Gradient Domain Editing of Deforming Mesh Sequences

Weiwei Xu Kun Zhou Yizhou Yu* Qifeng Tan† Qunsheng Peng† Baining Guo

Microsoft Research Asia *University of Illinois at Urbana-Champaign †State Key Lab of CAD&CG, Zhejiang Univ.

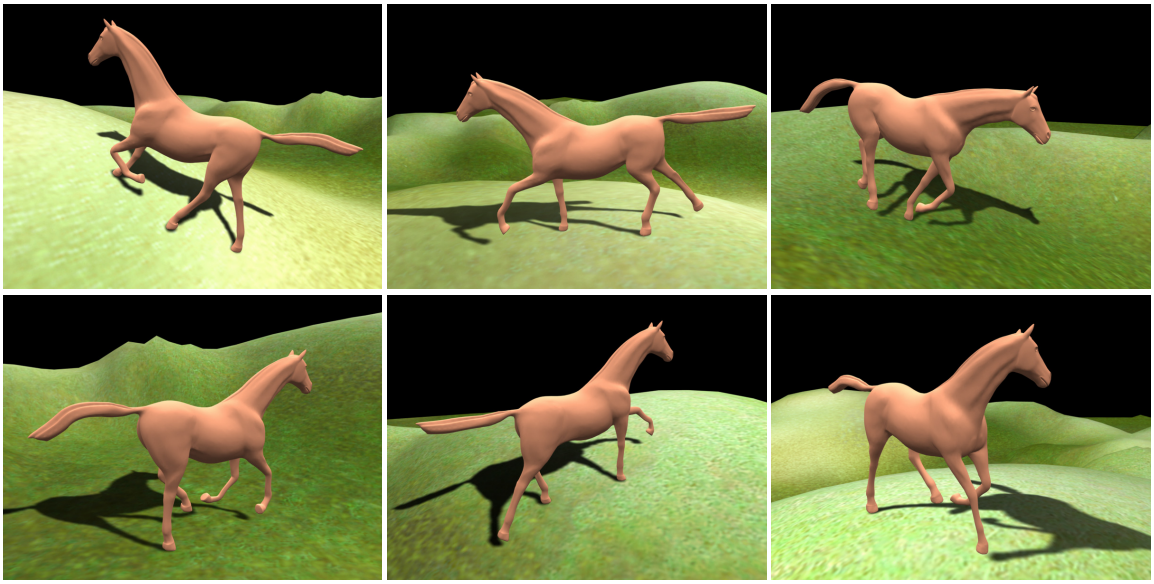

Figure 1: *A straight run is adapted to a curved path on an uneven terrain. The original deforming mesh sequence moves along a straight line on a plane. We first make the* HORSE *move along a curve using path editing, and then adapt the sequence onto the terrain using footprint editing.*

Abstract

Many graphics applications, including computer games and 3D animated films, make heavy use of deforming mesh sequences. In this paper, we generalize gradient domain editing to deforming mesh sequences. Our framework is keyframe based. Given sparse and irregularly distributed constraints at unevenly spaced keyframes, our solution first adjusts the meshes at the keyframes to satisfy these constraints, and then smoothly propagate the constraints and deformations at keyframes to the whole sequence to generate new deforming mesh sequence. To achieve convenient keyframe editing, we have developed an efficient alternating least-squares method. It harnesses the power of subspace deformation and two-pass linear methods to achieve high-quality deformations. We have also developed an effective algorithm to define boundary conditions for all frames using handle trajectory editing. Our deforming mesh editing framework has been successfully applied to a number of editing scenarios with increasing complexity, including footprint editing, path editing, temporal filtering, handle-based deformation mixing, and spacetime morphing.

†This work was done while Qifeng Tan was an intern at Microsoft Research Asia.

Keywords: Mesh Deformation, Keyframes, Control Meshes, Local Frames, Handle Trajectory, Rotation Interpolation

1 Introduction

Many computer graphics applications, including computer games and 3D animated films, make heavy use of deforming mesh sequences. Designing and producing visually pleasing mesh sequences, either manually or through physically based simulation, is a costly and time-consuming process, during which one needs to further consider effects caused by interactions between the deforming object and its surroundings. For example, a running horse needs to adjust its pace and body configuration according to obstacles and turns in the path as well as undulations on the terrain. This motivates a methodology that creates novel mesh sequences by reusing and adapting existing ones.

The goal of adapting existing deforming mesh sequences is to conveniently produce desired ones that satisfy requirements from both the user and the environment. Compared with static mesh editing, deforming mesh sequences have an additional temporal dimension, which leads to much increased data complexity and a few new technical challenges. First, a paramount demand is to minimize the amount of user intervention. This is especially important for long sequences and dictates the usability of the entire editing system. Second, with minimal user intervention, the system should still permit both flexible and precise user control. Third, given very sparse constraints, the system should be able to produce desired results that preserve both temporal coherence and important characteristics of the deformations in original mesh sequences.

This paper generalizes, for the first time, gradient domain static mesh editing [Alexa 2003; Sorkine et al. 2004; Yu et al. 2004; Lipman et al. 2005; Huang et al. 2006; Lipman et al. 2006; Shi et al.

2006] to deforming mesh sequences. Our framework is keyframe based, and it can meet all the aforementioned challenges. The user can choose to edit any frame in the original mesh sequence as well as any handle on that frame. The environment may also induce constraints at certain frames. Any frame with environment-induced or user-supplied constraints subsequently becomes a keyframe. The location of these constraints vary among different keyframes. Adapting the original deforming mesh to satisfy such sparse and irregularly distributed constraints at unevenly spaced keyframes is a daunting task. Our solution first adjusts the meshes at the keyframes to satisfy these constraints, and then smoothly "propagate" the constraints and deformations from keyframes to the entire sequence to generate a new deforming mesh sequence.

Since all keyframes are edited independently, adjusting each of them to satisfy its constraints is actually a static mesh editing problem. Present gradient domain mesh deformation techniques have limitations when used repeatedly for many keyframes. Two-pass linear methods, such as the one in [Lipman et al. 2005], require additional rotational constraints, which are not very convenient to supply when such constraints are needed at many keyframes. Sometimes, it is even impossible to specify rotations for environment-induced constraints. On the other hand, geometric subspace deformation [Huang et al. 2006] can deduce rotations from positional constraints via nonlinear optimization. Nevertheless, it requires a control mesh for every frame and it is sensitive to the quality of the control mesh. Meanwhile, it is very hard to guarantee the quality of control meshes when they are automatically generated for all the frames in a sequence. To this end, we have developed an efficient alternating least-squares method for keyframe editing. It harnesses the power of subspace deformation and two-pass linear methods to achieve high-quality deformations even when the control mesh is problematic.

After keyframe editing, both the deformations and constraints at keyframes need to be smoothly propagated to the rest of the frames. The deformations are defined as local transforms of the differential coordinates between a pair of original and deformed keyframes. The propagated deformations are used for computing target differential coordinates for every frame. Subsequent mesh reconstruction from target differential coordinates requires a boundary condition. Therefore, we have also designed a least-squares handle trajectory editing algorithm to propagate the constraints at keyframes to the rest of the frames to serve as boundary conditions.

We have successfully applied our deforming mesh editing framework to a number of editing scenarios with increasing complexity, including footprint editing, path editing, temporal filtering, handle-based deformation mixing, and spacetime morphing. We have developed techniques that make such applications possible as well as user interfaces with further simplified user interaction specifically tailored for each of these scenarios. For example, in path editing, one only needs to quickly sketch a new path to make an entire mesh sequence automatically follow that path. Moreover, our method is not specifically designed for skeleton based mesh animations. We demonstrate editing capabilities for non-articulated deformations, including cloth animation and spacetime morphing.

1.1 Related Work

This paper is made possible by many inspirations from previous work on surface and meshless deformations [Alexa 2003; Sheffer and Kraevoy 2004; Sorkine et al. 2004; Yu et al. 2004; Lipman et al. 2005; Müller et al. 2005; Zhou et al. 2005; Zayer et al. 2005; Botsch et al. 2006; Huang et al. 2006; Lipman et al. 2006; Shi et al. 2006; Au et al. 2006], multiresolution mesh editing [Zorin et al. 1997; Kobbelt et al. 1998; Guskov et al. 1999; Kircher and Garland 2006], mesh skinning [Mohr and Gleicher 2003; James and Twigg 2005], mesh inverse kinematics (mesh-IK) [Sumner et al. 2005; Der et al. 2006], and shape interpolation and manipulation [Alexa et al. 2000; Igarashi et al. 2005]. Surface-based mesh modeling and multiresolution mesh editing produce new results from a single input mesh while skinning and mesh-IK require multiple existing deformation examples.

Laplacian mesh editing has received much attention recently. This approach extracts intrinsic geometric properties, such as differential coordinates, from the input mesh, lets them subject to local transformations during editing, and finally reconstructs new meshes from the transformed differential coordinates by solving a global system of equations. The reconstruction step makes local editing in differential coordinates have global effects on the new mesh. The composition of these steps leads to an overall nonlinear process. Due to the existence of efficient solvers for sparse linear systems, much effort has been devoted in the past few years to obtain approximate solutions using either linearizations [Sorkine et al. 2004] or multiple linear passes [Lipman et al. 2005; Zayer et al. 2005; Lipman et al. 2006; Shi et al. 2006]. The latter typically requires explicit rotational constraints. Alternatively, one can directly cast the problem as a nonlinear optimization without rotational constraints. Since nonlinear optimizations require more expensive iterative steps, subspace [Huang et al. 2006] methods have been developed to achieve acceleration. A fast nonlinear optimization framework for mesh deformation based on shape matching among rigid prisms enveloping the mesh faces has been presented in [Botsch et al. 2006]. In this paper, we show that nonlinear Laplacian mesh editing can be solved more accurately using alternating least-squares, which has provable convergence. Note that Laplacian mesh editing has not been previously generalized to the spacetime domain.

Flexible mesh editing can also be achieved with a multiresolution decomposition of the original mesh [Zorin et al. 1997; Kobbelt et al. 1998; Guskov et al. 1999]. By working at a specific resolution, one can manipulate the mesh at a desired scale to reduce the amount of user intervention. Multiresolution mesh editing has been recently extended to deforming mesh editing in [Kircher and Garland 2006] with an effective technique to maintain temporal coherence. However, without solving a global system as proposed in [Kobbelt et al. 1998] as well as in Laplacian mesh editing, effects achievable by the method in [Kircher and Garland 2006] is rather local in the spatial domain even at a very coarse scale, which means that precise control of large deformations can only be achieved with much more user intervention. We will compare our method with the one in [Kircher and Garland 2006] in Section 3.

Skinning has been a popular method for producing deforming mesh sequences suited for real-time rendering on graphics hardware. Typically, there should be an underlying skeleton to make this method applicable. The movement of every vertex is controlled by a linear blend of the movement of a few nearby rigid bones. Blending coefficients are trained from deformation examples. James and Twigg [2005] introduces a practical technique that converts a deforming mesh sequence to a linear blend skinning model. It automatically extracts a set of "bones" from the input meshes instead of requiring them as part of the input. However, being a hardware friendly mesh representation, skinning is not well suited for generic mesh editing, especially in the presence of highly nonrigid deformations.

Recent techniques on mesh-IK [Sumner et al. 2005; Der et al. 2006] share similarity with nonlinear Laplacian mesh editing. Both approaches avoid rotational constraints and allow the user to deform meshes by specifying sparse positional constraints only. An important distinction is that mesh-IK requires multiple existing deformation examples in order to deform a single input mesh. In its current form, the input to mesh-IK is a static mesh instead of a deforming

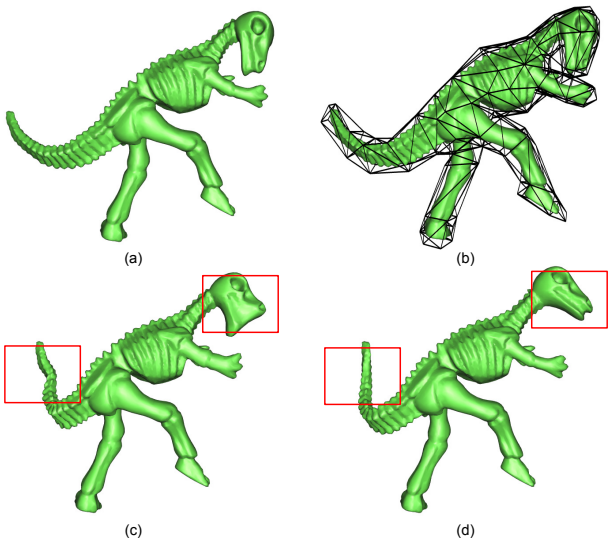

Figure 2: *Comparison between the subspace method and our alternating least-squares method. (a) The rest pose of a mesh. (b) Its control mesh. Since the* HEAD *and* HAND *of the* DINOSAUR *is too close, it causes self-intersections in the control mesh. As a result, the subspace method produces unnatural results shown in (c), while the alternating least-squares method can preserve the details of the original shape very well (d).*

mesh sequence.

2 Alternating Least-Squares for Mesh Deformation

Let us first focus on individual keyframes and introduce an accurate nonlinear solution method for static mesh deformation. A deformed mesh should satisfy deformation constraints while optimally preserving the local relative configurations among neighboring vertices under rotation invariant criteria, which motivated the introduction of a local frame at every vertex in [Lipman et al. 2005], where a solution of the deformed mesh is obtained by solving the altered local frames and vertex positions in two sequential linear steps. Suppose $\hat{\mathbf{v}}_i$ and $\mathbf{v}_i$ represent the same vertex in the original and deformed meshes, respectively. One of the rotation-invariant properties in this approach can be expressed as follows.

$$\mathbf{v}_j - \mathbf{v}_i = \mathbf{R}_i \hat{\mathbf{v}}_{i \to j}, j \in N_i, \tag{1}$$

where N_i represents the index set of the 1-ring neighbors of $\mathbf{v}_i$, $\mathbf{R}_i$ is a 3×3 rotation matrix that represents the altered local frame at $\mathbf{v}_i$, and $\hat{\mathbf{v}}_{i \to j}$ represents the local coordinates of $\hat{\mathbf{v}}_j$ in the local frame, $\hat{\mathbf{R}}_i$, at $\hat{\mathbf{v}}_i$ in the original mesh. Note that the columns of $\mathbf{R}_i$ consist of the three orthonormal axes, $(\mathbf{b}_1^i, \mathbf{b}_2^i, \mathbf{N}^i)$, of the local frame. Once the altered local frames are known, the equations in (1) over all vertices give rise to an overdetermined linear system for vertex positions and can be solved using least squares if a boundary condition is given.

It should be noted that it is suboptimal to first solve local frames and then solve vertex positions in the sense that they may not optimally satisfy the overdetermined system expressed in (1). Instead, we would like to solve both local frames and vertex positions simultaneously by minimizing the following objective function.

$$\sum_i \sum_{j \in N_i} \|\mathbf{v}_j - \mathbf{v}_i - \mathbf{R}_i \hat{\mathbf{v}}_{i \to j}\|^2 + \|\mathbf{CV} - \mathbf{U}\|^2, \tag{2}$$

where $\mathbf{v}_i$, $\mathbf{v}_j$ and $\mathbf{R}_i$ are unknowns, $\mathbf{V}$ represents the set of constrained vertices on handles, $\mathbf{C}$ is the positional constraint matrix, and $\mathbf{U}$ is the target positions of the constrained vertices.

Simultaneously optimizing all the unknowns in (2) gives rise to an expensive nonlinear least-squares problem. Fortunately, an initial inaccurate solution to this nonlinear problem can be refined until convergence by iteratively alternating two simpler and more efficient least-squares steps, which are respectively responsible for improving the estimation of the local frames and vertex positions. The first of these two steps minimizes (2) by fixing the vertex positions and optimizing $\mathbf{R}_i$'s only. The second step does the opposite. Note that the second step results in the same overdetermined linear system as in (1). Although the first step is nonlinear, the fixed vertex positions let us solve the optimal local frame at each vertex independently. Thus, minimizing (2) becomes equivalent to minimizing the following local objective function once for each local frame.

$$\sum_{j \in N_i} \|(\mathbf{v}_j - \mathbf{v}_i) - \mathbf{R}_i \hat{\mathbf{v}}_{i \to j}\|^2, \tag{3}$$

where the optimal rotation matrix $\mathbf{R}_i$ can be conveniently obtained from the closed-form quaternion-based solution provided in [Horn 1987]. This is actually a local shape matching problem similar to the ones that occur in [Müller et al. 2005; Botsch et al. 2006; Park and Hodgins 2006]. It has a unique solution as long as $\mathbf{v}_i$ has two or more 1-ring neighbors and $\mathbf{v}_i$ is not collinear with them. Note that since the number of vertices in a 1-ring neighborhood is practically bounded, solving the local frames at all vertices using this closed-form solution has a linear complexity. Overall, we successfully minimize (2) using a combination of local shape matching and sparse linear systems without expensive global shape matching typically solved using nonlinear optimization. The convergence of the alternating least square server can be guaranteed, since the two alternating steps respectively obtain optimal least-squares solutions with respect to the subset of the variables they optimize. The first step optimizes the rotations using Horn's closed-form solution, which has the same least-squares objective function in Equation 2. The second step optimizes vertex positions while fixing rotations. Both steps monotonically decrease the least-squares objective function defined in Equation (2) and, therefore, guarantee convergence.

An initial solution to (2) is still required to start the iterations. The subspace method in [Huang et al. 2006] uses a coarse control mesh to obtain an approximate solution inside a subspace. This approximate subspace solution can serve as a good initial solution for our alternating least squares. In addition, this subspace method only requires positional constraints. Even though our iterative method involves rotations, the initial rotations can be estimated from the initially deformed vertex positions supplied by the subspace method. As a result, our method can also work with positional constraints only. In all our experiments, achieving smooth and visually pleasing results with our alternating least-squares method requires less than 10 iterations. Using the sparse linear solver from CHOLMOD [Davis 2006], the running time for each iteration is 0.03 second on a 3.2GHz Pentium processor for a mesh with 10,000 vertices.

Since our iterative method can reach a more globally optimal solution than the methods in [Lipman et al. 2005; Huang et al. 2006], it can achieve better visual results as well. On the other hand, the methods in [Lipman et al. 2005; Lipman et al. 2006; Shi et al. 2006] need user-supplied rotational constraints while our method does not. Although we do not explicitly enforce the smoothness of the rotation field over the deformed mesh as in [Lipman et al. 2006], the local frames as well as the resulting rotation field we obtain are in fact smoothly varying because they are extracted from 1-ring neighborhoods as in (3) and the 1-ring neighborhoods of adjacent vertices have partial overlap. Fig. 2 compares our method with the subspace method in [Huang et al. 2006]. Self-intersections in the control mesh as well as its coarse resolution can result in unnatural deformation results from the subspace method while our alternating least-squares method can successfully eliminate the problems

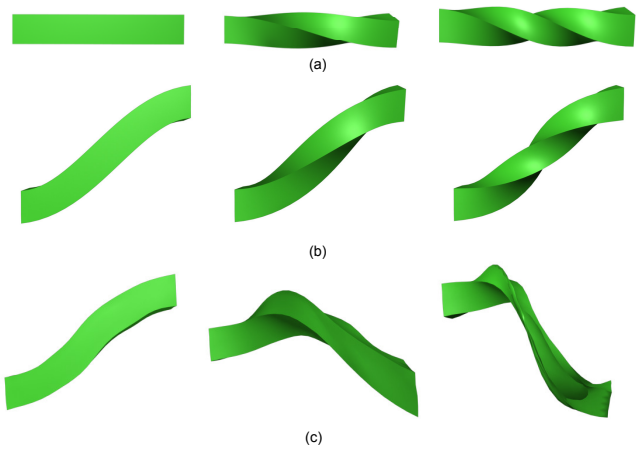

Figure 3: *Comparison between our gradient domain method and the multiresolution method for deforming mesh editing in [Kircher and Garland 2006]. (a) Original box twisting sequence. (b) The edited result from our system. (c) The edited result from the multiresolution method. In this example, we edit the first frame which becomes a keyframe, and then propagate the deformation at the first frame to the entire sequence. The left column shows the original and edited first frame. The middle and right columns show two frames in resulting mesh sequences. Note that our method generates natural results while the results from the multiresolution method have severe distortions. This is because the multiresolution method is based on local detail vectors only while our method is based on local rotations.*

in such results.

We note that several other nonlinear algorithms, including Primo [Botsch et al. 2006], can be used to produce good deformation results for the static mesh shown in Figure 2 as well. Our alternative deformation method is built upon the subspace method in [Huang et al. 2006] that supports many useful controls over deformation results, including the volume constraint, skeleton constraint, and projection constraint. Therefore, we deem our alternative deformation method a more convenient tool in interactive keyframe editing.

3 Editing Deforming Mesh Sequences

In our spacetime framework, given a deforming mesh sequence, the user can choose to edit an arbitrary subset of frames. In the following, these chosen frames are called *keyframes*. At each keyframe, the user can also choose to edit an arbitrary subset of handles. A handle is defined as a subset of nearby vertices within the same frame. When manipulating a handle, the user only needs to drag one vertex in the handle to provide a positional constraint. The rotation of the handle will be obtained automatically from the nonlinear optimization introduced in the previous section. We assume that the mesh connectivity remains the same across all frames.

Given sparse positional constraints on the handles, an overall objective function for spacetime editing across all frames can be formulated as

$$\sum_k \sum_i \|\mathbf{L}^k \mathbf{v}_i^k - \mathbf{R}_i^k \hat{\mathbf{d}}_i^k\|^2 + \sum_k \sum_i \|\Delta_t \mathbf{R}_i^k\|_F^2 + \sum_k \|\mathbf{C}^k \mathbf{V}^k - \mathbf{U}^k\|^2 \quad (4)$$

where $\mathbf{L}^k$ is the matrix for computing spatial Laplacian differential coordinates at the k-th frame, $\hat{\mathbf{d}}_i^k$ is the vector of differential coordinates at the i-th vertex of the original mesh at the k-th frame, $\mathbf{R}_i^k$ is the local rotation matrix at $\mathbf{v}_i^k$, $\Delta_t \mathbf{R}_i^k$ represents the differences of corresponding rotation matrices in consecutive frames, and $\mathbf{C}^k$, $\mathbf{V}^k$ and $\mathbf{U}^k$ are similar to $\mathbf{C}$, $\mathbf{V}$ and $\mathbf{U}$ in (2). The second term in (4) enforces temporal coherence of local rotations. In this objective function, for the sake of clarity, we have left out terms accommodating additional constraints, such as volume preservation. Directly optimizing such a nonlinear objective function with respect to vertex positions and local rotations across all the frames is extremely expensive. More importantly, we have observed in our experiments that simultaneous optimization in both spatial and temporal domains does not well preserve the shape of the meshes at individual frames because the solution needs to provide a tradeoff between these two domains.

In practice, we have devised an efficient algorithm to obtain a numerically suboptimal solution, which nonetheless is capable of producing visually pleasing results. Once constraints are set up, our algorithm performs the following sequential steps to minimize (4) and produce a new deforming mesh sequence that is consistent with all constraints while maintaining temporal coherence of both local rotations and handle positions. Our system is based on the mesh deformation technique presented in the previous section.

- First, since obtaining the initial solution for alternating least-squares requires a coarse control mesh, to make it applicable in the spacetime domain, we need a control mesh for every frame in the input sequence. To minimize user interaction, our framework only requires a control mesh for the first frame and automatically generates a control mesh for each subsequent frame by adjusting the first one.

- Second, the mesh at every keyframe is deformed to satisfy the positional constraints at that keyframe using alternating least squares introduced in the previous section. As a product, we obtain an altered local coordinate frame as well as an associated rotation at every vertex over the deformed mesh.

- Third, these per-vertex rotations at the keyframes are smoothly interpolated to obtain dense rotational constraints at every intermediate frame. Rotations are represented as unit quaternions, and the logarithm of the quaternions are interpolated using Hermite splines. Those frames at the very beginning or end of the mesh sequence may not be bracketed by sufficient number of keyframes and, therefore, are only linearly interpolated or not interpolated at all. The user can optionally designate an influence interval for a keyframe to have finer control over the interpolation.

- Fourth, we extract an average translation and rotation for each handle from the translations and rotations associated with individual vertices in the handle. Since each handle has its own temporal trajectory in the original mesh sequence, this trajectory is automatically edited and updated using the extracted average translations and rotations at keyframes to guarantee its temporal consistency.

- Lastly, a deformed mesh is solved for every intermediate frame using the interpolated rotational constraints as well as the positional constraints from the updated handles. Here, since we already have a rotational constraint at every free vertex, we can obtain a deformed mesh quickly by simply solving the overdetermined linear system in (1) without invoking the more expensive subspace method. A few iterations of alternating least squares can be optionally applied to improve the deformed mesh.

In terms of the objective function in (4), its first and third terms are minimized by the second and last steps respectively for keyframes and intermediate frames. Its second term is reduced by the third step. The positional constraints at intermediate frames are automatically set up by the fourth step. Comparing to simultaneous optimization of all the unknowns, our sequential steps can better preserve the shapes at individual frames. More details regarding automatic control mesh generation (the first step) and handle propagation (the fourth step) follow.

Figure 3 shows a comparison between our gradient domain editing method and the multiresolution method in [Kircher and Garland

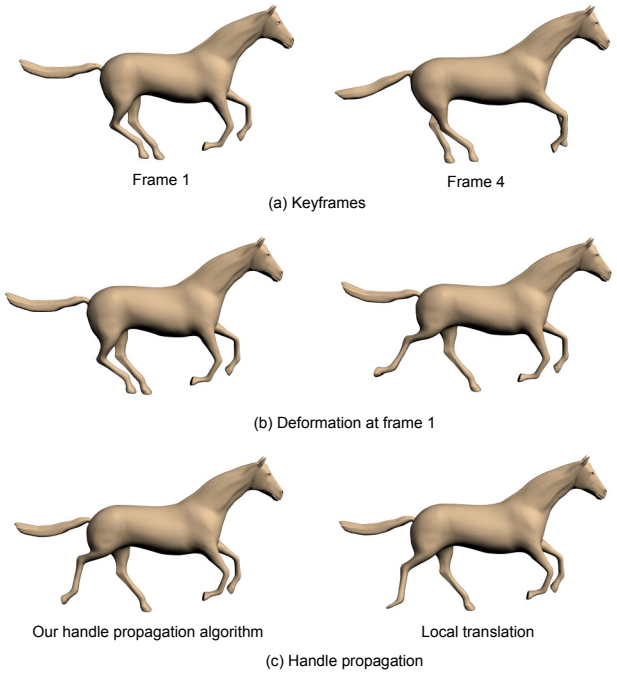

Figure 4: *Handle trajectory editing. (a) Frames #1 and #4 in a horse running sequence are set as keyframes. (b) Edited frame #1 (c) Edited results at frame #2. Editing based on simple local translation generates severe distortion in the edited leg while our method generates natural results.*

2006]. Our method can produce more natural results because our method is based on per-vertex local rotations while the method in [Kircher and Garland 2006] is based on "edit replicators" which are actually displacement vectors represented in the local coordinate frames defined at the vertices of the fine mesh. Once the bar is twisted, the orientation of a local coordinate frame is changed significantly from frame to frame. As a result, the same local displacement vectors become pointing in drastically different directions once they have been transformed into the world coordinate system. In addition, displacement vectors have inherent ambiguity in representing 3D transformations such as twisting.

3.1 Handle Trajectory Editing

It is critical to correctly propagate the handle editing results from the keyframes to all intermediate frames since altered handles will serve as new boundary conditions when the meshes at the intermediate frames are deformed and reconstructed. Because there are multiple vertices in the same handle and interpolating each vertex position individually at intermediate frames would severely distort the overall shape of the handle, these vertices are forced to move together rigidly using the extracted average translation and rotation. We define a single local coordinate frame for each handle and represent the vertices in that handle using their local coordinates. Formally, a deformation handle i at frame k is denoted as $\mathbf{H}_i^k = \{\mathbf{c}_i^k, \mathbf{F}_i^k, \{\mathbf{v}_{i_m}^k | m = 0, ..., n_p - 1\}\}$, where $\{v_{i_m}^k | m = 0, ..., n_p - 1\}$ is the set of vertices in this handle, $\mathbf{c}_i^k$ and columns of $\mathbf{F}_i^k$ define the origin and axes of the local coordinate frame. $\mathbf{c}_i^k$ can be the centroid of the vertices and the initial axes can be defined arbitrarily since we will only consider the rotation between the initial and altered axes.

Suppose there are n_f frames in the entire mesh sequence. If we consider a set of corresponding handles, $\{\mathbf{H}_i^k, 0 \le k \le n_f\}$, collectively, we can define a piecewise linear curve in the temporal domain and $\{\mathbf{c}_i^k, 0 \le k \le n_f\}$ is the set of vertices of this curve. We can also use

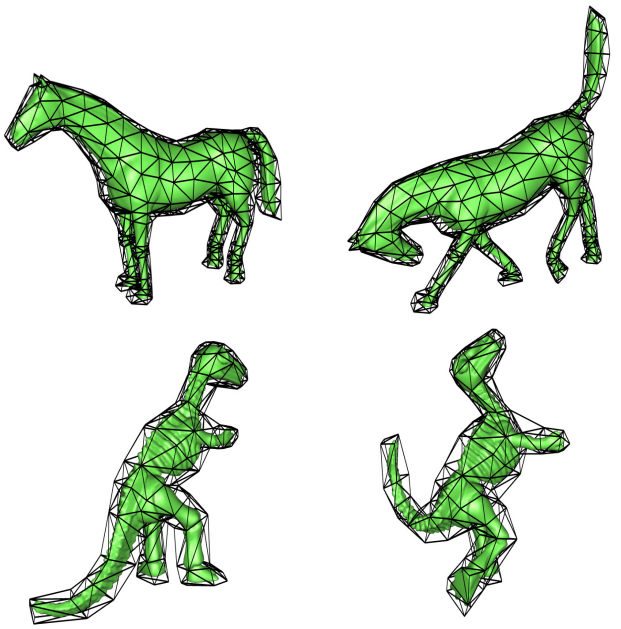

Figure 5: *Control mesh transfer. Left: the first frame with assigned control mesh in two deforming mesh sequences. Right: the automatically transferred control meshes at a later frame.*

$\{\mathbf{F}_i^k, 0 \le k \le n_f\}$ to define the axes of the local coordinate frames at these vertices. Once the meshes at keyframes have been adjusted, a new local coordinate frame is defined at each altered handle using the extracted average translation and rotation. These new local coordinate frames impose both positional and rotational constraints on the curve. Under these constraints, we need to figure out the new positions and rotations for the rest of the vertices on the curve. Thus, handle trajectory editing can be cast as a temporal curve editing problem.

We extend the rotation invariant property in (1) from spatial meshes to temporal curves. Thus, (1) can be reformulated as

$$\mathbf{c}_i^l - \mathbf{c}_i^k = \mathbf{F}_i^k \hat{\mathbf{c}}_i^{k \to l}, l \in N^k, \tag{5}$$

where $\hat{\mathbf{c}}_i^k$ and $\mathbf{c}_i^k$ represent the original and new handle centroids, N^k represents the index set of the immediate neighbors of $\mathbf{c}_i^k$, $|N^k| \le 2$, and $\hat{\mathbf{c}}_i^{k \to l}$ represents the local coordinates of $\hat{\mathbf{c}}_i^l$ in the local frame, $\hat{\mathbf{F}}_i^k$, defined at $\hat{\mathbf{c}}_i^k$ in the original curve. We simply interpolate corresponding handle rotations at keyframes over the rest of the vertices on the curve to figure out the orientation of all the new local coordinate frames, $\{\mathbf{F}_i^k\}$, at these vertices. Once these altered local frames are known, the equations in (5) over all unconstrained vertices give rise to an overdetermined linear system for vertex positions and can be solved using least squares. We solve such a least-squares problem for each curve. Since the number of vertices on a curve is relatively small, it can be solved very quickly. The new world coordinates of the vertices inside each handle at an intermediate frame can be obtained by maintaining their original local coordinates in the new local coordinate frame at that handle.

Figure 4 shows a comparison between our handle trajectory editing algorithm and simple local translation at each vertex. We can see that local translation obviously shrinks the size of the horse leg while the results from our method remain natural.

3.2 Control Mesh Transfer

In the subspace method in [Huang et al. 2006], a coarse control mesh is used for creating a deformed version of a finer mesh

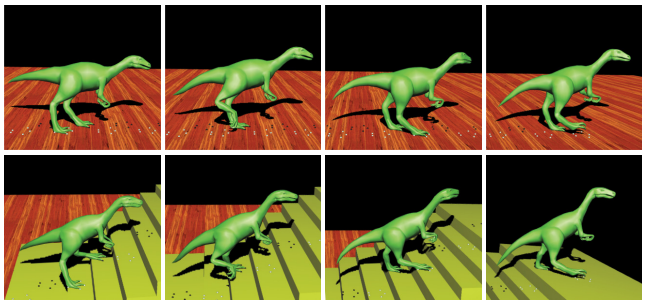

Figure 6: *Footprint editing. A planar walking sequence is adapted to a stair walking. Black handles represent the footprints of the left leg, and white handles represent those of the right leg. In this example, the user only needs to relocate the footprints to the stairs, and our system can automatically generate the stair walking sequence.*

through a linear relationship,

$$\mathbf{V}^f = \mathbf{W}\mathbf{V}^c, \tag{6}$$

where $\mathbf{V}^f$ represents vertices of the finer mesh, $\mathbf{V}^c$ represents vertices of the control mesh, and $\mathbf{W}$ is a matrix of mean value coordinates [Ju et al. 2005]. In the current context, given the control mesh of the first frame in a deforming mesh sequence, we need to exploit the frame-to-frame deformation of the finer mesh to automatically construct an altered control mesh for every keyframe in the sequence. The altered control mesh has altered vertex positions but the same mesh connectivity.

Here we draw an analogy between this problem and linear blend skinning, and consider the control mesh as the "skin" while a subset of vertices on the finer mesh as the "bones". Our control mesh transfer algorithm first binds the control mesh to the finer mesh at the first frame, and then uses the same binding throughout the rest of the sequence. At the first frame, for each vertex $\tilde{\mathbf{v}}_j$ of the control mesh, find n_b vertices, $\{\mathbf{v}_{i_m}\}_{m=0}^{n_b-1}$, on the finer mesh where the mean value coordinate from $\tilde{\mathbf{v}}_j$ is the largest. We achieve this by searching the j-th column of $\mathbf{W}$. We represent $\tilde{\mathbf{v}}_j$ in the local frames of these n_b vertices. That is, $\tilde{\mathbf{v}}_j = \mathbf{v}_{i_m} + \mathbf{R}_{i_m}\tilde{\mathbf{v}}_{i_m\to j}, m = 0,\ldots,n_b-1$, where $\tilde{\mathbf{v}}_{i_m\to j}$ represents $\tilde{\mathbf{v}}_j$'s local coordinates in $\mathbf{v}_{i_m}$'s local coordinate frame. When we need to work out $\tilde{\mathbf{v}}_j$'s new world coordinates for another frame in the mesh sequence, we first find the corresponding vertices of $\{\mathbf{v}_{i_m}\}_{m=0}^{n_b-1}$ in the finer mesh at that frame, and then transfer $\tilde{\mathbf{v}}_j$'s local coordinates, $\{\tilde{\mathbf{v}}_{i_m\to j}\}_{m=0}^{n_b-1}$, to the local coordinate frames at these corresponding vertices. Potentially different world coordinates can be recovered from these multiple local frames. And we simply use their linear blend as $\tilde{\mathbf{v}}_j$'s new world coordinates. We find out that even uniform weighting in the linear blending can achieve good results. Figure 5 shows two typical results from this control mesh transfer algorithm.

4 Advanced Editing

Even though the editing framework in the previous section is already powerful, sometimes, it may become tedious for the user to generate more advanced editing effects. For example, porting a walking sequence from a plane to an uneven terrain would need much user interaction using this editing framework. In this section, we present advanced editing modes built on top of this editing framework. Among these advanced modes, footprint editing and path editing are specifically designed for motion similar to walking and running. Handle-based deformation mixing can be used for duplicating handle movements from a source sequence to a target sequence. In advanced editing modes, we can not only alter the shape of the deforming mesh sequence but also the overall motion of the shape, which is very hard to achieve using multiresolution mesh editing [Kircher and Garland 2006].

Figure 7: *Path editing. The path of a planar walking sequence is adapted to a user sketched curve (cyan). The original sequence is the same as in Figure 6.*

4.1 Footprint Editing

During walking or running, humans or animals may have at least one support leg in contact with the ground. It will leave a so-called footprint there. In our system, we define the footprint to be an interval of the frames where a handle keeps fixed on the ground. In footprint editing, we revise the original footprints, and let the system automatically adapt the original walking motion to the new footprints. There are two major advantages of footprint editing. First, when manipulating footprints, the user actually saves time by editing the same handle at several frames simultaneously. Second, footprints correctly capture the constraints that should be satisfied in walking motion. So the user does not need to take time to explicitly specify keyframes to avoid annoying footskating.

To extract footprints, the user needs to first define a handle that represents the foot. After that, the footprints are automatically detected by checking in what interval the position of the handle is unchanged or the changes are less than a threshold. Any frame that contains a footprint is automatically set as a keyframe. We further allow the user to either directly place or project footprints onto a terrain. The footprint can be rotated during projection by aligning the local surface normal of the terrain and the up axis of local frame at footprint.

Figure 6 shows an example of footprint editing. The user only needs to relocate the footprints, and our system can automatically generate the edited sequence.

4.2 Path Editing

During path editing, the user only needs to sketch a curve on the ground as a new motion path. Our system can automatically adapt the original mesh sequence to follow the new path. Path editing has been addressed in the editing of motion capture data [Gleicher 2001]. We adapt this idea to deforming mesh sequences.

To estimate the motion path of the original mesh sequence, we first project the centroids of the meshes in the original sequence onto the ground, and then fit a B-spline curve through these projected points. The path editing algorithm builds correspondence between the original path $p_{orig}(s)$ and the new one $p_{new}(t)$ using arc length. There is a default local coordinate frame at any point on a path. It is defined by the tangent, normal and binormal at that point. Suppose $p_{orig}(s_k)$ is the point for frame k on the original path, and it corresponds to $p_{new}(t_k)$ on the new path. We use the rigid body transform between the two local frames at these two points to transform the mesh at frame k in the original sequence so that the transformed mesh has a position and orientation consistent with the new path. There is one additional issue with footprints though. Since a footprint may last a few frames each of which may have different transforms, the same footprint in the original sequence may be transformed to multiple different locations. We simply average these multiple locations and set the result as the corresponding footprint for the edited sequence. To make the transformed meshes consistent with the new footprints, we need to apply our basic spacetime solver to these transformed meshes while considering the new footprints as handle constraints.

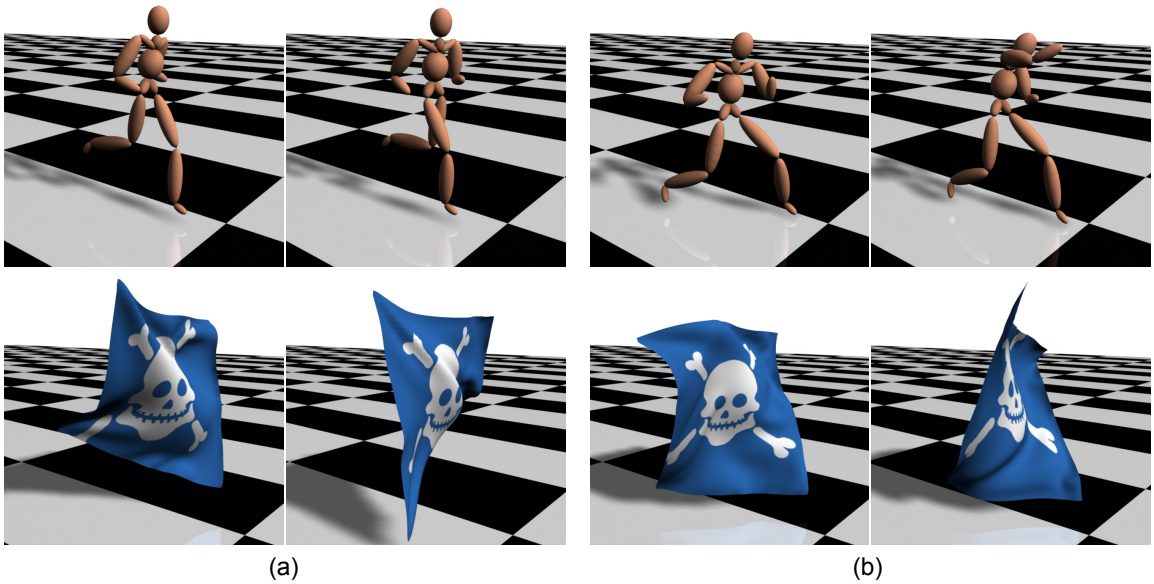

Figure 8: *Deformation mixing. Top: MoCAP data for running. Bottom: the running sequence is mixed with a flag animation. The wrinkles on the cloth are generated using cloth simulation. The user defines corresponding handles between the human model and cloth. For instance, handles on the arms of the human model correspond to upper corners of the cloth while handles on the feet of the human model correspond to lower corners of the cloth.*

Figure 7 illustrates path editing. We use the same original mesh sequence as in Figure 6. Figure 1 shows a more complex example where a straight run is adapted to a curved path on an uneven terrain. During editing, we adjust the pose of the horse's head as well to make the animation look more natural.

4.3 Handle Based Deformation Mixing

Deformation transfer has been used for making a static mesh follow the deformation sequence of a second mesh. The original deformation transfer algorithm [Sumner and Popović 2004] requires a complete correspondence between triangles on the two meshes. In the current context, we propose deformation mixing, which means that we start with two deforming meshes and generate a new sequence that mixes the large-scale deformations of the first mesh with the small-scale deformations of the second. We use the motion trajectories of a sparse set of handles on the first mesh to define its large-scale deformations, and force the corresponding handles on the second mesh to follow these trajectories. Thus, we only need to define the correspondence between handles on the two meshes.

Each handle on these two meshes generates a motion trajectory within its own mesh sequence. Since there may be global differences between two corresponding trajectories, directly forcing the second handle to follow the trajectory of the first one may produce noticeable inconsistencies between the transferred handle trajectory and the rest of the deformation on the second mesh. We therefore typically align the two corresponding trajectories first using a global transformation, including scaling, translation, and rotation, between them. After that, we set the transformed handle positions and rotations from the first trajectory as constraints for the corresponding handle on the second mesh. We repeatedly set up such constraints on the second mesh for all the handles defined on the first mesh, and then apply our basic spacetime solver to deform every frame of the second sequence.

Figure 8 shows our deformation mixing results. We mix human motion, such as running and boxing, with a cloth animation (a flag blown in a wind). Note that the wrinkles are well preserved using our method.

4.4 Mesh Deformation Filtering

Once multiple handles have been defined on a mesh sequence, the trajectory of the center of each handle becomes a temporal signal. Instead of transferring them to a second mesh as in the previous section, we can apply signal processing techniques to produce filtered versions of the trajectories, and then use these filtered trajectories as constraints on the handles to deform the original mesh sequence. As we know, in a deforming mesh sequence, the mesh surface itself has characteristic small-scale details in addition to frame-to-frame deformations. Deformation filtering is designed to alter the latter but preserve the details. Performing filtering at the level of handles lets us achieve this goal while directly filtering the trajectory of every vertex would destroy such details.

Our system further allows the user to create a hierarchy among the handles. The root of the hierarchy is always the centroid of the entire mesh. The centroids of the meshes at all frames defines the trajectory of the root. All the user-defined handles are children of the root. The user can optionally set one handle as a child of another handle. There is a local coordinate frame at each node of the hierarchy. The trajectory of each handle is transformed to the local coordinate frame at its parent. All these relative trajectories as well as the global trajectory at the root are subject to filtering. However, there is one exception. Detected footprints are not filtered to avoid footskating. The filtered relative trajectories are transformed back to the world coordinate system before used as constraints in the spacetime solver.

The cartoon animation filter introduced in [Wang et al. 2006] is an interesting filter that can produce exaggerated cartoon style motion. It is actually a signal enhancement filter defined as follows.

$$X^*(t) = X(t) - X(t) \otimes LOG, \tag{7}$$

where LOG is the Laplacian of the Gaussian, $X(t)$ is the original signal, and $X^*(t)$ is the filtered signal. We have successfully experimented with this filter. In Figure 9, we show one filtering example. Figure 9(a) and (b) illustrates the handles and their initial hierarchy defined for the DINOSAUR. The trajectory of each handle is filtered to create cartoon style motion. To maintain existing contact constraints, the foot trajectories are not filtered. Please see the

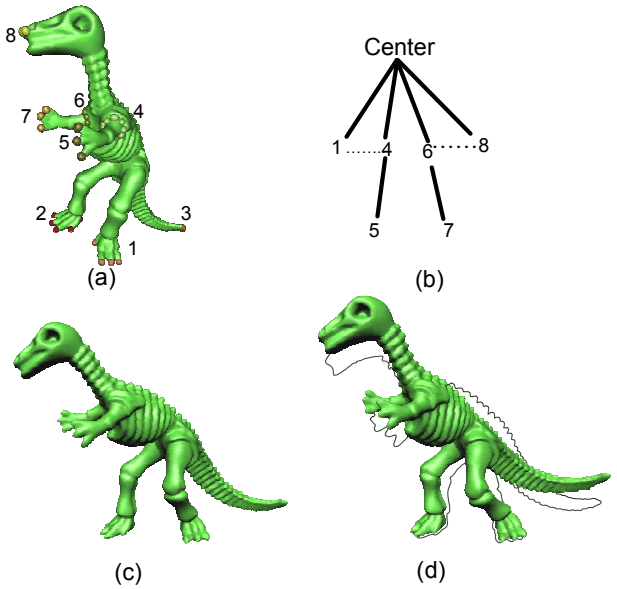

Figure 9: *Handle Hierarchy and filtering (handles have distinct colors.) (a) A* DINOSAUR *model with user-specified handles. (b) An example hierarchy among the handles. (c) One frame in the original deforming mesh sequence. (d) Filtered result for this frame with the silhouette of the original shape in the background.*

accompanied video for the filtered sequence.

4.5 Spacetime Morphing

Spacetime morphing is a novel application based on our deforming mesh sequences editing algorithm. It morphs a source deforming mesh A^s to a target deforming mesh A^t in terms of both shape and deformation. For example, given a walking dinosaur as the source mesh and a walking lion as the target mesh, spacetime morphing tries to generate a new sequence, where the shape of the dinosaur is morphed into the shape of the lion while the dinosaur walking transits to the lion walking.

In a preprocessing step, both deforming sequences are remeshed so that every pair of corresponding frames have the same topology. To achieve this, we first perform cross-parametrization between the first pair of corresponding frames as in [Kraevoy and Sheffer 2004], and then bind the resulting meshes to their respective control meshes by computing mean value coordinates for their vertices: $\tilde{\mathbf{V}}^{f,0} = \mathbf{W}^0\mathbf{V}^{c,0}$, where $\tilde{\mathbf{V}}^{f,0}$ represents the set of vertices from a resulting mesh, $\mathbf{W}^0$ is the matrix of mean value coordinates, and $\mathbf{V}^{c,0}$ is the set of vertices from its control mesh. Then every subsequent frame is generated as follows: maintain the same topology as the cross-parameterized mesh at the first frame and compute the new vertex positions using the formula, $\tilde{\mathbf{V}}^{f,k} = \mathbf{W}^0\mathbf{V}^{c,k}$, where $\tilde{\mathbf{V}}^{f,k}$ represents the new vertex positions at the k-th frame, and $\mathbf{V}^{c,k}$ represents the vertices of the control mesh at the k-th frame. Note that $\mathbf{V}^{c,k}$ was computed as in Section 3.2 using the original mesh topology.

Before start of morphing, we designate a temporal morph interval for A^s and A^t respectively. Let us assume it is (f_1^s, f_2^s) for the source and (f_1^t, f_2^t) for the target. Note that the number of frames in the two intervals can be different. Second, we align the two mesh sequences in the world coordinate system. It is achieved by computing an optimal rigid transformation to make the mesh at f_1^t align with the mesh at f_1^s, or it can be specified by user. The computed transformation is then applied to every frame in A^t to align the two sequences. The final step is to blend the portion of the two sequences inside the morph intervals. For each blended frame, we calculate a parameter t ($\in [0,1]$) to sample a source and target mesh from (f_1^s, f_2^s) and (f_1^t, f_2^t) respectively. The local frames and Laplacian differential coordinates of the sampled meshes are interpolated using a method similar to the one presented in [Lipman et al. 2005], and a mesh for the blended frame is reconstructed from the interpolated Laplacian coordinates.

We need to pay attention to constraints, such as footprints in a walking motion, in spacetime morphing as well. In the morph interval, we mark the frames where a constraint should be satisfied and maintain that constraint when we solve the meshes for the blended frames.

Figure 10 demonstrates spacetime morphing. We concatenate dinosaur walking with lion walking by morphing the DINOSAUR into the LION in one and half walking cycles (starting out with right foot). The middle three images in figure 10 show how the shape and motion of the two original sequences are blended to generate the final results.

5 Results

We have demonstrated a variety of deforming mesh sequence editing results throughout the paper. Prior to interactively editing a deforming mesh sequence, we need to transfer the control mesh at the first frame to all frames, and compute the mean-value matrix from the control mesh to the fine mesh for every frame. Laplacian matrices necessary for minimizing Eq. (4) are precomputed. Once the user have specified handles, we obtain sparse coefficient matrices resulting from the steps in Section 3, and precompute the factorization of such sparse matrices for every frame.

An editing session has both online and offline stages. During online keyframe editing, our system provides interactive frame rates. This is achieved by using the subspace solver to show approximate initial solutions quickly. The alternating least-squares solver is activated by the user only when the subspace solver cannot produce good results. After keyframe editing, we perform offline computation to obtain the entire edited sequence because there are usually more than 100 frames in the sequence and we need to solve for every frame.

In the current implementation, we use UMFPACK [2005] to perform LU decomposition of the matrices. Table 1 summarizes the statistics and timing for the deforming mesh sequences used in this paper. The performance of our system can be much improved if we switch to faster sparse linear solvers, such as those in [Davis 2006; Shi et al. 2006].

6 Conclusions and Future Work

We have successfully generalized gradient domain static mesh editing to deforming mesh sequences editing. Given sparse and irregularly distributed constraints at keyframes throughout the deforming sequence, our system first adjusts the meshes at keyframes to satisfy the constraints, and then propagates the constraints and deformations at keyframes to the entire sequence to generate new deforming mesh sequences. Our framework enables several novel applications, including footprint editing, path editing, handle based deformation mixing, deformation filtering and space-time morphing. In addition, an alternating least-squares method has been developed to obtain high-quality deformation results at keyframes.

Since the problem scale of deforming mesh sequences editing is much larger than static mesh editing. It is necessary to develop novel acceleration techniques to achieve fast response. Even though we have employed precomputation to save time during interaction, the solution time for an entire sequence is still reasonably long.

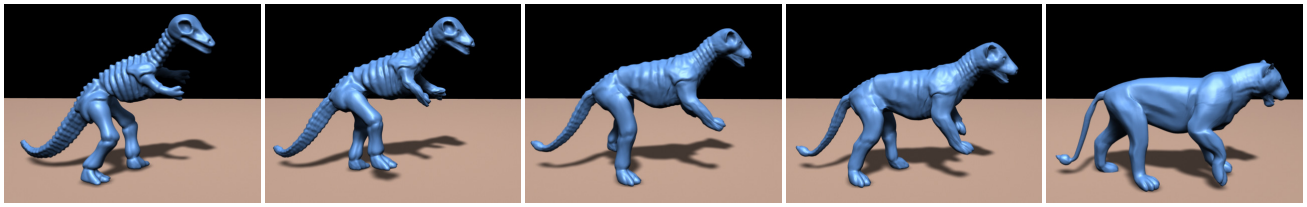

Figure 10: *Spacetime morphing. Dinosaur walking is morphed into lion walking.*

Mesh	Triangles	Editing	Frames	Precomputation	Solve	Session Time
box	1664	Basic editing	20	1.3s	0.2s	30s
cloth	5000	Running	87	$\sim$ 12s	6.15s	$\sim$ 8m
Dinosaur-1	8476	Footprint Editing && Path editing	154	$\sim$ 3m	28.14s	$\sim$ 15m
Dinosaur-2	20k	deformation filtering	190	$\sim$ 22m	108.61s	$\sim$ 5m
frog	23k	Footprint editing	187	$\sim$ 25m	201.43s	$\sim$10m
horse	29k	Footprint editing & Path editing	420	$\sim$50m	30m	$\sim$ 1 hour

Table 1: *Statistics and timings. All the timing data are measured on a 3.2GHz Intel Xeon workstation with 4GB memory. Precomputation time includes the time for control mesh transfer and factorization of sparse matrices. Session time means user interaction time. Since the horse sequence is too large to entirely load into the physical memory, the timing for this example includes virtual memory paging. The entire interactive session took about 60 minutes because of the complexity of this editing scenario.*

A subspace or multigrid method built simultaneously on the spatial and temporal domains should be able to significantly shorten the solution time. Meanwhile, the reconstruction of intermediate frames can be easily made multiple times faster by exploiting multithreading on multicore processors.

Acknowledgements

The authors would like to thank Lincan zhou and Kit Thambiratnam for their help in video production. Special thanks to Junzhi Lu and Shanchuan Huang for preparing the mesh animation data. We are also very grateful to the anonymous reviewers for their helpful suggestions and comments to improve the paper. The boxing and running sequence used in deformation mixing were obtained from CMU motion capture database. Qunsheng Peng is partially supported by the 973 program of China (No. 2002CB312101).

References

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *SIGGRAPH 2000 Conference Proceedings*, 157–164.

ALEXA, M. 2003. Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 2, 105–114.

AU, O. K.-C., TAI, C.-L., LIU, L., AND FU, H. 2006. Dual laplacian editing for meshes. *IEEE Transactions on Visualization and Computer Graphics 12*, 3, 386–395.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: Coupled prisms for intuitive surface modeling. 11–20.

DAVIS, T. A. 2005. Umfpack version 4.4 user guide. Tech. Rep. TR-04-003, University of Florida.

DAVIS, T. A. 2006. User guide for cholmod. Tech. rep., University of Florida.

DER, K., SUMNER, R., AND POPOVIĆ, J. 2006. Inverse kinematics for reduced deformable models. *ACM Transactions on Graphics 25*, 3, 1174–1179.

GLEICHER, M. 2001. Motion path editing. In *Symposium on Interactive 3D Graphics*, 195–202.

GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution signal processing for meshes. In *Proc. SIGGRAPH'99*, 325–334.

HORN, B. 1987. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A 4*, 4, 629–642.

HUANG, J., SHI, X., LIU, X., ZHOU, K., WEI, L.-Y., TENG, S.-H., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. Subspace gradient domain mesh deformation. *ACM Transactions on Graphics 25*, 3, 1126–1134.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. 2005. As-rigid-as-possible shape manipulation. *ACM TOG 24*, 3, 1134–1141.

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

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics 24*, 3, 561–566.

KIRCHER, S., AND GARLAND, M. 2006. Editing arbitrarily deforming surface animations. *ACM Transactions on Graphics 25*, 3, 1098–1107.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. SIGGRAPH'98*, 105–114.

KRAEVOY, V., AND SHEFFER, A. 2004. Cross-parameterization and compatible remeshing of 3d models. *ACM Transactions on Graphics 23*, 3, 861–869.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Transactions on Graphics 24*, 3.

LIPMAN, Y., COHEN-OR, D., GAL, R., AND LEVIN, D. 2006. Volume and shape preservation via moving frame manipulation. *Technical report*.

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

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Transactions on Graphics 24*, 3.

PARK, S., AND HODGINS, J. 2006. Capturing and animating skin deformation in human motion. *ACM Transactions on Graphics 25*, 3, 881–889.

SHEFFER, A., AND KRAEVOY, V. 2004. Pyramid coordinates for morphing and deformation. In *Second International Symposium on 3D Data Processing, Visualization, and Transmission*, 68–75.

SHI, L., YU, Y., BELL, N., AND FENG, W.-W. 2006. A fast multigrid algorithm for mesh deformation. *ACM Transactions on Graphics 25*, 3, 1108–1117.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Symposium of Geometry Processing*.

SUMNER, R., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Transactions on Graphics 23*, 3, 397–403.

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

WANG, J., DRUCKER, S., AGRAWALA, M., AND COHEN, M. 2006. The cartoon animation filter. *ACM Transactions on Graphics 25*, 3, 1169–1173.

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. *ACM Transactions on Graphics (special issue for SIGGRAPH 2004) 23*, 3, 641–648.

ZAYER, R., RÖSSL, C., KARNI, Z., AND SEIDEL, H.-P. 2005. Harmonic guidance for surface deformation. *Computer Graphics Forum (Eurographics 2005) 24*, 3.

ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2005. Large mesh deformation using the volumetric graph laplacian. *ACM Transactions on Graphics 24*, 3, 496–503.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive mutiresolution mesh editing. In *SIGGRAPH 97 Proceedings*, 259–268.