Interactive Editing of Deformable Simulations

Jernej Barbič
University of Southern California

Funshing Sin
University of Southern California

Eitan Grinspun
Columbia University

Abstract

We present an interactive animation editor for complex deformable object animations. Given an existing animation, the artist directly manipulates the deformable body at any time frame, and the surrounding animation immediately adjusts in response. The automatic adjustments are designed to respect physics, preserve detail in both the input motion and geometry, respect prescribed bilateral contact constraints, and controllably and smoothly decay in spacetime. While the utility of interactive editing for rigid body and articulated figure animations is widely recognized, a corresponding approach to deformable bodies has not been technically feasible before. We achieve interactive rates by combining spacetime model reduction, rotation-strain coordinate warping, linearized elasticity, and direct manipulation. This direct editing tool can serve the final stages of animation production, which often call for detailed, direct adjustments that are otherwise tedious to realize by re-simulation or frame-by-frame editing.

CR Categories: I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

Keywords: animation, edit, interactive, contact, physically based

Links: DL PDF WEB VIDEO

1 Introduction

From a *technical* perspective, the animation of deformable objects is often framed as a problem in *forward simulation*, where dynamical trajectories evolve subject to initial and boundary conditions. From an *artistic* perspective, animation is usually framed as a problem of *design*, an iterative process of evaluation and revision seeking a balance among multiple competing criteria, both quantitative ("obeys physical laws") and qualitative ("a matter of taste that is hard to put into words, let alone algorithms").

In an iterative design processes, the *cost*—in time, effort, and resources—of making a revision can drastically influence the nonlinear path in design space chosen by the artist. In essence, the artist is continuously balancing what they want against what they can achieve, each consideration influencing the other. Forward simulations on their own are a severely limited tool for revising an animation: the most minor revision requires a complete simulation run; and it may be tedious or impossible to set the simulation parameters to obtain exactly the desired revision. At the other end of the spectrum, direct keyframe editing is also too limited for revising complex animations, as it affords complete artistic control but burdens the artist with maintaining physical plausibility.

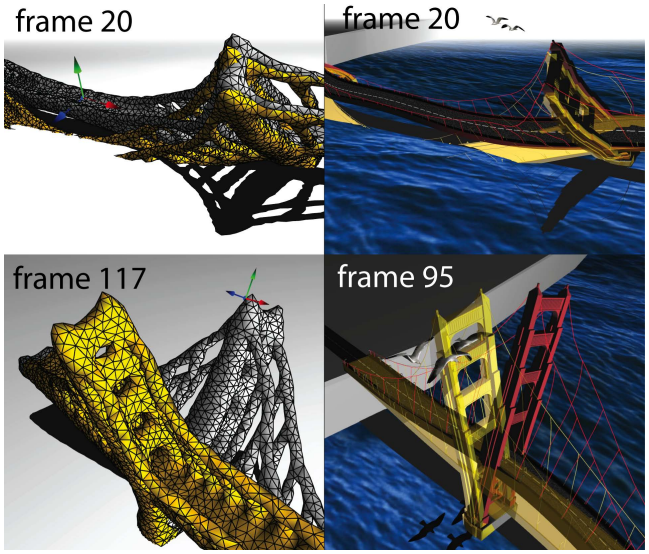

Figure 1: Adjusting complex input motion: *The bridge motion is edited interactively (21 fps) to correct unwanted collisions with the ocean (frame 20) and two groups of scripted seagulls (one visible on frame 95). Eight vertices were adjusted in an editing session that lasted 5 minutes. The input motion was computed using a FEM simulation, and the edits preserve its complexity. Input motion is shown yellow. Editing is performed on the tetrahedral simulation mesh; the right view shows the embedded rendering triangle mesh.*

What is needed is a tool that provides the immediacy of interaction of keyframe editing, with the automatic "physical understanding" that is built into forward simulations. We develop such a tool with the goal of revising complex deformable object animations at interactive rates. Given an input animation, our system enables artists to select and drag an arbitrary vertex (or set of vertices) at arbitrary animation frames, while the system automatically adjusts the entire animation to accommodate the edit and keep the animation as true to the original as possible. As the user drags a vertex, the entire shape at that moment of time, and at previous and future times, automatically reconfigures in a manner that satisfies several configurable criteria, including (a) preservation of detail in the input animation, likely laboriously created using previous resource-intensive forward simulations; (b) spatiotemporal smoothness and locality of any introduced adjustments; (c) introduction of physics-based secondary motion resulting from the edit (see Figure 2). The feedback to the user is instant and interactive. To achieve this, we introduce *spacetime Green's functions* that generalize spatial Green's functions [James 2001] to animations. Such Green's functions reduce editing feedback to a simple superposition of a few basis animations and can be quickly recomputed using incremental solvers each time the set of user-manipulated vertices changes.

2 Related Work

Animation editing is a "high-dimensional" problem, in that it requires editing of many shapes, one at each instant in time. Witkin and Popović [1995] edited animations by applying spline interpola-

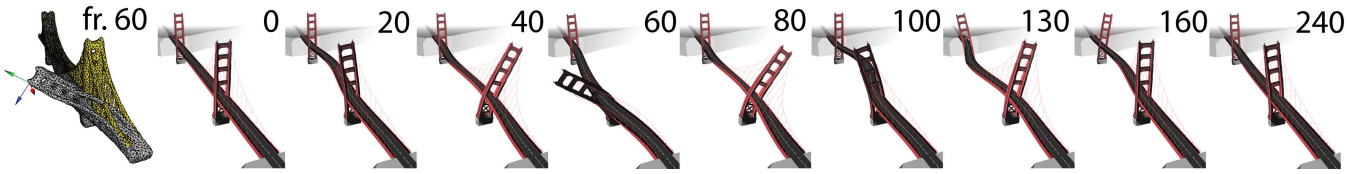

Figure 2: Spacetime Green's function *corresponding to constraining a single mast vertex at frame 60. Secondary motion (e.g., mast pre-swinging, natural bridge pathway deflections, swinging of the other mast) is clearly visible and appears automatically in the Green's function. User selected the vertex at frame 60 and pulled the vertex handle in the direction perpendicular to the mast; no other effort was necessary. Bridge example (*31,746 *tetrahedra,* 1,000 *modes,* 240 *frames), zero input animation, free end boundary condition, 21 fps.*

tion and warping to keyframed edits. Gleicher [1997] introduced interactive animation editing for articulated skeletons, a line of work that has continued even recently [Lee and Shin 1999; Tak et al. 2002; Min et al. 2009; Sok et al. 2010]. Popović et al. [2000; 2003] introduced interactive editing for rigid body animations. These techniques employ incremental updates to enforce constraints at interactive rates, as underscored in the seminal *differential manipulation* of Gleicher and Witkin [1991]. We build on these methods, confronting the challenge that the full space of shape *evolutions* of a deformable body is intractable for interactive animation editing.

While we focus on interaction as the key to rapid design, earlier methods considered other usage patterns to facilitate animation design. *Tracking solvers* [Bergou et al. 2007] begin with a given coarse simulation and enrich the output with additional physical detail in an offline simulation. *Proportional-derivative controllers* [Kondo et al. 2005] use a forward simulation augmented with forces that guide the object to prescribed temporal keyframes. For deformable objects with an embedded skeleton, a PD controller can provide constraints on instantaneous joint accelerations, which can serve as an intuitive interface for creating animations [Kim and Pollard 2011b; Kim and Pollard 2011a]. *Spacetime constraints* [Witkin and Kass 1988; Popović and Witkin 1999; Fang and Pollard 2003; Wojtan et al. 2006] pose control as an optimization problem, which may be accelerated via coarse-to-fine and windowing techniques [Cohen 1992; Liu et al. 1994], the *adjoint method* [McNamara et al. 2004], and *reduced control* [Safonova et al. 2004; Barbič and Popović 2008; Barbič et al. 2009]. While previous reduction-based accelerations required that the *entire* input animation lies in the reduced space, our method asks that the *adjustment* to the animation lies in the reduced space*time*, enabling the editor to preserve input detail. One could perform simple edits using a real-time tracking optimal controller [Barbič and Popović 2008], by tuning perturbation forces in spacetime. In our work, the user can prescribe vertex positions, which provides more direct control over the animation. Furthermore, with tracking controllers, one must perform an entire (reduced) simulation run each time a force is altered, whereas our editing feedback is instant.

The principle of *interaction* is deeply embedded into techniques for *static* shape modeling [Zorin et al. 1997; Alexa et al. 2000; Lipman et al. 2004; Botsch et al. 2006; Gal et al. 2009]; some static modeling tools rely on deformable model simulation [Umetani et al. 2011]. Animations can be created by interpolating keyframes obtained using such static techniques, either using spline interpolation, or by properly interpolating rotations using as-rigid-as-possible interpolation [Alexa et al. 2000; Sumner and Popović 2004; Botsch et al. 2006]. Such geometric techniques, however, cannot provide dynamics in the edits.

In 2008, Kass and Anderson [2008] introduced *wiggly splines*: complex-number-valued 1D modal oscillators that incorporate not only the usual cubic spline controls but also parameters corresponding to phase, resonance, and damping. Artists can use wiggly splines to interactively apply dynamic small-deformation adjustments to animations, by tuning individual 1D oscillators. Huang et al. [2011] employed modal oscillators for dynamic *interpolation* of deformations, and introduced *rotation-strain warping* to avoid linearization artifacts under large edits. We build on these works: to support direct vertex manipulation, contact constraints, and edits that are both large and high-dimensional, the oscillators must be coupled. We demonstrate how to achieve such coupling, at interactive rates, by combining our novel spacetime Green's functions with spatial model reduction and warping.

3 Manipulation Objective

The input to our system is (i) the rest pose of a deformable object represented by a collection of n vertices, (ii) a mass matrix M and stiffness matrix K associated with this rest pose, and (iii) a sequence of object deformations comprising an animation. The mesh may be anchored (a subset of the vertices is fixed to the rest position), or unanchored. The mass and stiffness matrices are easily obtained using standard simulation tools (e.g., FEM, mass-spring systems, thin shells, cloth solver). Our approach relies on these three inputs as "black boxes" and does not impose additional physical structure on the input animation; thus animations produced by hand (keyframing, skinning, etc.) may also be edited with our tool.

Notation: Denote the sequence of input frames by $\bar{u}_0, \ldots, \bar{u}_{T-1}$, likewise the output by $u_0, \ldots, u_{T-1}$. Each frame $\bar{u}_i \in \mathbb{R}^{3n}$ records vertex *displacements* relative to a time-invariant rest pose. The difference $q_i = u_i - \bar{u}_i$, between outputs and inputs is the *realized edit*. The realized edit observed by the user is not computed directly by our method; instead, we first internally compute a *linearized edit* $p_i \in \mathbb{R}^{3n}$ from which we later compute $q_i \in \mathbb{R}^{3n}$ via warping (§6).

Spacetime Objective: We begin by posing a quadratic function that evaluates the cost of an edit. We postulate that the input animation is "natural:" without user constraints, the input animation is our output. Our technique supports arbitrary quadratic energy functions of the linearized edit sequence $\{p_i\}$. We employ a function that favors edits that are closest to physical, by minimizing the residual forces f in linearized elastic equations of motion $Mp'' + Dp' + Kp = f$ (see, e.g., [Shabana 1990], p. 301). After discretizing p'' and p' in time, we obtain our energy,

$$E = \frac{h}{2} \sum_{i=1}^{T-2} || \frac{1}{h^2} M \hat{p}_i + \frac{1}{h} D(p_{i+1} - p_i) + K p_i ||^2_{M^{-1}} , \quad (1)$$

where the hat operator is $\hat{p}_i = p_{i+1} - 2p_i + p_{i-1}$ and h is the timestep. We employ Rayleigh damping, $D = \alpha M + \beta K$, with tunable parameters $\alpha \geq 0$ and $\beta \geq 0$. As standard in mechanics, we weight the forces by M^{-1}. This energy generalizes the spline energy proposed by Kass and Anderson [2008] from a single damped harmonic oscillator to a linearized high-dimensional elastic system.

In control terminology, our energy E corresponds to a fully actuated system (all forces are permitted). Therefore, a solution to the given user constraints always exists and can be found quickly. We considered underactuating our edits, say, by restricting forces to a subset of "muscle" forces. However, one then has to optimize underactuated systems, which pose a fundamentally more difficult problem. In the initial stages of our research, we also considered energy functions where the Kp_i term is replaced by nonlinear elasticity. Although producing improved large edits, such computation is very complex: at interactive rates, it did not scale beyond examples with approximately 20 vertices.

Reduction: We accelerate computation by restricting the linearized edits to a subspace of r modes: we set $p_i = Uz_i$, where the columns of $U \in \mathbb{R}^{3n\times r}$ form the reduced basis, and $z_i \in \mathbb{R}^r$ is the coordinate vector of the edit with respect to the reduced basis. To obtain the reduced basis, we solve the generalized eigenproblem

$$Kx_j = \lambda_j M x_j$$

for the eigenvectors $\{x_j\}$ and corresponding ascending eigenvalues $\{\lambda_j\}$. We truncate the eigenbasis, retaining the first r vibration modes as columns of $U = (x_1 \dots x_r)$, and the corresponding eigenvalues in the diagonal matrix Λ. We typically retain several hundred modes (see Figure 3), and our technique scales well with increasing the number of modes (see Figure 5).

Applying $p_i = Uz_i$ to (1) yields the reduced coordinates objective

$$E = \frac{h}{2}\sum_{i=1}^{T-2} ||\frac{1}{h^2}\hat{z}_i + \frac{1}{h}(\alpha I + \beta\Lambda)(z_{i+1} - z_i) + \Lambda z_i||^2. \quad (2)$$

We obtained this simple form by noting that eigenvectors are mass-orthonormal, $U^T MU = I$, and they diagonalize the stiffness matrix, $U^T KU = \Lambda$. The truncated reduction has thus yielded a system of r damped harmonic oscillators. While they are decoupled in (2), we will need to address the fact that they are coupled by imposed user and contact constraints (§4).

We can now assemble linearized edits p_i into a vector $p = [p_0^T, \dots, p_{T-1}^T]^T \in \mathbb{R}^{3nT}$, and reduced coordinates into a vector $z = [z_0^T, \dots, z_{T-1}^T]^T \in \mathbb{R}^{rT}$. Then, at $p = 0$ (input animation), both E and its gradient with respect to p are zero. We can therefore succinctly rewrite (2) as the quadratic form

$$E = \frac{1}{2} z^T H z. \quad (3)$$

The symmetric positive-definite pentadiagonal Hessian matrix $H = d^2E/dz^2 \in \mathbb{R}^{rT\times rT}$ is given in Appendix A. It is constant at runtime and precomputed at startup from the undeformed mesh pose: it only depends on the eigenvalues of the input rest mesh, damping, the timestep, and the number of frames T.

Given the user-imposed vertex position constraints, at arbitrary frames, our system finds linearized edits $p_0, \dots, p_{T-1}$ that minimize E under those constraints. Because the system is linear, we precompute the solutions using spacetime Green's functions whenever the set of constrained vertices changes, and then merely superimpose the Green's functions as the user drags a vertex. We explain this process in the next section.

4 Manipulation and Green's Function

Animation degrees of freedom are ordered pairs (i, j), where i is the frame index and j is the degree of freedom in the mesh. Formally, the set of all animation DOFs is $\mathcal{D} = \{0, 1, \dots, T-1\} \times \{0, 1, \dots, 3n-1\}$. The user manipulates the animation by selecting degrees of freedom (x, y, z vertex deformations), at arbitrary frames, and dragging their positions with the mouse (see Figure 4). Any degree of freedom can be selected or de-selected, in arbitrary order. As the user drags a selected degree of freedom, the other selected DOFs keep their prescribed values. Note that there is no need for the animator to craft entire frames; sparse input in spacetime is sufficient. Our outputs were typically created by manipulating about 10-30 degrees of freedom. In our implementation, the user selects vertices, and we always select all three degrees of freedom

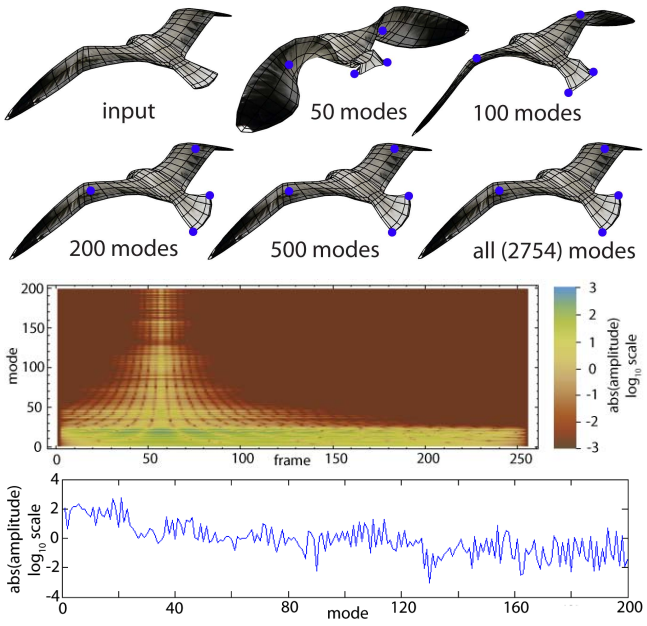

Figure 3: Convergence under an increasing number of modes: *We edited a 256-frame seagull animation, by imposing a detailed edit that twists the tail. Top: frame 56, from an animation computed using a varying number of linear modes. Solutions with 200 and 500 modes are visually very similar and identical to a full solution, respectively. Blue points are user constraints. Middle: the temporal diagram of absolute values ($\log_{10}$ scale) of modal amplitudes in a solution computed using a full basis, for first 200 modes. It can be seen that the output is high-dimensional with modes coupled in a non-trivial way. Our method automatically provides the coupling. Bottom: the vertical slice at frame=56.*

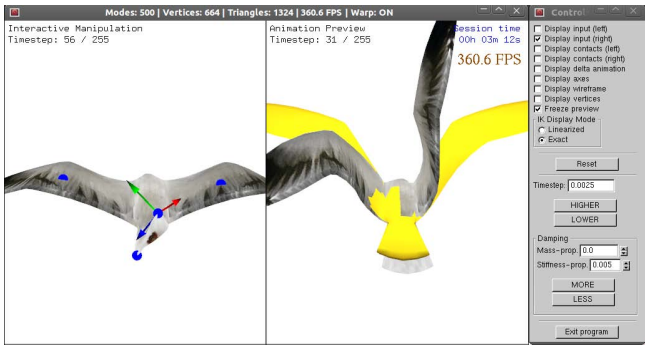

Figure 4: Our user interface: *Left part of the window permits the user to scroll to any frame and add, remove or adjust vertex constraints. On the right, the current output animation is continuously displayed. Seagull example, 500 modes, 255 frames.*

of a vertex simultaneously; we provide a coordinate-axis handle for easier manipulation (see Figure 4).

Formally, at any moment during the manipulation, the set $\mathcal{D}$ consists of two disjoint parts: the set of free DOFs $\mathcal{F}$, and the set of manipulated DOFs $\mathcal{M}$. Analogously, we can decompose the linearized edit vector p into $p^{\mathcal{M}} \in \mathbb{R}^c$ and $p^{\mathcal{F}} \in \mathbb{R}^{3nT-c}$. Positional constraints imposed by the user are linear constraints, $Cz = p^{\mathcal{M}}$, where the sparse matrix $C \in \mathbb{R}^{c \times rT}$ constrains the corresponding rows of the modal matrix U, positioned at the frame of the constraint. We obtain our reduced coordinate edit z by minimizing (3) subject to $Cz = p^{\mathcal{M}}$. Applying the method of Lagrange multipliers to the constrained minimization yields the linear system

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} z \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ p^{\mathcal{M}} \end{bmatrix}. \tag{4}$$

In order to accelerate runtime computation, we can precompute the solution to a unit change of each DOF of $\mathcal{M}$, by solving for the Green's functions matrix $\mathcal{G} \in \mathbb{R}^{rT \times c}$

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \mathcal{G} \\ \lambda' \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix}. \tag{5}$$

Each column of $\mathcal{G}$ is a *spacetime* Green's function: it gives the change in the entire animation as a result of displacing the x, y or z coordinate of a manipulated vertex. Such world-space constraints automatically produce edits where hundreds of modes activate in non-trivial patterns (see Figure 3); this distinguishes our method from earlier approaches that required users to manually tune individual oscillators. Green's functions automatically include secondary motion (see Figure 2); with traditional techniques, such detail would need to be keyframed manually.

Green's functions are globally supported both in space and time. If desired, they can be restricted to a submesh by computing the submesh modes under a fixed interface to the rest of the mesh. Restriction in time can be achieved by using a subrange of the input frames. The $(rT+c) \times (rT+c)$ system in (5) must be solved each time $(\mathcal{M}, \mathcal{F})$ changes. Because H is constant, symmetric positive-definite and pentadiagonal, its Cholesky decomposition only has three non-zero diagonals, and can be computed and stored very efficiently. At runtime, we can then efficiently solve (5) using Schur's complement of H, which we update each time a constraint is added or removed (Appendix B). Computing $\mathcal{G}$ is fast, even for hundreds of modes (see Figure 5). As the user is dragging a manipulated vertex, due to linearity, we can simply superimpose Green's functions:

$$z = \mathcal{G} p^{\mathcal{M}}, \qquad p_i = U z_i \quad (\text{for } i = 0, \ldots, T-1). \tag{6}$$

Note that we do not need to construct all frames of z at once, but only z_i and p_i currently previewed on the screen.

Boundary conditions: Equation 5 is singular because the entire animation can undergo a constant frame translation without changing the editing energy. Prior to solving (5), we therefore impose boundary conditions. We use two forms of boundary conditions: (1) imposing zero edits at frames $0, 1, T-2, T-1$, and (2) imposing zero edits at frames $0, 1$ only. The conditions are imposed by removing the corresponding rows and columns from the system matrix in (5). These conditions effectively set the edits and edit velocities to zero at the beginning and optionally the end of the animation. Typically, edits are performed in a temporal window, and condition (1) ensures that any pre-animation and post-animation blends in smoothly in positions and velocities to the edited segment. Condition (2) in turn, yields more vibrant edits, as the end of the animation is free; it is useful, for example, for creating new

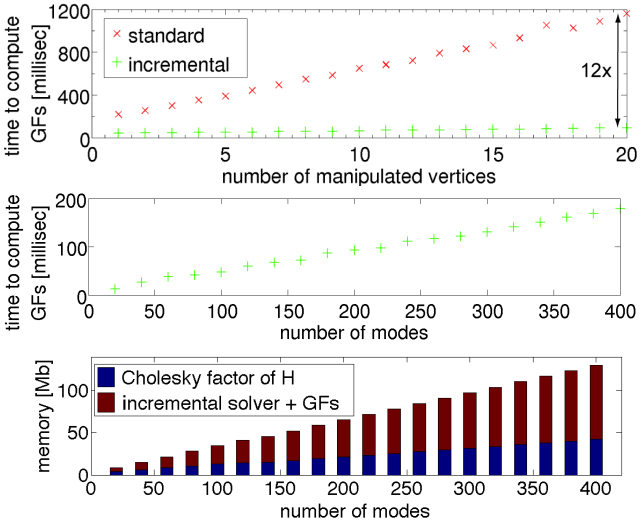

Figure 5: Green's functions can be computed in under 100 msec for complex meshes, long animations and many manipulated vertices: *Top: Bridge example, $r = 200$ modes, $T = 240$ frames. The standard method computes Green's functions by performing Cholesky decomposition on the system matrix of Eq. 5. Our incremental method uses Schur's complement of H (Appendix B) to incrementally update the inverse of the system matrix. Bottom: Computation time and memory under increasing r, for 20 simultaneously constrained vertices. If all modes were used ($r = 26{,}337$), incremental solver takes 24.5 seconds and requires 8.3 Gb of memory (136x slower and 64x more memory than with 400 modes).*

animations, with automatic secondary motion, by editing zero animations. The zero position and velocity boundary conditions could be replaced with a different regularization condition, such as adding an extra energy term that penalizes deviation from input positions or input velocities. Such terms only affect the main diagonal of H, and could be easily incorporated into our method.

5 Contact

Our system can edit input animations with contact. The contacts are preserved by the edits, and are allowed to slide in the contact plane, under the assumption that the set of contacts itself does not change. Each contact is given as a triple (i, v, n), where i is the contact frame, v the contact vertex, and n the contact normal. For each contact, a scalar constraint is formulated that keeps the contact vertex in its plane of contact. These one-dimensional constraints are then simply added to the constraints C in (5), in an identical way as the user constraints. Because the contacts never change during editing, we can compute Schur's complement of H plus the contact part of C, at startup. Our formulation also supports constraints on relative motion of two vertices, which can be used to preserve self-contact. Optionally, vertices can also be pinned to their input locations, to accommodate, e.g., no-slip contact conditions.

Determining contacts: Physically based simulations can use any collision detection and contact resolution method. The contacts should be passed as input to our editor alongside with the frame deformations. In some practical scenarios, contact information is not readily available, e.g., with commercial cloth solvers that do not explicitly expose contact information. In such cases we determine the contacts by performing collision detection on our input frames.

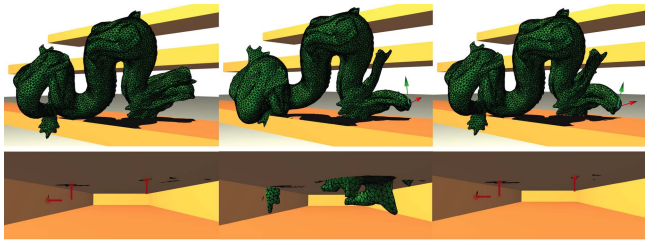

Figure 6: Editing simulations with contact: *The user adjusted the mouth deformation of this dragon tumbling down a staircase. Sliding contact constraints were used. Bottom row shows the view from inside the stair, including contacts (red points) as determined by our clustering algorithm and the contact normals. When contact handling is disabled, large penetrations occur, whereas with contact handling, contacts are preserved.*

Contact clustering: Adjacent contact vertices in space or time usually represent the same contact site. We remove such redundant contacts using a spacetime clustering of contacts. First, we form a graph that consists of T replicate copies of the mesh vertices; two nodes are adjacent if they have the same frame index and are mesh neighbors, or if they are identical vertices and neighbors in time. The contacts form a graph subset, and we set our clusters to its connected components. We replace each cluster by a single contact (i, v, n), where i is the nearest frame to the mean cluster time, v is the nearest mesh vertex to the mean of cluster vertices, and n is the (normalized) mean cluster normal. Such clustering reduced the number of contacts from $92{,}742$ to 410 in our dragon example (Figure 6). Thresholds could be set to avoid excessively large clusters. More advanced clustering strategies could further reduce the number of representative contacts. We analyze Green's function computation time under a progressive number of contacts in Figure 7. Clustering decreases computation time and improves controllability. A small basis may be unable to resolve contact points that are nearby in space and time, leading to well-known "ringing" artifacts of modal methods. To some degree, this is alleviated by the pseudoinverse in the incremental solver. It is advisable, however, that model reduction goes hand in hand with *contact reduction*.

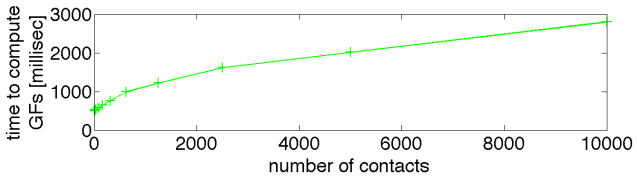

Figure 7: Contact clustering decreases Green's function computation time. *Dragon example. One manipulated vertex. Solver runs out of memory with approximately 15,000 contacts.*

6 Warping Implementation

The previous sections presented a complete editing system. The edits, however, would exhibit linearization artifacts when $p^{\mathcal{M}}$ is large (Figure 8, middle row). We employ the warping proposed by Huang et al. [2011], providing only a brief summary here, and pointing out how to handle contact constraints and input animation efficiently. Because of warping, the Hessian matrix H, computed at the time-invariant rest shape (Equation 3), is suitable even for large *input* deformations; but its expressiveness diminishes under large *edits* (where a re-linearization would be beneficial). Effectively,

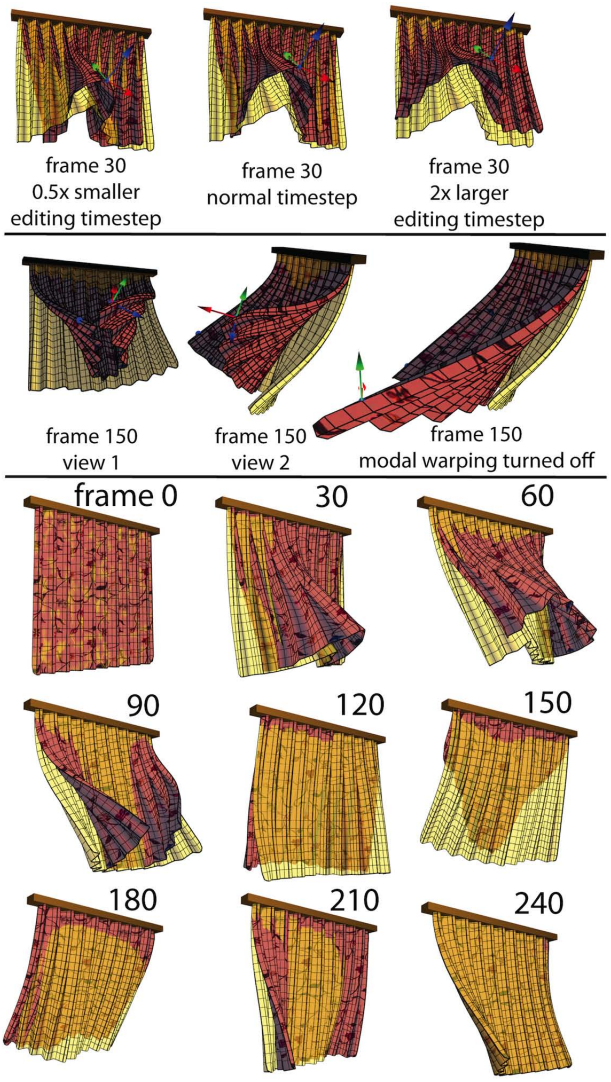

Figure 8: Editing cloth simulation: *The user adjusted three vertices (one at frame 30, two at 150) to make the cloth swing higher and flap its corners. Input simulation is shown yellow, output is shown red. $T = 240, r = 200$. Input motion was computed using the cloth simulator of Baraff and Witkin [1998]. Editing timestep need not match the simulation timestep; top row shows the effect of adjusting the editing timestep. Edits become spatially more global with larger editing timesteps. Middle row shows artifacts when warping is disabled. Bottom row shows selected animation frames. Secondary motion due to edits at frame 150 are clearly visible on frames 90 and 210. Frames $0, 1, 239, 240$ were constrained to zero edit (§ 4), so that the initial and final output position and velocity match the input; thus, the edit can smoothly blend into the input motion before and after the temporal editing window.*

our system is editing deformations around the time-invariant rest shape, whereas warping properly rotates and deforms them relative to input deformations. Initially, we attempted to precompute H and perform warping by linearizing about each input frame rather than the rest shape, but the long precomputation times and greater memory requirements made such an approach impractical.

Warping begins by transforming displacements $v \in \mathbb{R}^{3n}$ (measured from the rest configuration) to *rotation-strain coordinates*. In our

work, we will apply warping to $v = \bar{u}_i + p_i$, for all frames i, but the principle is general. Let $G^j \in \mathbb{R}^{9\times 3n}$ be the usual discrete gradient operator of tetrahedron j [Huang et al. 2011], i.e., the 3×3 deformation gradient of tetrahedron j equals $I + G^j v$. We decompose $G^j v$ into symmetric and antisymmetric components, $G^j v = (G^j v + (G^j v)^T)/2 + (G^j v - (G^j v)^T)/2$. Denoting the upper triangle of the symmetric part as $y^j_\epsilon \in \mathbb{R}^6$, and the skew-vector corresponding to the antisymmetric part as $y^j_\omega \in \mathbb{R}^3$, we assemble the rotation-strain coordinates $[y^j_\epsilon, y^j_\omega]$ for all tetrahedra in the vector $y(v) \in \mathbb{R}^{9e}$. Huang et al. computed y by superimposing precomputed modal vectors, whereas we directly compute the sparse product Gv, where $G \in \mathbb{R}^{9e\times 3n}$ is the gradient matrix assembled from all G^j. The warped displacements are the minimizers u of the constrained Poisson reconstruction quadratic objective

$$\sum_{j=1}^{e} V_j ||I + G^j u - \exp\big(y^j_\omega\big)\big(I + \widetilde{y^j_\epsilon}\big)||^2_F = ||VGq - b||^2_2, \quad (7)$$

where $\tilde{x}$ is the 3×3 symmetric matrix corresponding to the upper-triangle $x \in \mathbb{R}^6$, $\exp(w) \in \mathbb{R}^{3\times 3}$ is the rotation matrix corresponding to the rotation by angle $|w|$ around the axis w, $||\ ||_F$ denotes the Frobenius norm, V_j is the volume of tetrahedron j, and $V = \mathrm{diag}(\sqrt{V_1}, \sqrt{V_1}, \ldots, \sqrt{V_e})$ (each entry repeated 9x). The 9-block of vector $b \in \mathbb{R}^{9e}$ corresponding to tetrahedron j, expressed as a row-major 3×3 matrix, is

$$b_j = \sqrt{V_j}\big(\exp(y^j_\omega)(I + \widetilde{y^j_\epsilon}) - I\big). \quad (8)$$

Our constraints include pinned vertices for objects that are permanently rooted the ground, and contact constraints, but not the user constraints. For free-flying objects, we add a three-dimensional constraint that keeps the mesh centroid unmodified. The constrained quadratic minimization yields the linear system

$$\begin{bmatrix} L & d^T \\ d & 0 \end{bmatrix} \begin{bmatrix} q \\ \lambda \end{bmatrix} = \begin{bmatrix} (VG)^T b \\ s \end{bmatrix}, \quad (9)$$

where $L = G^T V^2 G$ is the discrete Laplacian of the mesh, d gives the constraints and s are values to be met by the constraints. Each sliding contact constraint contributes one sparse row to d, constraining the vertex in the plane of contact. Non-sliding contact constraints add three sparse rows to d. The centroid constraint also adds three rows to d, consisting of 3×3 diagonal matrices $w_i I$, where w_i is some meaningful weight assigned to vertex i, such as $w_i = 1/n$; rows of s give the input frame centroid.

The matrix L only depends on the input mesh geometry, and not on U or v. The system matrix in (9) is sparse, symmetric and constant, and can be pre-factored, so warping can be performed efficiently at runtime. For simulations with contact, constraints $d = d_i$ are frame-dependent, but do not change at runtime. Because L is constant, we can efficiently precompute Schur's complements for the combined (L, d_i) systems of each frame i, at startup. Because we warp each frame independently of the other frames, temporal discontinuities could appear at frames where the set of contact vertices changes, especially under large deformations of long objects. Although not necessary in our examples, temporal smoothness of large edits with transient contact can be improved by warping such frames twice: with contacts of the previous frame, and the current frame; then blending the difference to the neighboring frames.

Incorporating input frames: We combine input frames $\bar{u}_i$ with linearized edits p_i to produce good-looking output frames u_i under large edits, as follows. Frames $\bar{u}_i$ are arbitrary shapes, not necessarily obtained by warping, or low-dimensional. We exploit the property that the rotation-strain coordinates form a linear space, therefore they are combined simply using addition [Huang et al. 2011].

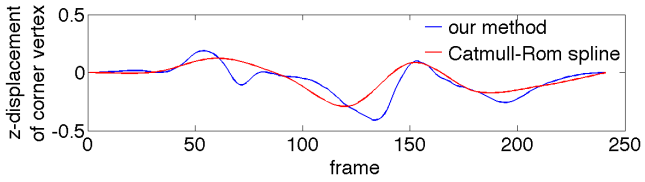

Figure 9: Comparison to splines: *We used Catmull-Rom splines to recreate the curtain motion of Figure 8, by using our output frames* $0, 30, 60, \ldots, 240$ *as keyframes. We plot the change in the z-coordinate of a selected vertex, relative to input motion. Our method produces adjustments that are richer in frequency content, even though the user input consisted of only three vertex constraints, whereas splines used 9 keyframes.*

At startup, we precompute the rotation-strain coordinates $y(\bar{u}_i)$ for all input frames, by performing polar decomposition on the input deformation gradient of every tetrahedron, $F = QS$, where Q is a rotation matrix and S is symmetric, and then taking the log of Q to construct $y(\bar{u}_i)$. The log is computed by converting Q to a quaternion, and then extracting the axis and angle of rotation. The computation of $y(\bar{u}_i)$ is fast (Table 1), and can be done in parallel. To warp a given linearized edit p_i at runtime, we use $y = y(p_i) + y(\bar{u}_i)$ as the rotation-strain coordinates in the Poisson objective. For $p_i = 0$, such warping will give $\bar{u}_i$, as intended. To the best of our knowledge, we are the first to demonstrate the retention of *fine* details in *complex* input motions. For triangle meshes, we follow the method described in [Sumner and Popović 2004], where each triangle is converted into a tetrahedron by adding a fourth vertex in a direction normal to the triangle plane. Free-flying animations are accommodated by precomputing (at startup, using polar decomposition) a global linear translation t_i and rotation R_i that best aligns the rest configuration to each input frame i. Each input frame is transformed by the inverse of this affine transformation, yielding input frames $\bar{u}_i$ to our system. For free-flying objects undergoing contact, the contact normals are transformed by R_i^T at startup.

7 Results

Table 1 gives the performance data for our examples. In our first example (Figure 1), we readjusted an invertible FEM simulation [Irving et al. 2004] so that the bridge does not collide with the ocean and the scripted birds. Automatic secondary motion was shown for solid FEM simulations (bridge; Figure 2) and cloth computed using the Baraff-Witkin cloth simulator [Baraff and Witkin 1998] (curtain example; Figure 8). The dragon input motion was computed using Maya's Digital Molecular Matter plugin [Parker and O'Brien 2009]; this example (Figure 6) demonstrated that our edits can preserve input contact. In the seagull example (Figures 3, 4), we edited an artist-keyframed triangle mesh animation. We used modes and frequencies computed using a FEM model obtained by voxelizing the rest (non-manifold) seagull triangle mesh. In the seagull and curtain examples, we used triangle mesh warping [Sumner and Popović 2004]. All other examples use solid warping.

Detail in the input motion is preserved in all examples. Figure 3 demonstrates convergence to unreduced linear elasticity with progressive r, and non-trivial modal excitation patterns resulting from world-space vertex constraints. Figures 5 and 7 analyzed Green's function computation time and convergence under an increasing number of modes and constraints. Figure 4 depicts our user interface. Figure 9 demonstrates that our method produces edits richer in frequency content than spline interpolation. Finally, we note that

	model	ver-t	tri	ver-v	e	r	T	input	modes	launch	GF	$p_i = Uz_i$	warp	fps
beam	solid	208	412	208	75(V)	500	360	zero anim.	1.9 s	3.5 s	0.29 s	7.8E-05 s	0.00046 s	550
bridge	solid	100,759	186,218	9,244	31,746(T)	1,000	240	1.3 min	774 s	10.1 s	0.41 s	0.019 s	0.028 s	21
seagull	solid	664	1,324	918	432(V)	500	255	artist keyf.	26 s	2.5 s	0.22 s	0.0006 s	0.0038 s	350
dragon	solid	13,367	19,040	13,367	48,010(T)	500	600	257 min	383 s	35 s	0.82 s	0.014 s	0.057 s	10
curtain	cloth	1,281	2,400	–	–	200	240	0.12 min	7.3 s	1.8 s	0.08 s	0.0006 s	0.0080 s	180

Table 1: Simulation statistics *for #triangle mesh vertices (ver-t), #triangles (tri), #volumetric mesh vertices (ver-v), #elements (e, V=voxels, T=tetrahedra), #modes (r), #frames (T), time to compute input motion (input), linear modes (modes), and Green's functions (GF; one vertex constraint), time to launch the editor (launch; including all precomputation), and construct deformations from modal coordinates ($p_i = Uz_i$), warping time (warp), and editing frame rate including rendering (fps). Machine specs: Intel Core i7-980X, 6-Core, 3.33 GHz, 24 GB memory.*

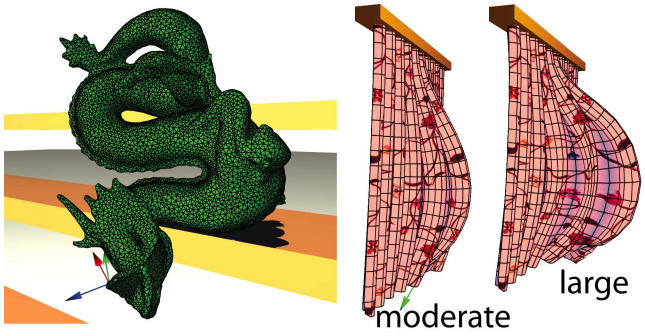

Figure 10: Limitations: *Left: Large edits may introduce self-collisions (dragon's mouth). Middle, right: A bottom-central vertex was pulled up. Warping is on. Under large deformations (right), quadrilaterals grow in size due to linearization, even with warping.*

we also successfully considered an alternative editing energy,

$$E' = \frac{\mu}{2h^3} \sum_{i=1}^{T-2} \hat{p}_i^T M \hat{p}_i + \frac{h}{2} \sum_{i=0}^{T-1} p_i^T K p_i. \qquad (10)$$

This energy seeks a balance between temporal and spatial smoothness of the edits, as controlled by the tunable parameter $\mu \geq 0$. It does not produce wiggles or oscillations and is useful in situations where secondary motion is not required.

8 Discussion

We presented a method for interactive editing and design of deformable object animations. We have found that minimizing the force residual objective (2) works well in our system; however, our acceleration techniques are not specific to this objective, and other interesting quadratic metrics of the edit cost could be substituted. We introduced spacetime Green's functions, which enable instant editing feedback to the user, even with complex animations. The method exploits model reduction and linearization for speed, and incorporates bilateral contact. We found that immediate, real-time, feedback makes it possible to rapidly explore the design space of our method, and greatly improves our animation editing process.

Our Green's functions are real-valued, but could be extended to complex values to directly control phase and overlap [Kass and Anderson 2008]. Speed could be increased further using a hierarchical approach in time [Lee and Shin 1999]. We do not automatically detect and correct new contacts during editing; this task is currently left to the user, but could in the future be addressed using real-time collision detection. Our method aims primarily at moderate edits of existing animations, where it offers direct control and faster turnaround than rerunning a simulation. We demonstrated large edits when the bird wing and tail fin are bent 45 and 60 degrees, respectively; the beam is twisted 720 degrees; we flap the curtain cloth corner by 180 degrees, and bend the bridge mast and dragon's head and mouth over 45 degrees. Going beyond such edits would require unilateral constraints, real-time collision detection across all frames, and arbitrarily nonlinear elastic potentials, aspects we could not accommodate in realtime; see Figure 10 for a demonstration of the limitations. Our modes and dynamics are linearized at the rest configuration of the object. While it is easily possible to "bake-in" the current edits and re-linearize, direct nonlinear extensions of our editing energy are left for future work.

Acknowledgements: This research was sponsored in part by the National Science Foundation (CAREER-53-4509-6600, IIS-11-17257, IIS-10-48948, IIS-09-16129, CCF-06-43268), the James H. Zumberge Research and Innovation Fund at USC, the Sloan Foundation, and generous gifts from Adobe, Autodesk, Intel, and The Walt Disney Company.

A Hessian of the Editing Energy

As the modes are decoupled in the energy (2), the Hessian of E consists of r independent blocks H_i of size $T \times T$, corresponding to the temporal evolution of each modal oscillator i, for $i = 1, \ldots, r$. The pentadiagonal matrix H_i is obtained by "splatting" a temporally constant 3×3 matrix $A = aa^T/h^3$ down the diagonal of H_i, where $a = [1, -2 - d_i h + h^2 \lambda_i, 1 + d_i h]^T$, and d_i, λ_i are the entries of diagonal matrices $\alpha I + \beta \Lambda$ and Λ, respectively. Matrix A is the energy Hessian of an individual mode and timestep.

B Matrix Inverse Update

Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix with a known inverse A^{-1}. Form the matrix

$$\hat{A} = \begin{bmatrix} A & B^T \\ B & C \end{bmatrix}, \qquad (11)$$

where $B \in \mathbb{R}^{m \times n}$, and $C \in \mathbb{R}^{m \times m}$ is symmetric. When m is small, the inverse can be computed efficiently as

$$\hat{A}^{-1} = \begin{bmatrix} A^{-1} + FDF^T & -FD \\ -DF^T & D \end{bmatrix}, \qquad (12)$$

where $D = S^{-1} \in \mathbb{R}^{m \times m}$ for $S = C - BA^{-1}B^T \in \mathbb{R}^{m \times m}$, and $F = A^{-1}B^T \in \mathbb{R}^{n \times m}$. Solving individual systems $\hat{A}x = b$ can be done efficiently even if A^{-1} is not known explicitly; the ability to multiply $A^{-1}x$ efficiently is sufficient, e.g., via known Cholesky factors of A. When (12) is applied to solve the system (5), the computation simplifies to $\mathcal{G} = -FD$. In order to handle matrices B that may be rank-deficient, e.g., arising from (near-)redundant constraints, we compute D using a SVD-based pseudoinverse: we truncate all singular values smaller than $\epsilon\sigma_0$, where $\epsilon = 10^{-6}$ and σ_0 is the largest singular value of S.

References

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

BARAFF, D., AND WITKIN, A. P. 1998. Large Steps in Cloth Simulation. In *Proc. of ACM SIGGRAPH 98*, 43–54.

BARBIČ, J., AND POPOVIĆ, J. 2008. Real-time control of physically based simulations using gentle forces. *ACM Trans. on Graphics (SIGGRAPH Asia 2008) 27*, 5, 163:1–163:10.

BARBIČ, J., DA SILVA, M., AND POPOVIĆ, J. 2009. Deformable object animation using reduced optimal control. *ACM Trans. on Graphics (SIGGRAPH 2009) 28*, 3, 53:1–53:9.

BERGOU, M., MATHUR, S., WARDETZKY, M., AND GRINSPUN, E. 2007. TRACKS: Toward directable thin shells. *ACM Trans. on Graphics (SIGGRAPH 2007) 26*, 3, 50:1–50:10.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. PriMo: Coupled Prisms for Intuitive Surface Modeling. In *Eurographics Symp. on Geometry Processing*, 11–20.

COHEN, M. F. 1992. Interactive spacetime control for animation. In *Computer Graphics (Proc. of SIGGRAPH 92)*, vol. 26, 293–302.

FANG, A. C., AND POLLARD, N. S. 2003. Efficient synthesis of physically valid human motion. *ACM Trans. on Graphics (SIGGRAPH 2003) 22*, 3, 417–426.

GAL, R., SORKINE, O., MITRA, N., AND COHEN-OR, D. 2009. iWIRES: An Analyze-and-Edit Approach to Shape Manipulation. *ACM Trans. on Graphics (SIGGRAPH 2009) 28*, 3, 33:1–33:10.

GLEICHER, M., AND WITKIN, A. 1991. Differential manipulation. In *Graphics Interface*, 61–67.

GLEICHER, M. 1997. Motion editing with spacetime constraints. In *Proc. ACM Symp. on Interactive 3D Graphics*, 139–148.

HUANG, J., TONG, Y., ZHOU, K., BAO, H., AND DESBRUN, M. 2011. Interactive shape interpolation through controllable dynamic deformation. *IEEE Trans. on Visualization and Computer Graphics 17*, 7, 983–992.

IRVING, G., TERAN, J., AND FEDKIW, R. 2004. Invertible Finite Elements for Robust Simulation of Large Deformation. In *Proc. of the Symp. on Comp. Animation 2004*, 131–140.

JAMES, D. 2001. *Multiresolution Green's Function Methods for Interactive Simulation of Large-scale Elastostatic Objects and Other Physical Systems in Equilibrium*. PhD thesis, University of British Columbia.

KASS, M., AND ANDERSON, J. 2008. Animating oscillatory motion with overlap: Wiggly splines. *ACM Trans. on Graphics (SIGGRAPH 2008) 27*, 3, 28:1–28:8.

KIM, J., AND POLLARD, N. S. 2011. Direct control of simulated non-human characters. *IEEE Computer Graphics and Applications 31*, 4, 55–65.

KIM, J., AND POLLARD, N. S. 2011. Fast simulation of skeleton-driven deformable body characters. *ACM Trans. on Graphics 30*, 5, 121:1–121:19.

KONDO, R., KANAI, T., AND ANJYO, K. 2005. Directable animation of elastic objects. In *Symp. on Computer Animation (SCA)*, 127–134.

LEE, J., AND SHIN, S. Y. 1999. A Hierarchical Approach to Interactive Motion Editing for Human-like Figures. In *Proc. of ACM SIGGRAPH 99*, 39–48.

LIPMAN, Y., SORKINE, O., COHEN-OR, D., LEVIN, D., RÖSSL, C., AND SEIDEL, H.-P. 2004. Differential coordinates for interactive mesh editing. In *Proc. of Shape Modeling International*, 181–190.

LIU, Z., GORTLER, S. J., AND COHEN, M. F. 1994. Hierarchical spacetime control. In *Computer Graphics (Proc. of SIGGRAPH 94)*, 35–42.

MCNAMARA, A., TREUILLE, A., POPOVIĆ, Z., AND STAM, J. 2004. Fluid control using the adjoint method. *ACM Trans. on Graphics (SIGGRAPH 2004) 23*, 3, 449–456.

MIN, J., CHEN, Y.-L., AND CHAI, J. 2009. Interactive generation of human animation with deformable motion models. *ACM Trans. on Graphics 28*, 1, 9:1–9:12.

PARKER, E. G., AND O'BRIEN, J. F. 2009. Real-time deformation and fracture in a game environment. In *Symp. on Computer Animation (SCA)*, 156–166.

POPOVIĆ, Z., AND WITKIN, A. P. 1999. Physically based motion transformation. In *Proc. of ACM SIGGRAPH 99*, 11–20.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. 2000. Interactive manipulation of rigid body simulations. In *Proc. of ACM SIGGRAPH 2000*, 209–218.

POPOVIĆ, J., SEITZ, S. M., AND ERDMANN, M. 2003. Motion sketching for control of rigid-body simulations. *ACM Trans. on Graphics 22*, 4, 1034–1054.

SAFONOVA, A., HODGINS, J., AND POLLARD, N. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. on Graphics (SIGGRAPH 2004) 23*, 3, 514–521.

SHABANA, A. A. 1990. *Theory of Vibration, Volume II: Discrete and Continuous Systems*. Springer–Verlag, New York, NY.

SOK, K. W., YAMANE, K., LEE, J., AND HODGINS, J. 2010. Editing dynamic human motions via momentum and force. In *Symp. on Computer Animation (SCA)*, 11–20.

SUMNER, R., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. on Graphics (SIGGRAPH 2004) 23*, 3, 399–405.

TAK, S., YOUNG SONG, O., AND KO, H.-S. 2002. Spacetime sweeping: An interactive dynamic constraints solver. In *Computer Animation 2002*, 261–270.

UMETANI, N., KAUFMAN, D., IGARASHI, T., AND GRINSPUN, E. 2011. Sensitive couture for interactive garment modeling and editing. *ACM Trans. on Graphics (SIGGRAPH 2011) 30*, 4, 90:1–90:12.

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *Computer Graphics (Proc. of SIGGRAPH 88)*, vol. 22, 159–168.

WITKIN, A., AND POPOVIC, Z. 1995. Motion warping. In *Proc. of ACM SIGGRAPH 98*, 105–108.

WOJTAN, C., MUCHA, P. J., AND TURK, G. 2006. Keyframe control of complex particle systems using the adjoint method. In *Symp. on Computer Animation (SCA)*, 15–23.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *Proc. of ACM SIGGRAPH 97*, 256–268.