FEPR: Fast Energy Projection for Real-Time Simulation of Deformable Objects

DIMITAR DINEV*, University of Utah
TIANTIAN LIU*, University of Pennsylvania
JING LI, University of Utah
BERNHARD THOMASZEWSKI, Université de Montréal
LADISLAV KAVAN, University of Utah

Fig. 1. Our fast energy projection method produces vivid motion for an animated rabbit.

We propose a novel projection scheme that corrects energy fluctuations in simulations of deformable objects, thereby removing unwanted numerical dissipation and numerical "explosions". The key idea of our method is to first take a step using a conventional integrator, then project the result back to the constant energy-momentum manifold. We implement this strategy using *fast projection*, which only adds a small amount of overhead to existing physics-based solvers. We test our method with several implicit integration rules and demonstrate its benefits when used in conjunction with Position Based Dynamics and Projective Dynamics. When added to a dissipative integrator such as backward Euler, our method corrects the artificial damping and thus produces more vivid motion. Our projection scheme also effectively prevents instabilities that can arise due to approximate solves or large time steps. Our method is fast, stable, and easy to implement—traits that make it well-suited for real-time physics applications such as games or training simulators.

CCS Concepts: • **Computing methodologies → Physical simulation**;

Additional Key Words and Phrases: Physics-based animation, real-time simulation.

ACM Reference Format:
Dimitar Dinev, Tiantian Liu, Jing Li, Bernhard Thomaszewski, and Ladislav Kavan. 2018. FEPR: Fast Energy Projection for Real-Time Simulation of Deformable Objects. *ACM Trans. Graph.* 37, 4, Article 79 (August 2018), 12 pages. https://doi.org/10.1145/3197517.3201277

*The first two authors contributed equally to this work.

Authors' addresses: Dimitar Dinev, University of Utah; Tiantian Liu, University of Pennsylvania; Jing Li, University of Utah; Bernhard Thomaszewski, Université de Montréal; Ladislav Kavan, University of Utah.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
0730-0301/2018/8-ART79 $15.00
https://doi.org/10.1145/3197517.3201277

1 INTRODUCTION

Simulation of dynamic elastic objects is important in many areas of engineering where one of the key numerical considerations is accuracy. Consequently, many numerical integration techniques have been developed with an often implicitly assumed understanding that accuracy is paramount. This is perfectly justified in engineering applications when the goal is to simulate the dynamics of structures such as a bridge or an airplane wing. Quite different situations occur in interactive physics-based simulations, important e.g. in computer games or surgical simulators [Sui et al. 2017]. The key difference is that numerical time integration must be fast enough to provide solutions in real-time, which often means frame rates of 30Hz or more. The simulation results are immediately displayed to the user and therefore, the numerical methods must be stable, because it is unacceptable to display artifacts such as erroneously "exploded" frames to the user. With fast advances in technologies for virtual and augmented reality, we can expect increasing demands for real-time physics simulators.

Due to this intolerance to instabilities, real-time simulators typically rely on dissipative integrators such as backward Euler. However, these integrators come with two types of drawbacks: 1) the increased stability comes at the cost of artificial numerical damping which depends on the time step size and other parameters of the simulated system; 2) implicit integration rules are not easy to solve numerically. In theory, machine-precision-accurate solutions of implicit update rules can be computed, e.g., by iterating Newton's method to convergence. However, in practice, the implicit integration rules are often only solved approximatively, e.g., by linearizing the nonlinear implicit equations [Baraff and Witkin 1998]. There are also methods developed specifically for real-time physics, such

as eXtended Position Based Dynamics (XPBD) which approximates backward Euler [Macklin et al. 2016]. Unfortunately, these approximations introduce errorswhich, in turn, can lead to instabilities.

In this paper, we present a new light-weight technique to improve the quality and robustness of dynamic simulations, especially useful in real-time simulations. Our method is an "add-on" which can be applied after any combination of an integration rule and its iterative solution process. This means that software developers do not need to change their favorite integrator or their numerical solvers in order to incorporate it. Our method is motivated by two elementary observations: 1) numerical "explosions" are usually characterized by the total energy of the system increasing indefinitely; 2) artificial numerical damping is characterized by the decay of the total energy. These behaviors are due to numerical errors which contradict an important principle of nature – conservation of energy. More precisely, total energy, i.e., the sum of potential and kinetic energies, is constant in conservative dynamical systems. Nevertheless, numerical methods commonly used in real-time physics do *not* conserve energy, leading to artifacts such as the artificial damping of backward Euler. Our method can be viewed as the projection of the state of a dynamical system (positions and velocities) which restores the correct energy behavior. If this is done carefully, i.e., by not altering the state too much and paying attention also to linear and angular momentum in addition to the total energy, we observe qualitatively improved simulations in a number of practical real-time-physics settings. Our method guarantees that 1) the simulation cannot "explode" in the sense of unbounded energy increases; 2) artificial numerical damping – if present in the underlying integrator – is eliminated.

Contributions. Our main algorithmic contribution is a new numerical energy-momentum projection method which is very fast and thus capable of meeting the stringent requirements of real-time physics applications. We demonstrate that our method improves the quality of simulation results for a number of different combinations of implicit integration rules (backward Euler, BDF-2, implicit midpoint) and their numerical solution strategies (linearization, Position Based Dynamics, Projective Dynamics). Even though the principle of our method has solid theoretical foundations in classical mechanics, we caution that our approach is intended only as dynamics post-processing for computer graphics. Even though in some cases our results seem to be more accurate, we do not claim or guarantee the ability to provide more accurate numerical solutions to the equations of motion.

2 RELATED WORK

Integration in physics-based animation. Explicit integration methods are usually very fast and simple to implement, but they can be very unstable when applied to large time steps. Some schemes can be stable given certain assumptions [Zheng et al. 2017], and many adaptive schemes have been developed that can choose a suitable time step to guarantee accuracy [Debunne et al. 2001]. It is also possible to time-step different parts of the simulated objects asynchronously [Ainsley et al. 2012; Harmon et al. 2009; Thomaszewski et al. 2008]. However, these approaches are usually not used in real-time physics due to their variable computing requirements.

Implicit methods have been used in physics-based animation [Terzopoulos and Fleischer 1988a,b; Terzopoulos et al. 1987] due to their good behavior when using large time steps. Baraff and Witkin [1998] showed that backward Euler can work very well for cloth simulations using large time steps, while Choi and Ko [2002] proposed to reduce the artificial damping of backward Euler (BDF-1) by its second order version, BDF-2. Volino and Thalmann [2005] went even further and proposed to simulate cloth using implicit midpoint, which as a symplectic integrator does not introduce numerical dissipation. Combinations of implicit and explicit methods have also been explored with promising results [Bridson et al. 2003; Eberhardt et al. 2000; Fierz et al. 2011; Stern and Grinspun 2009].

Exponential integrators have been recently introduced to graphics [Chen et al. 2017; Michels et al. 2017, 2014]. Their key idea is to use the analytical solution (matrix exponential) for the fast (i.e., highly oscillatory) component of the potential. This makes them particularly well-suited for stiff systems. Lie group integrators [Iserles et al. 2000; Kobilarov et al. 2009] can also be used for specific types of problems.

Numerical Solutions for Integration Methods. Many recent methods for fast physics-based simulation rely on implicit integration, often formulated as a minimization problem [Gast et al. 2015; Martin et al. 2011]. The chief advantage of the optimization formulation is that instead of solving systems of (typically nonlinear) equations, we can instead solve a minimization problem which has numerical benefits [Kharevych et al. 2006]. The optimization formulation also enables generalizations such as inequality constraints or contact constraints [Andrews et al. 2017; Jin et al. 2017]. The specific choice of the objective function for different integration methods is discussed in the Appendix.

When computing time is limited, the minimization problem is usually not solved exactly and instead, an approximate solution is accepted. Many modern real-time physics simulators can be viewed as approximate solvers for the backward Euler optimization problem. One popular family of methods is known as Position Based Dynamics (PBD) [Bender et al. 2014; Frâncu and Moldoveanu 2017; Macklin et al. 2014; Müller et al. 2007]. Recently, it was shown that a small modification of the original PBD algorithm [Müller et al. 2007] termed XPBD makes the method converge to the backward Euler result [Macklin et al. 2016]. Faster constraint projection can be achieved using Newton's method [English and Bridson 2008; Goldenthal et al. 2007] or with Projective Dynamics (PD) [Bouaziz et al. 2014; Liu et al. 2013]. Wang et al. [2015] further accelerated PBD and PD by applying a Chebychev semi-iterative method on the GPU, and then generalized this approach to support more general materials [Wang and Yang 2016]. The *Vivace* solver [Fratarcangeli et al. 2016] accelerates the linear system solve in PD using a novel graph-coloring approach to achieve fast Gauss-Seidel solves on the GPU. Overby et al. [2017] showed that PD can be interpreted as a special case of Alternating Direction Method of Multipliers (ADMM) [Boyd et al. 2011] while [Liu et al. 2017] reformulated PD as a Quasi-Newton method. Both the ADMM and the Quasi-Newton formulations enable more general hyperelastic material models compared to the original PD constraints.

Conserved quantities. In physics, Noether's theorem says that symmetries of the Lagrangian give rise to conserved quantities,

such as energy and momenta. Another type of conserved quantity is the symplectic form, which provides constraints on how the motion can change with changing initial conditions. The ideal continuous solution of the equations of motions is symplectic and energy-momentum conserving. Unfortunately, it is usually impossible to preserve both symplecticity and energy-momentum with numerical integrators with a constant time step [Ge and Marsden 1988]. This negative result lead to the development of two branches of numerical methods: 1) symplectic integrators [Hairer et al. 2006; Marsden and West 2001] and 2) energy-momentum methods [Gonzalez 1996; Kuhl and Crisfield 1999; Kuhl and Ramm 1996; Simo et al. 1992]. Our approach can be viewed as a fast numerical method for energy-momentum conservation.

[Hughes et al. 1978; LaBudde and Greenspan 1975] are examples of early work on energy-preserving integration. However, projecting just the energy can be dangerous and diverge from the correct solution [Ortiz 1986], unless the momentum structure is respected in addition to energy [Hairer 2006]. Discrete gradients that preserve the energy-momentum structure of mechanical systems have been studied in mechanical engineering and applied mathematics [Gonzalez 1996, 2000; Harten et al. 1997; McLachlan et al. 1999]. We explored these methods but found that discrete gradients lead to highly non-linear equations which are very difficult to solve for large time steps; we discuss an example of this in Section 4. Energy-momentum methods have not enjoyed much attention in graphics. One exception is the method of [Su et al. 2013] which improves the motion quality in many cases. However, since their method cannot guarantee monotonicity in energy, simulations can still become unstable as we discuss in Section 4. Dinev et al. [2018] circumvent the issues with energy-momentum projection by blending implicit midpoint with backward Euler to prevent implicit midpoint's instabilities. As a side effect, however, the backward Euler step can introduce unwanted damping and decay of momenta.

3 METHOD

To set the stage for the technical description of our method, we start by introducing a number of important quantities. The total energy of our system is defined as $H(\mathbf{x},\mathbf{v}) = E(\mathbf{x}) + \frac{1}{2}||\mathbf{v}||^2_{\mathbf{M}}$ (where $||\mathbf{v}||^2_{\mathbf{M}} = \mathbf{v}^{\mathsf{T}}\mathbf{M}\mathbf{v}$ stands for the mass-weighted norm), i.e. the sum of potential and kinetic energies. We also define the total linear momentum $P(\mathbf{v}) = \sum_i m^i \mathbf{v}^i$ and total angular momentum $L(\mathbf{x},\mathbf{v}) = \sum_i \mathbf{x}^i \times m^i \mathbf{v}^i$ where i indexes individual particles, $\mathbf{x}^i \in \mathbb{R}^3$, $\mathbf{v}^i \in \mathbb{R}^3$ are positions and velocities of each particle and m^i are masses of corresponding vertices. If the potential is time invariant, i.e., the forces are conservative , the exact solution conserves the total energy: $H(\mathbf{x}_{n+1},\mathbf{v}_{n+1}) = H(\mathbf{x}_n,\mathbf{v}_n)$, where $(\mathbf{x}_n,\mathbf{v}_n)$ is the previous state and $(\mathbf{x}_{n+1},\mathbf{v}_{n+1})$ is the new state. If the potential is translation and rotation invariant (e.g., an elastic object without external forces or boundary conditions), the exact solution conserves momenta: $P(\mathbf{v}_{n+1}) = P(\mathbf{v}_n)$ and $L(\mathbf{x}_{n+1},\mathbf{v}_{n+1}) = L(\mathbf{x}_n,\mathbf{v}_n)$.

Numerical integrators do not compute the exact solution, but only an approximation. The error in this approximation can be further exacerbated if the integration rules are not solved exactly, but only approximated e.g. via linearization. Particularly problematic errors are consistent increases or decreases of the total energy, which can manifest themselves as "explosions" or numerical damping. To correct for these errors, we propose an energy-momentum projection method. We project the results of any time integration method onto a constant-energy manifold by solving the following optimization problem, a modified version of the projection in [Hairer 2006]:

$$\begin{aligned} \underset{\mathbf{x},\mathbf{v},s,t}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{x}-\mathbf{x}_{n+1}\|^2_{\mathbf{M}} + \frac{h^2}{2}\|\mathbf{v}-\mathbf{v}_{n+1}\|^2_{\mathbf{M}} + \frac{\epsilon}{2}(s^2+t^2) \\ \text{subj. to} \quad & H(\mathbf{x},\mathbf{v}) = H(\mathbf{x}_n,\mathbf{v}_n) \\ & P(\mathbf{v}) = P(\mathbf{v}_{n+1}) + s(P(\mathbf{v}_n) - P(\mathbf{v}_{n+1})) \\ & L(\mathbf{x},\mathbf{v}) = L(\mathbf{x}_{n+1},\mathbf{v}_{n+1}) + \\ & \qquad t(L(\mathbf{x}_n,\mathbf{v}_n) - L(\mathbf{x}_{n+1},\mathbf{v}_{n+1})) \end{aligned} \tag{1}$$

where $(\mathbf{x},\mathbf{v})$ is the resulting projected state and $s \in \mathbb{R}$ and $t \in \mathbb{R}$ are auxiliary variables (discussed below). The constant $(\mathbf{x}_{n+1},\mathbf{v}_{n+1})$ is the state computed with an arbitrary time integration method and the term $\|\mathbf{x}-\mathbf{x}_{n+1}\|^2_{\mathbf{M}} + \frac{h^2}{2}\|\mathbf{v}-\mathbf{v}_{n+1}\|^2_{\mathbf{M}}$ ensures that our projection deviates from $(\mathbf{x}_{n+1},\mathbf{v}_{n+1})$ as little as possible. In our experiments, the diagonal $\mathbf{M}$ matrix are either nodal masses for mass-spring systems or a lumped mass-matrix of linear finite elements. While not strictly necessary, we recommend to scale weight of the velocity term with h^2 (where h is the time step) in order to make the units in both terms consistent to facilitate numerical optimization. The constant ϵ is a regularization weight, encoding the mass and size of the mesh. To be specific, $\epsilon = cmr^2$ where c is a mesh independent small constant, m is the total mass, and r is the radius of the bounding sphere of the rest pose of the simulated mesh. Because our testing models have about the same size and mass, in our experiments we set ϵ directly to 0.001. However, in general we recommend using the $\epsilon = cmr^2$ formulation which ensures the regularization term has the same units as the objective.

Variables s, t**.** In an exact solution, if the potential E is translation invariant, linear momentum is conserved, i.e., $P(\mathbf{v}_n) = P(\mathbf{v}_{n+1})$ and the s parameter becomes moot as the constraint reduces to $P(\mathbf{v}) = P(\mathbf{v}_{n+1})$. Similarly, if E is rotation invariant, angular momentum is conserved, i.e., $L(\mathbf{x}_n,\mathbf{v}_n) = L(\mathbf{x}_{n+1},\mathbf{v}_{n+1})$ and the t parameter becomes moot as the constraint reduces to $L(\mathbf{x},\mathbf{v}) = L(\mathbf{x}_{n+1},\mathbf{v}_{n+1})$. Therefore if E is both translation and rotation invariant, we can discard the s and t variables and simplify the momentum constraints to: $P(\mathbf{v}) = P(\mathbf{v}_{n+1}), L(\mathbf{x},\mathbf{v}) = L(\mathbf{x}_{n+1},\mathbf{v}_{n+1})$. We initially tried this approach with momentum conserving integrators and tested with potentials E both with and without translation or rotation invariance. This did not always work, because if linear or angular momentum varies, numerical integrators (even exactly-solved momentum conserving integrators) do not compute the exact value of the momenta, but only their numerical approximation. Due to these numerical errors, requesting the projection step to exactly match the momenta $P(\mathbf{v}_{n+1})$ and $L(\mathbf{x}_{n+1},\mathbf{v}_{n+1})$ and the total energy $H(\mathbf{x}_n,\mathbf{v}_n)$ may be impossible. Specifically, in some cases we observed that the numerical integrator (specifically, we used accurately solved implicit midpoint) overshoots the momenta, resulting in momentum constraints incompatible with the total energy constraint – the momentum constraints are asking for so large velocities the total energy constraint cannot accommodate them. To be able to guarantee feasibility of our optimization problem, we introduced the auxiliary scalar variables s and t. To prove that the constrained optimization problem in Eq. 1

is always feasible, we plug in $\mathbf{x} := \mathbf{x}_n, \mathbf{v} := \mathbf{v}_n, s := 1, t := 1$ and see that all of the constraints are indeed satisfied.

Instead of using the variable s, t, one could instead toggle the momentum constraints on and off depending on which of these quantities are conserved. However, this toggling strategy needs a complicated mechanism to control, including determining which quantities are conserved in a given simulation. For example, in a spinning cube example as shown in Figure 18, only the angular momentum around the spinning axis is conserved (one scalar). This control mechanism would be more difficult to design in cases with multiple attachment constraints or collisions. Another downside of the toggling strategy is its binary nature – the conserved quantities are either required to be conserved exactly, or not at all. The latter case, corresponding to turning off some of the momentum constraints, allows the projection to change these quantities arbitrarily which may be dangerous. The s, t formulation allows for momentum changes during the projection if necessary, but restricts changes to an interpolation between the momenta of the current and previous simulation step.

Note that the s, t formulation can handle the situation where the potential E is only partially invariant to translations or rotations. For example, the linear gravity potential (i.e., the familiar *mgh*) is translation invariant but only with respect to translations in the x and z axis, assuming the gravity acts along the y axis. The vector $P(\mathbf{v}_n) - P(\mathbf{v}_{n+1}) \in \mathbb{R}^3$ used in Eq. 1 correctly reflects this property: its x and z components are always zero and linear momentum can be modified only along the y axis, as expected. An analogous situation occurs with the angular momentum, e.g., when the potential E is invariant only to a sub-group of rotations, such as an elastic cube spinning about a fixed axis.

3.1 Numerical Solution

Eq. 1 is an optimization problem with a quadratic objective and 7 equality constraints: 3 linear constraints for linear momentum, 3 quadratic constraints for angular momentum and 1 nonlinear constraint for energy. We had initially employed a general-purpose interior-point solver IPOPT [Wächter and Biegler 2006] which worked well, but was too slow. In this section we propose our own solver which takes advantage of the special structure of our optimization problem to achieve fast runtime performance.

First, we simplify the notation used in Eq. 1. Let $\mathbf{q} := [\mathbf{x}; \mathbf{v}; s; t] \in \mathbb{R}^{6m+2}$ be a stacked vector containing all variables, where m is the number of vertices of our simulated system. Furthermore, let $\mathbf{q}_{n+1} := [\mathbf{x}_{n+1}; \mathbf{v}_{n+1}; 0; 0] \in \mathbb{R}^{6m+2}$ (a constant vector), $\mathbf{D} := diag(\mathbf{M}; h^2\mathbf{M}; \epsilon; \epsilon) \in \mathbb{R}^{(6m+2)\times(6m+2)}$ (a diagonal matrix) and let $\mathbf{c} : \mathbb{R}^{6m+2} \rightarrow \mathbb{R}^7$ denote the constraints from Eq. 1 (a vector-valued function). All constraints are satisfied iff $\mathbf{c}(\mathbf{q}) = \mathbf{0}$. With this notation, we can rewrite Eq. 1 as:

$$\begin{aligned} \underset{\mathbf{q}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{q} - \mathbf{q}_{n+1}\|^2_{\mathbf{D}} \\ \text{subj. to} \quad & \mathbf{c}(\mathbf{q}) = \mathbf{0} \end{aligned} \tag{2}$$

where the minimizer $\mathbf{q}^*$ can be interpreted as the projection of $\mathbf{q}_{n+1}$ onto the energy-momentum manifold $\mathbf{c}(\mathbf{q}) = \mathbf{0}$ in the $\mathbf{D}$-metric. To find the constrained minimizer, we use Sequential Quadratic Programming (SQP) [Nocedal and Wright 2006]. The corresponding

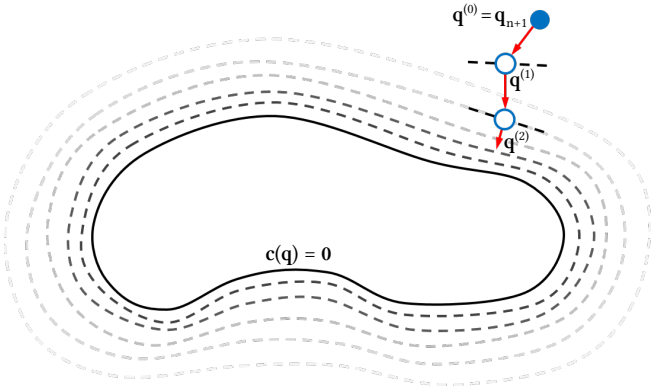

Fig. 2. Geometric interpretation of Eq. 8. Instead of projecting $\mathbf{q}_{n+1}$ directly onto the manifold $\mathbf{c}(\mathbf{q}) = \mathbf{0}$, Eq. 8 projects the next iterate $\mathbf{q}^{(k+1)}$ onto the linearized manifold $\mathbf{c}(\mathbf{q}^{(k)}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})^\mathsf{T}(\mathbf{q} - \mathbf{q}^{(k)}) = \mathbf{0}$ from the current iterate, $\mathbf{q}^{(k)}$.

Lagrangian function is $\mathcal{L}(\mathbf{q}, \lambda) = \frac{1}{2}||\mathbf{q} - \mathbf{q}_{n+1}||^2_{\mathbf{D}} + \mathbf{c}(\mathbf{q})^\mathsf{T}\lambda$ and setting its partial derivatives to zero leads to:

$$\mathbf{D}(\mathbf{q} - \mathbf{q}_{n+1}) + \nabla\mathbf{c}(\mathbf{q})\lambda = \mathbf{0} \tag{3}$$

$$\mathbf{c}(\mathbf{q}) = \mathbf{0} \tag{4}$$

where $\nabla\mathbf{c}(\mathbf{q}) = [\nabla c_1(\mathbf{q}), \nabla c_2(\mathbf{q}), \ldots, \nabla c_7(\mathbf{q})] \in \mathbb{R}^{(6m+2)\times 7}$ is the Jacobian of $\mathbf{c}$, and $c_i(\mathbf{q})$ denotes the i-th constraint. We solve the nonlinear equations using Newton's method, which leads to a sequence of iterates $(\mathbf{q}^{(1)}, \lambda^{(1)}), \ldots, (\mathbf{q}^{(k)}, \lambda^{(k)})$ converging to $(\mathbf{q}^*, \lambda^*)$. We start with an initial guess $\mathbf{q}^{(0)} = \mathbf{q}_{n+1}$ and $\lambda^{(0)} = \mathbf{0}$. Given $(\mathbf{q}^{(k)}, \lambda^{(k)})$ for any $k = 0, 1, 2, \ldots$, the next iterate $(\mathbf{q}^{(k+1)}, \lambda^{(k+1)})$ is computed by solving the following Newton-KKT (Karush-Kuhn-Tucker) system:

$$\begin{aligned} &\begin{bmatrix} \mathbf{D} + \Sigma_i \lambda_i^{(k)} \nabla^2 c_i(\mathbf{q}^{(k)}) & \nabla\mathbf{c}(\mathbf{q}^{(k)}) \\ \nabla\mathbf{c}(\mathbf{q}^{(k)})^\mathsf{T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{q}^{(k+1)} - \mathbf{q}^{(k)} \\ \lambda^{(k+1)} - \lambda^{(k)} \end{bmatrix} \\ &= -\begin{bmatrix} \mathbf{D}(\mathbf{q}^{(k)} - \mathbf{q}_{n+1}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})\lambda^{(k)} \\ \mathbf{c}(\mathbf{q}^{(k)}) \end{bmatrix} \end{aligned} \tag{5}$$

which can be simplified to:

$$\begin{aligned} &\begin{bmatrix} \mathbf{D} + \Sigma_i \lambda_i^{(k)} \nabla^2 c_i(\mathbf{q}^{(k)}) & \nabla\mathbf{c}(\mathbf{q}^{(k)}) \\ \nabla\mathbf{c}(\mathbf{q}^{(k)})^\mathsf{T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{q}^{(k+1)} - \mathbf{q}^{(k)} \\ \lambda^{(k+1)} \end{bmatrix} \\ &= -\begin{bmatrix} \mathbf{D}(\mathbf{q}^{(k)} - \mathbf{q}_{n+1}) \\ \mathbf{c}(\mathbf{q}^{(k)}) \end{bmatrix} \end{aligned} \tag{6}$$

This SQP approach converges quickly, but each iterate requires us to solve a $(6m + 9) \times (6m + 9)$ linear system to compute $(\mathbf{q}^{(k+1)}, \lambda^{(k+1)})$, which is too costly for real-time physics. We can accelerate the linear system solve using a quasi-Newton approximation by dropping the term $\Sigma_i \lambda_i^{(k)} \nabla^2 c_i(\mathbf{q}^{(k)})$, leaving only a diagonal matrix $\mathbf{D}$ as the upper-left block. This approximation is equivalent to linearizing all constraints at the current iterate $\mathbf{q}^{(k)}$ and computing the next iterate $\mathbf{q}^{(k+1)}$ as:

$$\begin{aligned} \mathbf{q}^{(k+1)} = \underset{\mathbf{q}}{\text{argmin}} \quad & \frac{1}{2}||\mathbf{q} - \mathbf{q}_{n+1}||^2_{\mathbf{D}} \\ \text{subj. to} \quad & \mathbf{c}(\mathbf{q}^{(k)}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})^\mathsf{T}(\mathbf{q} - \mathbf{q}^{(k)}) = \mathbf{0} \end{aligned} \tag{7}$$

However, even though each iteration of Eq. 7 can be computed quickly, the number of iterations required for convergence increase significantly. In particular, we observed oscillations in the iterates. Intuitively, this behavior can be explained by the fact that constraints linearized at different states can vary wildly and fight with objective term which pulls the result towards $\mathbf{q}_{n+1}$. We found that a simple modification avoids this problem:

$$\begin{aligned} \mathbf{q}^{(k+1)} := \underset{\mathbf{q}}{\text{argmin}} \quad & \frac{1}{2}||\mathbf{q}-\mathbf{q}^{(k)}||^2_{\mathbf{D}} \\ \text{subj. to} \quad & \mathbf{c}(\mathbf{q}^{(k)}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}(\mathbf{q}-\mathbf{q}^{(k)}) = \mathbf{0} \end{aligned} \tag{8}$$

The only difference is that in Eq. 8 we use $\mathbf{q}^{(k)}$ in the objective instead of $\mathbf{q}_{n+1}$ as in Eq. 7. This way, the oscillatory convergence paths are avoided and the iterative process quickly converges to a solution which exactly satisfies all of our constraints. A geometric interpretation of the sequence generated by this iterative process can be seen in Figure 2.

After this modification, the solution is no longer exactly minimizing the $\mathbf{D}-$distance from $\mathbf{q}_{n+1}$. However, since we use $\mathbf{q}_{n+1}$ as our initial guess, this approximation is sufficiently close and produces plausible results. In order to verify this experimentally, we ran the same simulation with both Eq. 2 and Eq. 8. As shown in Figure 3, there is only a minor visual difference between the two results (please see also the acccompanying video).

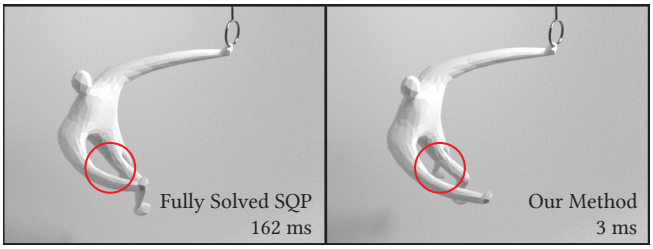

Fig. 3. Results produced by fully solved SQP (Eq. 2) (left) and by our modified optimization problem (Eq. 8) (right). Although different (as circled), our method produces qualitatively similar results with fully converged SQP, while being much faster.

In a similar way as before, we solve Eq. 8 using Lagrange multipliers. The corresponding Lagrangian is:

$$\hat{\mathcal{L}}(\mathbf{q},\lambda) = \frac{1}{2}||\mathbf{q}-\mathbf{q}^{(k)}||^2_{\mathbf{D}} + \left(\mathbf{c}(\mathbf{q}^{(k)}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}(\mathbf{q}-\mathbf{q}^{(k)})\right)^{\mathsf{T}}\lambda \tag{9}$$

and the solution $(\mathbf{q}^{(k+1)}, \lambda^{(k+1)})$ is characterized by vanishing partial derivatives of the Lagrangian:

$$\mathbf{D}\left(\mathbf{q}^{(k+1)} - \mathbf{q}^{(k)}\right) + \nabla\mathbf{c}(\mathbf{q}^{(k)})\lambda^{(k+1)} = \mathbf{0} \tag{10}$$

$$\mathbf{c}(\mathbf{q}^{(k)}) + \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}(\mathbf{q}^{(k+1)} - \mathbf{q}^{(k)}) = \mathbf{0} \tag{11}$$

We can rearrange these equations into the familiar Karush-Kuhn-Tucker (KKT) matrix form:

$$\begin{bmatrix} \mathbf{D} & \nabla\mathbf{c}(\mathbf{q}^{(k)}) \\ \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{q}^{(k+1)} - \mathbf{q}^{(k)} \\ \lambda^{(k+1)} \end{bmatrix} = - \begin{bmatrix} \mathbf{0} \\ \mathbf{c}(\mathbf{q}^{(k)}) \end{bmatrix} \tag{12}$$

This time, the matrix of this KKT system is a symmetric $(6m+9)\times(6m+9)$ matrix with special structure which can be exploited to achieve fast solves. In particular, since D is constant and diagonal, the Schur Complement $\mathbf{S} = \nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}\mathbf{D}^{-1}\nabla\mathbf{c}(\mathbf{q}^{(k)})$ can be computed very efficiently, leading to the reduced system:

$$\left(\nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}\mathbf{D}^{-1}\nabla\mathbf{c}(\mathbf{q}^{(k)})\right)\lambda^{(k+1)} = \mathbf{c}(\mathbf{q}^{(k)}) \tag{13}$$

This is a dense 7×7 linear system, therefore, solving for $\lambda^{(k+1)}$ is very efficient. The "slowest" part is the matrix multiplication in $\nabla\mathbf{c}(\mathbf{q}^{(k)})^{\mathsf{T}}\mathbf{D}^{-1}\nabla\mathbf{c}(\mathbf{q}^{(k)})$ which has asymptotic complexity $O(m)$. Because $\mathbf{D}$ is positive definite, the Schur complement $\mathbf{S}$ is guaranteed to be a positive semi-definite matrix. Furthermore, if the constraint Jacobian has full rank, $\mathbf{S}$ will be positive definite and thus invertible. In theory $\nabla\mathbf{c}$ can be rank-deficient in very special cases. For example, when an object is in its rest pose with zero velocity, the gradient of the energy conservation constraint will be zero, causing the entire first column of $\nabla\mathbf{c}$ to be zero. For robustness, we have implemented a test by checking the determinant of $\mathbf{S}$ (which is fast because $\mathbf{S}$ is just a 7×7 matrix). If the determinant is close to zero, we apply regularization $\mathbf{S} + 10^{-7}\mathbf{I}$. In practice, however, we have never observed this case in our experiments. Having computed $\lambda^{(k+1)}$, we obtain the new iterate as

$$\mathbf{q}^{(k+1)} = \mathbf{q}^{(k)} - \mathbf{D}^{-1}\nabla\mathbf{c}(\mathbf{q}^{(k)})\lambda^{(k+1)} \tag{14}$$

Eq. 14 has also asymptotic complexity $O(m)$ and all of the computations involve only dense numerical algebra subroutines with small dense matrices.

We stop iterating our projection when a state $\mathbf{q}^{(k)}$ is close enough to the energy-momentum manifold $\mathbf{c}(\mathbf{q}) = \mathbf{0}$. Specifically, we check the L^1-norm of the constraint function and stop if $|\mathbf{c}(\mathbf{q}^{(k)})|_1 < 10^{-7}$. In practice, the number of iterations for our projection method is quite small, see Table 1.

Our method is inspired by and similar to the fast projection scheme of Step and Project (SAP) [Goldenthal et al. 2007] even though our optimization problem is quite different. In particular, our optimization problem contains only seven nonlinear constraints, so we can comfortably rely on dense matrix algebra to solve Eq. 13.

3.2 Attachments, collisions and damping

Attachments. External forces such as gravity can be included as part of the potential function E and do not require any special treatment. We also implemented attachment constraints (also known as "pin constraints"), which are zero rest-length springs with one endpoint controlled kinematically [Bouaziz et al. 2014; Müller et al. 2007]. The attachment constraints can be either fixed in space or moving, e.g., to enable user interaction with the simulated object.

Collisions. A similar strategy is employed to handle collisions. If we detect inter-penetrations, we first project the collided vertices to their closest surface point as in PBD [Müller et al. 2007]. However, this strategy can lead to oscillations if the projected vertices immediately try to return back to their penetrated states. We prevent these oscillations by temporarily including the following "collision potential" similar to repulsion springs [McAdams et al. 2011] which repels the projected vertices from the penetrated configuration:

$$E_{\text{col}}(\mathbf{x}) = \begin{cases} -((\mathbf{S}\mathbf{x} - \mathbf{x}_{\text{surf}})^{\mathsf{T}}\mathbf{n})^3 & (\mathbf{S}\mathbf{x} - \mathbf{x}_{\text{surf}})^{\mathsf{T}}\mathbf{n} < 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{S}$ is a selector matrix extracting the colliding vertex from $\mathbf{x}$, $\mathbf{x}_{\text{surf}} \in \mathbb{R}^3$ is the surface point where the colliding vertex was projected and $\mathbf{n} \in \mathbb{R}^3$ is the surface normal at $\mathbf{x}_{\text{surf}}$. We then include E_{col} into our total energy function $H(\mathbf{x}_n, \mathbf{v}_n)$ so that our projection is aware of the collisions. E_{col} can be interpreted as a cubic penalty function for inequality constraint $(\mathbf{S}\mathbf{x} - \mathbf{x}_{\text{surf}})^{\mathsf{T}}\mathbf{n} \geq 0$. We chose a cubic penalty because it yields a C^2-continuous function E_{col}. See Figure 4 for an example of collision handling.

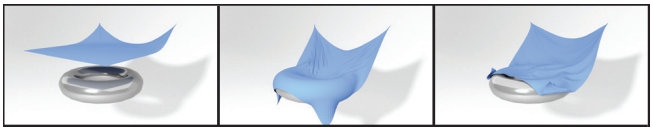

Fig. 4. A piece of draping cloth colliding with a torus.

Damping. Our method can be combined with a variety of damping models. In a similar way as [Bridson et al. 2003] who proposed to integrate elastic and damping forces in separate integration steps, we apply damping in a separate integration step executed after FEPR. After damping, we can also add forcing, such as changing the potential by moving attachment constraints. After this damping/forcing, we then recompute $H(\mathbf{x}_n, \mathbf{v}_n)$, $P(\mathbf{v}_n)$, $L(\mathbf{x}_n, \mathbf{v}_n)$ and use these updated values in the subsequent FEPR step. This way, the subsequent FEPR step will not "undo" the intentional, explicitly added damping – by recomputing $H(\mathbf{x}_n, \mathbf{v}_n)$, $P(\mathbf{v}_n)$, and $L(\mathbf{x}_n, \mathbf{v}_n)$ using the updated damped values $(\mathbf{x}_n, \mathbf{v}_n)$, the subsequent FEPR will respect the energy and momenta after the explicit damping (and forcing, if present).

A simple yet useful example is "ether drag", which corresponds to multiplying all velocities by a coefficient less than one, potentially spatially varying. This slows down all motion, including rigid-body modes. If we want to model damping only due to internal friction, rigid-body modes of motion should not be affected. This can be achieved by more sophisticated momentum-conserving damping models such as [Baraff and Witkin 1998] and [Kharevych et al. 2006] integrated in a separate implicit step [Bridson et al. 2003]. However, the computing overhead of these approaches may be too high for real-time physics. Instead, we implemented the simple yet fast momentum-conserving damping method used in Position Based Dynamics [Müller et al. 2007] which explicitly factors out global linear and angular velocities and damps out only the residual velocities corresponding to non-rigid motion.

4 RESULTS

Performance. Table 1 summarizes our testing scenarios and run times for both 1) an iterative solver of an integration rule and 2) our fast energy-projection (FEPR) algorithm. All experiments were executed on an Intel i7-6700HQ CPU at 2.6 GHz. All scenarios are simulated using a fixed time step $h = 1/30$ seconds. We tested different iterative solvers for the nonlinear optimization problem (see the Appendix for implementation details). In summary, we experimented with an L-BFGS-accelerated Projective Dynamics solver [Liu et al. 2017], eXtended Position Based Dynamics (XPBD) [Macklin et al. 2016] and a linearized solve analogous to one iteration of Newton's method [Baraff and Witkin 1998]. With implicit midpoint we also tried to compute a fully converged solution because we wanted to test whether the implicit midpoint instabilities are caused by an approximate solve. We found this was not the case – implicit midpoint explodes even if the update rules are resolved to machine-precision accuracy.

Our fast energy projection adds only a small computing overhead, typically around 10%. There are a few exceptions, for example, stiffer systems require more iterations resulting in longer runtimes. However, even these challenging examples are still faster than the solve of the integration rule.

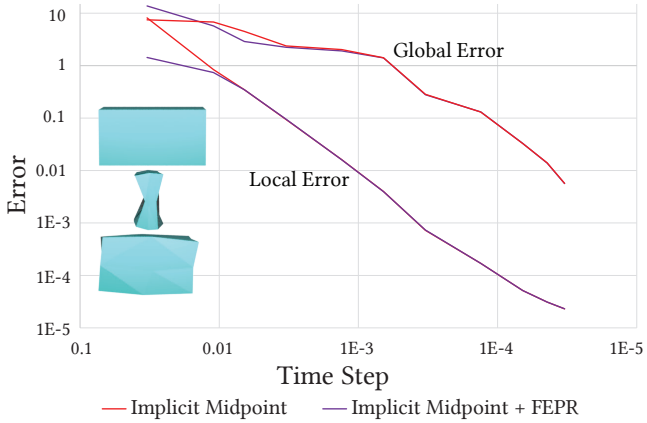

Fig. 5. We compare the effect that applying FEPR has to the order of accuracy (local and global error). Our method is consistent and does not change the order of accuracy of the underlying integrator.

Accuracy. To evaluate how FEPR affects the order of accuracy, we ran a test simulation of a pre-stretched cube and computed the accuracy of implicit midpoint without FEPR and implicit midpoint with FEPR, shown in Figure 5. We evaluated the local error by computing a ground truth solution $\mathbf{x}^*_{n+1}$ every frame starting at $\mathbf{x}_n$ and comparing it to our solution $\mathbf{x}_{n+1}$ using an absolute error metric $||\mathbf{x}^*_{n+1} - \mathbf{x}_{n+1}||$. Figure 5 shows the average local error over the course of the simulation. To compute the global error, we compared the positions $\mathbf{x}_{1.65}$ at a specific time (1.65s) to a ground-truth simulation $x^*_{1.65}$ and plotted the absolute error $||\mathbf{x}^*_{1.65} - \mathbf{x}_{1.65}||$ in Figure 5. We computed the implicit midpoint solution using Projective Dynamics with 100 iterations, corrected with our fast approximate energy projection (FEPR, Eq. 8).

From this numerical experiment we can see that FEPR shares the order of accuracy of the underlying integrator and that the method is consistent. This is not surprising: as we reduce the time step, the result of any consistent integrator converges to the ground-truth solution, which is energy-momentum conserving. An energy-momentum conserving state is a trivial solution to the optimization problem in Eq. 1 as the value of the objective is zero and all of the constraints are satisfied.

Artificial damping. Many integrators commonly used in real-time simulations are dissipative, i.e., introduce artificial numerical damping, such as backward Euler. While this improves stability, a side-effect is that the resulting motions can appear very damped.

Example		#Verts.	#Elems.	Material	Integration Method	Nonlinear Solver	Integration Time	FEPR #Iters.	FEPR. Time
Human	(Figure 17)	571	1886	Neo-Hookean	IM	PD (20 iter.)	30 ms	3.0	3 ms
Cube	(Figure 18)	602	1668	Corot	IM	PD (10 iter.)	18 ms	5.2	7 ms
Ball	(Figure 16)	889	1772	Corot	IM	PD (20 iter.)	30 ms	16.3	24 ms
Squirrel	(Figure 14)	1507	5330	Corot	IM	PD (20 iter.)	71 ms	2.7	11 ms
Hippo	(Figure 11)	2387	8879	St.VK	BE	Linearized (1 iter.)	190 ms	8.9	28 ms
Hippo	(Figure 12)	2387	8879	St.VK	BDF-2	Linearized (1 iter.)	208 ms	11.2	35 ms
Trampoline	(Figure 7)	3721	18120	Mass-Spring	BE	XPBD (40 iter.)	21 ms	4.8	7 ms
Trampoline	(Figure 8)	3721	18120	Mass-Spring	BE	PD (20 iter.)	53 ms	9.1	14 ms
Cloth	(Figure 4)	3721	18120	Mass-Spring	BE	PD (40 iter.)	92 ms	4.1	6 ms
Cactus	(Figure 13)	5261	17631	Corot	IM	Converged Solution	211 ms	2.0	21 ms
Jelly	(Figure 6)	6073	22054	Corot	BE	PD (20 iter.)	350 ms	9.2	119 ms
Octopus	(Figure 17)	7502	30010	Corot	IM	Converged Solution	1864 ms	5.1	120 ms
Rabbit	(Figure 10)	14261	52090	Corot	BDF-2	PD (40 iter.)	1868 ms	4.2	180 ms

Table 1. Statistics for all our testing scenarios. The reported times and numbers of iterations are averages over the entire simulation run. Both the "Integration Time" and the "FEPR Time" columns report total time, i.e., time for *all* iterations.

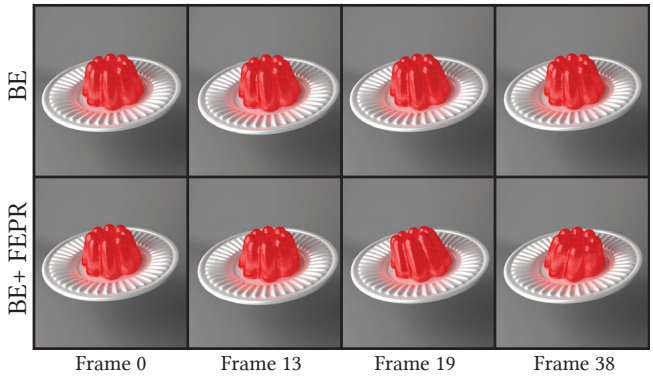

Fig. 6. A simulation of a serving of Jell-O being shaken. The motion produced by backward Euler (BE) looks rigid; adding our method (FEPR) produces vivid motion.

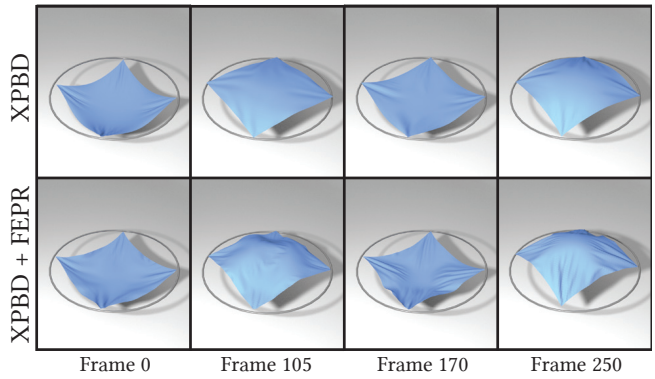

Fig. 7. FEPR (our method) applied on top of eXtended Position Based Dynamics leads to improved wrinkle formation.

Figure 6 shows a simulation where we shook a plate that had a delicious serving of Jell-O. Running this simulation using backward Euler resulted in a very rigid-looking snack, even though we did not add any damping to the simulation – all of the damping is due to backward Euler. When we applied our FEPR post-processing, we managed to restore the natural wiggling of the gelatin. In fact we added a small amount of damping using the PBD damping method (see Section 3.2), because real-world materials dissipate energy. However, aided by FEPR, this dissipation is user-controllable, which is not the case of backward Euler where the artificial damping is due to the integration rule itself and depends on the time step size and other simulation parameters.

A very popular method to simulate deformable objects is Position-Based Dynamics (PBD) [Müller et al. 2007]. Recently, Macklin et al. [2016] introduced a small but powerful modification called eXtended Position-Based Dynamics (XPBD) which correctly accounts for material stiffness and can thus theoretically converge to an exact backward Euler solution. However, in practice, XPBD is rarely iterated until convergence. Interestingly, numerical dissipation is not the main issue of early-terminated XPBD. Instead, XPBD can produce motion with artificially increased flexibility of the simulated object, as we can see in the *trampoline* example in Figure 7. Applying FEPR after XPBD results in improved wrinkle formation. It is important to note that our method cannot correct all of the errors introduced by an underlying simulator such as XPBD. In the *trampoline* example, the increased "stretchiness" caused by under-solved XPBD is still present even after FEPR, even though we make the cloth more wrinkled and its motion more vivid.

Projective Dynamics (PD) [Bouaziz et al. 2014] is another approximate solver for backward Euler specifically designed for real-time simulations. Just like XPBD, PD is also usually terminated after a fixed number of iterations, regardless of convergence. Unfortunately, this early termination seems to exacerbate the numerical damping of dissipative integrators. We ran the same *trampoline* example with Projective Dynamics, using the Quasi-Newton formulation of PD enhanced with L-BFGS [Liu et al. 2017]. The oscillations of the trampoline were very quickly damped out by PD, as shown in the

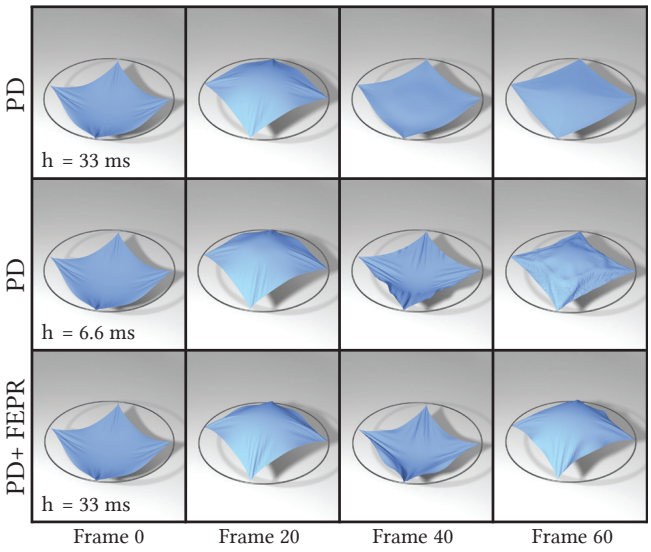

Fig. 8. FEPR fixes the damping of Projective Dynamics. Reducing the time step (as shown in the middle row) also alleviates this problem, but is much more expensive to compute than our method.

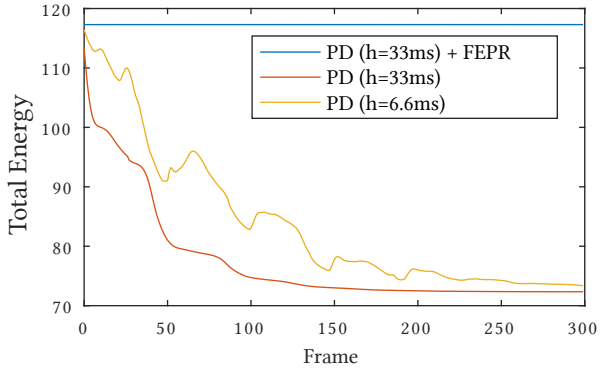

Fig. 9. Energy plot of Figure 8. Our method conserves the total energy, while backward Euler solved by Projective Dynamics damps a lot. Reducing time step helps but does not completely resolve the problem.

top row of Figure 8, leading to the trampoline looking very still. Applying FEPR to PD kept the trampoline oscillating, as shown in the bottom row of Figure 8. A graph plotting the total energy in the simulation over time can be seen in Figure 9. Indeed, our energy projection ensures the total energy is perfectly constant over the entire duration of the simulation run.

The artificial numerical damping of backward Euler can be mitigated by its second order version, BDF-2, which produces more vivid motion than backward Euler, but still contains numerical damping. In Figure 10, we took a comical rabbit character and tortured it by pelting it with a cannonball. While BDF-2 produced a nicely swinging rabbit, adding FEPR resulted in much more vivid animation and created a very panicked-looking rabbit.

Explosions. Instabilities are potentially even more problematic than numerical damping, especially in real-time simulations. Even

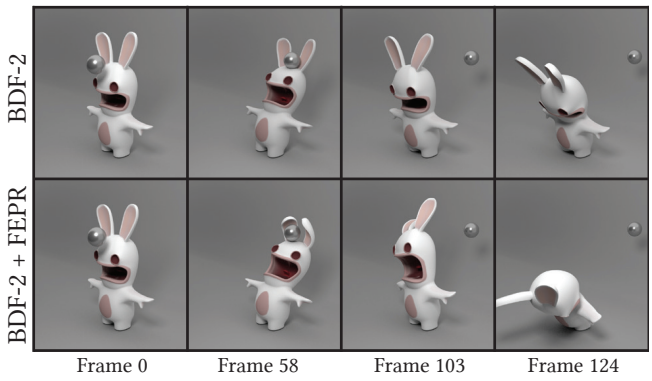

Fig. 10. Even though BDF-2 produces a nice-looking animation, adding FEPR makes it even more lively and humorous.

though integrators such as backward Euler are very stable due to their dissipative properties discussed above, this is not guaranteed if the implicit time stepping rules (nonlinear equations) are not solved exactly. A classical approximate solve of implicit integration is linearization of the non-linear equations, analogous to an undamped step of Newton's method. This approach is common in off-line simulators [Baraff and Witkin 1998], but also in real-time applications such as games [Parker and O'Brien 2009]. Unfortunately, this approach can introduce instabilities ("explosions"), as demonstrated on a *hippo* example in Figure 11. The simulation explodes due to the nonlinear backward Euler equations not being accurately solved. Applying FEPR stabilizes the simulation due to energy conservation and the hippo survives. The situation is even worse if instead of backward Euler we use BDF-2, because this integrator is less stable. Indeed, we found this to be true: we ran the same hippo simulation using linearized BDF-2 (see Figure 12) and the simulation exploded even earlier. However, our FEPR post-processing still succeeded in preventing explosions and produced a nice hippo animation.

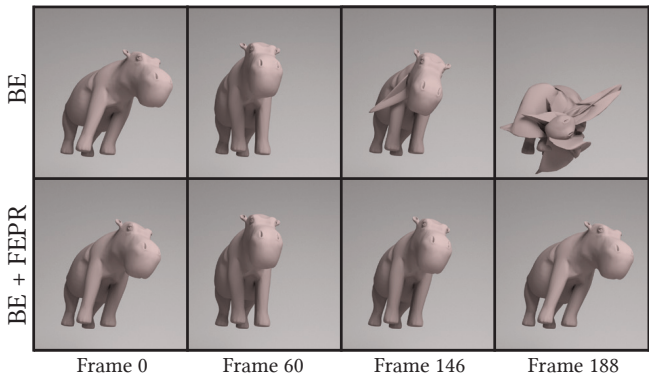

Fig. 11. A swaying hippo simulated with linearized backward Euler explodes due to linearization errors. FEPR prevents the explosions and produces plausible motion.

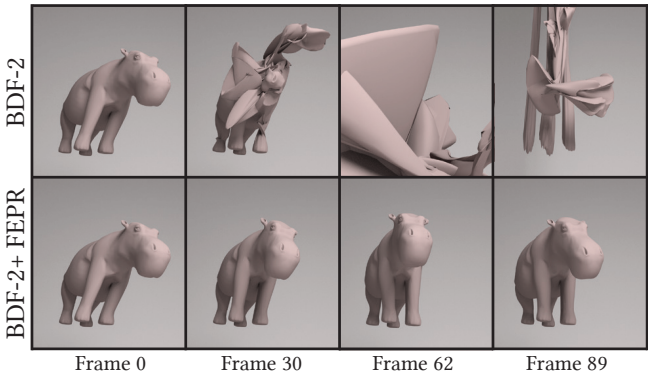

Fig. 12. Linearized BDF-2 explodes even faster than backward Euler. FEPR is still able to prevent the explosions and produce nice motion.

Things get even worse when using non-dissipative integrators, such as implicit midpoint. In this case, even machine-precision-accurate solution (fully-converged) of the implicit equations can lead to "explosions". Figure 13 shows a pre-deformed cactus simulated using fully converged implicit midpoint integration. Running implicit midpoint with a fixed time step $h = 33$ ms causes the cactus to explode, while adding our energy projection stabilizes the motion while keeping the vivid character of implicit midpoint.

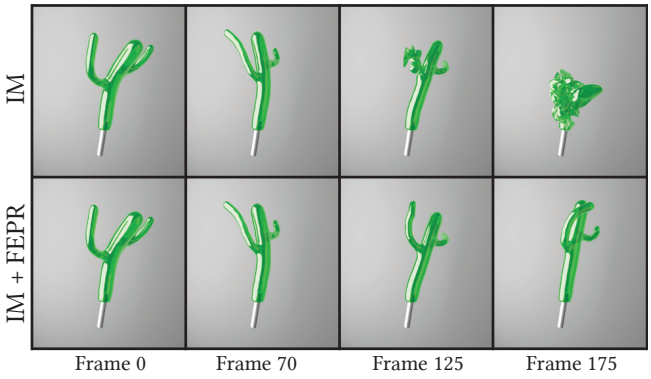

Fig. 13. Implicit midpoint explodes even when solved to full convergence. FEPR can be used to stabilize implicit midpoint and produce vivid jiggling.

Reducing the time step. Time-step reduction is the silver bullet of physics-based simulation – most problems tend to go away with smaller time steps. We simulated the trampoline model in Figure 8 with a time step of $h = 6.6$ ms, and sure enough the simulation looked much more lively, as shown in the middle row. However, this is much more expensive, taking 255 ms of computing time to advance the simulation by 1/30 second (our video frame rate). Our method produces results of similar visual quality, but costs only 67 ms of computing time per frame. Note that reducing the time step mitigates but does not fully eliminate the artificial damping problems of backward Euler, as we can see in the last column of Figure 8. Although producing more wrinkles, the trampoline simulated with reduced time steps (middle) still lags behind the one simulated with our method (bottom).

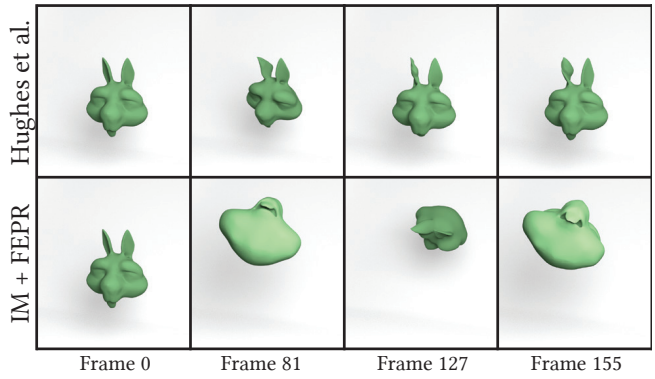

Fig. 14. Energy-conservation enforced using Lagrange multipliers [Hughes et al. 1978] eventually stops the global motion and produces high frequency vibrations that conserve energy. With our method the object continues to swing naturally.

Sufficiently small time steps can also prevent numerical explosions. We found that a time step of at least $h = 6.6$ ms was necessary to stabilize Figure 11, totaling to a computing time of 919 ms per frame. Our method is much faster, taking only 218 ms per frame. Similarly, the critical time step for Figure 12 was $h = 2.2$ ms, and Figure 13 required $h = 5.5$ ms to survive the entire simulation run, costing 2543 ms (hippo) and 8278 ms (cactus) per frame. Our method only requires 212 ms (hippo) and 232 ms (cactus) to stabilize the simulations and produces animations which are qualitatively similar to the results obtained with smaller time steps.

Previous energy-conserving methods. One of the early approaches for energy conservation was due to Hughes [1978], who added an energy-conserving constraint to the variational formulation of an implicit integrator and solved the resulting optimization problem using Lagrange multipliers. While this method perfectly conserves the energy, angular momentum is not conserved and visible artifacts may occur. In the example in Figure 14, this method loses angular momentum, but total energy is forced to be constant. This leads to non-physical transformation of the energy from global rotational motion into unnatural high-frequency vibrations, such as the oscillations in the ears and the teeth, while the global motion slows down and eventually stops. It is a well-known fact that preserving momentum in addition to energy helps [Hairer et al. 2006] and indeed, our method keeps the squirrel head swinging as expected.

Discrete gradient methods preserve energy and momentum by construction. There are several variants of these methods; for simulations of deformable objects, the most common is the midpoint discrete gradient ([Gonzalez 1996, 2000]). The midpoint discrete gradient has found use in mechanical engineering and can produce good results. Unfortunately, this is only the case with sufficiently small time steps, because the midpoint discrete gradient requires solving a root-finding problem $\mathbf{g}(\mathbf{x}, \mathbf{v}) = \mathbf{0}$. With larger time steps, the root finder can get stuck at a local minimum, failing to find a root. This can be observed even in a didactic one dimensional example similar to harmonic oscillator. Specifically, we ran a simulation using a quartic potential $E(x) = 128x^4$ using a time step of $h = 33$ ms and found a time step where the root solver failed. Figure 15 shows a contour plot of the merit function $\mathbf{g}(\mathbf{x}, \mathbf{v})^\mathsf{T}\mathbf{g}(\mathbf{x}, \mathbf{v})$. The goal

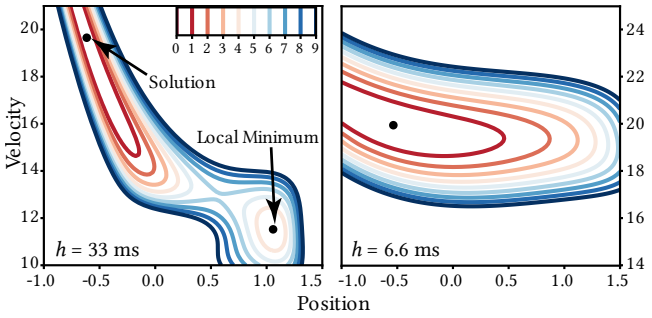

Fig. 15. A contour plot of the merit function for the discrete gradient method for a time step of 33 ms (left), where the root solver got stuck at a local minimum. With time step reduced to 6.6 ms (right) the root solver succeeded.

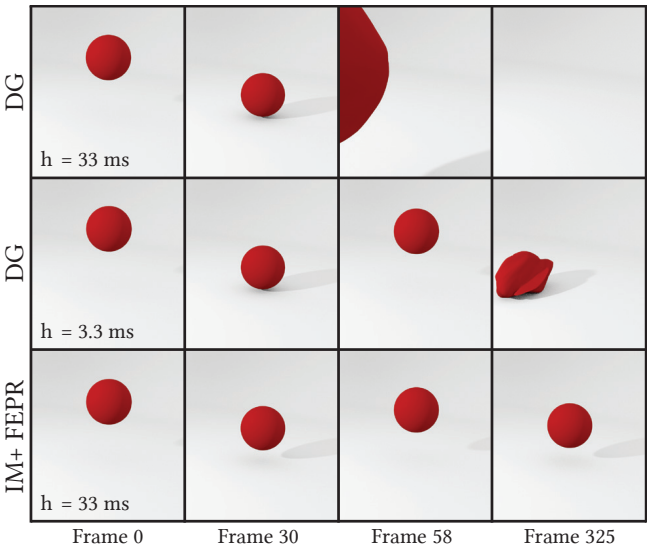

Fig. 16. The root solver used to evaluate a midpoint discrete gradient update rule can get stuck at a local minimum and produce incorrect results. This eventually happens even with ten times smaller time steps (middle row). Our method works even with large time steps and produces plausible animation (bottom).

of the root finder is to find zero values of the merit function, corresponding to roots $\mathbf{g}(\mathbf{x}, \mathbf{v}) = \mathbf{0}$. Unfortunately, depending on the initial guess, the root solver can get stuck at a local minimum of the merit function and fail. With the standard initial guess for implicit midpoint, $\mathbf{y}_n = \mathbf{x}_n + h\mathbf{v}_n$ for the position and $\mathbf{v}_n$ for the velocity, the root finder fails as illustrated in Figure 15 (left). This problem can be avoided by reducing the time step five times, to $h = 6.6$ ms, as shown in Figure 15 (right). Intuitively, this helps because with smaller time steps, the equations of the implicit update rule are "less non-linear" and local minima problems are less likely to happen.

Converging towards a local minimum produces a failure on finding the root for $\mathbf{g}(\mathbf{x}, \mathbf{v}) = \mathbf{0}$ for discrete gradient methods, therefore destroys the energy conservation property. The error of a failed step immediately gets accumulated into the dynamic system that produces significant visual artifacts in practical simulations. Our projection step cannot avoid local minima either. However, the energy constraint is guaranteed to be satisfied even in a local minimum using our optimization form in Eq. 1. We demonstrate this in Figure 16, showing a ball falling under gravity, evaluated using our method and the midpoint discrete gradient. At a time step of $h = 33$ ms, our method results in the ball bouncing up and down without any issues. The midpoint discrete gradient at the same time step runs into local minimum issues early on in the simulation and produces implausible results. Decreasing the time step by a factor of ten to $h = 3.3$ ms helps, but the midpoint discrete gradient still runs into trouble towards the end of the simulation run, see Figure 16 (middle row).

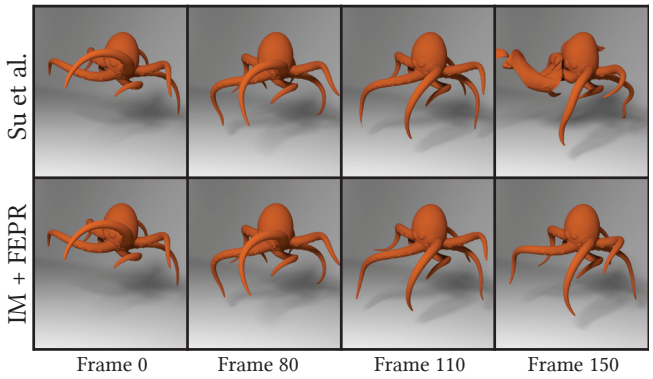

Fig. 17. Octopus hanging under gravity. Previous energy-projection methods [Simo et al. 1992; Su et al. 2013] help but can still explode. Our method produces stable and vivid motion.

Another previously explored strategy is adding an *explicit* projection step to enforce energy conservation by modifying only the velocities [Simo et al. 1992; Su et al. 2013]. This explicit projection is very fast, but large modifications of velocities may be needed because the positions are fixed. Also, the kinetic energy cannot decrease below zero, in which case the method fails to conserve energy and can explode. We demonstrate this on an octopus dangling from a fixed point, where we integrate it using a fully converged implicit midpoint integrator. Even though the velocity modification helps, energy can erroneously accumulate in the potential and the simulation explodes, see Figure 17. FEPR modifies both the positions and velocities to conserve energy, avoiding explosions. Note that even though it may seem dangerous to be modifying positions especially with stiff materials, where small changes of positions can produce large changes of potential energy, this is not a problem with our method. In such cases, FEPR changes the positions only slightly in order to achieve the desired energy level because it tries to depart from the initial guess as little as possible.

Dinev et al. [2018] proposed a method to correct the implicit midpoint result by blending it with a step of backward or forward Euler. This helps, but visible damping from backward Euler can creep in. We demonstrate this on a spinning cube example in Figure 18. In this example, we can see that when implicit midpoint overshoots the total energy, explosion is prevented by blending with backward Euler, which removes the excessive energy due to its numerical damping properties. Unfortunately, as a side effect, the blending

of backward Euler also reduces the angular momentum and slows down the global motion. Our method is an add-on on top of any integrator and avoids this problem by exactly projecting energy while also carefully taking into account both linear and angular momentum.

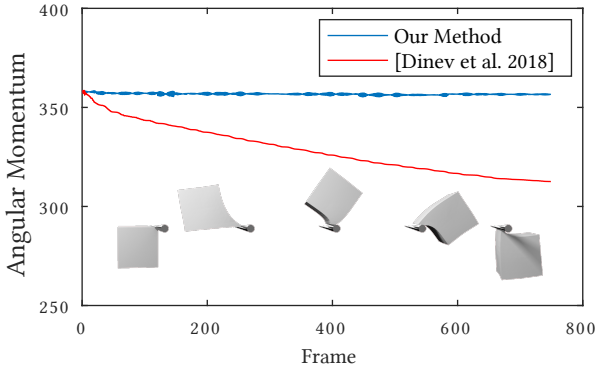

Fig. 18. Angular momentum of a spinning cube around its spinning axis. Unlike [Dinev et al. 2018], our method preserves the global rotational motion.

5 LIMITATIONS AND FUTURE WORK

While our method produces more visually plausible motion, we do not claim to produce a more numerically accurate solution to the underlying ODEs. Our method inherits some aspects of the integrator we apply it to. For example, even though our method produces more vivid motion in Figure 8, we do not obtain the same wrinkles as with a simulation with reduced time step. When added to non-dissipative integrators such as implicit midpoint, our method does not always eliminate the high-frequency oscillations introduced by implicit midpoint at large time steps. An energy-momentum projection capable of controlling frequencies of the resulting motion would be an interesting direction for future work.

Since this work focused on real-time simulation of deformable objects, we did not explore how our energy-momentum projection scheme would perform in other simulation settings such as celestial mechanics. In the future, we plan to explore additional applications areas of fast numerical methods for energy-momentum projections, e.g., correcting the "energy drift" problem in molecular dynamics [Engle et al. 2005].

6 CONCLUSION

Real-time simulations impose a strict computation budget for solving the equations of motion which in practice usually translates to large time steps and numerical approximations of implicit integration rules. This can lead to visual artifacts, such as artificial numerical damping or numerical "explosions". In this paper, we proposed a post-processing energy-projection method that corrects these artifacts, producing vivid yet stable motion. Our projection can be applied to many different physics-based simulation methods and our experiments show that it improves the visual quality of the resulting simulations. We proposed a fast numerical algorithm for energy-momentum projection which makes the method appealing for real-time simulations in applications such as games or training simulators.

ACKNOWLEDGEMENTS

We thank Robert Bridson, Mathieu Desbrun, Dominik Michels, Junior Rojas, Eftychios Sifakis, Daniel Sýkora, Nghia Troung and Cem Yuksel for many insightful discussions. We also thank Saman Sepehri Nejad for modelling and animating the delicious Jell-O and Nathan Marshak for proofreading. This material is based upon work supported by the National Science Foundation under Grant Numbers IIS-1617172 and IIS-1622360. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge the support of Activision and hardware donation from NVIDIA Corporation.

REFERENCES

Samantha Ainsley, Etienne Vouga, Eitan Grinspun, and Rasmus Tamstorf. 2012. Speculative parallel asynchronous contact mechanics. *ACM Trans. Graph.* 31, 6 (2012), 151.

Sheldon Andrews, Kenny Erleben, Paul G Kry, and Marek Teichmann. 2017. Constraint reordering for iterative multi-body simulation with contact. In *ECCOMAS Thematic Conference on Multibody Dynamic*.

David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. In *Proc. of ACM SIGGRAPH*. 43–54.

Jan Bender, Matthias Müller, Miguel A Otaduy, Matthias Teschner, and Miles Macklin. 2014. A Survey on Position-Based Simulation Methods in Computer Graphics. In *Comput. Graph. Forum*, Vol. 33. 228–251.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. Graph.* 33, 4 (2014), 154.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning* 3, 1 (2011), 1–122.

Robert Bridson, Sebastian Marino, and Ronald Fedkiw. 2003. Simulation of clothing with folds and wrinkles. *Proc. EG/ACM Symp. Computer Animation*, 28–36.

Yu Ju Chen, Uri Ascher, and Dinesh Pai. 2017. Exponential Rosenbrock-Euler Integrators for Elastodynamic Simulation. *TVCG* (2017).

Kwang-Jin Choi and Hyeong-Seok Ko. 2002. Stable but responsive cloth. *ACM Trans. Graph.* 21, 3 (2002), 604–611.

Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H Barr. 2001. Dynamic real-time deformations using space & time adaptive sampling. In *Proc. of Computer graphics and interactive techniques*. 31–36.

Dimitar Dinev, Tiantian Liu, and Ladislav Kavan. 2018. Stabilizing Integrators for Real-Time Physics. *ACM Trans. Graph.* 37, 1 (Jan. 2018), 9:1–9:19.

Bernhard Eberhardt, Olaf Etzmuß, and Michael Hauth. 2000. *Implicit-explicit schemes for fast animation with particle systems*. Springer.

Robert D Engle, Robert D Skeel, and Matthew Drees. 2005. Monitoring energy drift with shadow Hamiltonians. *J. Comput. Phys.* 206, 2 (2005), 432–452.

Elliot English and Robert Bridson. 2008. Animating developable surfaces using non-conforming elements. *ACM Trans. Graph.* 27, 3, 66.

Basil Fierz, Jonas Spillmann, and Matthias Harders. 2011. Element-wise mixed implicit-explicit integration for stable dynamic simulation of deformable objects. *Proc. EG/ACM Symp. Computer Animation*, 257–266.

Mihai Frâncu and Florica Moldoveanu. 2017. Position based simulation of solids with accurate contact handling. *Computers & Graphics* 69 (2017), 12–23.

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.* 35, 6 (2016), 214.

Theodore F Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M Teran. 2015. Optimization integrator for large time steps. *TVCG* 21, 10 (2015), 1103–1115.

Zhong Ge and Jerrold E Marsden. 1988. Lie-poisson hamilton-jacobi theory and lie-poisson integrators. *Physics Letters A* 133, 3 (1988), 134–139.

Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. 2007. Efficient simulation of inextensible cloth. *ACM Trans. Graph.* 26, 3 (2007), 49.

Oscar Gonzalez. 1996. Time integration and discrete Hamiltonian systems. *Journal of Nonlinear Science* 6, 5 (1996), 449–467.

Oscar Gonzalez. 2000. Exact energy and momentum conserving algorithms for general models in nonlinear elasticity. *Computer Methods in Applied Mechanics and Engineering* 190, 13 (2000), 1763–1783.
Ernst Hairer. 2006. *Long-time energy conservation of numerical integrators.* 162–180.
Ernst Hairer, Christian Lubich, and Gerhard Wanner. 2006. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations.* Vol. 31.
David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. 2009. Asynchronous contact mechanics. In *ACM Trans. Graph.*, Vol. 28. 87.
Amiram Harten, Peter D Lax, and Bram Van Leer. 1997. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. In *Upwind and High-Resolution Schemes.* Springer, 53–79.
TJR Hughes, TK Caughey, and WK Liu. 1978. Finite-element methods for nonlinear elastodynamics which conserve energy. *Journal of Applied Mechanics* 45, 2 (1978), 366–370.
Arieh Iserles, Hans Z Munthe-Kaas, Syvert P Nørsett, and Antonella Zanna. 2000. Lie-group methods. *Acta Numerica 2000* 9 (2000), 215–365.
Ning Jin, Wenlong Lu, Zhenglin Geng, and Ronald P Fedkiw. 2017. Inequality cloth. In *Proc. EG/ACM Symp. Computer Animation.* ACM, 16.
Liliya Kharevych, Weiwei Yang, Yiying Tong, Eva Kanso, Jerrold E Marsden, Peter Schröder, and Matthieu Desbrun. 2006. Geometric, variational integrators for computer animation. *Proc. EG/ACM Symp. Computer Animation*, 43–51.
Marin Kobilarov, Keenan Crane, and Mathieu Desbrun. 2009. Lie group integrators for animation and control of vehicles. *ACM Trans. Graph.* 28, 2 (2009), 16.
D Kuhl and MA Crisfield. 1999. Energy-conserving and decaying algorithms in nonlinear structural dynamics. *International journal for numerical methods in engineering* 45, 5 (1999), 569–599.
Detlef Kuhl and Ekkehard Ramm. 1996. Constraint energy momentum algorithm and its application to non-linear dynamics of shells. *Computer methods in applied mechanics and engineering* 136, 3-4 (1996), 293–315.
Robert A LaBudde and Donald Greenspan. 1975. Energy and momentum conserving methods of arbitrary order for the numerical integration of equations of motion. *Numer. Math.* 25, 4 (1975), 323–346.
Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. 2013. Fast simulation of mass-spring systems. *ACM Trans. Graph.* 32, 6 (2013), 209:1–7.
Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Trans. Graph.* 36, 3 (2017), 23.
Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: position-based simulation of compliant constrained dynamics. In *Proc. of Motion in Games.* 49–54.
Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2014. Unified particle physics for real-time applications. *ACM Trans. Graph.* 33, 4 (2014), 153.
Jerrold E Marsden and Matthew West. 2001. Discrete mechanics and variational integrators. *Acta Numerica 2001* 10 (2001), 357–514.
Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-based elastic materials. In *ACM Trans. Graph.*, Vol. 30. ACM, 72.
Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient elasticity for character skinning with contact and collisions. In *ACM Trans. Graph.*, Vol. 30. 37.
Robert I McLachlan, GRW Quispel, and Nicolas Robidoux. 1999. Geometric integration using discrete gradients. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 357, 1754 (1999), 1021–1045.
Dominik Michels, Vu Thai Luan, and Mayya Tokman. 2017. A Stiffly Accurate Integrator for Elastodynamic Problems. *ACM Trans. Graph.* (2017).
Dominik L Michels, Gerrit A Sobottka, and Andreas G Weber. 2014. Exponential integrators for stiff elastodynamic problems. *ACM Trans. Graph.* 33, 1 (2014), 7.
Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.
Jorge Nocedal and Stephen Wright. 2006. *Numerical optimization.*
Michael Ortiz. 1986. A note on energy conservation and stability of nonlinear time-stepping algorithms. *Computers & structures* 24, 1 (1986), 167–168.
Matthew Overby, George E Brown, Jie Li, and Rahul Narain. 2017. ADMM ⊇ Projective Dynamics: Fast Simulation of Hyperelastic Models with Dynamic Constraints. *TVCG* 23, 10 (2017), 2222–2234.
Eric G Parker and James F O'Brien. 2009. Real-time deformation and fracture in a game environment. *Proc. EG/ACM Symp. Computer Animation*, 165–175.
Juan C Simo, N Tarnow, and KK Wong. 1992. Exact energy-momentum conserving algorithms and symplectic schemes for nonlinear dynamics. *Computer methods in applied mechanics and engineering* 100, 1 (1992), 63–116.
Ari Stern and Eitan Grinspun. 2009. Implicit-explicit variational integration of highly oscillatory problems. *Multiscale Modeling & Simulation* 7, 4 (2009), 1779–1794.
Jonathan Su, Rahul Sheth, and Ronald Fedkiw. 2013. Energy conservation for the simulation of deformable bodies. *TVCG* 19, 2 (2013), 189–200.
Yuan Sui, Jun J Pan, Hong Qin, Hao Liu, and Yun Lu. 2017. Real-time simulation of soft tissue deformation and electrocautery procedures in laparoscopic rectal cancer radical surgery. *The International Journal of Medical Robotics and Computer Assisted Surgery* (2017).
Demetri Terzopoulos and Kurt Fleischer. 1988a. Deformable models. *The Visual Computer* 4, 6 (1988), 306–331.
Demetri Terzopoulos and Kurt Fleischer. 1988b. Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In *Computer Graphics (Proceedings of SIGGRAPH)*, Vol. 22. 269–278.
Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically deformable models. In *Computer Graphics (Proceedings of SIGGRAPH)*, Vol. 21. 205–214.
Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straßer. 2008. Asynchronous cloth simulation. *Computer Graphics International.*
Pascal Volino and Nadia Magnenat-Thalmann. 2005. Implicit midpoint integration and adaptive damping for efficient cloth simulation. *Computer Animation and Virtual Worlds* 16, 3-4 (2005), 163–175.
Andreas Wächter and Lorenz T Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* 106, 1 (2006), 25–57.
Huamin Wang. 2015. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph.* 34, 6 (2015), 246.
Huamin Wang and Yin Yang. 2016. Descent methods for elastic body simulation on the GPU. *ACM Trans. Graph.* 35, 6 (2016), 212.
Mianlun Zheng, Zhiyong Yuan, Qianqian Tong, Guian Zhang, and Weixu Zhu. 2017. A novel unconditionally stable explicit integration method for finite element method. *The Visual Computer* (2017), 1–13.

APPENDIX

Optimization Form for Time Integration Methods. Many popular implicit integration schemes can be expressed as special cases of the following optimization problem:

$$g(\mathbf{x}) = \frac{1}{2}||\mathbf{x}-\mathbf{y}||^2_{\mathbf{M}} + \alpha h^2 E(\beta\mathbf{x}+\mathbf{z}) \tag{15}$$

where h is the time step size, $||\mathbf{x}||^2_{\mathbf{M}} = \mathbf{x}^{\mathsf{T}}\mathbf{M}\mathbf{x}$ stands for the mass-weighted norm and E is the potential energy of the system. Two scalars α, β and two vectors $\mathbf{y}$, $\mathbf{z}$ are time-integrator-dependent constants. The goal is to minimize $g(\mathbf{x})$ to find the next state position $\mathbf{x}_{n+1} := \arg\min_{\mathbf{x}} g(\mathbf{x})$ (local minimum is sufficient). Once the positions $\mathbf{x}_{n+1}$ are found, we compute the velocities $\mathbf{v}_{n+1}$ explicitly based on the update rules of the specific integrator. The specific integrator-dependent constants and update rules are as follows.

For backward Euler (BDF-1), we have:

$$\alpha = 1 \quad \beta = 1$$
$$\mathbf{y} = \mathbf{x}_n + h\mathbf{v}_n \quad \mathbf{z} = \mathbf{0}$$
$$\mathbf{v}_{n+1} := \frac{1}{h}(\mathbf{x}_{n+1} - \mathbf{x}_n)$$

For BDF-2, we have:

$$\alpha = \frac{4}{9} \quad \beta = 1$$
$$\mathbf{y} = \frac{4\mathbf{x}_n - \mathbf{x}_{n-1}}{3} + h\frac{8\mathbf{v}_n - 2\mathbf{v}_{n-1}}{9} \quad \mathbf{z} = \mathbf{0}$$
$$\mathbf{v}_{n+1} := \frac{3}{2h}(\mathbf{x}_{n+1} - \frac{4\mathbf{x}_n - \mathbf{x}_{n-1}}{3})$$

For implicit midpoint, we have:

$$\alpha = 1 \quad \beta = \frac{1}{2}$$
$$\mathbf{y} = \mathbf{x}_n + h\mathbf{v}_n \quad \mathbf{z} = \frac{\mathbf{x}_n}{2}$$
$$\mathbf{v}_{n+1} := \frac{2}{h}(\mathbf{x}_{n+1} - \mathbf{x}_n) - \mathbf{v}_n$$