Deformable Object Animation Using Reduced Optimal Control

Jernej Barbič[1] Marco da Silva[1] Jovan Popović[1,2,3]

[1]Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

[2]Advanced Technology Labs, Adobe Systems Incorporated
[3]University of Washington

Abstract

Keyframe animation is a common technique to generate animations of deformable characters and other soft bodies. With spline interpolation, however, it can be difficult to achieve secondary motion effects such as plausible dynamics when there are thousands of degrees of freedom to animate. Physical methods can provide more realism with less user effort, but it is challenging to apply them to quickly create *specific* animations that closely follow prescribed animator goals. We present a fast space-time optimization method to author physically based deformable object simulations that conform to animator-specified keyframes. We demonstrate our method with FEM deformable objects and mass-spring systems.

Our method minimizes an objective function that penalizes the sum of keyframe deviations plus the deviation of the trajectory from physics. With existing methods, such minimizations operate in high dimensions, are slow, memory consuming, and prone to local minima. We demonstrate that significant computational speedups and robustness improvements can be achieved if the optimization problem is properly solved in a low-dimensional space. Selecting a low-dimensional space so that the intent of the animator is accommodated, and that at the same time space-time optimization is convergent and fast, is difficult. We present a method that generates a quality low-dimensional space using the given keyframes. It is then possible to find quality solutions to difficult space-time optimization problems robustly and in a manner of minutes.

CR Categories: I.6.8 [Simulation and Modeling]: Types of Simulation—Animation, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality

Keywords: deformations, space-time, keyframes, control, model reduction

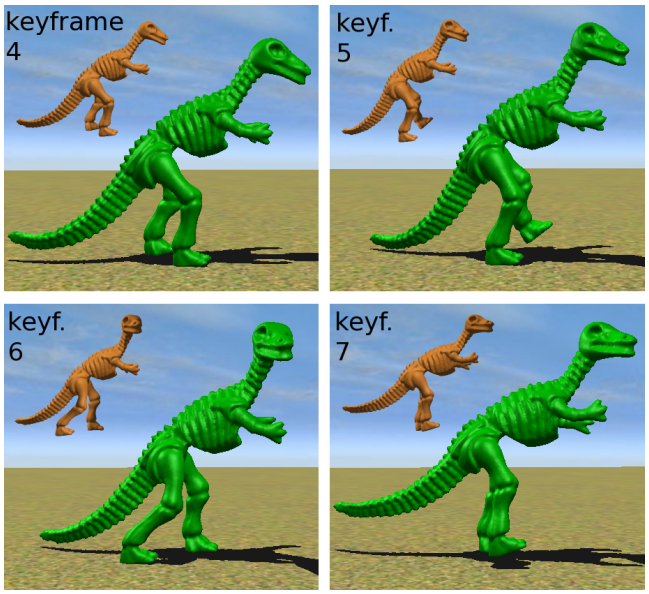

Figure 1: Fast authoring of animations with dynamics: *This soft-body dinosaur sequence consists of five walking steps, and includes dynamic deformation effects due to inertia and impact forces. Each step was generated by solving a space-time optimization problem, involving 3 user-provided keyframes, and requiring only 3 minutes total to solve due to a proper application of model reduction to the control problem. Unreduced optimization took 1 hour for each step. The four images show output poses at times corresponding to four consecutive keyframes (out of 11 total). For comparison, the keyframe is shown in the top-left of each image.*

1 Introduction

Generating animations that satisfy the goals of the animator yet look realistic is one of the key tasks in computer animation. In this paper, we present a fast method to generate such animations for solid 3D deformable objects such as large deformation Finite Element Method (FEM) models and mass-spring systems (see Figure 1). With deformable objects, animators often specify their goals by generating a set of keyframes to be met at a sparse set of points in time. The deformable object trajectory is then commonly obtained using spline interpolation. However, manipulating splines can become tedious when hundreds or even thousands degrees of freedom are involved. Unless a large amount of manual work is invested in generating dense keyframes, spline trajectories will lack deformation dynamics, mass inertia and other secondary motion effects.

Physically based simulation can provide realism with significantly less animator effort. However, physically based animations do not easily meet animator's goals such as a set of keyframes because they must obey the equations of motion in order to stay realistic. One can meet a simplistic set of keyframes by tweaking initial animation poses and velocities ("*shooting*"). In order to follow a complex keyframe sequence it is, however, inevitable to deviate from purely physical trajectories. Deviation from physics can be defined as imbalance in the equations of motion of the deformable object, and is therefore equivalent to injecting (fictitious) control forces into the physically based simulation. We present a method where these control forces are minimized, by solving an optimization problem with an objective function that minimizes a weighted sum of keyframe deviations and the amount of injected control.

Such space-time optimization problems are very common in computer graphics, and many techniques exist to solve them. With complex deformable systems, however, the existing methods are slow, memory consuming and prone to local minima. Our paper addresses two main challenges in applying space-time optimization

to 3D deformable solids. First, we increase optimization speed and robustness by solving the optimization problem in a reduced space, which greatly decreases the number of free parameters in the optimization. We demonstrate how to generate a small, yet expressive, low-dimensional space for the reduced optimization. This space contains the keyframes, and is augmented with "tangent" deformations that naturally interpolate the keyframes, enabling quick and stable convergence. The basis can be computed quickly and without user intervention, and captures both global and local deformations present in the keyframes. Our basis can be computed both for models with permanently constrained vertices, and for "free-flying" models where no vertices are constrained (see Figure 2).

Second, while the keyframes can be created using any of the mesh modeling methods, standard methods can create keyframes with high internal strain energies, leading to poor space-time convergence or suboptimal results. We introduce a physically based keyframing tool that uses the same simulator that will later be used to solve the space-time problem. This results in *low strain energy* keyframes, suitable for space-time optimization. The keyframing tool works by employing an interactive static solver. The user applies forces to the model and/or fixes desired model vertices, while the system interactively computes static equilibria under the applied forces and constraints, permitting the user to progressively shape the object. We also present a method to create keyframes from a given external triangle mesh animation. In this case, we use a preprocessing optimization that fits a volumetric mesh deformation to the input triangle mesh keyframes. Our optimization function combines fit quality with a preference to low strain energy shapes.

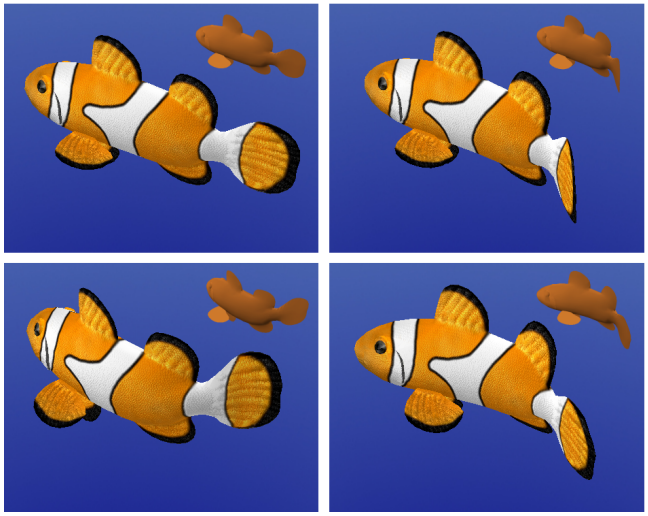

Figure 2: Our method supports unconstrained models, *as can be seen in this fish animation. The four images show output poses corresponding to the times of the first four keyframes. Keyframes are shown in the top-right of each image.*

2 Related work

In optimal control, one seeks minimal control forces for a physical system to perform a certain task. Typically, these minimal forces achieve the goal by cooperating with the natural dynamics of the system. Because the resulting motions are only minimally perturbed from control-free trajectories, they often appear "natural", making them desirable in computer animation [Brotman and Netravali 1988]. Computing optimal control requires solving a *space-time optimization problem*. In computer animation, space-time optimization was pioneered by Witkin and Kass [1988]. Many researchers have since improved the method (see [Fang and Pollard 2003] and [Safonova et al. 2004] for good surveys). Researchers have improved user interaction and accelerated computation [Cohen 1992; Liu 1996; Grzeszczuk et al. 1998], or targeted specific physical systems, such as human motion [Rose et al. 1996; Gleicher 1997; Popović and Witkin 1999; Fang and Pollard 2003; Safonova et al. 2004; Sulejmanpasić and Popović 2005; Liu et al. 2005], rigid-body simulations [Popović et al. 2003], and fluids. Treuille et al. [2003] and McNamara et al. [2004] controlled fluids using multiple shooting and the adjoint method, respectively.

Our paper presents a fast approximation scheme to solve optimal control problems for deformable objects. In computer graphics, there are many methods to simulate deformable objects, but there are fewer papers that also *control* them. With complex meshes, such control is generally slow and prone to local minima due to the large number of deformable degrees of freedom. Deformable objects can be controlled using PD controllers [Kondo et al. 2005] or spatially localized controllers [Jeon and Choi 2007], which are fast and conceptually simple. The forces are, however, computed instantaneously, without a longer planning horizon or incorporating the system dynamics. In our work, we *minimize* the control forces, which permits us to meet the keyframes more closely with less force. Deformable object animations can also be created by properly interpolating static keyframes generated using interactive shape deformation methods [Der et al. 2006; Huang et al. 2006; Adams et al. 2008]. Our animations, however, follow physical equations of motion, and therefore exhibit physically based *dynamics*, or deviate from it gracefully. Proper time-evolution of deformations and secondary soft-body motion occur automatically in our framework.

Optimal control has been previously applied to particle systems and cloth [Wojtan et al. 2006], by using the adjoint method, analogous to how McNamara et. al [2004] controlled fluids. These previous adjoint method applications simulated fluid or cloth in the full high-dimensional space of fluid grid node velocities or model vertex deformations. In our paper, we also use the adjoint method, but we do so in a quality low-dimensional space where the adjoint iterations are orders of magnitude faster, yielding significantly shorter optimization times and smaller memory footprints (Table 1, page 7).

Constrained Lagrangian solvers such as TRACKS [Bergou et al. 2007] can generate a detailed simulation by tracking a given input coarse animation. For deformable solids, our method could serve as a source of such coarse animations. While both our method and TRACKS can generate dynamic simulations from a rudimentary input, TRACKS has been designed for dense animation input, whereas we assume sparse keyframes. TRACKS enforces the low spatial frequency part of the motion with an exact constraint, which avoids optimization (increasing method speed), but might require high forces in the direction normal to the constraint. Our method does not use a constraint, but weights keyframe enforcement against control effort using an optimization. The trade-off is adjustable. The reduced degrees of freedom can deviate from the guiding input (the keyframes), as dictated by the natural dynamics.

Deformable simulations could also be generated by randomly sampling forces in space-time [Chenney and Forsyth 2000], similar to how Twigg and James [2007] were able to browse rigid object animations. With deformable objects, however, there is a choice of a force on every vertex at every timestep. This can quickly lead to a dimensional explosion in the number of space-time force samples, each of which will require an expensive full forward deformable simulation to evaluate. Given a space-time force sample, our method computes the space-time force change which decreases an objective function the most. This enables us to find optimal space-time forces more quickly, akin to how a 1D function can be minimized more quickly if derivative information is available.

Recently, Kass and Anderson [2008] demonstrated how standard splines can be extended to easily keyframe a broad class of oscillatory 1D curves. These curves were then used to drive modal deformations of detailed meshes using pairs of complex-valued phase-shifted modal shapes. Each modal pair either required tuning its own spline, or the animator needed to tune phase shifts among the different pairs. Our method also combines several deformation modes to generate an animation, but it computes the modal coupling automatically, for an arbitrary number of modes. For example, this enables us to support large deformations, where the nonlinear forces couple the different modes in a non-obvious way that would be difficult to hand-tune for animators.

In the solid mechanics community, reduction has been used to forward-simulate nonlinear elastic models [Krysl et al. 2001], and to control complex *linear*, small deformation elasticity (c.f. [Gildin 2006]). It has, however, not been previously employed to control complex nonlinear 3D deformable objects undergoing large deformations. In graphics, reduction has been applied to control the motion of a full-body human skeleton [Safonova et al. 2004], with a basis obtained from a temporally dense motion capture database. Reduction has also been employed to forward-simulate FEM deformable objects [Barbič and James 2005] and fluids [Treuille et al. 2006] at interactive rates. The reduction bases of [Barbič and James 2005] were, however, designed for (large) deformations around the rest shape, as opposed to sparse keyframe input. While one could combine, say, the linear modes and their derivatives computed at every keyframe, the bases would quickly grow large in size, and would not necessarily contain the degrees of freedom most important for the optimization (Section 4.3). Given a *pre-existing* animation $\mathcal{A}$, we previously demonstrated [Barbič and Popović 2008] how to use reduction to expand $\mathcal{A}$ into a "tube" of animations centered around $\mathcal{A}$, making it possible for a real-time simulation to deviate from $\mathcal{A}$ due to, for example, user input. The tracking controller required a temporally dense animation as input. In this work, we show how to author entirely new animations *from scratch*, given only a temporally sparse set of keyframes, enabling the following complementary process: (1) create the keyframes, (2) generate the animation $\mathcal{A}$ using reduced optimization (this paper), (3) use $\mathcal{A}$ to construct a real-time tracking controller [Barbič and Popović 2008].

3 The reduced control problem

The input to our method is a set of unreduced keyframe deformation vectors $\overline{q}_1, \ldots, \overline{q}_K$, to be met at timesteps $t_1 < t_2 < \ldots < t_K$. We use a fixed timestep size h, therefore $t_i = hk_i$, for some integers $k_1 < k_2 < \ldots < k_K$. Time $t = 0$ corresponds to the beginning of the animation. In our examples, initial conditions are specified by the user, but they could easily also be subject to optimization.

Our method first uses the keyframes to construct a low-dimensional space, tailored to the keyframes and typical deformations in between the keyframes (see Figure 3). Next, it projects the keyframes to this low-dimensional space, obtaining *reduced keyframes*. The output animation is then computed by solving a space-time optimization problem in the low-dimensional space, using the adjoint method and conjugate gradient optimization. We will now briefly introduce reduced simulations, and then formulate the reduced optimization problem. In Section 4, we describe our method to generate the low-dimensional basis, and in Section 5, we explain how we solve the resulting low-dimensional optimization problem.

3.1 Tutorial: Full and reduced simulation

Full simulations are simulations without reduction. Assuming linear control, we express deformable simulations as the following (high-dimensional) second order system of ODEs:

$$\ddot{q} = F(q, \dot{q}, t) + Bu. \tag{1}$$

Here, $q \in \mathbb{R}^n$ is the state vector (n will typically be at least several thousands), $F(q, \dot{q}, t) \in \mathbb{R}^n$ is some (nonlinear) function specifying the dynamics of the deformable object, $B \in \mathbb{R}^{n \times m}$ is a constant *control matrix*, and $u \in \mathbb{R}^m$ is the *control vector*. We demonstrate our method using geometrically nonlinear FEM deformable objects [Capell et al. 2002] and mass-spring systems (both supporting large deformations). The state vector q consists of displacements of the vertices of a 3D volumetric mesh, with respect to some fixed rest configuration.

Reduced simulations are obtained by projecting Equation 1 onto a r-dimensional *subspace*, spanned by columns of some basis matrix $U \in \mathbb{R}^{n \times r}$ (typically, $r \sim 20$ in our examples). We orthonormalize our bases with respect to the mesh mass matrix ($U^{\mathrm{T}} M U = I$). The full state is approximated as $q = Uz$, where $z \in \mathbb{R}^r$ is the *reduced state*. The resulting low-dimensional system of ODEs

$$\ddot{z} = \widetilde{F}(z, \dot{z}, t) + \widetilde{B} w, \quad \text{for} \quad \widetilde{F}(z, \dot{z}, t) = U^{\mathrm{T}} F(Uz, U\dot{z}, t), \tag{2}$$

approximates the high-dimensional system provided that the true solution states q are well-captured by the chosen basis U. Here, $\widetilde{B} \in \mathbb{R}^{r \times s}$ is a constant matrix, and $w \in \mathbb{R}^s$ is the *reduced control vector* (we use $s = r$ in our work). With geometrically nonlinear FEM deformable objects, there exists an efficient cubic polynomial formula for $\widetilde{F}(z, \dot{z}, t)$ [Barbič and James 2005], which we use to accelerate space-time optimization with our FEM examples.

3.2 The objective function

Our goal is to generate animations where keyframes are met as closely as possible, with the least amount of error in physics (injected control forces). In an unreduced problem, one seeks control forces $u_0, \ldots, u_{T-1}$ that minimize the objective function

$$E = \frac{1}{2} \sum_{i=1}^{K} \big(q(t_i) - \overline{q}_i\big)^{\mathrm{T}} Q_i \big(q(t_i) - \overline{q}_i\big) + \frac{1}{2} \sum_{i=0}^{T-1} u_i^{\mathrm{T}} R_i u_i. \tag{3}$$

Here, Q_i and R_i are state error and control effort cost matrices, respectively. Each state $q(t_i)$ implicitly depends on control vectors $u_0, \ldots, u_{i-1}$. Therefore, E is a nonlinear function of the control sequence $\{u_i\}_i$. This sequence consists of T control vectors, each of which is n-dimensional, causing minimization algorithms to be slow, memory-consuming, and prone to local minima. Instead, we perform an optimization in a low-dimensional space, spanned by columns of the basis matrix U. We convert the keyframes to their low-dimensional representations $\overline{z}_i = U^{\mathrm{T}} M \overline{q}_i$ (the projection is mass-weighted to support non-uniform meshes), and then minimize

$$\widetilde{E} = \frac{1}{2} \sum_{i=1}^{K} \big(z(t_i) - \overline{z}_i\big)^{\mathrm{T}} \widetilde{Q}_i \big(z(t_i) - \overline{z}_i\big) + \frac{1}{2} \sum_{i=0}^{T-1} w_i^{\mathrm{T}} \widetilde{R}_i w_i, \tag{4}$$

under the dynamics of Equation 2. We typically set $\widetilde{Q}_i$ to the identity matrix, which can be shown to penalize the total keyframe mass deviation. One can also set some diagonal entries to zero, to only penalize the error in a subset of the modes. For control cost, we typically use a scalar multiple of the identity matrix. Minimizing Equation 4 gives the reduced control $\{w_i\}_i$, and the reduced animation $\{z_i\}_i$. Our output is the animation $\{Uz_i\}_i$.

The keyframe constraints only appear as a term in the optimization function of Equation 4, and are therefore met approximately, weighted against control effort (*soft* keyframes). This is appealing

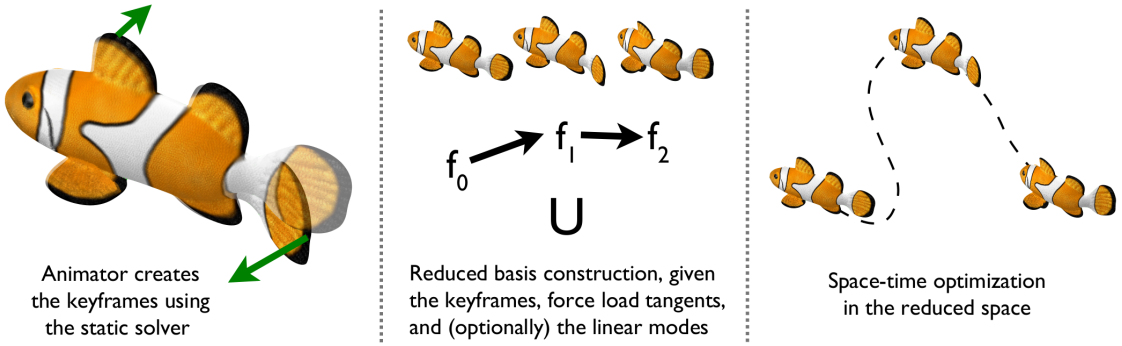

Figure 3: Overview: *The animator provides the keys using a static solver (or they are fitted to external data using an optimization). Our system then computes a reduced basis tailored to the keyframes, and then solves a space-time optimization problem in that basis.*

because, by the nature of the artistic process, the animator's input is often not meant to be absolute, complete, or necessarily enforced exactly. Sometimes, it might not even be physically well-formed. However, in our examples we observed that (if desired) it is possible to meet the keyframes very closely, by setting the control cost sufficiently low. Alternatively, exact keyframes could be enforced by applying direct transcription to Equation 2, combined with a constrained optimizer such as SNOPT (see [Safonova et al. 2004]).

4 Keyframes and basis selection

We now describe how we generate the keyframes and a low-dimensional basis U for reduced optimization. Space-time optimization is sensitive to how keyframes and the basis are selected. With suboptimal choices, the optimizer fails to converge, or converges to a visibly suboptimal solution. Selecting the keyframes and a basis that are able to "harness" solid deformable object space-time optimization is challenging and presents our key contribution.

4.1 Keyframes from a physically based modeler

Keyframes are static shapes. Therefore, an artist can generate them using any of the many shape deformation techniques proposed in computer graphics [Gain and Bechmann 2008], applied to a volumetric simulation mesh. However, because a geometric method might not be aware of the underlying physical model, shapes obtained using purely geometric techniques do not always serve as optimal keyframes for a physical simulation. For example, such shapes could have large elastic strain energies, forcing the subsequent space-time optimization to exert a lot of unnatural effort to reach the shapes in a dynamic simulation. Mezger [2008] recently proposed a method that uses a FEM physically based simulation for geometric modeling of static shapes. We adopt a similar approach for keyframe generation. We use one physical simulator both for keyframe modeling and for subsequent space-time optimization. This results in "natural" keyframe shapes with low elastic strain energies, amenable to space-time optimization. Also, such an approach simplifies implementation, as only a single simulator is required. The keyframes are also very suitable for constructing a quality low-dimensional optimization space (Section 4.3).

To keyframe, we use an unreduced static solver (Mezger [2008] used plasticity flow). The user is presented with an interactive system where they can select an arbitrary vertex (or set of vertices), and apply forces to them, either with the mouse or a 3-DOF input device (see Figure 3, left). The system interactively computes the static configuration under the currently applied force loads, by solving the equation $R(q) = f$, where R are the (non-linear) internal elastic forces, f is the current global force load vector, and q is the deformation. The nonlinear equation is solved using a Newton-Raphson procedure, by repeatedly forming the tangent stiffness matrix $K = dR/dq$. At any moment, the user can freeze the current load, and continue adding other loads on top of it, essentially stacking the loads. Also, the user can at any time pin the current position of any vertex or a set of vertices. Such constraints can be implemented by removing proper rows from K (Lagrange multipliers are not necessary). The combination of pinning vertices and stacking force loads permits the user to generate rich deformations. The computational bottleneck of such a system are typically the evaluation of internal forces and stiffness matrices (dominant with smaller models), and sparse linear system solves (with large models). In our system, we use a multi-threaded direct PARDISO solver, and multi-threaded evaluation of internal forces and stiffness matrices, which made our static solver interactive ($>$ 3 fps) for all of our models.

4.2 Keyframes from an external triangle mesh modeler

Our method can also start with keyframes in the form of deformations of a *triangle mesh*, generated, say, using an external geometric shape modeler. Given a triangle mesh keyframe sequence, we must construct a sequence of volumetric mesh (the "cage") deformations (keyframes to our method) that reconstruct the triangle mesh deformations as closely as possible. We compute each volumetric mesh deformation u separately, by solving the optimization problem

$$u = \arg\min_{\hat{u}} \Big(||A\hat{u} - \overline{u}||_2 + \beta \text{ elastic\_strain\_energy}(\hat{u}) \Big), \quad (5)$$

where A is a large sparse matrix of barycentric weights giving the deformations at the triangle mesh vertices, $\overline{u}$ is the desired triangle deformation, and β controls the trade-off between matching the triangle mesh deformation and minimizing elastic mechanical strain energy. We initialize the optimization by setting the deformation of every volumetric mesh vertex to the deformation of the nearest (in the undeformed configuration) triangle mesh vertex. The elastic strain energy term biases volumetric keyframes toward low strain energy. It decreases or removes any irregularities in the triangle mesh deformations, producing "physical" volumetric keyframes. It also correctly positions any volumetric mesh elements that do not contain any triangle mesh vertices. The 2-norm can be weighted with the surface area belonging to each triangle mesh vertex.

4.3 Basis generation

Given the keyframes $\overline{q}_1, \ldots, \overline{q}_K$, the most straightforward way to generate a basis is to concatenate all keyframes into a basis, followed by mass-Gramm-Schmidt orthonormalization. Such a basis

is able to express all linear interpolations of the keyframes. This works well when keyframes are deformed little relative to one another, but fails when they are separated by large deformations. In such cases, linear interpolation can only express shapes that are visibly non-physical, such as volume-inflated shapes resulting from the inability to properly interpolate rotations (see Figure 4, left). Worse even, if such a basis is used for space-time optimization, the control forces have to squeeze the object into these non-natural shapes, which leads to very strong forces, locking, and convergence problems. For these reasons, we augment our bases as follows. For every keyframe $\bar{q}_i$, one can evaluate the internal elastic forces, $R(\bar{q}_i)$, acting on the object in configuration $\bar{q}_i$. A system in configuration $\bar{q}_i$ is then in static equilibrium under external forces $f_i = R(\bar{q}_i)$. For every $\alpha \in [0,1]$, one can define an external force load $f(\alpha) = (1-\alpha) f_i + \alpha f_{i+1}$, and then find the deformation $q(\alpha)$ which is the static equilibrium under $f(\alpha)$, i.e., $R(q(\alpha)) = f(\alpha)$. Note that, unless the stiffness matrix is close to a singularity, such a deformation will exist and be unique. Also note that the nonlinear internal force function R will ensure that $q(\alpha)$ is a "good-looking" deformation, different from a mere linear interpolation of $(1-\alpha)q_i + \alpha q_{i+1}$ (see Figure 4, left). For example, if $f_i = 0$ and f_{i+1} consists of a force F applied to a single vertex, then $q(\alpha)$ (for $\alpha \in [0,1]$) will be the static shape under a decreased load αF.

For each i, traversing $\alpha \in [0,1]$ gives an arc in the high-dimensional deformation space. The consecutive arcs are connected, forming a curve in the deformation space (see Figure 4, right), which we call the *keyframe deformation curve*. For geometrically nonlinear FEM models and mass-spring networks, this curve is C^∞ in between the keyframes and C^0 at the keyframes. It contains "natural" deformations for an animation specified by keyframes $\bar{q}_1, \ldots, \bar{q}_K$, and we use it to generate our motion basis U. We do so by computing tangent vectors $dq/d\alpha$ to the curve at the endpoints of each arc. Tangent vectors T_i^0 and T_i^1 are defined to be the derivatives of $q_i(\alpha)$ at $\alpha = 0$ and $\alpha = 1$, and can be computed by differentiating $R(q(\alpha)) = f(\alpha)$ with respect to α :

$$K(\bar{q}_i)T_i^0 = f_{i+1} - f_i, \qquad K(\bar{q}_{i+1})T_i^1 = f_{i+1} - f_i, \qquad (6)$$

where $K(q) = dR/dq$ is the tangent stiffness matrix at q. We generate the basis by concatenating all keyframes and tangent vectors into a basis, followed by mass-Gramm-Schmidt orthonormalization. If the resulting basis is large (e.g., larger than 20), one can optionally apply Principal Component Analysis (PCA) to the basis, and retain a smaller number of dimensions. Although our basis is small compared to the size of the system, it will capture any local deformations in the keyframes, which will then appear in the optimized dynamic simulation. This is in contrast to previous methods of model reduction of solids [Barbič and James 2005] which typically simulated global deformations. We note that it can be shown mathematically that the tangents are invariant with respect to picking a particular mesh rest shape (even under rest shape rotations).

Instead of using the tangents, a basis could be obtained by sampling the $q(\alpha)$ curve for a sufficient number of α values, using a static solver. We opted for the tangents because the user then does not have to specify the number of samples. The basis with tangents performed well despite some very large gaps between keyframes in our examples. With extreme keyframe gaps (for example, 180 degree twists), the user can either insert an extra keyframe, or combine tangents with sampling.

Basis for unconstrained models: With unconstrained models, the internal forces do not resist rigid translations and rotations, causing the stiffness matrices K of Equation 6 to be singular. This singularity, however, is only of dimension six in the undeformed configuration (nullspace consists of translations and infinitesimal rotations), and typically three-dimensional elsewhere (nullspace consists of translations). Because K is symmetric, its nullspace $\mathcal{N}$ is orthogonal to its range $\mathcal{R}$. For unconstrained models, we define the keyframe deformation curve differentially, by defining its derivative at α as follows (solution to $q'(\alpha)$ is unique):

$$K(q(\alpha))q'(\alpha) = \text{projection}_{\mathcal{R}(K(q(\alpha)))}(f_{i+1} - f_i), \qquad (7)$$

$$q'(\alpha) \perp \mathcal{N}(K(q(\alpha))). \qquad (8)$$

In order to compute the tangents, we therefore project the right hand sides of Equation 6 to $\mathcal{R}(K(\bar{q}_i))$ and $\mathcal{R}(K(\bar{q}_{i+1}))$, and solve for T_i^0 and T_i^1 using a sparse solver capable of handling moderate degeneracies in the system matrix.

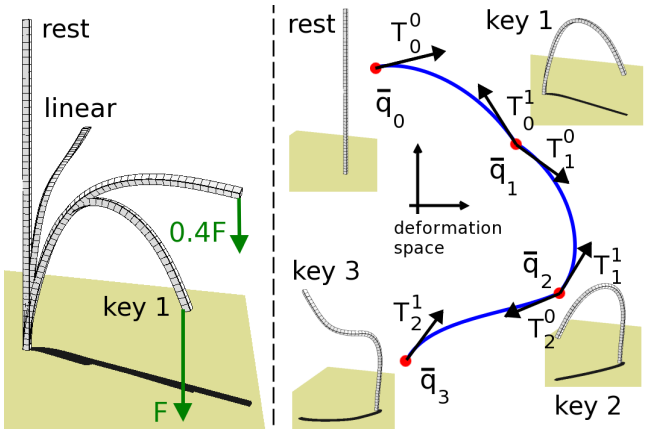

Figure 4: Our basis captures the natural deformations connecting the keyframes: *Left: User sets a keyframe by pulling on the end of a beam (force F), using an interactive static solver. The static equilibrium under an interpolated load ($\alpha = 0.4$) is a natural shape, whereas a mere linear interpolation is severely distorted. Right: Three consecutive arcs on the keyframe deformation curve.*

Basis enrichment: Because our subspace was constructed using keyframe data, it can provide the dynamics (including proper frame timing) along the natural trajectories between consecutive keyframes. In order to enrich the animations, the keyframe-tangent basis U_K can be extended by providing additional deformation examples. One option is to let the user provide these examples, by interacting with the deformable model in a simulator, and recording the resulting deformations. Alternatively, one can augment the basis *automatically*, by computing some natural deformation shapes of an object, such as the natural modes of vibration. In either case, it is important to keep the original keyframe content in the basis so that these crucial degrees of freedom are available to space-time optimization. Given the additional deformation examples $q_1^A, \ldots, q_N^A$, we first perform a mass-orthogonal projection to remove any content already included in U_K :

$$\tilde{q}_i^A = q_i^A - U_K U_K^{\mathrm{T}} M q_i^A. \qquad (9)$$

We then apply mass-PCA [Barbič and James 2005] on $\tilde{q}_1^A, \ldots, \tilde{q}_N^A$, to obtain a basis U_A, spanning the most significant dimensions (typically 10-20 in our examples) of the additional content. Finally, we set the basis U to be a concatenation of U_K and U_A.

5 Minimizing the objective function

Objective functions such as ours (Equation 4) are very common in computer graphics and their minimization has been addressed in

several papers. There are no closed form solutions for either local or global minima. Only local minima can be found in practice, and selection of a good initial guess is important. We optimize the objective function using conjugate gradient optimization [Press et al. 2007], where the gradients are computed using the adjoint method [McNamara et al. 2004; Wojtan et al. 2006]. Unlike previous methods, however, our adjoint method operates in the reduced space, enabling significantly faster iterations and convergence.

The adjoint method can be seen as a "black box", which, given the current control sequence $\{w_i\}_i$, computes the value of the objective function *and* its gradient with respect to all components of the sequence $\{w_i\}_i$. The adjoint method is standard; see Appendix A for details of our implementation. The adjoint method in the reduced space is faster and less memory consuming than the adjoint method in the full simulation space because large sparse linear system solves are replaced with small dense linear systems. This applies both to forward simulations and to backward passes to compute the gradient. We use implicit Newmark integration, and perform backward passes where the computed gradients are *exact* with respect to our Newmark discretization scheme. This is different from the gradient approximation of [Wojtan et al. 2006] where the Hessian was avoided to ease implementation and increase speed. In our examples, we found that computing exact gradients leads to faster convergence, often to visibly better minima. Obtaining exact gradients requires computing the Hessian (second derivative) of the internal elastic forces. With reduced geometrically nonlinear FEM, the Hessian can be computed by taking derivatives of the second-order multivariate polynomials (entries of the reduced stiffness matrix), which can be done symbolically during pre-process (fast, a few milliseconds). With mass-spring systems, the spring forces are simple expressions and the Hessian is easily manageable with a short algebraic derivation. In both cases, we managed to compute the Hessian equally fast or within the same order of magnitude as computing internal forces and stiffness matrices.

Conjugate gradient optimization: The simplest strategy to (locally) minimize a scalar function with computable gradients is to always walk in the direction of the gradient (steepest descent). Much like with iterative methods for linear systems, however, convergence can be significantly accelerated by using the conjugate gradient (CG) optimization algorithm. After computing the gradient, CG properly factors out the gradient directions explored during previous iterations, yielding an improved search direction d. It then performs a 1D line search along d. We use a line search where only function values are probed (as opposed to also derivatives); therefore, our line searches consist of a series of forward simulations. This results in optimizations where there are more forward simulations than backward gradient computation passes, typically with a ratio of about 8:1. In our experiments, CG converged to the same solution as steepest descent 5-40 times faster.

Initial guess: Often, we were able to initialize the optimization with zero control and still find the very nonlinear solution to the problem. In some cases, however, it helped if the initial guess was reasonably close to a good solution. We used a PD controller to initialize our method in such cases: at every moment of time, we push the object to the linear interpolation of the two consecutive keyframes, combined with a proper amount of damping. We note that it is often difficult to match the keyframes with such a PD controller, unless the controller is made very stiff. Optimized solutions, however, use a minimized amount of injected control forces, all the while meeting the keyframes very closely.

Timestep selection: The total physical time duration of our animations is hT, where h is the timestep length, and T is the number of timesteps. The parameter h should be set such that the resulting time-scale matches that of the underlying natural physical dynamics. While the match only needs to be approximate, a good match is important. Timesteps orders of magnitude too large might lead to animations that "idle", then shoot for a keyframe over the last few frames, and values several orders of magnitude too small give animations that only move a small fraction toward the keyframe. The animator can employ the following strategy to select a reasonable timestep h, and then refine by trial and error as necessary. Given keyframes $\bar{q}_0$ and $\bar{q}_1$, one can compute the natural vibration frequency ω of the tangential vector T_0, and set the timestep h so that $\bar{q}_1$ is reached at approximately $1/4$ of the oscillation period:

$$\omega^2 = \frac{T_0^{\mathrm{T}} K(\bar{q}_0) T_0}{T_0^{\mathrm{T}} M T_0}, \qquad h = \frac{\pi}{2\omega}. \tag{10}$$

In case of many keyframes, one can use the average value of h, or vary the timestep along the animation.

6 Results

In our first example, we show a keyframed animation of a walking dinosaur (Figure 1). Each step of this five-step sequence was generated in minutes. The dinosaur matches the keyframes, while undergoing unscripted deformation dynamics. The basis was enriched with 18 linear modes, computed with respect to both legs pinned to the ground. The basis generation time in Table 1 includes 2 seconds to compute the linear modes. Each walking step was a separate optimization. For each step, the support foot was constrained, and the motion was computed by specifying two keyframes, one with the swing leg fully lifted up and the other with the swing leg landed. For the next step, we reversed the roles: the vertices on the landed leg become fixed, while the other leg was unpinned and became the swing leg. The final deformation position and velocity at the end of a step were used as initial conditions for the next step.

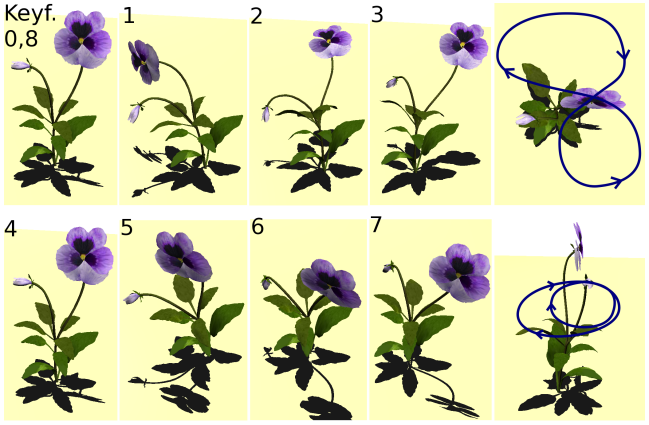

Figure 5: Animated purple pansy: *The first four columns give the keyframes. The rightmost column outlines the figure"8" motion of the main bloom (bird's eye view), and the (quasi-)circular motion of the secondary bloom. Keyframes 0,4,8 are identical.*

We also animated a pansy flower, by manually positioning the keyframes (using a static solver) so that the main bloom follows a figure "8" (as seen from a bird's eye perspective), and that the small bloom performs a circular motion (see Figure 5). This example demonstrates that our method can handle long animation sequences with many (eight) keyframes. Such animations would be difficult to generate through random external force sampling or trial-and-error

	v	elements	T	K	r	basis time		forward pass		backward pass		total space-time		memory	
						generate	cubic poly.	red.	full	red.	full	red.	full	red.	full
dinosaur	1493	5249	120	2	25	5.5s	90s	0.036s	17s	0.066s	58s	84s	1 hour	94 KB	16 MB
flower	2713	7602	300	8	22	25s	106s	0.069s	56s	0.111s	208s	140s	24 hours	206 KB	75 MB
fish	885	3477	210	5	24	11.3s	54s	0.059s	18s	0.100s	60s	345s	3.3hours (F)	158 KB	17 MB
elephant	4736	19386	240	8	16	30s	245s	0.050s	194s	0.067s	516s	30s	17.5hours (F)	120 KB	104 MB
bridge	9244	31764	150	3	18	5.0s	mass-spring	9 s	100s	34s	130s	50min	failed (F)	84 KB	127 MB

Table 1: Optimization statistics: *v=#vertices in tetrahedral mesh, T=#frames, K=#keyframes, r=basis dimension.* F = *converged to a visibly suboptimal local minimum. Machine specs: Apple Mac Pro, 2 x 2.8 GHz Quad-Core Intel Xeon processor, 4 GB memory.*

approaches. For example, we implemented a PD controller that applies a force which (at any moment of time) guides the object to a trajectory obtained by the Catmull-Rom spline interpolation of the keyframes. We found the PD controller to be difficult and slow to tune, as seeing the output under each set of PD parameters requires one full simulation, typically exceeding the cost of our reduced pipeline already after one or two such samples. In addition to providing the output animation, our method also provides control which can be used to re-generate the motion in a simulator. Therefore, we were able to use the output of our method as input to a real-time tracking controller [Barbič and Popović 2008].

The keyframes form the core of our bases, the tangents "fill up" the gaps in between the sparse keyframes, and the linear modes or user data provide the dimensions for the extra, unscripted, dynamics. Without the tangents, the simulation cannot bridge any large keyframe gaps. The optimization output is then limited to small deformations, and appears stiff (see Figure 6). This is especially pronounced with simulations where nonlinearities are essential, such as any simulation where the elastic forces are strong enough to prevent the material from collapsing. If the material is very soft, the control forces can simply squish the object along a linear interpolation of the keyframes, in which case the tangents are less important. With the keyframes and tangents, but without the linear modes, there is less dynamics, and the simulations more closely follow the keyframe deformation curves. Unlike splines, however, the spacetime optimization will automatically provide for a proper timing along the deformation curve. The static basis might not be optimal for simulations with large kinetic energies where simulations could deviate very far from static solutions. However, in the absence of any other information about the animation other than the keyframes, the static basis is a reasonable choice.

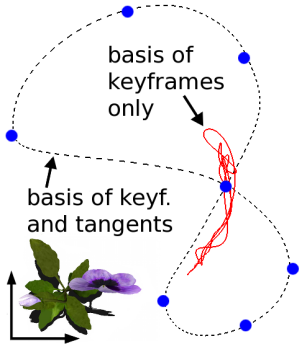

Figure 6: *Bird's eye view on the trajectory of a top flower vertex, computed using bases with and without the tangents.*

The fish example (Figure 2) demonstrates that our method can be used for objects where no vertices are constrained. The basis was enriched by the user pulling on the fins in an interactive simulation, and recording the resulting deformations. Motion of fins other than the tail was not keyframed, but occurs for "free", as a part of the natural system dynamics. In this example, a full optimization failed to converge to a plausible solution after 3 hours of optimization.

The bridge example (Figure 7) uses a mass-spring system, where there is no simple formula for reduced internal forces and stiffness matrices, so their evaluation has to proceed according to Equation 2. However, reduction is still beneficial both in time (about 10x in our example) and memory because it avoids a large sparse linear system solve both in the forward and backward adjoint passes. The memory footprint is reduced as one only needs to store the reduced animation to perform each backward pass, and not the entire animation. The basis was enriched with 10 linear vibration modes.

Figure 7: Our method supports mass-spring systems. *We keyframed a deformable animation of this elastic bridge.*

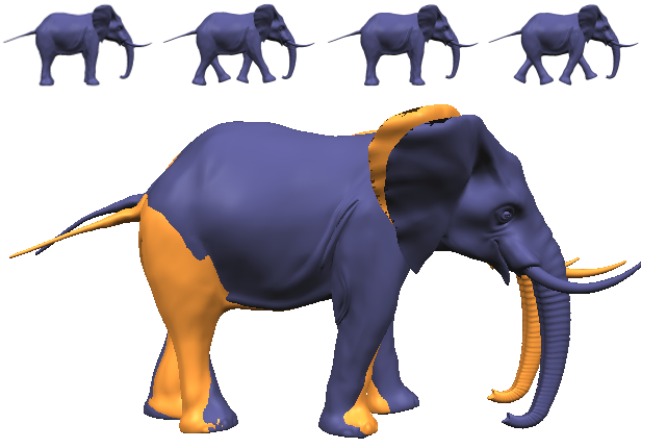

Figure 8: Interpolation with dynamics: *Our method can provide a dense "dynamic interpolation" of the input sparse keyframes. Top: the first four keyframes in our sequence. Bottom: an animation frame with our dynamic result shown in blue, and Catmull-Rom spline interpolation of the input keyframes overlaid in yellow.*

A static solver was used to generate the keyframes in all examples, except in the elephant example (Figure 8) where we tested the procedure of fitting volumetric mesh keyframes to external triangle mesh keyframes. We generated the triangle mesh keyframes (42,321 vertices) using the skinning engine in Maya. It took about 1 minute to fit each volumetric keyframe, using strain energy optimization of Equation 5. Elastic strain energy computation was multi-threaded (8 cores). This example uses a free-flying basis, which was enriched with 10 linear vibration modes.

All our examples use tetrahedral meshes with an embedded triangle mesh for rendering purposes. The bridge example uses tetrahedral edges and vertices to form the mass-spring network. We op-

timized both reduced and full optimization to the best of our abilities. Full optimization uses the multi-threaded direct PARDISO solver to solve all sparse linear systems encountered by the optimization, with the number of threads (typically 2 or 3) optimized to maximize performance. Unreduced FEM internal force and stiffness matrix evaluation was multi-threaded (8 cores). Mass-spring force and stiffness matrix evaluation, however, are simpler and fast and threading did not significantly improve our performance.

Table 1 gives statistics on our optimizations. In particular, it gives a comparison to an unreduced adjoint optimization [Wojtan et al. 2006]. It can be seen that optimization with reduction is orders of magnitude faster, and consumes less memory. This enables us to generate longer sequences with several keyframes. In Table 1, we reported the running times to produce the final result. In practice, one can continuously visualize the output iterations of the conjugate gradient solver, and can abort early (often within a matter of seconds) if necessary. In Figure 9, we compare our method to Catmull-Rom spline interpolation. It can be seen that spline interpolation causes visible artifacts, especially when there are large rotations in the deformation field between the two keyframes.

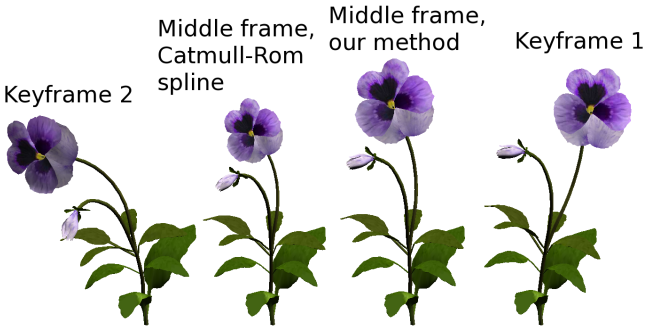

Figure 9: Spline interpolation causes loss of volume, *whereas our subspace produces more plausible shapes. The two middle frames were sampled at the same instance of time. Similar artifacts were observed with linear interpolation.*

7 Conclusion

This paper introduced a method for physically based keyframe animation of deformable models. The animator crafts keyframes in an interactive physical simulator. Our method then automatically constructs a reduced control basis using these keyframes and their tangents. Optionally, the basis is augmented with natural modes of the deformable model. The basis is then used to solve a reduced space-time optimization problem with tens of controls per time step instead of thousands. Building a reduced basis for the control of deformable models is not straightforward. Keyframes should be created using simulation so that they are compatible with optimization.

Reducing the optimization basis decreases the amount of time needed to optimize an animation. Furthermore, it increases robustness as higher-dimensional optimizations often get stuck in undesirable local minima. With our examples, reduced optimization converged in a couple of minutes, whereas full optimizations took several hours and in many cases yielded inferior results. This speed increase, while not interactive, makes optimization a more useful tool as animators can quickly iterate on a particular motion sequence.

When designing long animations, it might make sense to break up the problem into several consecutive windowed optimizations, each of which operates on a few keyframes [Cohen 1992]. With a fixed total number of frames T, there is almost no slowdown when using dense keyframes. For future work, we would like to explore applications where the keyframes are as dense as at every simulation step, for example, to add dynamics to pre-existing static animations.

Many characters have a natural skeletal structure, and could be animated by controlling a rigid skeleton with a wrapped passive dynamic deformable skin [Capell et al. 2002]. Such methods will, however, be less useful for deformable objects that lack a clearly defined skeleton (e.g., bridge, flower). Also, complex skeletons (typically around 15-20 DOFs in [Safonova et al. 2004]) might need reduction to improve the speed and convergence of control, which requires data, or insight about a specific skeleton. Our method computes the reduced space automatically and in a manner of seconds, using mesh geometry and keyframe input. It could be combined with a skeleton method, to simultaneously control both the rigid and deformable aspects of a multi-body dynamics simulation.

It is generally difficult to build optimal controllers for scenarios rich with contact, due to the discontinuous and non-sticky nature of (potentially unilateral) contact forces and their gradients. In our walking examples, however, we demonstrated how simulations with contact can be generated nonetheless, by breaking up the motion into several separate optimizations, each of which has a fixed bilateral contact state. For future work, we would like to couple rigid body motion of the character with deformation dynamics. In the fish example, rigid body motion and deformations are decoupled, and rigid body motion was scripted. While we focused on generating keyframed animations in this paper, proper coupling would allow us to solve more general control problems, such as finding the most efficient swimming patterns [Tu and Terzopoulos 1994].

Acknowledgements: This work was supported by grants from the Singapore-MIT Gambit Game Lab, the National Science Foundation (CCF-0810888), Adobe Systems, Pixar Animation Studios, and software donations from Autodesk and Adobe Systems.

Appendix

A Adjoint method

Assuming fixed initial conditions, the entire sequence of deformations and velocities $\hat{q}_i = (q_i, \dot{q}_i)$ is uniquely determined by the control sequence u_i, $i = 0, \ldots, T-1$ (Equation 1). Therefore, the objective function of Equation 3 is a function of the elements of the control sequence $\{u_i\}_i$. The adjoint method [McNamara et al. 2004; Wojtan et al. 2006] is an efficient algorithm to compute its gradient. The algorithm can be applied both to full and reduced simulation; with reduction, it operates with respect to the reduced control sequence $\{w_i\}_i$ and Equations 2 and 4. The algorithm proceeds backward from the last timestep, computing a sequence of *adjoint vectors* r_i :

$$r_i = \left(\frac{\partial f_i}{\partial \hat{q}_i}\right)^{\mathrm{T}} r_{i+1} + \left(\frac{\partial E}{\partial \hat{q}_i}\right)^{\mathrm{T}}, \qquad r_T = \left(\frac{\partial E}{\partial \hat{q}_T}\right)^{\mathrm{T}}, \tag{11}$$

from which the gradient can be computed as

$$\frac{\partial E}{\partial u} = \left[\, r_1^{\mathrm{T}} \frac{\partial f_0}{\partial u_0} + \frac{\partial E}{\partial u_0}, \;\; \ldots \;\; , r_T^{\mathrm{T}} \frac{\partial f_{T-1}}{\partial u_{T-1}} + \frac{\partial E}{\partial u_{T-1}} \,\right]. \tag{12}$$

Here, E is the objective function of Equation 3, and f_i denotes the function that maps $\hat{q}_i$ to $\hat{q}_{i+1}$, i.e., $\hat{q}_{i+1} = f_i(\hat{q}_i, u_i)$. Function f_i incorporates F and B from Equation 1, and the particular numerical integrator. With implicit integration, f_i and $\partial f_i / \partial \hat{q}_i$ cannot be evaluated directly, but are computed by solving a linear system. With implicit Newmark integration [Barbič and James 2005], we found it convenient to expand $\hat{q}_i$ to also include $\ddot{q}_i$.

References

ADAMS, B., OVSJANIKOV, M., WAND, M., SEIDEL, H.-P., AND GUIBAS, L. J. 2008. Meshless modeling of deformable shapes and their motion. In *Symp. on Computer Animation (SCA)*, 77–86.

BARBIČ, J., AND JAMES, D. L. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. on Graphics (SIGGRAPH 2005) 24*, 3, 982–990.

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.

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.

BROTMAN, L. S., AND NETRAVALI, A. N. 1988. Motion interpolation by optimal control. In *Computer Graphics (Proc. of SIGGRAPH 88)*, vol. 22, 309–315.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformations. *ACM Trans. on Graphics (SIGGRAPH 2002) 21*, 3, 586–593.

CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *Proc. of ACM SIGGRAPH 2000*, 219–228.

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

DER, K. G., SUMNER, R. W., AND POPOVIĆ, J. 2006. Inverse kinematics for reduced deformable models. *ACM Trans. on Graphics (SIGGRAPH 2006) 25*, 3, 1174–1179.

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.

GAIN, J., AND BECHMANN, D. 2008. A survey of spatial deformation from a user-centered perspective. *ACM Trans. on Graphics 27*, 4, 1–21.

GILDIN, E. 2006. *Model and controller reduction of large-scale structures based on projection methods*. PhD thesis, The Institute for Computational Engineering and Sciences, University of Texas at Austin.

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

GRZESZCZUK, R., TERZOPOULOS, D., AND HINTON, G. 1998. NeuroAnimator: Fast neural network emulation and control of physics-based models. In *Proc. of ACM SIGGRAPH 98*, 9–20.

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 Trans. on Graphics (SIGGRAPH 2006) 25*, 3, 1126–1134.

JEON, H., AND CHOI, M.-H. 2007. Interactive motion control of deformable objects using localized optimal control. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2582–2587.

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.

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

KRYSL, P., LALL, S., AND MARSDEN, J. E. 2001. Dimensional model reduction in non-linear finite element dynamics of solids and structures. *Int. J. for Numerical Methods in Engineering 51*, 479–504.

LIU, C. K., HERTZMANN, A., AND POPOVIĆ, Z. 2005. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. on Graphics (SIGGRAPH 2005) 24*, 3, 1071–1081.

LIU, Z. 1996. *Efficient animation techniques balancing both user control and physical realism*. PhD thesis, Department of Comp. Science, Princeton University.

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.

MEZGER, J., THOMASZEWSKI, B., PABST, S., AND STRASSER, W. 2008. Interactive physically-based shape editing. In *Proc. of the ACM symposium on Solid and physical modeling*, 79–89.

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

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.

PRESS, W., TEUKOLSKY, S., VETTERLING, W., AND FLANNERY, B. 2007. *Numerical recipes: The art of scientific computing*, third ed. Cambridge University Press, Cambridge, UK.

ROSE, C., GUENTER, B., BODENHEIMER, B., AND COHEN, M. 1996. Efficient generation of motion transitions using spacetime constraints. In *Proc. of ACM SIGGRAPH 96*, 147–154.

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.

SULEJMANPASIĆ, A., AND POPOVIĆ, J. 2005. Adaptation of performed ballistic motion. *ACM Trans. on Graphics 24*, 1, 165–179.

TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Trans. on Graphics (SIGGRAPH 2003) 22*, 3, 716–723.

TREUILLE, A., LEWIS, A., AND POPOVIĆ, Z. 2006. Model reduction for real-time fluids. *ACM Trans. on Graphics (SIGGRAPH 2006) 25*, 3, 826–834.

TU, X., AND TERZOPOULOS, D. 1994. Artificial fishes: Physics, locomotion, perception, behavior. In *Proc. of ACM SIGGRAPH 94*, 43–50.

TWIGG, C. D., AND JAMES, D. L. 2007. Many-worlds browsing for control of multibody dynamics. *ACM Trans. on Graphics (SIGGRAPH 2007) 26*, 3, 14:1–14:8.

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

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.