www.cs.berkeley.edu/~ryanw/research/tr_cloth.html.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *SIGGRAPH 1997*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 259–268.