Stabilizing Integrators for Real-Time Physics

DIMITAR DINEV, University of Utah
TIANTIAN LIU, University of Pennsylvania
LADISLAV KAVAN, University of Utah

We present a new time integration method featuring excellent stability and energy conservation properties, making it particularly suitable for real-time physics. The commonly used backward Euler method is stable but introduces artificial damping. Methods such as implicit midpoint do not suffer from artificial damping but are unstable in many common simulation scenarios. We propose an algorithm that blends between the implicit midpoint and forward/backward Euler integrators such that the resulting simulation is stable while introducing only minimal artificial damping. We achieve this by tracking the total energy of the simulated system, taking into account energy-changing events: damping and forcing. To facilitate real-time simulations, we propose a local/global solver, similar to Projective Dynamics, as an alternative to Newton's method. Compared to the original Projective Dynamics, which is derived from backward Euler, our final method introduces much less numerical damping at the cost of minimal computing overhead. Stability guarantees of our method are derived from the stability of backward Euler, whose stability is a widely accepted empirical fact. However, to our knowledge, theoretical guarantees have so far only been proven for linear ODEs. We provide preliminary theoretical results proving the stability of backward Euler also for certain cases of nonlinear potential functions.

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

Additional Key Words and Phrases: Real-time, physics-based animation, stability, energy conservation

ACM Reference format:
Dimitar Dinev, Tiantian Liu, and Ladislav Kavan. 2018. Stabilizing Integrators for Real-Time Physics. *ACM Trans. Graph.* 37, 1, Article 9 (January 2018), 19 pages.
https://doi.org/10.1145/3153420

1 INTRODUCTION

Numerical time integration of the equations of motion has been a classical problem in engineering since the seminal work of Euler. Numerical integration is also a key ingredient of physics-based animation. However, in computer graphics, we do not necessarily strive for accurate numerical solutions of differential equations but rather for physically plausible results. Simply put, the resulting motion needs to *look right*, depending on the needs of a particular application. The discrepancy between accuracy and plausibility is especially pronounced in real-time applications, such as computer games or surgery simulators. Interactive applications need to refresh the screen at fixed time intervals, typically 33ms or even less, to create the illusion of smooth motion. In real-time simulations, we need to advance the state of the virtual world by 33ms while using strictly less than 33ms of computing time on a given hardware (CPU/GPU). Games or training simulators are complex software systems composed of many subsystems (rendering, networking, human-computer interaction, etc.), and therefore the time budget for physics will be typically only a small fraction of the total frame time (33ms). Historically, rigid body physics has been the first success story of real-time simulations; each rigid body has only six degrees of freedom. The situation is more complicated with deformable objects, such as biological soft tissues, which require many more degrees of freedom and, consequently, much more computation.

Adaptive timestepping methods with error control are popular in scientific computing. These methods are necessary in engineering applications, such as when designing an airplane or nuclear reactor, where simulation accuracy may be critical. Unfortunately, adaptive timestepping is incompatible with the requirements of real-time applications, where we have only a limited computing budget per frame. Despite progress in parallel-in-time methods, timestepping is a fundamentally sequential process, difficult to parallelize. Real-time simulations therefore have to compromise accuracy to retain interactivity. However, the goal is to do this gracefully and retain physical plausibility by keeping the inevitable errors under control. The most striking manifestation of inaccuracy is the case of instabilities ("explosions"), where small discretization errors compound and the discrete approximation departs dramatically from the true continuous solution. The classical way of avoiding such catastrophic failures is to reduce the timestep. Unfortunately, this is not an option in real-time physics, which needs to operate within a limited computing budget.

An important feature of numerical time integrators is their conservation properties (i.e., which features of the exact continuous solution are mimicked by the numerical integrator). Nondissipative mechanical systems conserve many first integrals, notably the Hamiltonian (i.e., total energy) and momentum (both angular and linear), as well as the symplectic form of the system. Previous work can be broadly classified in two main branches: energy-momentum conserving methods and symplectic methods. Energy-momentum methods [48] try to exactly preserve the energy and momenta; however, this does not always result in plausible simulations, as we explain in Section 2. Symplectic integrators [52] focus on conserving the symplectic form, which implies momentum conservation and also good energy behavior—the total energy oscillates around the correct value. Unfortunately, with stiff systems and fixed timesteps, these oscillations can be extreme and result

This material was based on work supported by the National Science Foundation under grants 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.
Authors' addresses: D. Dinev and L. Kavan, 50 S. Central Campus Drive, University of Utah, Salt Lake City, UT 84112-9205; emails: {ddinev, ladislav}@cs.utah.edu; T. Liu, Moore 103, 3330 Walnut Street, Philadelphia, PA 19104; email: ltt1598@gmail.com.

https://doi.org/10.1145/3153420

in visual "instabilities" or "explosions" (see Section 3 for a didactic example). There are various definitions of stability in the literature, but an informal understanding is that simulations dramatically depart from the true solution. Since neither symplectic methods nor energy-momentum methods can guarantee visual plausibility for large timesteps, most real-time physics-based simulators continue to rely on backward Euler time integration, despite its numerical damping. Higher-order backward methods, such as BDF2, reduce this damping, but the remaining numerical dissipation is still obvious.

In this article, we propose a novel time integration method for real-time simulation of deformable objects that inherits the stability of backward Euler but does not suffer from artificial numerical damping. We achieve this by starting with the implicit midpoint integrator. Implicit midpoint does not introduce artificial damping, but unfortunately the energy oscillations can be dramatic and produce visually catastrophic results. We observe that these oscillations can be tamed by adding only a small contribution of backward Euler integration. Similarly, in cases where the initial implicit midpoint solve underestimates the total energy, we can correct this by blending with forward Euler. The key is to determine the right amount of blending between implicit midpoint and forward/backward Euler. We do this by tracking the total energy of our simulated system, taking account of energy-changing events such as damping (energy dissipation) or forcing (energy injection). When we detect that implicit midpoint overshoots the total energy (indicating that future timesteps could develop instabilities), we calculate which blend between implicit midpoint and backward Euler will result in an "as-energy-conserving-as-possible" state. Usually, only a very small contribution of backward Euler is sufficient to stabilize the simulation while introducing only a minimal amount of numerical damping. Similarly, we use forward Euler blending when implicit midpoint loses energy. This is not critical for guaranteeing stability but it helps to improve the visual quality of the resulting motion.

Our method derives its stability from backward Euler. Accurately solved backward Euler is known to be stable in practice, even with complicated nonlinear elastic potentials. Insufficiently accurate numerical solutions of the backward Euler equations can introduce instabilities [4]; however, these instabilities are avoidable by iterating Newton's method until convergence, resulting in highly accurate backward Euler solutions. The stability of (exact) backward Euler is well studied in case of quadratic potentials, corresponding to linear ODEs [2]. However, nonlinear deformation energies such as mass-spring systems or corotated elasticity are common in graphics applications. To our knowledge, there is no backward Euler stability proof for such nonlinear potentials. In the appendix, we provide a backward Euler stability proof for convex potential functions. Further in the appendix, we discuss the challenges of generalizing this proof to nonconvex potentials and present a stability proof for a simple yet nonconvex test potential (two connected springs). Even though this does not settle the discussion of backward Euler stability with nonlinear potentials, it is our hope that our results will inspire future research in this direction.

To obtain accurate solutions of backward Euler, we have to iterate Newton's method until convergence, which is impractically slow in real-time simulations. Therefore, a special class of quasi-Newton methods known as Projective Dynamics [7, 33], based on local/global optimization, has been developed as a computationally efficient alternative to Newton's method. Projective Dynamics is derived from backward Euler integration and therefore inherits the undesired numerical damping. In this article, we also present a local/global acceleration for implicit midpoint and its stabilization via forward/backward Euler to facilitate high-quality real-time animations. Even though local/global solves are typically not iterated until convergence, our experiments demonstrate that we still obtain stable simulations while avoiding numerical damping. The additional computing overhead over standard Projective Dynamics is small, typically on the order of 20% to 30% of extra computing time.

In summary, we present three main contributions: (1) stabilization of implicit midpoint via energy tracking and blending with forward/backward Euler, (2) a fast local/global numerical procedure for implicit midpoint, and (3) proofs of backward Euler stability for certain types of nonlinear potential functions. The first two contributions lead to a fast integrator with good conservation properties, and the third provides some theoretical insights into its stability guarantees. It is our hope that our method will be useful especially in real-time simulations with immediate applications, such as in computer games or training simulators.

2 RELATED WORK

Integration in physics-based animation. The dramatic instabilities of forward Euler have been observed since in the early days of physics-based animation. Pioneering work from the 1980s applied backward Euler integration [54–56] to remedy these issues. Subsequent works explored explicit methods such as the popular Runge-Kutta family, featuring simpler implementation and faster runtime, until Baraff and Witkin [4] demonstrated the advantages of approximately solving backward Euler, using a method analogous to one iteration of Newton's method without a line search. Despite the fact that the numerical damping of backward Euler is a well-known issue [11, 52], many physics-based systems continue to use this method. Alternative methods such as BDF2 [6, 11] reduce the numerical damping but do not eliminate it completely. Symplectic methods, including the Newmark family of integrators [49] (which for $\gamma = 1/2$ is symplectic even though not in the traditional sense [61]), do not exhibit the undesired numerical dissipation. Unfortunately, this good behavior applies only for sufficiently small timesteps, where "sufficiently" depends on the parameters of the simulated system [18, 49]. At larger timesteps, these oscillations can be very large. One solution is to adaptively change the timestep [25]. Asynchronous integrators [31, 57] offer even more flexibility by allowing us to vary the timestep sizes spatially [1, 21, 62]. Instead of varying the timestep, one could also use adaptive remeshing [45]. However, these methods are not suitable for real-time physics, where we cannot afford fluctuations in computing time.

Stability. Stability of numerical integrators is theoretically well understood in the context of linear ODEs, where the problem essentially reduces to eigenanalysis [2, 23]. However, the situation is much more complicated with nonlinear ODEs which are common in physics-based animation. As an illustration of this fact,

note that the implicit midpoint method (a symplectic integrator) is unconditionally stable when applied to linear ODEs—for instance, the stability of the numerical approximation using implicit midpoint is equivalent to the stability of the true continuous solution regardless of the size of the timestep. However, this result does not hold with nonlinear ODEs where implicit midpoint becomes only conditionally stable [18, 28]. Previous work in physics-based animation cautions against the instabilities of symplectic schemes [26, 49]. Numerical instabilities can be cured by reducing the timestep size, but unfortunately this is not possible in real-time applications that need to operate under strict computing limits.

Another way of analyzing the stability of numerical integrators is by observing their energy behavior. For example, forward Euler causes unbounded increases in the total energy during the course of a simulation, manifesting as "explosions." In this article, we consider a simulation *stable* if the total energy (the Hamiltonian) is bounded over an arbitrary number of integrator steps, assuming that the potential is bounded from below (see Appendix B for a more formal discussion).

Enforcing energy conservation. Unstable simulations are characterized by the total energy dramatically departing from its true value. A logical way to avoid this problem is by explicitly enforcing constraints on total energy. This can be implemented either as a postprocessing (projection) step after running an arbitrary integrator [29, 48, 53] or by imposing an energy-conserving constraint using Lagrange multipliers [22]. Even though enforcing energy conservation can help in some cases, there are no guarantees of improved accuracy. It has been demonstrated that enforcing energy conservation can lead to less accurate results [19]. This may seem counterintuitive; after all, the exact continuous solutions of Hamiltonian systems exactly conserve energy, so one could expect that enforcing this constraint would lead to better results. Unfortunately, this is not always the case, as numerical schemes inevitably deviate from the true continuous solution due to discretization errors. Strictly enforcing some aspects of the ideal continuous solution is dangerous because we may destroy other desirable properties, such as symplecticity. For example, if numerical error is concentrated at one part of the simulated object, the energy conservation mechanism can nonphysically "teleport" energy to remote parts of the object, which can lead to unrealistic motion (often manifesting itself as unnatural oscillations). Another caveat is that stability is not guaranteed with projection-type methods [48, 53], as kinetic energy cannot be decreased below zero. Lagrange multiplier approaches guarantee stability with arbitrarily large timesteps [22] but do not guarantee accuracy. Figure 1 shows a simple mass-spring system cloth falling under gravity with energy conservation enforced using Lagrange multipliers [22]. Even though the energy is conserved perfectly, the simulation is not visually plausible—the cloth gets "stuck" in one position and the individual elements begin to oscillate in place. This is because Newton's method arrives at a local minimum that satisfies the energy-conserving constraint but does not correspond to plausible motion.

Symplectic methods. Flows of mechanical systems without dissipation (Hamiltonian systems) exactly conserve not only energy and momenta but also the symplectic form [14]. Numerical integrators that conserve the symplectic form are called *symplectic integrators*. Symplectic integrators can be elegantly derived by discretizing the principle of least action [20, 52, 60]. Using backward error analysis, it is possible to show that symplectic integrators exhibit good long-time energy behavior [20]; in particular, the energy oscillates about its true value. However, this is true only with sufficiently small timesteps; otherwise, symplectic integrators (both explicit and implicit ones) can oscillate dramatically [3, 28], or even diverge. One can attempt to correct this via energy preservation as discussed in the previous paragraph. Unfortunately, there are known theoretical limitations—fixed timestep methods cannot have all three of the following properties: (1) symplecticity, (2) energy conservation, and (3) momentum conservation [63]. It is possible to achieve all three with adaptive timestepping [25]. However, as discussed previously, adaptive timestepping does not meet the needs of real-time simulations. Recently, there has been renewed interest in exponential integrators that combine analytical solutions of the linear part of the ODEs with numerical solutions for the nonlinear residual [39]. Exponential integrators are computationally efficient, robust, and well suited for applications such as molecular dynamics [38]. However, just like with other symplectic schemes, stability is not guaranteed, which is problematic in real-time simulations where there is no "second chance" to rerun the simulation in case of instability.

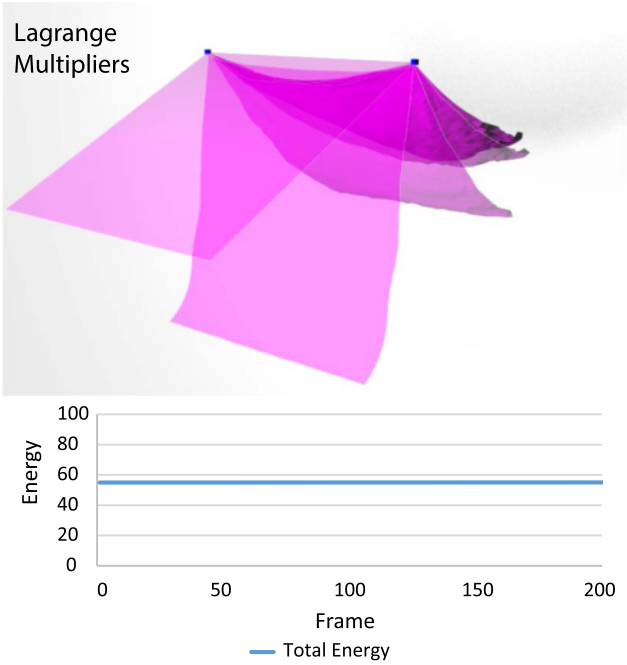

Fig. 1. A mass-spring cloth model with gravity, with energy conservation enforced using the method of Lagrange multipliers as described in Huges et al. [22] (top) and the corresponding energy graph (bottom). Although the energy is perfectly conserved, the resulting motion gets "stuck," resulting in a very poor animation (see the accompanying video).

Generalized-α methods. Good energy behavior is exhibited also by methods such as the well-known Newmark family of algorithms [46]. More general schemes, such as the generalized alpha method, have been developed [12] and applied to computer graphics [50].

These integration methods provide several parameters. Kane [24] showed that the Newmark-β method is symplectic with the common setting $\beta = 1/4, \gamma = 1/2$, even though not in the traditional sense of the canonical symplectic form. Unfortunately, similarly to the symplectic methods discussed earlier, with a fixed timestep there is no guarantee of visually plausible results.

Position-based methods. Position-based dynamics [43] is a physics-based simulation approach that specifically caters to the needs of real-time applications such as computer games. Closely related techniques have been followed in the Nucleus solver [51]. Position-based methods have been extended with more advanced solvers [27, 40, 59], finite element models [5, 41], fluid simulation [35], and a unified simulator supporting multiple phases of matter [36]. As pointed out by Liu [32], position-based dynamics can be derived from backward Euler integration (BDF1) and therefore inherits its artificial numerical damping. It is possible to upgrade position-based dynamics to BDF2 [6]. This helps, but it does not completely eliminate the undesired numerical dissipation. BDF2 can be replaced with, for example, implicit midpoint or another implicit symplectic scheme that avoids numerical damping, but this compromises stability. Real-time applications cannot risk instability, and therefore current systems accept the "lesser evil" of uncontrollable numerical damping.

Projective Dynamics. Backward Euler is often used for real-time applications due to its stability. Newton's method is the classic method for computing accurate solutions to optimization problems and is typically used to compute the solution to backward Euler in real-time animation. Unfortunately, it can be slow because the Hessian matrix changes at every iteration. In real-time applications, accuracy is usually a secondary concern to visual plausibility. To remedy these problems and provide an integration technique suitable for real-time applications, Projective Dynamics, introduced by Bouaziz et al. [7], provides a fast method for solving backward Euler that trades accuracy for speed while providing stability with fixed timesteps. Projective Dynamics has been further accelerated by advanced numerical techniques [58] and extended to more general materials [33, 44]. However, all of these methods are still based on backward Euler and inherit its artificial damping properties.

3 BACKGROUND

Forward Euler. Forward Euler (sometimes called *explicit Euler*) uses only the current state to update the next state; it is an explicit method. The update rules are very simple:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_n \tag{1}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1} f(\mathbf{x}_n), \tag{2}$$

where $\mathbf{x}_{n+1}$ and $\mathbf{v}_{n+1}$ are the positions and velocities at the next state, $\mathbf{x}_n$ and $\mathbf{v}_n$ are the positions and velocities at the current state, $\mathbf{M}$ is the mass matrix, h is the timestep, and $f(\mathbf{x}_n)$ is the net force evaluated at $\mathbf{x}_n$. Unfortunately, this scheme is not very robust, unless the time step is very small. Even though using only the current state to estimate the next state is straightforward and intuitive, it typically results in an overestimation of the total energy at every timestep. In Appendix A, we prove that for convex potential functions, forward Euler cannot decrease the total energy of the system. With larger timesteps, there are often significant increases of the total energy, leading to the commonly observed instabilities (or "explosions"). The analysis is more complicated for nonconvex potentials, as we discuss in Appendix C.

Backward Euler. Whereas forward Euler uses the current state to compute the next state, backward Euler uses the next state, $\mathbf{x}_{n+1}$ and $\mathbf{v}_{n+1}$. Since we no longer have closed-form formulas for calculating the next state, we have to solve a system of (typically nonlinear) equations to obtain $\mathbf{x}_{n+1}$ and $\mathbf{v}_{n+1}$:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_{n+1} \tag{3}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1} f(\mathbf{x}_{n+1}). \tag{4}$$

This is more computationally intensive than explicit methods, but the advantage is that the usual Courant-Friedrichs-Lewy (CFL) limitations no longer apply. Simply put, explicit methods can propagate forces only to the immediately adjacent nodes. For example, when simulating a cloth picked up at one corner, the external contact forces propagate very slowly throughout the entire system. Implicit methods can achieve such propagation during *one step*. The use of backward Euler in physics-based animation has been popularized by Baraff and Witkin [4], who demonstrated its superior behavior compared to forward Euler, especially with larger timesteps and stiff systems. Backward Euler has the exact opposite error behavior compared to forward Euler: it underestimates the energy of the true solution. This is not always a problem, because some amount of dissipation can in fact improve the visually plausibility of the motion. Unfortunately, the amount of numerical dissipation of backward Euler is not explicitly controllable by the user and instead indirectly depends on resolution, timestep, and stiffness. Backward Euler also results in loss angular momentum. Higher-order backward methods such as BDF2 produce more accurate solutions, but uncontrolled numerical dissipation is still present. Asynchronous timestepping methods such as those of Zhao et al. [62] also mitigate the numerical damping of backward Euler, but at the cost of introducing variable computing demands. This is not desirable in real-time simulations where each frame should ideally take the same amount of computing resources.

Backward Euler is known to be very stable but without much understanding as to why. Although proofs exist for the unconditional stability of backward Euler for linear ODEs and basic stability analysis has been done for some nonlinear ODEs [13, 30], we are not aware of any general proofs in the nonlinear case, such as for the deformation energies commonly used in computer animation. In Appendix B, we present a stability proof for backward Euler for convex potential functions. As in the forward Euler case, the analysis becomes more difficult for nonconvex potentials, even simple ones such as mass-spring systems (see Appendix C). Note that our theoretical analysis presented in the appendix assumes exact solutions of Equations (3) and (4). Instability can be also introduced by inaccurate numerical solvers (which compute only approximate solutions of Equations (3) and (4)), such as using only one iteration of Newton's method). These numerics-induced instabilities have been observed already by Baraff and Witkin [4], who proposed a solution by adapting the timestep.

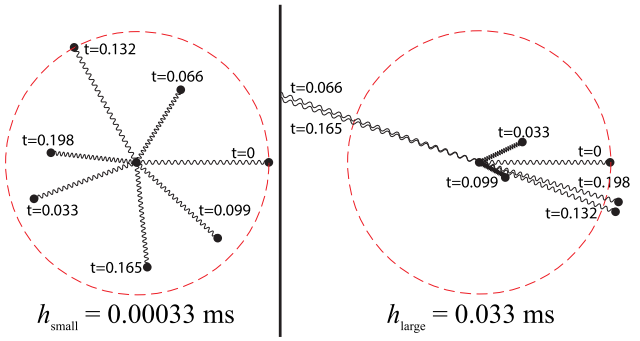

Fig. 2. A simple spring-type potential run using an implicit midpoint at a timestep $h_{\text{small}} = 0.00033$ (left) and $h_{\text{large}} = 0.033$ (right). Each point represents a state at a time t. The red circle illustrates the initial stretched length of the spring.

Implicit midpoint. Intuitively, it is obvious that both forward and backward Euler are biased: the former relies entirely on the current state, whereas the latter relies entirely on the next state. Implicit midpoint can be seen as a compromise between the two: instead of using just the current or just the next state, implicit midpoint uses their average:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}(\mathbf{v}_{n+1} + \mathbf{v}_n) \tag{5}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1} f\left(\frac{\mathbf{x}_{n+1} + \mathbf{x}_n}{2}\right). \tag{6}$$

This integrator has very nice properties—namely, it is symplectic and momentum preserving. A side effect of the preservation of the canonical symplectic form is good energy-preserving behavior [20]. However, this does not mean that the total energy has to be close to its exact value. Even in very simple simulations, it is not hard to obtain extreme energy oscillations. For example, the simulation in Figure 2 uses the potential function:

$$E_{\text{simpleSpring}}(\mathbf{x}) = \frac{1}{2}k(||\mathbf{x}||^2 - 1)^2, \tag{7}$$

which is a squared version of a classical Hookean spring of length 1m with one endpoint fixed at the origin. For our experiment, we used stiffness $k = 10^5$N/m, a mass of 0.5kg, two timesteps $h_{\text{small}} = 0.00033$s (Figure 2, left) and $h_{\text{large}} = 0.033$s (Figure 2, right), an initial velocity of (0, 10, 0)m/s, and initial length of 1.5 (corresponding to the spring being prestretched). Figure 2 shows that with h_{large}, the simulation does not behave plausibly even during the first few timesteps—contrary to our expectations, the rotating spring does not return to its rest length (1m) but instead further extends beyond the initial length of 1.5m (indicated by the red circle). If we decrease the timestep to h_{small}, this behavior disappears and the revolving spring contracts as expected, staying within the red circle.

What happened with the larger timestep $h = 0.033$s? We plot the total energy of this simple system in Figure 3. We can see that the simulation reaches incredibly large energy levels—the system starts out at an energy level of around 25,000J but quickly proceeds to energy levels of several orders of magnitude higher, at which point the moving endpoint flies off the screen with unrealistic speed. Even though the energy is fluctuating wildly, the angular momentum is still conserved, as shown in Figure 3. This poor behavior of implicit midpoint can be observed even with more complex models, such as the rotating cube modeled with corotated elasticity in Figure 4.

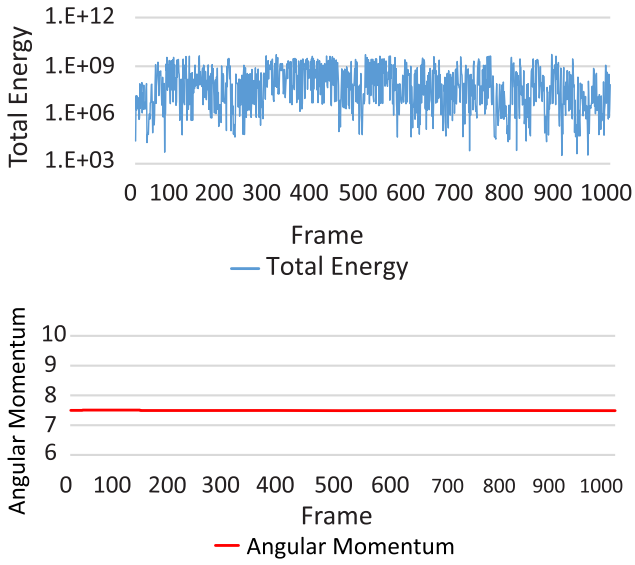

Fig. 3. Energy (top) and angular momentum (bottom) corresponding to the simulation from Figure 2 with $h = 0.033$s.

A related integrator comes from using the trapezoid rule instead of the midpoint rule. This corresponds to the Newmark integrator with $\gamma = 1/2$ and $\beta = 1/4$. Although this integrator can behave better than implicit midpoint, it is still prone to explosions (see Figure 4).

As a final remark, we note that implicit midpoint can also be written as a composition of backward and forward Euler:

$$\mathbf{x}_{n+\frac{1}{2}} = \mathbf{x}_n + \frac{h}{2}\mathbf{v}_{n+\frac{1}{2}} \tag{8}$$

$$\mathbf{v}_{n+\frac{1}{2}} = \mathbf{v}_n + \frac{h}{2}\mathbf{M}^{-1} f(\mathbf{x}_{n+\frac{1}{2}}) \tag{9}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_{n+\frac{1}{2}} + \frac{h}{2}\mathbf{v}_{n+\frac{1}{2}} \tag{10}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_{n+\frac{1}{2}} + \frac{h}{2}\mathbf{M}^{-1} f(\mathbf{x}_{n+\frac{1}{2}}). \tag{11}$$

By substituting Equation (10) into Equation (8) and substituting Equation (11) into Equation (9), we get

$$\mathbf{x}_{n+\frac{1}{2}} = \frac{\mathbf{x}_n + \mathbf{x}_{n+1}}{2} \tag{12}$$

$$\mathbf{v}_{n+\frac{1}{2}} = \frac{\mathbf{v}_n + \mathbf{v}_{n+1}}{2}. \tag{13}$$

When we substitute these back into Equation (10) and Equation (11), we get the implicit midpoint update equations (Equations (5) and (6)). If we instead do a step of forward Euler first and backward Euler second, we get the trapezoidal rule. These interesting facts motivate our method: perhaps some other combination of basic integrators could lead to a better time integration method for real-time physics?

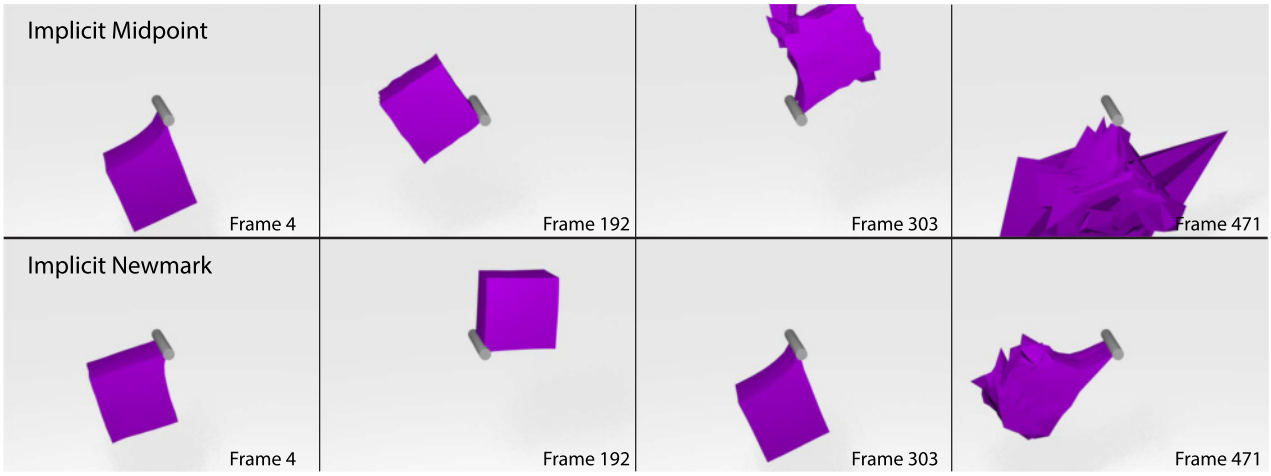

Fig. 4. A spinning deformable cube modeled with corotated elasticity simulated with timestep $h = 0.033$. After several hundred timesteps, the implicit midpoint and implicit Newmark integrator start producing visually implausible results.

4 METHOD

Overview. We will first explain our method in the case of a Hamiltonian system—that is, a dynamic system without any forcing or damping (we will discuss how to support these phenomena in Section 4.3). Our method is composed of two main steps. First, we calculate a timestep according to the implicit midpoint rule, which conserves momenta and the symplectic form and therefore provides an excellent "initial guess." As demonstrated in the previous section, the main problem of implicit midpoint is that wide oscillations of the total energy can occur, departing dramatically from a visually plausible solution. In the second step, we therefore calculate the energy increase/decrease due to implicit midpoint (with a Hamiltonian system, the exact solution conserves the total energy). If the energy erroneously increases, we correct this by computing a backward Euler step, which dissipates energy. A detail that will be important later is that the backward Euler solution process uses the state computed by implicit midpoint as an initial guess. We then solve for the optimal linear blending parameter, which will bring the total energy as close as possible to its original value. Similarly, if we detect that implicit midpoint erroneously decreased energy, we perform analogous blending with the forward Euler step, taking advantage of its energy injection property. Our experiments reveal that each timestep typically only required a small amount of blending, meaning that we do not depart too far from the symplectic and momentum-conserving solution given by implicit midpoint . This is also why we choose implicit midpoint instead of an explicit symplectic integrator; explicit integrators "explode" more dramatically at large timesteps and would lead to a larger amount of backward Euler blending. We address this in more detail in Section 5.

4.1 Optimization Form

Both implicit midpoint and backward Euler methods require an iterative solution process. In this section, we focus on accurate (up to rounding errors) solutions using Newton's method, and in Section 4.2, we discuss fast approximate solvers based on a local/global approach. In both cases, it is advantageous to formulate the implicit midpoint rule as an optimization problem (as Gast and Schroeder [16] and Gast et al. [17] did for backward Euler). To derive this optimization problem, we start by substituting Equation (6) into Equation (5), resulting in

$$\mathbf{x}_{n+1}^{\mathrm{IM}} = \mathbf{x}_n + h\mathbf{v}_n + \frac{h^2}{2}\mathbf{M}^{-1} f\left(\frac{\mathbf{x}_{n+1}^{\mathrm{IM}} + \mathbf{x}_n}{2}\right). \tag{14}$$

We assume that both $\mathbf{x}_n$ and $\mathbf{v}_n$ are known. For conciseness, in the following we denote the unknown state as $\mathbf{x}$ (instead of $\mathbf{x}_{n+1}^{\mathrm{IM}}$) and define a new symbol $\mathbf{y} := \mathbf{x}_n + h\mathbf{v}_n$. Rearranging terms, we obtain

$$\mathbf{M}(\mathbf{x} - \mathbf{y}) - \frac{h^2}{2} f\left(\frac{\mathbf{x} + \mathbf{x}_n}{2}\right) = 0. \tag{15}$$

To turn this into an optimization problem that finds $\mathbf{x}$, we need to antidifferentiate Equation (15) with respect to $\mathbf{x}$. With conservative forces (we discuss nonconservative forces in Section 4.3), we can write $f(\mathbf{x}) = -\nabla E(\mathbf{x})$, where E is an energy potential function. This allows us to perform the antidifferentiation:

$$g(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{y})^T \mathbf{M}(\mathbf{x} - \mathbf{y}) + h^2 E\left(\frac{\mathbf{x} + \mathbf{x}_n}{2}\right). \tag{16}$$

The critical points of g (i.e., $\nabla_{\mathbf{x}} g(\mathbf{x}) = 0$) correspond to the solutions of Equation (15). Having computed the positions $\mathbf{x}$ (which correspond to $\mathbf{x}_{n+1}^{\mathrm{IM}}$), computing the velocities is straightforward from Equation (5):

$$\mathbf{v}_{n+1}^{\mathrm{IM}} = \frac{2}{h}\left(\mathbf{x}_{n+1}^{\mathrm{IM}} - \mathbf{x}_n\right) - \mathbf{v}_n. \tag{17}$$

To apply Newton's method to minimize g, we need the gradient and Hessian of $g(\mathbf{x})$. These can be computed easily:

$$\nabla_{\mathbf{x}} g(\mathbf{x}) = \mathbf{M}(\mathbf{x} - \mathbf{y}) + \frac{h^2}{2}\nabla E\left(\frac{\mathbf{x} + \mathbf{x}_n}{2}\right) \tag{18}$$

$$\nabla_{\mathbf{xx}}^2 g(\mathbf{x}) = \mathbf{M} + \frac{h^2}{4}\nabla^2 E\left(\frac{\mathbf{x} + \mathbf{x}_n}{2}\right). \tag{19}$$

In this section, we assume that a solution $(\mathbf{x}_{n+1}^{\mathrm{IM}}, \mathbf{v}_{n+1}^{\mathrm{IM}})$ has been computed by iterating Newton's method until convergence (i.e.,

achieving high numerical accuracy). The next task is to compute our final state $(\mathbf{x}^{\text{our}}_{n+1}, \mathbf{v}^{\text{our}}_{n+1})$ by correcting for the total energy error in $(\mathbf{x}^{\text{IM}}_{n+1}, \mathbf{v}^{\text{IM}}_{n+1})$.

Energy error. Minimizing the energy error corresponds to bringing the following function as close as possible to zero:

$$e(\mathbf{x}, \mathbf{v}) = E(\mathbf{x}) + K(\mathbf{v}) - H_{\text{total}}, \tag{20}$$

where $K(\mathbf{v}) = \frac{1}{2}\mathbf{v}^{\mathsf{T}}\mathbf{M}\mathbf{v}$ is the kinetic energy, $E(\mathbf{x})$ is the potential energy, and H_{total} is the initial total energy of the system, specified by the initial conditions $(\mathbf{x}_0, \mathbf{v}_0)$. If $e(\mathbf{x}^{\text{IM}}_{n+1}, \mathbf{v}^{\text{IM}}_{n+1}) > 0$, it means that implicit midpoint erroneously increased the total energy, in which case we take advantage of the numerical dissipation properties of backward Euler to reduce e. When $e(\mathbf{x}^{\text{IM}}_{n+1}, \mathbf{v}^{\text{IM}}_{n+1}) < 0$, it means that implicit midpoint incorrectly decreased the total energy, which we correct using forward Euler.

Blending. The way we stabilize energy overshoots is by calculating a blend between the implicit midpoint result (which resulted in erroneous energy injection) and the backward Euler result (which may potentially remove too much energy). The results of two integrators are blended linearly, using a blending parameter $\alpha \in [0, 1]$:

$$\mathbf{x}_{n+1} = (1-\alpha)\mathbf{x}^{\text{IM}}_{n+1} + \alpha\mathbf{x}^{\text{BE}}_{n+1} \tag{21}$$

$$\mathbf{v}_{n+1} = (1-\alpha)\mathbf{v}^{\text{IM}}_{n+1} + \alpha\mathbf{v}^{\text{BE}}_{n+1}. \tag{22}$$

A higher α would yield a solution closer to backward Euler. At the extreme, $\alpha = 1$ would result in a full step of backward Euler. Assuming that backward Euler is stable, this implies stability of our method. Even though in Appendix C we prove stability of backward Euler only for certain potential functions, our experiments suggest that our method is stable even for more complex potentials. Even extreme initial conditions, as shown later in Figure 7, do not introduce instabilities in our method. Most practical simulations do not contain such large deformations, and we only need to introduce a small amount of backward Euler (i.e., α is close to zero). These small modifications early on in the simulation make a large difference in the long-term stability and perceived vividness of the motion. Without these small α modifications, the energy will gradually drift upward over a long period of time, leading to "explosions" [15]. This can be observed in Figure 4.

In an analogous fashion, we handle energy underestimates, which we correct by blending with forward Euler. Keeping the convention that higher α yields a more "damped" solution, we replace $(\mathbf{x}^{\text{IM}}_{n+1}, \mathbf{v}^{\text{IM}}_{n+1})$ with $(\mathbf{x}^{\text{FE}}_{n+1}, \mathbf{v}^{\text{FE}}_{n+1})$ and replace $(\mathbf{x}^{\text{BE}}_{n+1}, \mathbf{v}^{\text{BE}}_{n+1})$ with $(\mathbf{x}^{\text{IM}}_{n+1}, \mathbf{v}^{\text{IM}}_{n+1})$ in Equation (21) and Equation (22). Now, $\alpha = 0$ corresponds to a full step of forward Euler and $\alpha = 1$ corresponds to a full step of implicit midpoint. This allows us to adjust the energy in a symmetric way, compensating for both over- and undershoots.

We can reformulate $e(\mathbf{x}, \mathbf{v})$ from Equation (20) as a function of α. In the case of blending with backward Euler, we substitute Equation (21) and Equation (22) into Equation (20), obtaining

$$\begin{aligned} e(\alpha) = {} & E\left((1-\alpha)\mathbf{x}^{\text{IM}}_{n+1} + \alpha\mathbf{x}^{\text{BE}}_{n+1}\right) \\ & + K\left((1-\alpha)\mathbf{v}^{\text{IM}}_{n+1} + \alpha\mathbf{v}^{\text{BE}}_{n+1}\right) - H_{\text{total}}. \end{aligned} \tag{23}$$

The problem now becomes finding an α value that brings the residual energy as close as possible to zero. This can be done using a simple binary search algorithm because $e(\alpha)$ is continuous. During our experiments, we found that $e(\alpha)$ is always monotonic, making the binary search rapidly converge to a solution. Binary search requires a terminating condition. We base our terminating condition on the total energy at the last frame:

$$|e(\alpha)| < \epsilon H_{\text{total}}(\mathbf{x}_n, \mathbf{v}_n). \tag{24}$$

$\epsilon \in [0, 1]$ is a variable that controls how accurately we will conserve the energy. A small value will result in strict energy conservation, whereas a larger ϵ will result in a looser terminating condition. In practice, we found that a value of $\epsilon = 0.01$ (corresponding to a 1% error in the energy) produced good results.

In rare cases, backward Euler may not decrease the energy (we discuss an example in Appendix C). If both implicit midpoint and backward Euler increase the energy, then the binary search algorithm will not be able to find a root. When this happens, we simply take the endpoint $\alpha = 1$ as a solution. In other words, we take a full step of backward Euler, and the extra energy will be removed in future frames.

4.2 Local/Global Solver

Projective Dynamics provides a computational advantage over Newton's method by solving a backward Euler optimization problem using a local/global method, which allows us to precompute sparse Cholesky factors of the system matrix. In this section, we provide an overview of Projective Dynamics and then present a similar local/global optimization strategy for solving the implicit midpoint optimization problem (Equation (16)).

Projective Dynamics approximately solves the minimization formulation of backward Euler, which we can derive in the same manner as we derived the implicit midpoint formulation in Section 4.1:

$$g(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{y})^T\mathbf{M}(\mathbf{x} - \mathbf{y}) + h^2 E(\mathbf{x}). \tag{25}$$

The key idea behind Projective Dynamics is constraint *projection*. First, we introduce an auxiliary "projection" variable $\mathbf{p}$. To take advantage of this auxiliary variable, we need to define a special energy $E(\mathbf{x})$ for each element i:

$$E_i(\mathbf{x}) = \min_{\mathbf{p}_i \in \mathcal{M}} E(\mathbf{x}, \mathbf{S}_i\mathbf{p}), E(\mathbf{x}, \mathbf{S}_i\mathbf{p}) = ||\mathbf{G}_i\mathbf{x} - \mathbf{S}_i\mathbf{p}||^2_F, \tag{26}$$

where $\mathbf{S}_i$ is a selection matrix, $\mathbf{p}$ is a stacked vector of the projection variables that project onto the manifold $\mathcal{M}$, and $\mathbf{G}_i$ is a discrete differential operator. This Projective Dynamics energy can be used to express many common potentials, although not all potentials can be written in this form. For example, to model a mass-spring system, as Liu et al. [32] did, we use a sphere as our constraint manifold $\mathcal{M}$, and $\mathbf{G}_i \in \mathbb{R}^{3\times 3n}$ is an operator that subtracts two endpoints. $\mathbf{p} \in \mathbb{R}^{3s\times 1}$, then, is a variable that projects the current state onto the sphere $\mathcal{M}$, and the selection matrix has dimensions $\mathbf{S}_i \in \mathbb{R}^{3\times 3s}$ (where n is the number of vertices and s is the number of springs). To get a rigid-as-possible model [10], we instead use the manifold $\mathcal{M} = SO(3)$ and the deformation gradient operator [47] for $\mathbf{G}$.

The total potential $E(\mathbf{x})$ is simply a weighted sum of the element-wise potentials in Equation (26) (i.e., $E(\mathbf{x}) = \sum_i w_i E_i(\mathbf{x})$). The weights w_i are nonzero positive values that are typically the product of the rest-pose volume and stiffness of the element i. We

can now rewrite the optimization from Equation (25) using a few extra variables:

$$g(\mathbf{x},\mathbf{p}) = \frac{1}{2}(\mathbf{x}-\mathbf{y})^{\mathsf{T}}\mathbf{M}(\mathbf{x}-\mathbf{y}) + h^2\left(\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{L}\mathbf{x} - \mathbf{x}^{\mathsf{T}}\mathbf{J}\mathbf{p}\right). \quad (27)$$

$\mathbf{L} := (\sum w_i \mathbf{G}_i \mathbf{G}_i^{\mathsf{T}}) \otimes \mathbf{I}_3$ and $\mathbf{J} := (\sum w_i \mathbf{G}_i^{\mathsf{T}} \mathbf{S}_i) \otimes \mathbf{I}_3$, $\mathbf{I}_3 \in \mathbb{R}^{3\times 3}$ is the identity matrix, and $\otimes$ is the Kronecker product. The optimization is split up into a local and global step. In the local step, the positions $\mathbf{x}$ are assumed to be fixed and the projection variables $\mathbf{p}$ are solved for by being projected onto the constraint manifold (e.g., a sphere for simple springs or $SO(3)$ for corotated elasticity). The global step fixes the resulting projections $\mathbf{p}$ and solves a linear system to compute the new state. Taking $\nabla g(\mathbf{x},\mathbf{p}) = 0$ and rearranging the terms, we get the linear system:

$$(\mathbf{M} + h^2\mathbf{L})\mathbf{x} = (h^2\mathbf{J}\mathbf{p} + \mathbf{M}\mathbf{y}), \quad (28)$$

which is then solved to get the new $\mathbf{x}$ values. The state matrix $\mathbf{M} + h^2\mathbf{L}$ does not depend on $\mathbf{x}$, so it can be prefactorized and reused. This prefactorization is where the speedup compared to Newton's method comes from.

We need to make a slight modification to Equation (27) to use the local/global process for implicit midpoint. The potential energy needs to be evaluated at $\frac{\mathbf{x}+\mathbf{x}_n}{2}$ due to the update rules for implicit midpoint (Equation (16)). This changes the objective to

$$g(\mathbf{x},\mathbf{p}) = \frac{1}{2}(\mathbf{x}-\mathbf{y})^{\mathsf{T}}\mathbf{M}(\mathbf{x}-\mathbf{y}) + h^2\left(\frac{1}{2}\left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right)^{\mathsf{T}}\mathbf{L}\left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right) - \left(\frac{\mathbf{x}+\mathbf{x}_n}{2}\right)^{\mathsf{T}}\mathbf{J}\mathbf{p}\right). \quad (29)$$

Just like before, for the global solve, we set $\nabla_{\mathbf{x}} g(\mathbf{x},\mathbf{p}) = 0$ and rearrange to get the linear system we need to solve:

$$\left(\mathbf{M} + \frac{h^2}{4}\mathbf{L}\right)\mathbf{x} = \left(\mathbf{M}\mathbf{y} + \frac{h^2}{2}\mathbf{J}\mathbf{p} - \frac{h^2}{4}\mathbf{L}\mathbf{x}_n\right). \quad (30)$$

The system matrix for implicit midpoint is $\mathbf{M} + \frac{h^2}{4}\mathbf{L}$, which is slightly different from the original Projective Dynamics matrix $\mathbf{M} + h^2\mathbf{L}$ derived from backward Euler. In our method, we prefactorize both of these matrices. The sparse Cholesky factors of $\mathbf{M} + \frac{h^2}{4}\mathbf{L}$ are used in the initial implicit midpoint iterations, and the factors of $\mathbf{M} + h^2\mathbf{L}$ are used for the backward Euler stabilization (blending with forward Euler does not require any linear solves). When using backward Euler to stabilize the local/global approximation of implicit midpoint, the initial guess becomes important. Projective Dynamics typically uses the inertia term $\mathbf{y} := \mathbf{x}_n + h\mathbf{v}_n$ as the initial guess, but since we already have an approximate implicit midpoint solution, we can use it as a more effective initial guess.

Projective Dynamics is usually iterated only for a fixed number of iterations [7, 33]. For our method, we typically run 10 iterations of the local/global process for the initial implicit midpoint solve and only one iteration for the backward Euler for stabilization. This is sufficient due to the fact that we use the result of implicit midpoint as a starting point, yielding a good backward Euler solution even with only one local/global iteration (see Section 5). In addition to backward Euler, our method also uses forward Euler to inject energy into the system if necessary. However, since the forward Euler step can be easily computed explicitly, we do not need any approximate numerical solve; we directly use the update rules (Equation (1), Equation (2)) to blend with implicit midpoint.

4.3 Nonconservative Forces

Damping is an important aspect of real-world material behavior. In physics-based animation, damping is often used to control the amount of dynamic motion. As discussed in Section 4.3 of Su et al. [53], undamped simulations may result in high-frequency oscillations that expose the underlying mesh structure, which is undesired. A very simple explicit damping model is an ether drag model:

$$\mathbf{v}_i^{\text{damped}} = \mathbf{v}_i - k\frac{h}{m_i}\mathbf{v}_i. \quad (31)$$

This model has many problems, mainly that it also damps out the global motion. More sophisticated models, such as the implicit damping model proposed by Kharevych et al. [26], can mitigate these problems at the expense of extra computation. Although this model worked well in our experiments, the implicit solve that it requires was too slow for real-time physics.

This motivates our choice of damping model. Physically, we want damping that models the energy dissipation due to internal friction in the simulated material, transforming mechanical energy into heat (as opposed to modeling dissipation due to outside forces such as air drag). We also need our model to be fast while preserving the rigid body modes of the motion. Therefore, in our system, we chose to use the momentum-preserving damping model introduced in Müller et al. [43]. The key idea is to calculate the difference between the velocity of each vertex and its velocity in the best-fit rigid body approximation and damp out only these nonrigid velocity components:

$$\Delta\mathbf{v}_i = \mathbf{v}_i - \mathbf{v}_{\text{cm}} + \omega_{\text{cm}} \times \mathbf{r}_i \quad (32)$$

$$\mathbf{v}_i^{\text{damped}} = \mathbf{v}_i - k\Delta\mathbf{v}_i, \quad (33)$$

where $k \in [0,1]$ is the damping coefficient, $\mathbf{v}_i$ is the original (predamping) velocity of vertex i, $\mathbf{v}_{\text{cm}}$ is the velocity of the center of mass, and $\mathbf{r}_i = \mathbf{x}_i - \mathbf{x}_{\text{cm}}$ is the vector that points from the current position ($\mathbf{x}_i$) to the center of mass ($\mathbf{x}_{\text{cm}}$). $\mathbf{x}_{\text{cm}}$ and $\mathbf{v}_{\text{cm}}$ can be easily computed:

$$\mathbf{x}_{\text{cm}} = \frac{\sum_i m_i\mathbf{x}_i}{\sum_i m_i} \quad \mathbf{v}_{\text{cm}} = \frac{\sum_i m_i\mathbf{v}_i}{\sum_i m_i}, \quad (34)$$

where m_i is the mass of vertex i. To compute the angular velocity ω_{cm}, we need the angular momentum $\mathbf{L}$ and inertia matrix $\mathbf{I}_{\text{cm}}$. $\mathbf{L}$ is easy to find using the definition of angular momentum:

$$\mathbf{L} = \sum_i \mathbf{r}_i \times m_i\mathbf{v}_i. \quad (35)$$

The inertia matrix $\mathbf{I}_{\text{cm}}$ can be computed as

$$\mathbf{I}_{\text{cm}} = \sum_i m_i\mathbf{R}_{\times i}\mathbf{R}_{\times i}^{\mathsf{T}} \quad (36)$$

where $\mathbf{R}_{\times i} \in \mathbb{R}^{3\times 3}$ is the cross-product matrix of $\mathbf{r}_i$ (i.e., given an arbitrary vector $\mathbf{a}$, $\mathbf{R}_{\times}\mathbf{a} = \mathbf{r} \times \mathbf{a}$). Finally, we can compute the angular velocity as $\omega_{\text{cm}} = \mathbf{I}_{\text{cm}}^{-1}\mathbf{L}$. The rigid body component of the motion is fully described by $\mathbf{v}_{\text{cm}}$ and ω_{cm}. The damping model according to Equation (33) can be thought of as damping velocities that go against this rigid body motion. Choosing $k = 0$ corresponds

to no damping at all. Choosing $k = 1$ means that all nonrigid velocity components will be eliminated, resulting in pure rigid body motion. Intermediate values of k allow us to control the nonrigid modes of the motion and are useful to suppress fast oscillations, which may not be visually appealing.

This is a fast damping model that preserves the rigid motion; however, it is not a physically accurate damping model. If a body is highly deformed (e.g., a rope that is coiled), this model can cause nonphysical damping of the nonrigid motion. Furthermore, it is not dependent on the timestep, so changing the timestep can change the result. However, we choose to use this damping model due to its explicit computation, which fits our real-time constraints.

Regardless of the damping model chosen, we only need to make a slight modification to our error function. Rather than viewing H_{total} in Equation (20) as the starting energy, we can view it as the target energy we want to reach. In damped simulations, we need modify H_{total} by subtracting energy that dissipated away due to damping. We will call this value H_{diss}—the total energy intentionally dissipated from the system by our damping model. This approach enforces some limitation on the damping model—namely, it has to be done in a separate step from the optimization. We can apply the same logic for forcing—that is, intentional energy injections into the system (e.g., if the user perturbs the simulated object by applying external forces). Similarly to damping, in the case of forcing, we again update H_{total} to take into account this extra injected energy. We introduce variable H_{added} that will represent the amount of energy intentionally inserted into the system.

ALGORITHM 1: Our Method

$\mathbf{x} := \mathbf{x}_0$
$\mathbf{v} := \mathbf{v}_0$
$H_{\text{total}} :=$ calculateTotalEnergy($\mathbf{x}$, $\mathbf{v}$)
while $!exit$ **do**
  $(\mathbf{x}_n^{\text{UI}}, \mathbf{v}_n^{\text{UI}}) =$ userInteraction($\mathbf{x}_n$, $\mathbf{v}_n$)
  $H_{\text{added}} =$ calculateTotalEnergy($\mathbf{x}_n^{\text{UI}}$, $\mathbf{v}_n^{\text{UI}}$)−
    calculateTotalEnergy($\mathbf{x}_n$, $\mathbf{v}_n$)
  $H_{\text{total}} := H_{\text{total}} + H_{\text{added}}$
  $(\mathbf{x}_{\text{IM}}, \mathbf{v}_{\text{IM}}) =$ IMsolve($\mathbf{x}_n^{\text{UI}}$, $\mathbf{v}_n^{\text{UI}}$)
  if $e(\mathbf{x}_{IM}, \mathbf{v}_{IM}) > 0$ **then**
    $(\mathbf{x}_{\text{BE}}, \mathbf{v}_{\text{BE}}) :=$ BEsolve($\mathbf{x}_{\text{IM}}$, $\mathbf{v}_{\text{IM}}$, $\mathbf{x}_n$, $\mathbf{v}_n$)
    $(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) :=$ blend($\mathbf{x}_{\text{IM}}$, $\mathbf{v}_{\text{IM}}$, $\mathbf{x}_{\text{BE}}$, $\mathbf{v}_{\text{BE}}$)
  else if $e(\mathbf{x}_{IM}, \mathbf{v}_{IM}) < 0$ **then**
    $(\mathbf{x}_{\text{FE}}, \mathbf{v}_{\text{FE}}) :=$ FEsolve($\mathbf{x}_n$, $\mathbf{v}_n$)
    $(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) :=$ blend($\mathbf{x}_{\text{FE}}$, $\mathbf{v}_{\text{FE}}$, $\mathbf{x}_{\text{IM}}$, $\mathbf{v}_{\text{IM}}$)
  end
  $\mathbf{v}_{\text{damped}} :=$ damp($\mathbf{v}_{n+1}$)
  $H_{\text{diss}} =$ calculateTotalEnergy($\mathbf{x}_{n+1}$, $\mathbf{v}_{n+1}$)−
    calculateTotalEnergy($\mathbf{x}_{n+1}$, $\mathbf{v}_{\text{damped}}$)
  $H_{\text{total}} := H_{\text{total}} - H_{\text{diss}}$
  $\mathbf{v}_{n+1} := \mathbf{v}_{\text{damped}}$
  $n := n + 1$
end

There are several ways to implement forcing. In our implementation, the user can interact with the simulated object by using the mouse to move a special set of user-controlled vertices. These user-controlled vertices are not degrees of freedom of the simulation (i.e., they are not part of the system state $\mathbf{x}$ or velocities $\mathbf{v}$), but they are connected to the actual degrees of freedom (free vertices). For example, our hanging cloth example uses two user-controlled vertices as "attachment points" that are connected to the actual simulated vertices via springs, which propagate the user-specified (e.g., keyframed) motion to the simulation. The springs connecting the user-controlled and simulated vertices are included in the potential function. At the beginning of the frame, before we perform any integration, we check if the user has moved the user-controlled vertices compared to the last frame. If they have, then we can compute H_{added} by subtracting the potential value *after* user manipulation from the previous potential value, *before* user manipulation. Since we have not yet performed any updates to the system, we know that this difference in potential was entirely a result of user interaction (i.e., the energy the user injected into the system).

Combining this with the damping term, we get

$$H_{\text{total}} = H_{\text{initial}} + H_{\text{added}} - H_{\text{diss}}, \tag{37}$$

where H_{initial} is the initial total energy of the system according to the initial conditions (e.g., prestretched starting states or initial velocities correspond to larger H_{initial}). This energy-tracking process is crucial to our blending as it allows our method to handle damping and forcing, which are very common actions in interactive simulations. The pseudocode of our method is outlined in Algorithm 1. Here, BEsolve($\mathbf{x}_{\text{IM}}, \mathbf{v}_{\text{IM}}, \mathbf{x}_n, \mathbf{v}_n$) means running a backward Euler solve using $\mathbf{x}_{\text{IM}}, \mathbf{v}_{\text{IM}}$ as an initial guess. FEsolve($\mathbf{x}_n, \mathbf{v}_n$) means performing a standard forward Euler step.

Collisions. Although Bridson et al. [9] presented robust models for handling collisions (including self-collisions) using adaptive timestepping, robust real-time solutions remain a challenge. To support collisions in our current system, we use the standard model of repulsion springs [37]. Specifically, if an interpenetration is detected, we introduce collision springs with the following potential function:

$$E_c(\mathbf{x}) = \begin{cases} \frac{1}{2} k_c ||\mathbf{d} \cdot \mathbf{n}||^2 & \mathbf{d} \cdot \mathbf{n} \leq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{38}$$

Here, k_c is the stiffness for the collision springs, $\mathbf{d} = \mathbf{x} - \mathbf{x}_s$ is penetration depth of the vertex ($\mathbf{x}$ is the current position, and $\mathbf{x}_s$ is the projection of $\mathbf{x}$ onto the nearest point on the surface), and $\mathbf{n}$ is the surface normal at $\mathbf{x}_s$. This potential is included in $E(\mathbf{x})$ along with regular deformation energies. The collision detection is executed at the beginning of every frame, and all of the necessary collision springs are inserted into the potential function and taken into account during the subsequent integration step.

An important aspect of modeling collisions is modeling friction between the two colliding objects. We model friction by applying a damping force after the integration [17]. For every colliding vertex, we subtract the component of the acceleration that is tangent to collision normal $\mathbf{n}$:

$$v_i^f = v_i - k_f a_\perp(x_i), \tag{39}$$

where $k_f \in (0, 1)$ is a friction coefficient and $a_\perp(x_i) = h\frac{\nabla E_c(x_i)_\perp}{m_i}$ is the tangent component of the discrete acceleration caused by the collision penalty forces. From here, we can treat it just like

damping—we simply update the H_{diss} term from Algorithm 1 in the same way to take this intentionally dissipated energy into account. Like the damping model, this friction model is physically inaccurate and is chosen for its explicit evaluation, leading to a fast computation.

Perfectly elastic collisions (i.e., collisions without any damping) are energy-conserving phenomena. Whereas the total energy of an individual object can increase (i.e., a large moving object colliding with a small stationary object will insert energy into the small object), the total energy of the system cannot increase. For example, a ball falling onto a surface should not bounce higher than its starting point. The extra energy induced from the collision springs is therefore not included in our energy tracking (Equation (37)), as it is not energy that was originally in either object before colliding; it is temporarily introduced to handle interpenetrations. However, sometimes we want to model inelastic collisions, where energy is lost during the collision event. Although we did not use this in our examples, this energy dissipation could be tracked in exactly the same way as other damping models—for instance, by accounting for the amount of energy intentionally dissipated during inelastic collisions in the H_{diss} term (Equation (37)).

5 RESULTS

Energy conservation. In Figure 5, we compare the energy preservation behavior of our method and other methods over the course of several thousand timesteps. The simulation we used was a deformable cube modeled using corotated elasticity with linear finite elements [10, 47] spinning around a fixed axis. Backward Euler (BDF1) and its second-order counterpart (BDF2) damp out most of the energy, whereas pure implicit midpoint quickly explodes. Implicit Newmark (using $\gamma = 1/2, \beta = 1/4$, i.e., the trapezoidal rule) survives longer than implicit midpoint but still eventually explodes. We also tried to apply the energy budgeting method of Su et al. [53] to correct the implicit midpoint behavior. Even though the energy-budgeted simulation survives longer, the method fails to stabilize systems in the long term and also explodes. This is due to the fact that projection methods such as those of Su et al. [53] only modify the velocities, not positions, and therefore large erroneous potential energy can build up in the deformation modes. Our method avoids this problem by changing both positions and velocities and maintains the total energy over a long run.

Momentum conservation. Momentum conservation is another important aspect of numerical time integration; Figure 5 also compares the angular momentum conservation properties of our and previous methods. Although our method does not conserve angular momentum exactly, it preserves angular momentum much better than BDF1 or BDF2. Implicit midpoint preserves the angular momentum exactly but often exhibits dramatic errors in total energy. Our method uses implicit midpoint as a starting point and corrects the energy under/overshoots using forward/backward Euler. Even though these corrections are often small, our method is not exactly angular momentum conserving.

Alpha value. The chosen alpha for the blending is typically close to zero in reasonable simulations. Figure 6 shows the alpha values chosen for the cube example in Figure 5. The forward Euler alphas have been rescaled to an interval of $[-1, 0]$ for the purposes of

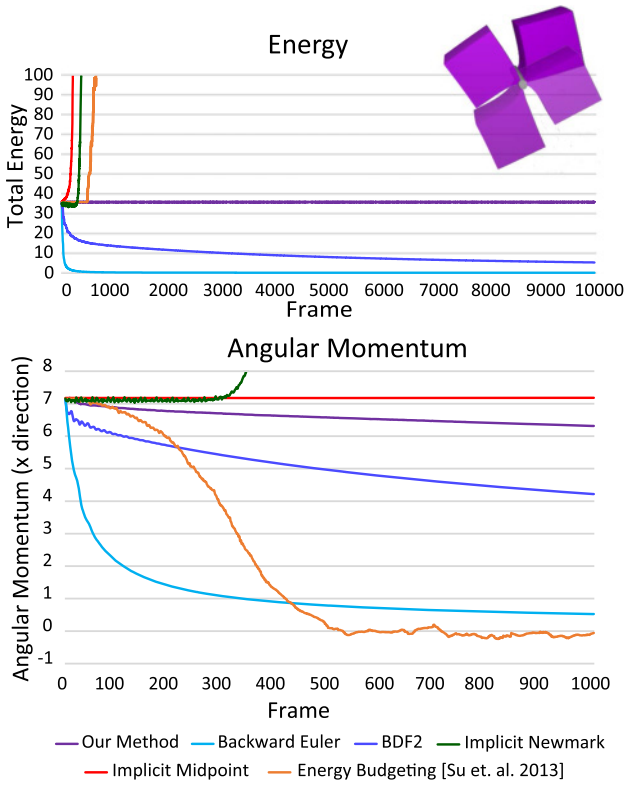

Fig. 5. A graph showing the total energy (top) and angular momentum (bottom) of a spinning elastic cube integrated using various methods all evaluated using a timestep of $h = 0.033$s. Our method is stable and preserves energy very well even in long simulation runs. Although our method does not exactly conserve angular momentum, it preserves it better than the other stable methods (backward Euler and BDF2).

showing both the forward and backward Euler blending. A value of -1 corresponds to a full forward Euler solution, and a value of 1 corresponds to a full backward Euler solution. Alpha values close to zero lead to less forward/backward Euler blending, which results in smoother and more vivid motion. Large alpha values correspond to large amounts of backward Euler blending, which can lead to a lot of damping in the angular momentum, as is the case if we use explicit symplectic Euler as a starting point. Explicit symplectic Euler's tendency to explode faster at large timesteps causes our algorithm to use an α of almost 1 for most of the simulation, which causes the angular momentum to drop to nearly 0 very rapidly.

Stability. Figure 7 shows a stability stress test of our method, featuring a cactus mesh subject to extreme initial conditions. Specifically, the initial vertex positions were randomized, as seen in Frame 1. The energy-conserving behavior keeps the simulation stable and at a constant energy level, so introducing a small amount of damping allows us to recover the original mesh. We used the momentum-preserving damping model described by Equations (32) and (33) with a damping coefficient $k = 0.08$. Implicit midpoint is unable to recover the cactus' rest shape with the same amount of damping, as the numerical instabilities overpower

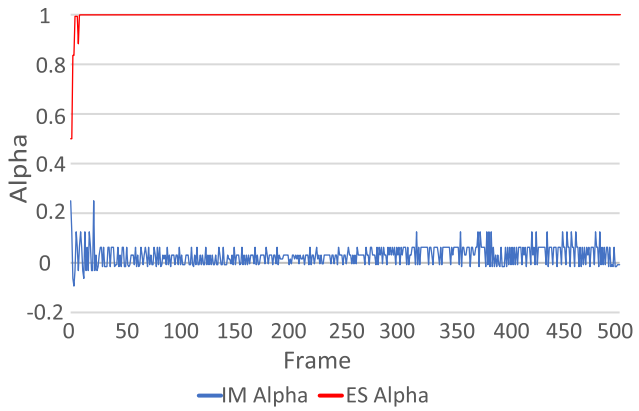

Fig. 6. The choice of blending alpha values that were used for the simulation in Figure 5, using implicit midpoint (IM) and explicit symplectic Euler (ES) as starting points.

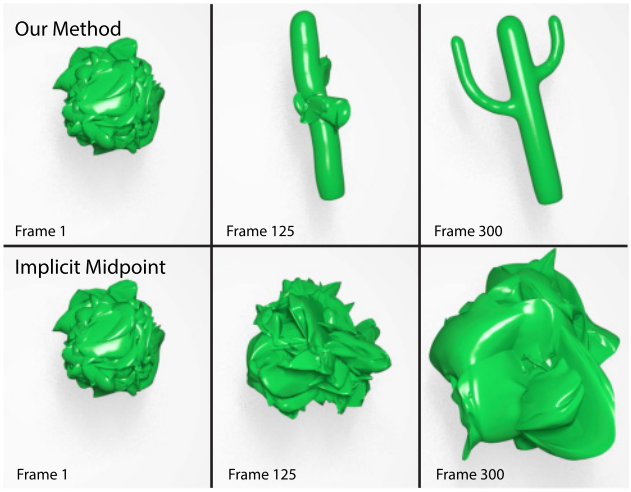

Fig. 7. An elastic cactus with randomized initial positions integrated using our method (top) and implicit midpoint (bottom), both using timestep $h = 0.033$s. We added a small amount of damping so that the cactus eventually returns to its rest pose. However, the energy overshooting of implicit midpoint overpowered the damping, and the cactus failed to come to rest even after 10,000 timesteps. This was not a problem with our method, which quickly recovered the rest pose.

the damping effect. We ran the simulation for implicit midpoint for much longer (another 10,000 frames) to see if it would eventually recover the rest pose, but the energy kept increasing despite the damping. Both simulations were run using a timestep of $h = 0.033$s.

Visual motion quality. As mentioned previously, backward Euler and its higher-order version BDF2 both cause artificial damping, which is particularly strong in higher-frequency deformation modes. This can lead to less visually attractive results. Later, Figure 10 shows how our method produces more realistic wrinkles in a cloth simulation where the cloth is being shook by one of its corners. This causes waves to propagate through the cloth, which are quickly damped out by backward Euler and BDF2 but are preserved by our method. Another example where this high-frequency damping causes undesired results is in Figure 11 (shown later). In this simulation, we aimed for a cartoon-like effect where an elastic dog's nose is stretched out and released, resulting in a humorous jiggling effect on the snout and lips. With backward Euler or BDF2, we were unable to get the desired effect, as all of the motion quickly died away and the dog's nose returned to the rest pose. With our method, we were able to get a comical, vivid motion. If required, the vividness can be reduced by introducing user-controlled damping, allowing us to achieve the desired visual effect. Please see the accompanying video.

The numerical dissipation of backward Euler manifests itself also in low-frequency deformation modes. In Figure 8, we show a damping-free elastic bar stretching under gravity. The numerical damping induced by backward Euler causes the bar to fail to return to its starting position. Using our method, the bar continues to happily bounce back to its rest state.

Similarly, Figure 9 shows a deformable bunny pinned at the ears, swinging in a pendulum-like motion. Backward Euler damps the deformation modes in the ears, causing the swinging motion to damp out. Our method keeps the bunny swinging in a lively manner due to conserving the total energy.

Our method can be less strict in the conservation of energy, depending on the choice of ϵ in Equation 24. For the results in this article, we used $\epsilon = 0.01$ (i.e., tolerate an increase or decrease of 1%). In Figure 12, we demonstrate what happens when we use an unreasonably loose $\epsilon = 0.5$. Although the animation does not explode, the heavy oscillation between the damping of backward Euler and the energy injecting of forward Euler manifests itself as a slowing of the global motion and unreasonable vibrations in the mesh. Please refer to the video for a much clearer demonstration.

Collisions. To test our method's ability to cope with collisions, we designed an experiment featuring an elastic hippo colliding with spheres (Figure 13). Because our collision detection and instancing of repulsion springs is executed only once per frame, we run this simulation at a slightly lower timestep of $h = 0.01$s to resolve all of the collisions accurately.

The artificial damping of backward Euler degrades the visual quality of collision resolution because the repulsion springs are usually stiff (to quickly remove interpenetrations). This leads to significant artificial energy dissipation during collisions, resulting in rigid-like motion that looks unrealistic for elastic objects, as demonstrated later in Figure 15. Our method executed in the same scenario produces a more elastic bouncing of the cube and less damping (see Figure 15 and the accompanying video).

We can incorporate frictional forces between colliding surfaces into our method. Figure 14 shows our method applied on a bunny sliding along a flat surface with and without friction. The frictionless surface results in the bunny sliding along the plane indefinitely. If we add some friction ($k_f = 0.8$), the bunny not only slows down but also begins rotating due to the asymmetric base. Since the frictional forces only affect the base, the bunny's ears also start moving as they are pulled back from their motion by the elastic forces of the mesh.

Performance. Table 1 shows details about our experiments, including performance measurements. All experiments used a

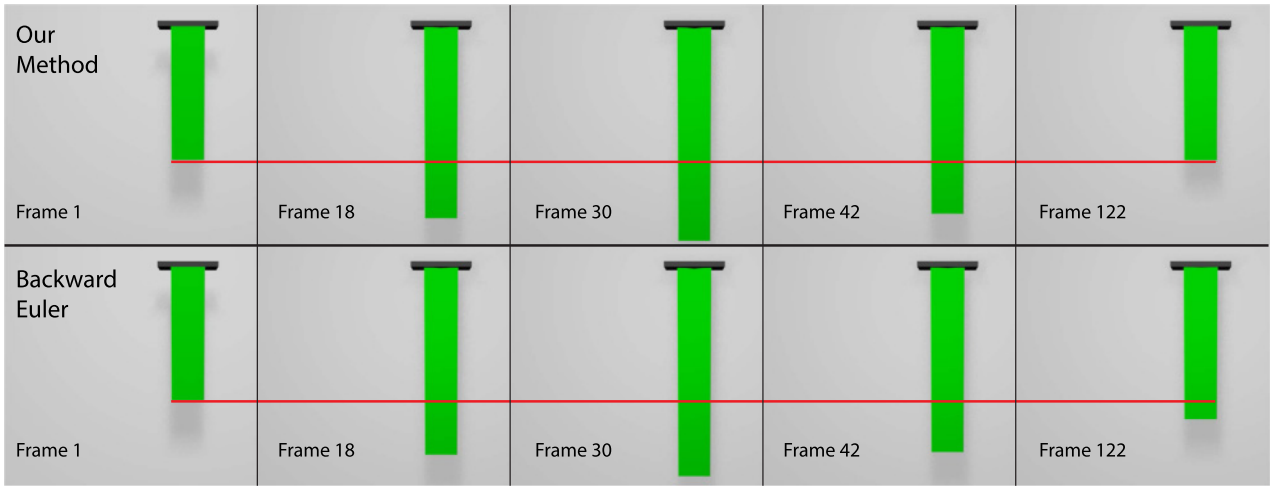

Fig. 8. An elastic tetrahedral bar stretching under gravity simulated using our method (top) and backward Euler (bottom). Frame 30 corresponds to the minimum height reached by the bar during the first bounce for both simulations, and frame 122 corresponds to the maximum height reached by the bar during the second bounce for both simulations. Backward Euler results in the bar deviating farther and farther from the starting position with each bounce, whereas our method correctly restores the bar to its starting position even after multiple bounces.

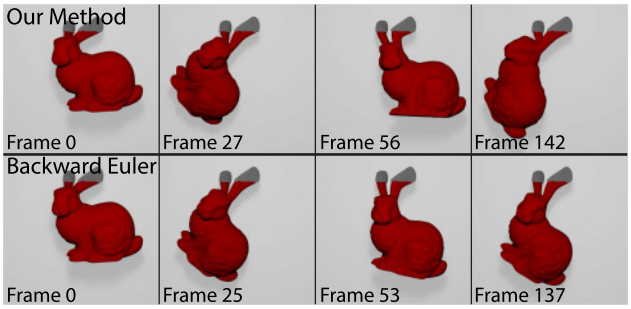

Fig. 9. An elastic bunny swings under gravity. Our method (top) results in the bunny swinging vividly, whereas backward Euler (bottom) damps out the pendulum motion. The frames chosen correspond to the apex of the pendulum motion for each example.

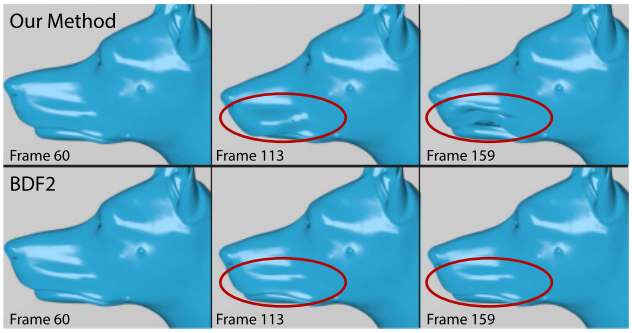

Fig. 11. An elastic dog face model starting with a prestretched nose. Our method keeps the face oscillations, whereas BDF2 damps out the motion.

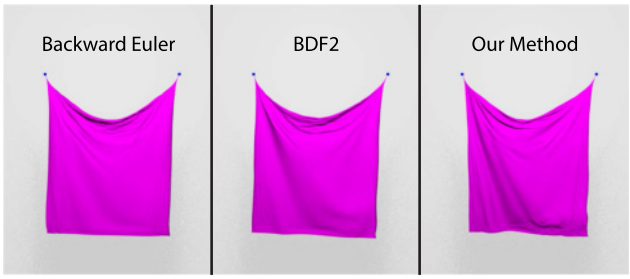

Fig. 10. Mass-spring system cloth simulated with backward Euler, BDF2, and our method. Our method produces more vivid wrinkles due to lower numerical dissipation.

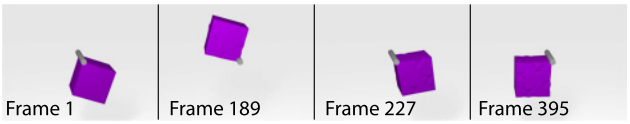

Fig. 12. Using a looser energy conservation term ($\epsilon = 0.5$) results in poor visual quality (please see the accompanying video).

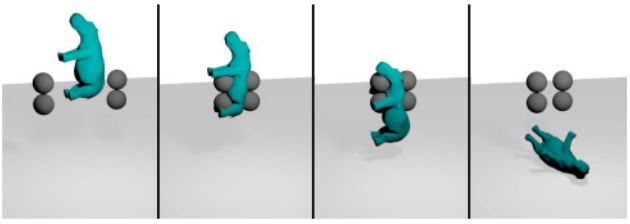

Fig. 13. Collision test with our method: an elastic hippo collides with four spheres.

lumped mass matrix for the masses. We use backward Euler as a baseline for our comparisons, and we consider two types of numerical solvers: (1) Newton's method iterated until convergence and (2) local/global with a fixed number of iterations. Let us first

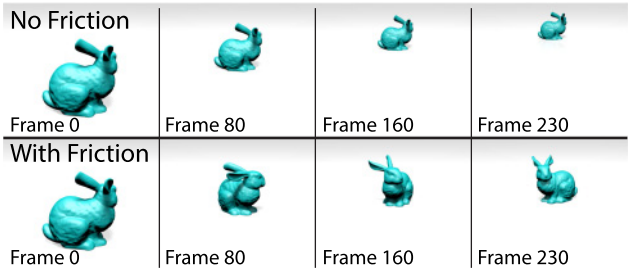

Fig. 14. A bunny sliding along a plane using our method with no friction (top) and with friction (bottom). Please refer to the video for a clearer demonstration.

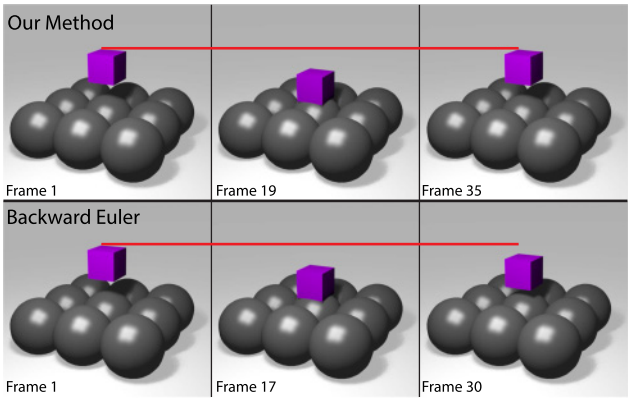

Fig. 15. An elastic cube falls on a collection of spheres, simulated using our method (top) and backward Euler (bottom). The frames chosen are the maximum and minimum heights of the cube after one bounce for each method. The cube bounces back to the original height with our method but not with backward Euler due to its numerical damping.

discuss the Newton solver. Our method executes one Newton solve for the initial implicit midpoint step, and in case this step overshoots energy, there is another Newton solve for the backward Euler stabilization step. Each of the Newton solves is iterated until convergence. This explains why our method is slower than using pure backward Euler, which needs only one Newton solve. The α-search process used to determine the amount of blending between implicit midpoint and backward Euler steps is not a bottleneck, as can be seen from the measurements in Table 1. With the local/global solver, the situation is more favorable for our method, as the stabilization step uses only one local/global iteration. The motion quality is high because we blend the single-iteration result with a more accurate solution (10 iterations) of implicit midpoint. In our experience, this results in higher motion quality with only small computing overhead.

Our method derives many of its desirable properties from implicit midpoint. What if instead of using our method we simply reduced the timestep for implicit midpoint? We study this question in Table 2, where we use the same simulation scenarios as in Table 1, but this time we compare against implicit midpoint with substepping—that is, splitting one timestep into several smaller "substeps." For example, with the original $h = 0.033$s and substepping factor 10, we use 10 steps with 0.0033s each, advancing

Fig. 16. High-frequency oscillations can occur in the horn of this dragon in an undamped simulation. This is more obvious in the accompanying video.

the simulated time by the same amount. Decreasing the timestep improves the motion quality generated by implicit midpoint; however, the extra computing resources are significant, and stability is not guaranteed even with very many substeps. This is especially obvious in collision scenarios, where even very small timesteps are not sufficient to guarantee stability. For example, in our hippo example (see Figure 13), even 100 substeps of implicit midpoint was not enough to produce a visually plausible result. Our method produces stable results despite relatively large timesteps. In addition to stability, our method also delivers much better energy and angular momentum preservation than backward Euler. Applying a local/global solver rather than using Newton's method results in a fast yet robust integration algorithm, with only minimal computing overheads compared to Projective Dynamics [7].

6 LIMITATIONS AND FUTURE WORK

One limitation of our method is that it is not perfectly momentum-conserving. This is because in cases where implicit midpoint produces energy overshoots, we correct this using backward Euler, which does not conserve angular momentum. Fortunately, we typically need only a small amount of backward Euler contribution to stabilize the simulation, resulting in much better energy and momentum conservation than backward Euler (BDF1) or BDF2. However, an interesting question for future work is whether we could better preserve angular momentum without compromising stability despite relatively large, fixed timesteps.

Implicit midpoint (and the trapezoidal rule) that is run at large timesteps can introduce local high-frequency oscillations that lead to explosions. Although blending with backward Euler makes the simulation stable and mitigates these localized oscillations, they are not always entirely eliminated and show up as artifacts, particularly in undamped simulations, as shown in Figure 16. The accompanying video shows these artifacts more clearly. The obvious solution is to introduce some damping to the simulation, but an interesting future direction is to eliminate these high-frequency oscillations of implicit midpoint without using damping.

Another direction for future work is collision handling. Currently, we use a simple method that only handles collisions with static objects. We did not implement self-collisions in our framework, but we believe that our method can be extended to support methods such as air meshes [42] or penalty methods [34] to handle self-collisions.

A limitation of our energy tracking algorithm is that it requires that damping and friction are applied in a separate step after the integration. Furthermore, the damping and friction models that we chose to use for this work are not physically accurate and have issues particularly in the case of large deformations. We chose them

Table 1. Computing Speed: Comparison With backward Euler

				Newton's Method Solver			Local/Global Solver (10 Iterations)		
				Backward	Our Method		Backward	Our Method	
Model	Ver. (#)	Ele. (#)	Timestep	Euler	Total Time	α Search	Euler	Total Time	α Search
Bar	290	716	0.033	148ms	231ms	3.27ms	3.62ms	5.73ms	1.85ms
Cube (Figure 5)	386	996	0.033	277ms	505ms	1.86ms	5.84ms	8.41ms	1.66ms
Cube (Figure 15)	386	996	0.01	365ms	597ms	0.76ms	6.93ms	9.50ms	0.93ms
Hippo	2,387	8,406	0.01	3,985ms	6,296ms	5.00ms	66.4ms	73.3ms	4.93ms
Bunny (Figure 14)	4,497	15,408	0.033	6,288ms	10,720ms	13.8ms	194ms	212ms	9.62ms
Cactus	5,261	17,187	0.033	5,050ms	7,577ms	41.6ms	115ms	174ms	39.90ms
Cloth†	10,201	50,200	0.033	24,958ms	42,758ms	6.11ms	110ms	127ms	7.58ms
Dog	28,390	117,423	0.033	39,101ms	99,138ms	212ms	916ms	1.205ms	193ms
Bunny (Figure 9)	34,844	121,058	0.033	31,624ms	64,037ms	227ms	1.181ms	1.510ms	215ms

Note: We compare the performance of our method to backward Euler to two types of solvers: (1) Newton's method (middle, iterated until convergence) and (2) local/global (right, using 10 iterations). All models use corotated elasticity with linear finite elements, except for the cloth† model, which is a mass-spring system. The α-search time is accounted for in the "total time" of our method. With the local/global solver, both backward Euler (i.e., Projective Dynamics) and implicit midpoint used in the first phase of our method always use 10 iterations. In the second (stabilizing) phase of our method, we use only one iteration of backward Euler. All of the reported times were taken as an average over 30 frames. The cube (fall) and hippo are collision tests, using a smaller timestep to accurately resolve collisions. The cactus example is a stress test with randomized initial positions (i.e., not a typical case in practical smulations), which explains the longer α-search time. Using Newton's method, the dog example takes longer than the bunny example despite being a smaller mesh because the dog simulation requires more iterations for convergence.

Table 2. Comparison of Our Method and Implicit Midpoint (local/Global Solver, 10 Iterations). Our Examples From Table 1 Compared Against Implicit Midpoint with Substepping, i.e., Reducing the Time Step Size by 5, 10, 20, and 100 Times. A Cell Color of Green Indicates that the Simulation is Stable at this Substep Level, While a Cell Color of Red Indicates that the Simulation is not Stable. Reducing the Time Step Improves Stability for Implicit Midpoint But at Significant Computing Costs

Model	Ver. (#)	Ele. (#)	Timestep	Our Method	Implicit Midpoint				
					1 Substep	5 Ssubsteps	10 Substeps	20 Substeps	100 Substeps
Bar	290	716	0.033	5.73ms	3.79ms	18.22ms	36.42ms	72.73ms	361ms
Cube (Figure 5)	386	996	0.033	8.41ms	6.03ms	29.86ms	54.95ms	111ms	541ms
Cube (Figure 15)	386	996	0.01	9.50ms	7.16ms	56.85ms	122ms	245ms	1,232ms
Hippo	2,387	8,406	0.01	73.3ms	68.4ms	312ms	666ms	1,230ms	6,615ms
Bunny (Figure 14)	4,497	15,408	0.033	212ms	184ms	830ms	2,014ms	3,568ms	18,907ms
Cactus	5,261	17,187	0.033	174ms	120ms	590ms	1,242ms	2,442ms	13,849ms
Cloth	10,201	50,200	0.033	127ms	109ms	478ms	997ms	1,976ms	8,388ms
Dog	28,390	117,423	0.033	1,205ms	904ms	4,413ms	9,079ms	16,660ms	79,369ms
Bunny (Figure 9)	34,844	121,058	0.033	1,510ms	1,101ms	5,616ms	11,293ms	22,577ms	120,631ms

due to their speed and relatively nice numeric properties, which is what we required for our real-time constraints. More physically accurate damping models that can be run in real time without adding much overhead to our method is an interesting future direction.

Finally, we believe that our analysis of backward Euler stability discussed in the appendix could be extended to more general cases, such as general mass-spring systems. Empirically, it seems the stability result still holds, as running numerous stress tests did not discover any counterexamples with mass-spring systems. Even more interesting would be to generalize these results to more complex energy potentials, such as FEM with nonlinear materials. Most likely, not all nonconvex potential functions will lead to stable backward Euler results. Discovering the conditions on deformation energy functions that are necessary/sufficient for backward Euler stability would be a very interesting theoretical investigation that could lead to new practical insights.

7 CONCLUSIONS

We presented a method that is stable and has good energy-preserving behavior, even for large fixed timesteps such as the ones required in real-time physics applications. The key idea was to take advantage of the artificial damping or artificial energy injection introduced by the backward and forward Euler methods to "stabilize" the results of implicit midpoint. We determined the right blending weight between implicit midpoint and backward/forward Euler by tracking the total energy, explicitly taking into account energy-changing events: damping and forcing. This resulted in an integrator that is stable but does not have the undesirable artificial damping of implicit methods such as backward Euler or BDF2. Finally, in the appendix, we study the energy behavior of common integrators, showing that with convex potentials, forward Euler weakly increases energy and backward Euler weakly decreases energy. We also provide a backward Euler stability proof for a simple example of nonconvex potential, hoping to inspire future work in this direction.

APPENDIXES

A FORWARD EULER INSTABILITY

In this section, we present a proof for the instability of forward Euler from a Hamiltonian point of view. The Hamiltonian is the

sum of the potential and the kinetic energies of the system:

$$H(\mathbf{x},\mathbf{v}) = \frac{1}{2}||\mathbf{v}||^2_{\mathbf{M}} + E(\mathbf{x}), \tag{40}$$

where $||\mathbf{v}||^2_{\mathbf{M}} := \mathbf{v}^\mathsf{T}\mathbf{M}\mathbf{v}$ is a mass-matrix norm and $E(\mathbf{x})$ the potential energy function. To simplify our discussion, we will assume that $E(\mathbf{x})$ is defined and finite for any state $\mathbf{x}$. Because we assume that the potential energy is time invariant, in the continuous setting the Hamiltonian is exactly conserved—for instance, at any time $t > 0$, the Hamiltonian $H(\mathbf{x}(t),\mathbf{v}(t)) = H(\mathbf{x}_0,\mathbf{v}_0)$, where $\mathbf{x}_0, \mathbf{v}_0$ are the initial conditions. This equality is no longer true in the case of numerical time integration due to discretization error. We can quantify the Hamiltonian error of a numerical scheme simply by subtracting the Hamiltonians at two consecutive steps:

$$H(\mathbf{x}_{n+1},\mathbf{v}_{n+1}) - H(\mathbf{x}_n,\mathbf{v}_n). \tag{41}$$

This error depends on a specific numerical integration scheme. In this section, we focus on the forward Euler method. Using the forward Euler update rules (Equations (1) and (2)), we can write

$$\begin{aligned} H(\mathbf{x}_{n+1},\mathbf{v}_{n+1}) &= \frac{1}{2}||\mathbf{v}_{n+1}||^2_{\mathbf{M}} + E(\mathbf{x}_{n+1}) \\ &= \frac{1}{2}||\mathbf{v}_n - h\mathbf{M}^{-1}\nabla E(\mathbf{x}_n)||^2_{\mathbf{M}} + E(\mathbf{x}_{n+1}) \\ &= \frac{1}{2}||\mathbf{v}_n||^2_{\mathbf{M}} - (\mathbf{x}_{n+1}-\mathbf{x}_n)^\mathsf{T}\nabla E(\mathbf{x}_n) \\ &\quad + \frac{1}{2}h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_n)||^2_{\mathbf{M}} + E(\mathbf{x}_{n+1}). \end{aligned} \tag{42}$$

Plugging this expression into Equation (41) results in

$$\begin{aligned} H(\mathbf{x}_{n+1},&\mathbf{v}_{n+1}) - H(\mathbf{x}_n,\mathbf{v}_n) \\ &= E(\mathbf{x}_{n+1}) - E(\mathbf{x}_n) - (\mathbf{x}_{n+1}-\mathbf{x}_n)^\mathsf{T}\nabla E(\mathbf{x}_n) \\ &\quad + \frac{1}{2}h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_n)||^2_{\mathbf{M}}. \end{aligned} \tag{43}$$

This formula leads to an interesting fact if we assume that the potential function E is convex (most practical potential functions are not convex; however, it is insightful to first study the case of convex E). With convex E, first-order convexity conditions (Section 3.1.3 in Boyd and Vandenberghe [8]) imply that $E(\mathbf{x}_{n+1}) - E(\mathbf{x}_n) - (\mathbf{x}_{n+1}-\mathbf{x}_n)^\mathsf{T}\nabla E(\mathbf{x}_n) \geq 0$ for any $\mathbf{x}_n$ and $\mathbf{x}_{n+1}$. Because the term $h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_n)||^2_{\mathbf{M}}$ is always nonnegative (as the mass-matrix $\mathbf{M}$ is positive definite), we have just proven that with forward Euler, the Hamiltonian is weakly increasing.

This shows why the forward Euler method is highly unstable for convex potential functions. The Hamiltonian cannot decrease, and therefore, in the best case scenario, it can remain constant. This is the expected behavior, as $h \to 0$ when the discrete approximation converges to the continuous solution (where the Hamiltonian is indeed constant). However, with larger timesteps h, we often observe significant increases of the Hamiltonian and the Hamiltonian quickly approaches infinity, indicating instability. This behavior is often empirically observed also with common nonconvex potential energy functions. However, with nonconvex potentials, it is possible to find examples where forward Euler actually decreases the Hamiltonian (i.e., $H(\mathbf{x}_{n+1},\mathbf{v}_{n+1}) < H(\mathbf{x}_n,\mathbf{v}_n)$). The analysis of nonconvex potentials is more complicated. as we will discuss in Appendix C.

B BACKWARD EULER STABILITY

Let us start with a quote: "Forward Euler takes no notice of wildly changing derivatives, and proceeds forward quite blindly. Backward Euler, however, forces one to find an output state whose derivative at least points back to where you came from, imparting, essentially, an additional layer of consistency (or sanity-checking, if you will)" (footnote 4, [4]). This remark provides beautiful intuition why backward Euler is more stable than forward Euler. In this section, we formalize this intuition by providing a formal stability proof of backward Euler for convex potential functions. With forward Euler, instability is characterized by increasing the Hamiltonian above all bounds. There are various interpretations of the term *stability* in the literature. In this article, we say that simulation is stable if there is a constant $C > 0$ such that $H(\mathbf{x}_n,\mathbf{v}_n) \leq C$ for all $n = 0, 1, 2, \ldots$. The constant C can depend on the initial conditions and the parameters of the system but cannot depend on n. In other words, regardless of how many timesteps we compute, the Hamiltonian can never increase above C—it must remain bounded. We also assume that our potential $E(\mathbf{x})$ is bounded from below. This is trivially satisfied for all elastic energies. If we use the standard linear gravity potential, we assume that there is a ground plane below which the object cannot fall.

Unlike forward Euler, backward Euler does not provide explicit formulas to compute the next step $(\mathbf{x}_{n+1},\mathbf{v}_{n+1})$; instead, the next step is given implicitly by a system of (typically nonlinear) equations. Nevertheless, we can still use a similar analysis as in Appendix A—the main trick being in performing the analysis backward in time. We start by applying the backward Euler rules (Equations (3) and (4)) to expand:

$$\begin{aligned} H(\mathbf{x}_n,\mathbf{v}_n) &= \frac{1}{2}||\mathbf{v}_{n+1} + h\mathbf{M}^{-1}\nabla E(\mathbf{x}_{n+1})||^2_{\mathbf{M}} + E(\mathbf{x}_n) \\ &= \frac{1}{2}||\mathbf{v}_{n+1}||^2_{\mathbf{M}} + (\mathbf{x}_{n+1}-\mathbf{x}_n)^\mathsf{T}\nabla E(\mathbf{x}_{n+1}) \\ &\quad + \frac{1}{2}h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_{n+1})||^2_{\mathbf{M}} + E(\mathbf{x}_n). \end{aligned} \tag{44}$$

Subtracting the Hamiltonian of the next step yields

$$\begin{aligned} H(\mathbf{x}_n,&\mathbf{v}_n) - H(\mathbf{x}_{n+1},\mathbf{v}_{n+1}) \\ &= E(\mathbf{x}_n) - E(\mathbf{x}_{n+1}) + (\mathbf{x}_{n+1}-\mathbf{x}_n)^\mathsf{T}\nabla E(\mathbf{x}_{n+1}) \\ &\quad + \frac{1}{2}h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_{n+1})||^2_{\mathbf{M}}. \end{aligned} \tag{45}$$

If we assume that the potential E is convex, the first-order convexity conditions (Section 3.1.3 in Boyd and Vandenberghe [8]) imply that

$$E(\mathbf{x}_n) - E(\mathbf{x}_{n+1}) + (\mathbf{x}_{n+1}-\mathbf{x}_n)^\mathsf{T}\nabla E(\mathbf{x}_{n+1}) \geq 0 \tag{46}$$

for any $\mathbf{x}_n$ and $\mathbf{x}_{n+1}$. The term $h^2||\mathbf{M}^{-1}\nabla E(\mathbf{x}_{n+1})||^2_{\mathbf{M}}$ is nonnegative because $h > 0$ and $\mathbf{M}$ is a positive-definite mass matrix. Therefore, with convex E, we can conclude that $H(\mathbf{x}_n,\mathbf{v}_n) \geq H(\mathbf{x}_{n+1},\mathbf{v}_{n+1})$. This means that we can define the upper bound C simply as $C := H(\mathbf{x}_0,\mathbf{v}_0)$—in other words, the Hamiltonian never rises above its value specified by the initial positions and velocities. With $h \to 0$, we converge to the continuous case where the Hamiltonian is conserved. In real-time simulations, it is common to use relatively large h, in which case there can be relatively large decreases of the Hamiltonian, corresponding to the numerical dissipation of backward Euler.

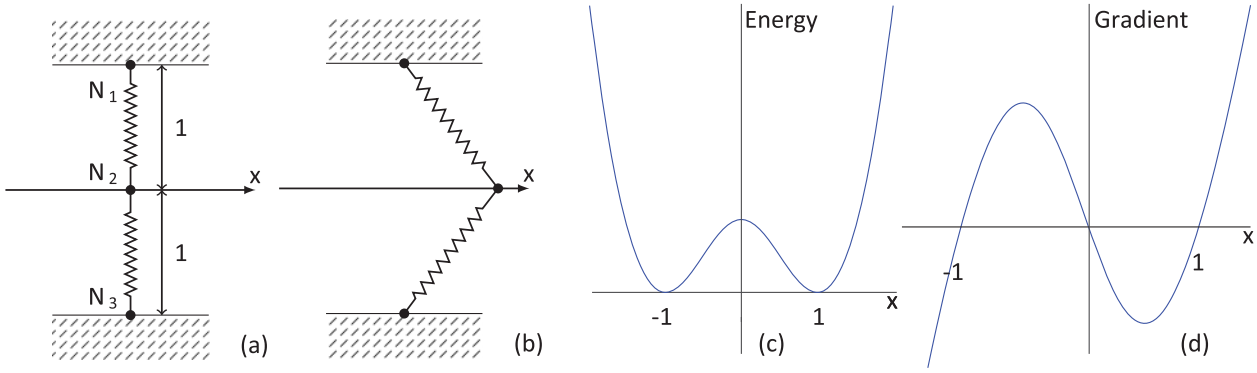

Fig. 17. A simple two spring system where N_1, N_2, and N_3 are connected by springs. N_1 and N_3 are fixed at a distance of 1 from the x-axis (a), and N_2 is free to move along the x-axis (b). The resulting potential function is nonconvex (c); its derivative is shown in (d).

With nonconvex potentials, this analysis does not hold, and it is possible to find counterexamples where $H(\mathbf{x}_n, \mathbf{v}_n) > H(\mathbf{x}_0, \mathbf{v}_0)$. We will discuss one such example in Appendix C.

C NONCONVEX POTENTIAL FUNCTIONS

Many potential functions used in physics-based animation are not convex—even simple mass-spring systems. To our knowledge, analysis of backward Euler's behavior with nonconvex potentials is an open problem. In this appendix, we merely scratch the surface by analyzing a simple toy example of a nonconvex potential. In this didactic case, we show that key result still holds—that is, with backward Euler, there exists an upper bound on the Hamiltonian $H(\mathbf{x}_n, \mathbf{v}_n)$. However, the analysis is more complicated than in the convex case studied in Appendix B where the Hamiltonian is weakly decreasing. In the nonconvex case, the Hamiltonian can temporarily increase, and we need to show that these increases are bounded.

We will analyze a very simple mass-spring system, consisting of three vertices connected by two springs, each of which has rest length $\sqrt{2}$ (Figure 17). In this system, the two outside nodes $(N1, N3)$ are fixed, and the middle node $(N2)$ is free to move along the x-axis. The state $x = 0$ corresponds to the situation in Figure 17(a), where both springs are compressed to length 1. Figure 17(b) depicts the case of $x = 1$, where both springs are at their rest length (i.e., the potential energy is zero). Let us derive a formula of the potential as a function of the free variable $x \in \mathbb{R}$. The length of each spring is, according to the Pythagorean theorem, $\sqrt{x^2+1}$. Plugging this into Hooke's law and summing the two springs, we obtain

$$E_{\text{test}}(x) = k\left(\sqrt{x^2+1} - \sqrt{2}\right)^2, \tag{47}$$

where $k > 0$ is the spring stiffness, which we assume is the same for both springs. The potential of our test system is graphed in Figure 17(c). We can immediately notice that the potential is nonconvex, with two local minima at -1 and 1 and a local maximum at 0. Let us also compute the derivative of this potential function:

$$E'_{\text{test}}(x) = 2k\left(\sqrt{x^2+1} - \sqrt{2}\right)\frac{x}{\sqrt{x^2+1}}. \tag{48}$$

We graph it in Figure 17(d).

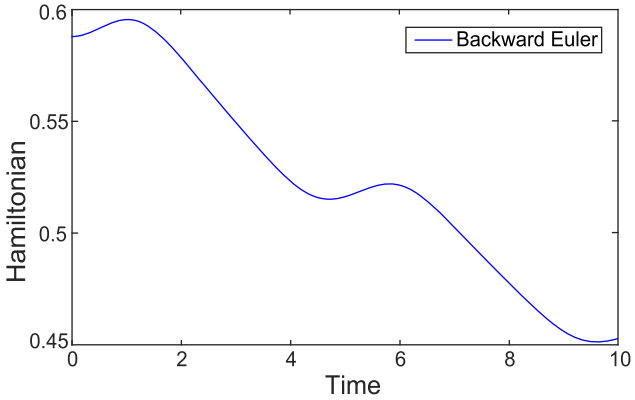

Fig. 18. The Hamiltonian for a simulation of our simple test system from Figure 17 computed using backward Euler. The Hamiltonian can occasionally increase despite the overall dissipative trend.

We can use the potential function E_{test} to construct an example where a backward Euler step actually increases energy, providing a counterexample that the approach from Appendix B does not trivially generalize to nonconvex potentials. To construct such an example, we can pick any starting point $x_0 \in (0, 1)$ and set the initial velocity $v_0 = -x_0/h$ (where h is the timestep). In this case, the backward Euler rules (Equations (3) and (4)) are satisfied with $x_1 = 0, v_1 = v_0$, because $E'_{\text{test}}(0) = 0$. In this case, the kinetic energy remains the same because $v_1 = v_0$, but the potential energy increases because $E_{\text{total}}(0) > E_{\text{total}}(x_0)$ (in fact, we could even pick x_0 slightly larger than 1 and this inequality would still hold). In other words, in this case, the nonconvexity of E_{test} caused backward Euler to increase the Hamiltonian.

Numerical simulation of our simple test system reveals that backward Euler can consistently increase the Hamiltonian over several timesteps, as shown in Figure 18. The increases are visible as little bumps in the otherwise generally decreasing Hamiltonian. This means that we cannot hope to prove that the Hamiltonian will be weakly decreasing with nonconvex potentials. However, we can still hope to prove stability if we can show that the occasional Hamiltonian increases due to potential nonconvexity are

in fact bounded. The graph in Figure 18 suggests that this indeed may be the case. In the following, we prove this for the case of our simple potential E_{test}. The key idea of the proof is the fact that the nonconvexity of E_{test} is localized to the interval $[-1, 1]$. In the following three lemmas, we consider three different cases depending on which intervals x_n and x_{n+1} are in.

Lemma 1. *Let $x_n, x_{n+1} \in \mathbb{R}$ be two consecutive timesteps of backward Euler applied to E_{test}. If $|x_{n+1}| > 1$, then $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$.*

Proof. First, we define a function $G(x)$ such that

$$G(x) = \begin{cases} 0 & x \in [-1, 1] \\ E_{\text{test}}(x) & \text{otherwise.} \end{cases} \tag{49}$$

Because $E_{\text{test}}(x)$ is a nonnegative function, $G(x) \leq E_{\text{test}}(x)\ \forall x \in \mathbb{R}$. Because $G(x)$ is convex and differentiable (note that $E'_{\text{test}}(\pm 1) = 0$), we can apply the first-order convexity condition to get the inequality:

$$G(x_n) \geq G(x_{n+1}) + (x_n - x_{n+1})G'(x_{n+1}). \tag{50}$$

Because we assumed that $|x_{n+1}| > 1$, we can write $G(x_{n+1}) = E_{\text{test}}(x_{n+1})$ and $G'(x_{n+1}) = E'_{\text{test}}(x_{n+1})$. Putting these facts together yields the following inequality for E_{test}:

$$E_{\text{test}}(x_n) \geq G(x_n) \geq E_{\text{test}}(x_{n+1}) + (x_n - x_{n+1})E'_{\text{test}}(x_{n+1}). \tag{51}$$

Plugging this inequality into Equation (45) shows that $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$ just like in the convex case (Appendix B). □

Lemma 2. *There is a constant $\beta > 1$ such that for any two consecutive time steps $x_n, x_{n+1} \in \mathbb{R}$ of backward Euler applied to E_{test}, it is true that if $|x_n| \geq \beta$, $|x_{n+1}| \leq 1$ then $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$.*

Proof. Lemma 1 showed that if $|x_{n+1}| > 1$, then $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$ regardless of the value of x_n. However, if $x_{n+1} \in [-1, 1]$, there are situations when $H(x_n, v_n) \leq H(x_{n+1}, v_{n+1})$. Now we will show that this cannot happen if $|x_n| \geq \beta$. First, we will assume that $x_n \geq \beta$ (the case of $\mathbf{x}_n \leq -\beta$ is analogous) and show that

$$E_{\text{test}}(x_n) \geq E_{\text{test}}(x_{n+1}) + (x_n - x_{n+1})E'_{test}(x_{n+1}). \tag{52}$$

This inequality leads to $H(x_n, v_n) \geq H(x_{n+1}, v_{n+1})$ as in the convex case (Equation (45)). Our first task is to find a suitable β. Let us consider the line

$$l(x) = E_{\text{test}}(0) + (x + 1)E'_{\text{test}}(x_{\max}). \tag{53}$$

The value $x_{\max}$ is defined as $x_{\max} := \text{argmax}_{x \in [-1,1]} E'(x)$ (i.e., the maximal derivative on the interval $[-1, 1]$). $E_{\text{test}}(0)$ is the maximum value of the potential on the interval $[-1, 1]$. We will then define β as the intersection point between $l(x)$ and $E_{\text{test}}(x)$ with $x > 0$ (Figure 19). In other words, we pick β as the positive solution of

$$E_{\text{test}}(\beta) = E_{\text{test}}(0) + (\beta + 1)E'_{\text{test}}(x_{\max}). \tag{54}$$

It is hard to evaluate β symbolically; however, we can easily approximate it numerically: $\beta \approx 2.209$. An important fact is that the β depends only on the potential E_{test} (i.e., it does not depend on the states x_n, x_{n+1}).

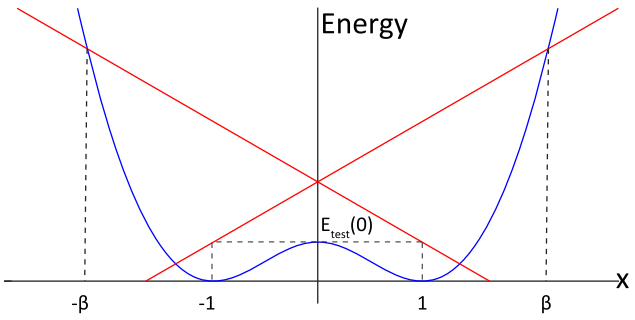

Fig. 19. Illustration for Lemma 2: the potential function $E_{\text{test}}(x)$ (blue) and our lines $l(x)$ (red) determining the constant β.

Using the definition of the line in Equation (53), we can show that the following inequality is also satisfied $\forall x_{n+1} \in [-1, 1]$:

$$E_{\text{test}}(\beta) \geq E_{\text{test}}(x_{n+1}) + (\beta - x_{n+1})E'_{\text{test}}(x_{n+1}). \tag{55}$$

To show this, we refer to Equation (54) and observe that $E_{\text{test}}(0) \geq E_{\text{test}}(x_{n+1})$ because $E_{\text{test}}(0) \geq E_{\text{test}}(x)\ \forall x \in [-1, 1]$. The fact that $E'_{\text{test}}(x_{\max}) \geq E'_{\text{test}}(x_{n+1})$ follows from the definition of $x_{\max}$. Finally, $\beta + 1 \geq \beta - x_{n+1}$ follows from $|x_{n+1}| \leq 1$.

E_{test} is a convex function on interval $[1, \infty]$, which implies that

$$E'_{\text{test}}(\beta) \geq E'_{\text{test}}(x_{\max}) \tag{56}$$

because β is the intersection between $l(x)$ and $E_{\text{test}}(x)$. The convexity on $[1, \infty]$ further implies that

$$E_{\text{test}}(x_n) \geq E_{\text{test}}(\beta) + (x_n - \beta)E'_{\text{test}}(\beta) \tag{57}$$

due to first-order convexity conditions [8]. We can now substitute Equation (55) into Equation (57):

$$\begin{aligned} E_{\text{test}}(x_n) \geq\ & E_{\text{test}}(x_{n+1}) + (\beta - x_{n+1})E'_{\text{test}}(x_{n+1}) \\ & + (x_n - \beta)E'_{\text{test}}(\beta). \end{aligned} \tag{58}$$

Plugging in Equation (56) and using the fact that $E'_{\text{test}}(x_{\max}) \geq E'_{\text{test}}(x_{n+1})$ and our assumption $x_n \geq \beta$ results, after some simplifications, in

$$E_{\text{test}}(x_n) \geq E_{\text{test}}(x_{n+1}) + (x_n - x_{n+1})E'_{\text{test}}(x_{n+1}). \tag{59}$$

This is the same inequality as Equation (46), and therefore the result $H(\mathbf{x}_n, \mathbf{v}_n) \geq H(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$ follows just like in the convex case (Appendix B). The proof for $x_n < -\beta$ is completely analogous due to the fact that $E_{\text{test}}(x) = E_{\text{test}}(-x)$. □

Lemma 3. *If $\beta > 1$ is as in Lemma 2, there exists a constant $H_c > 0$ such that for any two consecutive timesteps $x_n, x_{n+1} \in \mathbb{R}$ of backward Euler applied to E_{test}, such that $|x_{n+1}| \leq 1, |x_n| \leq \beta$, it is true that $H(x_{n+1}, v_{n+1}) \leq H_c$, where $H_c = E_{\text{test}}(0) + \frac{m}{2}(\frac{\beta+1}{h})^2$, where m is the mass of vertex N_2 and h is the timestep.*

Proof. Because $x_{n+1} \in [-1, 1]$, we immediately have a bound on the potential energy:

$$E_{\text{test}}(x_{n+1}) \leq E_{\text{test}}(0). \tag{60}$$

To obtain a bound on the kinetic energy, we use the update rule of backward Euler (Equation (3)) and the facts that $x_{n+1} \in [-1, 1]$

and $x_n \in [-\beta, \beta]$, which lead to

$$|v_{n+1}| = \left|\frac{x_{n+1} - x_n}{h}\right| \leq \frac{\beta + 1}{h}. \tag{61}$$

Combining Equation (60) and Equation (61) will get us a Hamiltonian bound:

$$H(x_{n+1}, v_{n+1}) = E_{\text{test}}(x_{n+1}) + \frac{m}{2}(v_{n+1})^2 \tag{62}$$

$$\leq E_{\text{test}}(0) + \frac{m}{2}\left(\frac{\beta + 1}{h}\right)^2 = H_c. \qquad \square \tag{63}$$

Finally, we put the results of the previous three lemmas together in the following theorem.

Theorem 1. *If $x_0, v_0 \in \mathbb{R}$ are arbitrary initial conditions of our test problem, then the Hamiltonian at any step n of exactly solved backward Euler integration satisfies*

$$H(x_n, v_n) \leq \max(H(x_0, v_0), H_c),$$

where $H_c = E_{\text{test}}(0) + \frac{m}{2}(\frac{\beta+1}{h})^2$ as in Lemma 3.

Proof. The proof is by induction. The theorem is trivially satisfied for $n = 0$. Moving from n to $n + 1$, there are several possibilities. If $|x_{n+1}| > 1$, Lemma 1 applies and shows that the Hamiltonian must weakly decrease. If $|x_{n+1}| \leq 1$ and $|x_n| \geq \beta$, Lemma 2 applies and also asserts that the Hamiltonian must weakly decrease. Finally, if $|x_{n+1}| \leq 1$ and $|x_n| < \beta$, the Hamiltonian may increase, but Lemma 3 shows that it cannot increase arbitrarily much, because $H(x_{n+1}, v_{n+1}) \leq H_c$. $\square$

We have shown that for our simple but nonconvex potential E_{test}, exactly solved backward Euler is stable. A logical question is whether a similar result would hold also for arbitrary mass-spring systems. Unfortunately, the situation becomes more complicated because our bounds would have to be extended to the multivariate case. For example, the two lines in Figure 19 could be replaced by a translated convex cone. However, there is a complication, because in general mass-spring systems, some springs can be contracted, whereas others can be extended. Even a single contracted spring can, in theory, introduce nonconvexities. The backward Euler stability proof for general mass-spring systems will therefore be more complicated, and we defer it to future work.

ACKNOWLEDGEMENTS

We thank Robert Bridson, Mathieu Desbrun, Eitan Grinspun, Dominik Michels, Daniele Panozzo, and Eftychios Sifakis for many inspiring discussions. We also thank Cem Yuksel, Petr Kadlecek, and Nghia Troung for proofreading. We also gratefully acknowledge the support of Activision and hardware donation from NVIDIA Corporation.

REFERENCES

[1] Samantha Ainsley, Etienne Vouga, Eitan Grinspun, and Rasmus Tamstorf. 2012. Speculative parallel asynchronous contact mechanics. *ACM Transactions on Graphics* 31, 6, 151.

[2] Uri M. Ascher and Linda R. Petzold. 1998. *Computer Methods for Ordinary Differential Equations and Differential-algebraic Equations.* Vol. 61. SIAM.

[3] Uri M. Ascher and Sebastian Reich. 1999. The midpoint scheme and variants for Hamiltonian systems: Advantages and pitfalls. *SIAM Journal on Scientific Computing* 21, 3, 1045–1065.

[4] David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'98).* ACM, New York, NY, 43–54. DOI: http://dx.doi.org/10.1145/280814.280821

[5] Jan Bender, Dan Koschier, Patrick Charrier, and Daniel Weber. 2014. Position-based simulation of continuous materials. *Computers and Graphics* 44, 1–10.

[6] Jan Bender, Matthias Müller, Miguel A. Otaduy, Matthias Teschner, and Miles Macklin. 2014. A survey on position-based simulation methods in computer graphics. *Computer Graphics Forum*, 33, 6, 228–251.

[7] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing constraint projections for fast simulation. *ACM Transactions on Graphics* 33, 4, 154.

[8] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization.* Cambridge University Press, New York, NY.

[9] Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics* 21, 3, 594–603.

[10] Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. 2010. A simple geometric model for elastic deformations. *ACM Transactions on Graphics* 29, 4, 38.

[11] Kwang-Jin Choi and Hyeong-Seok Ko. 2002. Stable but responsive cloth. *ACM Transactions on Graphics* 21, 3, 604–611.

[12] J. Chung and G. M. Hulbert. 1993. A time integration algorithm for structural dynamics with improved numerical dissipation: The generalized-α method. *Journal of Applied Mechanics* 60, 2, 371–375.

[13] K. Dekker and J. G. Verwer. 1987. Stability of Runge-Kutta methods for stiff nonlinear differential equations. *ZAMM—Journal of Applied Mathematics and Mechanics/Zeitschrift fr Angewandte Mathematik und Mechanik* 67, 1, 68. DOI: http://dx.doi.org/10.1002/zamm.19870670128

[14] Robert W. Easton. 1998. *Geometric Methods for Discrete Dynamical Systems.* Oxford University Press.

[15] Robert D. Engle, Robert D. Skeel, and Matthew Drees. 2005. Monitoring energy drift with shadow Hamiltonians. *JJournal of Computational Physics* 206, 2, 432–452.

[16] Theodore F. Gast and Craig Schroeder. 2014. Optimization integrator for large time steps. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Computer Animation.*

[17] Theodore F. Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M. Teran. 2015. Optimization integrator for large time steps. *IEEE Transactions on Visualization and Computer Graphics* 21, 10, 1103–1115.

[18] O. Gonzalez and J. C. Simo. 1996. On the stability of symplectic and energy-momentum algorithms for non-linear Hamiltonian systems with symmetry. *Computer Methods in Applied Mechanics and Engineering* 134, 3, 197–222.

[19] Ernst Hairer. 2006. *Long-Time Energy Conservation of Numerical Integrators.* Cambridge University Press, CambridgeNY.

[20] Ernst Hairer, Christian Lubich, and Gerhard Wanner. 2006. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations.* Vol. 31. Springer Science and Business Media.

[21] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. 2009. Asynchronous contact mechanics. *ACM Transactions on Graphics* 28, 3, Article No. 87.

[22] T. J. R. Hughes, T. K. Caughey, and W. K. Liu. 1978. Finite-element methods for nonlinear elastodynamics which conserve energy. *Journal of Applied Mechanics* 45, 2, 366–370.

[23] Arieh Iserles. 2009. *A First Course in the Numerical Analysis of Differential Equations.* Cambridge University Press, Cambridge, NY.

[24] Couro Kane. 1999. *Variational Integrators and the Newmark Algorithm for Conservative and Dissipative Mechanical Systems.* Ph.D. Dissertation. California Institute of Technology, Pasadena, CA.

[25] C. Kane, J. E. Marsden, and M. Ortiz. 1999. Symplectic-energy-momentum preserving variational integrators. *Journal of Mathematical Physics* 40, 7, 3353–3371.

[26] Liliya Kharevych, Weiwei Yang, Yiying Tong, Eva Kanso, Jerrold E. Marsden, Peter Schröder, and Matthieu Desbrun. 2006. Geometric, variational integrators for computer animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 43–51.

[27] Tae-Yong Kim, Nuttapong Chentanez, and Matthias Müller-Fischer. 2012. Long range attachments—a method to simulate inextensible clothing in computer games. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 305–310.

[28] D. Kuhl and M. A. Crisfield. 1999. Energy-conserving and decaying algorithms in non-linear structural dynamics. *International Journal for Numerical Methods in Engineering* 45, 5, 569–599.

[29] Robert A. LaBudde and Donald Greenspan. 1975. Energy and momentum conserving methods of arbitrary order for the numerical integration of equations of motion. *Numerische Mathematik* 25, 4, 323–346.

[30] J. D. Lambert. 1991. *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem.* John Wiley & Sons, New York, NY.

[31] Adrian Lew, Jerrold E. Marsden, Michael Ortiz, and Matthew West. 2003. Asynchronous variational integrators. *Archive for Rational Mechanics and Analysis* 167, 2, 85–146.
[32] Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. 2013. Fast simulation of mass-spring systems. *ACM Transactions on Graphics* 32, 6, 209:1–209:7. http://cg.cis.upenn.edu/publications/Liu-FMS
[33] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2016. Towards real-time simulation of hyperelastic materials. arXiv:1604.07378.
[34] Tiantian Liu, Ming Gao, Lifeng Zhu, Eftychios Sifakis, and Ladislav Kavan. 2016. Fast and robust inversion-free shape manipulation. *Computer Graphics Forum* 35, 2, 1–11.
[35] Miles Macklin and Matthias Müller. 2013. Position based fluids. *ACM Transactions on Graphics* 32, 4, 104.
[36] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2014. Unified particle physics for real-time applications. *ACM Transactions on Graphics* 33, 4, 153.
[37] 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. *ACM Transactions on Graphics* 30, 4, Article No. 37.
[38] Dominik L. Michels and Mathieu Desbrun. 2015. A semi-analytical approach to molecular dynamics. *Journal of Computational Physics* 303, 336–354.
[39] Dominik L. Michels, Gerrit A. Sobottka, and Andreas G. Weber. 2014. Exponential integrators for stiff elastodynamic problems. *ACM Transactions on Graphics* 33, 1, 7.
[40] Matthias Müller. 2008. Hierarchical position based dynamics. In *Proceedings of the Workshop in Virtual Reality Interactions and Physical Simulation (VRIPHYS'08)*. DOI : http://dx.doi.org/10.2312/PE/vriphys/vriphys08/001-010
[41] Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. 2014. Strain based dynamics. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation (SCA'14)*, Vol. 2.
[42] Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. 2015. Air meshes for robust collision handling. *ACM Transactions on Graphics* 34, 4, 133.
[43] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2, 109–118.
[44] Rahul Narain, Matthew Overby, and George E. Brown. 2016. ADMM ⊇ Projective Dynamics: Fast simulation of general constitutive models. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'16)*. 21–28.
[45] Rahul Narain, Armin Samii, and James F. O'Brien. 2012. Adaptive anisotropic remeshing for cloth simulation. *ACM Transactions on Graphics* 31, 6, 152.
[46] Nathan Mortimore Newmark. 1959. A method of computation for structural dynamics. *Journal of the Engineering Mechanics Division* 85, 3, 69–74.
[47] Eftychios Sifakis and Jernej Barbič. 2012. FEM simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*. ACM, New York, NY, 20.
[48] J. C. Simo, N. Tarnow, and K. K. Wong. 1992. Exact energy-momentum conserving algorithms and symplectic schemes for nonlinear dynamics. *Computer Methods in Applied Mechanics and Engineering* 100, 1, 63–116.
[49] F. S. Sin, D. Schroeder, and J. Barbič. 2013. Vega: Non-linear FEM deformable object simulator. *Computer Graphics Forum* 32, 1, 36–48.
[50] Gerrit Sobottka, Tomás Lay, and Andreas Weber. 2008. Stable integration of the dynamic Cosserat equations with application to hair modeling. *Journal of WSCG* 16, 73–80.
[51] Jos Stam. 2009. Nucleus: Towards a unified dynamics solver for computer graphics. In *Proceedings of the IEEE International Conference on Computer-Aided Design and Computer Graphics*.1–11.
[52] Ari Stern and Mathieu Desbrun. 2006. Discrete geometric mechanics for variational time integrators. In *ACM SIGGRAPH 2006 Courses*. ACM, New York, NY, 75–80.
[53] Jonathan Su, Rahul Sheth, and Ronald Fedkiw. 2013. Energy conservation for the simulation of deformable bodies. *IEEE Transactions on Visualization and Computer Grahics* 19, 2, 189–200.
[54] Demetri Terzopoulos and Kurt Fleischer. 1988. Deformable models. *Visual Computer* 4, 6, 306–331.
[55] Demetri Terzopoulos and Kurt Fleischer. 1988. Modeling inelastic deformation: Viscolelasticity, plasticity, fracture. In *ACM Siggraph Computer Graphics*, Vol. 22. ACM, 269–278.
[56] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically deformable models. *ACM SIGGRAPH Computer Graphics* 22, 4, 269–278.
[57] Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straßer. 2008. Asynchronous cloth simulation. In *Computer Graphics International*, Vol. 2. Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems.
[58] Huamin Wang. 2015. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics* 34, 6, 246.
[59] Huamin Wang, James O'Brien, and Ravi Ramamoorthi. 2010. Multi-resolution isotropic strain limiting. *ACM Transactions on Graphics* 29, 6, Article , 10 pages.
[60] Matthew West. 2004. *Variational Integrators*. Ph.D. Dissertation. California Institute of Technology.
[61] M. West, C. Kane, J. E. Marsden, and M. Ortiz. 1999. Variational integrators, the Newmark scheme, and dissipative systems. In *Proceedings of the International Conference on Differential Equations*, Vol. 1. World Scientific, 7.
[62] Danyong Zhao, Yijing Li Li, and Jernej Barbič. 2016. Asynchronous implicit backward Euler integration. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'16)*. 1–9.
[63] Ge Zhong and Jerrold E. Marsden. 1988. Lie-Poisson Hamilton-Jacobi theory and Lie-Poisson integrators. *Physics Letters A* 133, 3, 134–139.

Received September 2016; revised September 2017; accepted October 2017