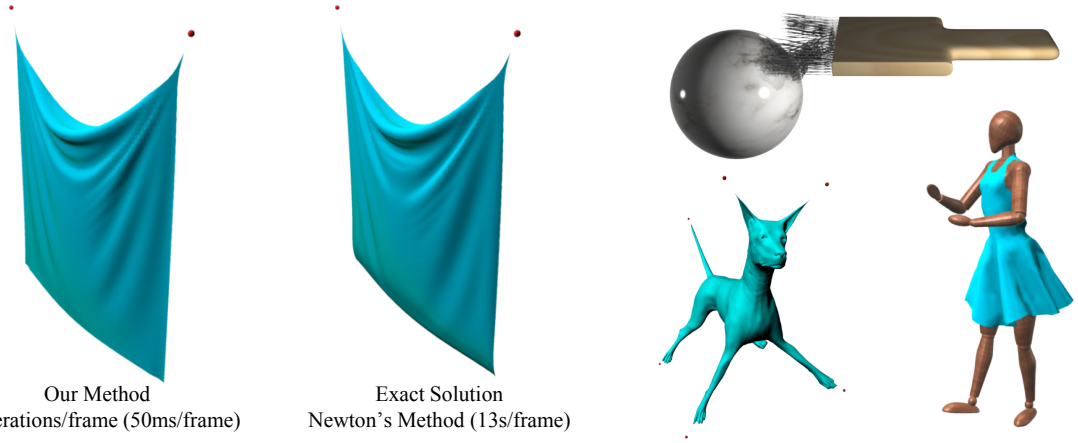
Fast Simulation of Mass-Spring Systems

Tiantian Liu
University of Pennsylvania

Adam W. Bargteil
University of Utah

James F. O'Brien
University of California, Berkeley

Ladislav Kavan*
University of Pennsylvania



Our Method
10 iterations/frame (50ms/frame)

Exact Solution
Newton's Method (13s/frame)

Figure 1: *When used to simulate the motion of a cloth sheet with 6561 vertices our method (left) produces real-time results on a single CPU comparable to those obtained with a much slower off-line method (middle). The method also performs well for one dimensional strands, volumetric objects, and character clothing (right).*

Abstract

We describe a scheme for time integration of mass-spring systems that makes use of a solver based on block coordinate descent. This scheme provides a fast solution for classical linear (Hookean) springs. We express the widely used implicit Euler method as an energy minimization problem and introduce spring directions as auxiliary unknown variables. The system is globally linear in the node positions, and the non-linear terms involving the directions are strictly local. Because the global linear system does not depend on run-time state, the matrix can be pre-factored, allowing for very fast iterations. Our method converges to the same final result as would be obtained by solving the standard form of implicit Euler using Newton's method. Although the asymptotic convergence of Newton's method is faster than ours, the initial ratio of work to error reduction with our method is much faster than Newton's. For real-time visual applications, where speed and stability are more important than precision, we obtain visually acceptable results at a total cost per timestep that is only a fraction of that required for a single Newton iteration. When higher accuracy is required, our algorithm can be used to compute a good starting point for subsequent Newton's iteration.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation.

*ladislav.kavan@gmail.com

ACM Reference Format
Liu, T., Bargteil, A., O'Brien, J., Kavan, L. 2013. Fast Simulation of Mass-Spring Systems. ACM Trans. Graph. 32, 6, Article 214 (November 2013), 7 pages. DOI = 10.1145/2508363.2508406 http://doi.acm.org/10.1145/2508363.2508406.

Copyright Notice
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
2013 Copyright held by the Owner/Author. Publication rights licensed to ACM.
0730-0301/13/11-ART214 $15.00.
DOI: http://dx.doi.org/10.1145/2508363.2508406

Keywords: Time integration, implicit Euler method, mass-spring systems.

Links: DL PDF VIDEO WEB

1 Introduction

Mass-spring systems provide a simple yet practical method for modeling a wide variety of objects, including cloth, hair, and deformable solids. However, as with other methods for modeling elasticity, obtaining realistic material behaviors typically requires constitutive parameters that result in numerically stiff systems. Explicit time integration methods are fast but when applied to these stiff systems they have stability problems and are prone to failure. Traditional methods for implicit integration remain stable but require solving large systems of equations [Baraff and Witkin 1998; Press et al. 2007]. The high cost of solving these systems of equations limits their utility for real-time applications (e.g., games) and slows production work flows in off-line settings (e.g., film and visual effects).

In this paper, we propose a fast implicit solver for standard mass-spring systems with spring forces governed by Hooke's law. We consider the optimization formulation of implicit Euler integration [Martin et al. 2011], where time-stepping is cast as a minimization problem. Our method works well with large timesteps—most of our examples assume a fixed timestep corresponding to the framerate, i.e., $h = 1/30s$. In contrast to the traditional approach of employing Newton's method, we reformulate this minimization problem by introducing auxiliary variables (spring directions). This allows us to apply a block coordinate descent method which alternates between finding optimal spring directions (local step) and finding node positions (global step). In the global step, we solve a linear system. The matrix of our linear system is independent of the current state, which allows us to benefit from a pre-computed sparse Cholesky factorization.

Newton's method is known for its excellent convergence properties. When the iterates are sufficiently close to the optimum, Newton's method exhibits quadratic convergence which out performs block

coordinate descent. However, each iteration of Newton's method requires solving a linear system which changes from one iteration to the next. Those solves are computationally expensive, and therefore commonly used techniques often do not iterate until convergence, but instead apply only *one* iteration of Newton's method. That approach is justified by the practical requirement of obtaining stable, visually plausible results with a limited computational budget.

Although the asymptotic convergence of our method is poor, its initial progress is quite fast. Specifically, the computational cost required to reach a relative error equivalent to a single iteration of Newton's method is only a small fraction of the cost of a single Newton iteration. Our method can be also terminated earlier, resulting in crude yet visually plausible approximation. Such functionality is difficult to achieve otherwise because early termination of conjugate gradients may produce unreliable descent directions due to the fact that conjugate gradients do not reduce the error smoothly [Shewchuk 1994]. In this light, we believe that our method will be appreciated in physics-based animation, because it can compute visually plausible results very quickly. The ability to compute fast yet stable simulations is critical in real-time applications, where speed is much more important than accuracy. In applications requiring higher accuracy, our method can be used to produce a good starting point for Newton's method, because its initial error reduction speed often outperforms the damped Newton phase.

2 Related Work

Mass-spring systems are conceptually simpler and easier to implement than more physically consistent models derived from continuum mechanics using the finite element method [Bonet and Wood 1997]. In contrast to scientific computing, highly accurate material modeling is not always necessary for physics-based animation. As a result, mass-spring systems are widely used for one and two-dimensional structures, such as hair [Selle et al. 2008] and cloth [Choi and Ko 2005], and to a lesser extent for elastic solids [Teschner et al. 2004]. For a more detailed discussion we recommend the survey article by Nealen et al. [2005].

Regardless of whether one employs a mass-spring system or another method based on continuum mechanics, some numerical time integration technique is necessary to simulate the system dynamics. The most straightforward integration methods are explicit, such as explicit Euler [Press et al. 2007]. For the purposes of physics-based animation, where performance concerns dictate large timesteps, explicit methods are often not sufficiently robust and experience stability problems. Seminal works [Terzopoulos et al. 1987; Baraff and Witkin 1998] introduced the implicit Euler method which offers robustness even for stiff systems and large timesteps. Unfortunately, the traditional numerical solution of implicit Euler employs Newton's method, which requires the solution of a sparse linear system at each iteration. The system matrix changes as the system evolves, which typically precludes pre-factorization in direct linear solvers [Botsch et al. 2005]. Recently, Hecht et al. [2012] proposed scheduled updates of Cholesky factors, trading off accuracy of the Hessian for its more efficient amortized evaluation. In contrast to Hecht et al. [2012], our system matrix is fully state-independent which allows us to completely rely on the pre-computed factorization as long as the system parameters and connectivity remain constant.

A drawback of implicit Euler is that integration error manifests as excessive numerical damping. Symplectic integrators [Stern and Desbrun 2006; Kharevych et al. 2006] are known for their energy conservation properties. A related time-stepping strategy involves the combination of implicit and explicit methods (IMEX) [Bridson et al. 2003; Stern and Grinspun 2009]. However, numerical damping may be tolerable in applications where energy conservation is not critical and many recent methods continue to rely on the implicit Euler method [Martin et al. 2011; Hahn et al. 2012]. A recently proposed technique that allows for a more direct control of damping is *energy budgeting* [Su et al. 2013]. Energy budgeting can be applied on top of any numerical time integration method, including ours.

An interesting alternative to classical force-based physics is Position Based Dynamics (PBD) [Müller et al. 2007]. Due to its robustness, speed, and simplicity, PBD has become very popular in the game and visual effects industries. The spring projection concept of PBD is also found in the Nucleus system [Stam 2009] and is also closely related to strain limiting [Provot 1995; Goldenthal et al. 2007; Thomaszewski et al. 2009; Wang et al. 2010; Narain et al. 2012]. PBD presents certain trade-offs by departing from the traditional elasticity models and relying on heuristic *constraint projection*, which utilizes parameters incompatible with standard models. Another issue is that the resulting stiffness of the simulated material depends on the number of PBD constraint projection iterations. In contrast, our method uses classical Hookean springs and converges to the exact implicit Euler solution.

Physics-based simulation is computationally expensive, especially for high-resolution models rich in detail. Long simulation times complicate animation workflows because changing parameters and re-simulating becomes time consuming, and achieving a specific desired behavior may be tedious. This problem was addressed by Bergou et al. [2007], who proposed to work with coarse, fast simulations until the desired behavior was achieved, and only then commit computational resources to high-resolution simulation that attempts to mimic (*track*) the coarse one. Two significant limitations are that an auxiliary coarse version of the model has to be developed and tuned, and that the tracking process may compromise the visual fidelity of the final high-resolution simulation. Our method enables fast approximate previews directly with the high-resolution models.

3 Background and Notation

This section reviews the basic method of implicit Euler and derives the method's optimization formulation. We assume a mechanical system with m points in 3D, evolving through a discrete set of time samples $t_1, t_2, \ldots$ with constant time step h (we typically use $h = 1/30s$). Let us denote the system configuration in time t_n as $\mathbf{q}_n \in \mathbb{R}^{3m}$. The system evolves in time according to Newton's laws of motion, where forces are represented by a non-linear function $\mathbf{f} : \mathbb{R}^{3m} \rightarrow \mathbb{R}^{3m}$, so that $\mathbf{f}(\mathbf{q}_n)$ is the vector of forces acting on all particles at time t_n. We consider only position dependent forces in this section and defer the discussion of damping to Section 4.1. We assume the forces are conservative, i.e., $\mathbf{f} = -\nabla E$, where $E : \mathbb{R}^{3m} \rightarrow \mathbb{R}$ is a potential function (often non-linear and non-convex), encompassing both internal and external forces. The task is to calculate system states $\mathbf{q}_1, \mathbf{q}_2, \ldots$ according to the laws of motion.

Given the diagonal (lumped) mass-matrix $\mathbf{M} \in \mathbb{R}^{3m \times 3m}$, implicit Euler time integration results in the following update rules [Baraff and Witkin 1998]:

$$\mathbf{q}_{n+1} = \mathbf{q}_n + h\mathbf{v}_{n+1} \quad (1)$$

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

where $\mathbf{v}_n$ represents velocity at time t_n. Using the integration rule eq. (1) we can express the velocities as:

$$h\mathbf{v}_n = \mathbf{q}_n - \mathbf{q}_{n-1} \quad (3)$$

$$h\mathbf{v}_{n+1} = \mathbf{q}_{n+1} - \mathbf{q}_n \quad (4)$$

Next, we eliminate velocities from eq. (2) by multiplying it with h and substituting eq. (3) and eq. (4):

$$\mathbf{q}_{n+1} - 2\mathbf{q}_n + \mathbf{q}_{n-1} = h^2\mathbf{M}^{-1}\mathbf{f}(\mathbf{q}_{n+1}) \tag{5}$$

This is nothing but a discretized version of Newton's second law (the well-known $F = ma$). If $\mathbf{q}_{n-1}$ and $\mathbf{q}_n$ are already known (previous states), we need to solve eq. (5) to obtain $\mathbf{q}_{n+1}$ (new state).

The classical recipe for solving the nonlinear system eq. (5) involves linearization of the forces in a known state [Baraff and Witkin 1998]:

$$\mathbf{f}(\mathbf{q}_{n+1}) \approx \mathbf{f}(\mathbf{q}_n) + \left(\nabla\mathbf{f}\big|_{\mathbf{q}_n}\right)(\mathbf{q}_{n+1} - \mathbf{q}_n) \tag{6}$$

where $\nabla\mathbf{f} = -\nabla^2 E \in \mathbb{R}^{3m\times 3m}$ is the Hessian, i.e., matrix of second derivatives, evaluated at $\mathbf{q}_n$. Eq. (6) is then substituted into eq. (5), reducing the problem to a linear system which is solved using e.g. preconditioned conjugate gradients. Alternatively, the nonlinear system in eq. (5) can be converted to an optimization problem. To simplify notation, we will denote the unknown state as $\mathbf{x} := \mathbf{q}_{n+1}$ and the known component as $\mathbf{y} := 2\mathbf{q}_n - \mathbf{q}_{n-1}$. Multiplying by the mass matrix $\mathbf{M}$, we can write eq. (5) succinctly:

$$\mathbf{M}(\mathbf{x} - \mathbf{y}) = h^2\mathbf{f}(\mathbf{x}) \tag{7}$$

The solutions of eq. (7) correspond to critical points of the following function:

$$g(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{y})^T\mathbf{M}(\mathbf{x} - \mathbf{y}) + h^2E(\mathbf{x}) \tag{8}$$

Indeed, $\nabla g = 0$ is exactly eq. (7). This leads to the optimization problem $\min_{\mathbf{x}} g(\mathbf{x})$; this formulation is known as variational implicit Euler [Martin et al. 2011]. To avoid confusion with symplectic methods (sometimes also called *variational* [Stern and Desbrun 2006]), we will refer to eq. (8) as *optimization* implicit Euler. The standard numerical solution of the optimization implicit Euler also employs Newton's method [Martin et al. 2011].

The optimization problem $\min_{\mathbf{x}} g(\mathbf{x})$ offers an interesting insight into Position Based Dynamics [Müller et al. 2007]. If we define an energy potential E_{PBD} as the sum of squares of the PBD constraints, we notice that the PBD constraint projection solver attempts to minimize $g(\mathbf{x})$ using a Gauss-Seidel-like method. Unfortunately, the PBD solver does not take the inertial term $(\mathbf{x} - \mathbf{y})^T\mathbf{M}(\mathbf{x} - \mathbf{y})$ explicitly into account, and further the projection of each individual constraint is not guaranteed to decrease E_{PBD} because the effect of the remaining constraints is ignored. Nevertheless, PBD is typically quite effective at reducing $g(\mathbf{x})$ and can be therefore understood as a heuristic variant of the implicit Euler method where $E := E_{PBD}$.

4 Method

The main idea of our technique is to reformulate the energy potential E in a way that will allow us to employ a block coordinate descent method. The crucial components of E are spring potentials. According to Hooke's law, the spring potential is defined as:

$$\frac{1}{2}k(||\mathbf{p}_1 - \mathbf{p}_2|| - r)^2 \tag{9}$$

where $\mathbf{p}_1, \mathbf{p}_2 \in \mathbb{R}^3$ are spring endpoints, $r \geq 0$ is the rest length, and $k \geq 0$ is the spring stiffness.

The key to our reformulation is the following fact showing that the spring potential eq. (9) is a solution to a specially designed constrained minimization problem.

Lemma. *For each* $\mathbf{p}_1, \mathbf{p}_2 \in \mathbb{R}^3$ *and* $r \geq 0$*:*

$$(||\mathbf{p}_1 - \mathbf{p}_2|| - r)^2 = \min_{||\mathbf{d}||=r} ||(\mathbf{p}_1 - \mathbf{p}_2) - \mathbf{d}||^2$$

Proof. For brevity we define $\mathbf{p}_{12} := \mathbf{p}_1 - \mathbf{p}_2$. Given the constraint $||\mathbf{d}|| = r$, we rewrite the left side of the equation:

$$(||\mathbf{p}_{12}|| - r)^2 = (||\mathbf{p}_{12}|| - ||\mathbf{d}||)^2$$

By applying the reverse triangle inequality, we have:

$$(||\mathbf{p}_{12}|| - ||\mathbf{d}||)^2 \leq ||\mathbf{p}_{12} - \mathbf{d}||^2$$

Next, if we substitute $\mathbf{d} = r\ (\mathbf{p}_{12}/||\mathbf{p}_{12}||)$ to the right side, we obtain:

$$\left|\left|\mathbf{p}_{12} - r\frac{\mathbf{p}_{12}}{||\mathbf{p}_{12}||}\right|\right|^2 = \left|\left|\frac{\mathbf{p}_{12}}{||\mathbf{p}_{12}||}(||\mathbf{p}_{12}|| - r)\right|\right|^2 = (||\mathbf{p}_{12}|| - r)^2$$

Therefore, when $\mathbf{d} = r\ (\mathbf{p}_{12}/||\mathbf{p}_{12}||)$, the right hand side of the equation produces its minimum value that equals to the left. □

The reformulation of Hooke's law into the minimization problem:

$$\min_{||\mathbf{d}||=r} ||(\mathbf{p}_1 - \mathbf{p}_2) - \mathbf{d}||^2 \tag{10}$$

is reminiscent of as-rigid-as-possible methods [Sorkine and Alexa 2007; Chao et al. 2010], because $\mathbf{d}/r$ can be interpreted as a rotated rest-pose spring direction. If we sum the contributions of all springs together, after some matrix algebra we obtain:

$$\frac{1}{2}\sum_{i=1}^{s} k_i||\mathbf{p}_{i_1} - \mathbf{p}_{i_2} - \mathbf{d}_i||^2 = \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x} - \mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{d} \tag{11}$$

where s is the total number of springs, $i_1, i_2 \in \{1, 2, \ldots, m\}$ are indices of spring i endpoints, and the vector $\mathbf{x} = (\mathbf{p}_1, \ldots, \mathbf{p}_m)$. The matrices $\mathbf{L} \in \mathbb{R}^{3m\times 3m}, \mathbf{J} \in \mathbb{R}^{3m\times 3s}$ are defined as follows:

$$\mathbf{L} = \left(\sum_{i=1}^{s} k_i\mathbf{A}_i\mathbf{A}_i^\mathsf{T}\right) \otimes \mathbf{I}_3,\ \mathbf{J} = \left(\sum_{i=1}^{s} k_i\mathbf{A}_i\mathbf{S}_i^\mathsf{T}\right) \otimes \mathbf{I}_3 \tag{12}$$

where $\mathbf{A}_i \in \mathbb{R}^m$ is the incidence vector of i-th spring, i.e., $A_{i,i_1} = 1, A_{i,i_2} = -1$, and zero otherwise. Similarly, $\mathbf{S}_i \in \mathbb{R}^s$ is the i-th spring indicator, i.e., $S_{i,j} = \delta_{i,j}$. The matrix $\mathbf{I}_3 \in \mathbb{R}^{3\times 3}$ is the identity matrix and $\otimes$ denotes Kronecker product. Note that the matrix $\mathbf{L}$ is nothing but a stiffness-weighted Laplacian of the mass-spring system graph.

If we denote external forces (gravity, user interaction forces, and collision response forces) as $\mathbf{f}_{ext} \in \mathbb{R}^{3m}$, we can write the potential of our system as:

$$E(\mathbf{x}) = \min_{\mathbf{d}\in U}\ \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x} - \mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{d} + \mathbf{x}^\mathsf{T}\mathbf{f}_{ext} \tag{13}$$

where $U = \{(\mathbf{d}_1, \ldots, \mathbf{d}_s) \in \mathbb{R}^{2s} : ||\mathbf{d}_i|| = r_i\}$ is the set of rest-length spring directions. We plug this into the minimization objective eq. (8), arriving at the final optimization problem:

$$\min_{\mathbf{x}\in\mathbb{R}^{3m},\ \mathbf{d}\in U}\ \frac{1}{2}\mathbf{x}^\mathsf{T}(\mathbf{M} + h^2\mathbf{L})\mathbf{x} - h^2\mathbf{x}^\mathsf{T}\mathbf{J}\mathbf{d} + \mathbf{x}^\mathsf{T}\mathbf{b} \tag{14}$$

where we have aggregated the external forces and inertia ($\mathbf{y}$ in Section 3) into vector $\mathbf{b} \in \mathbb{R}^{3m}$ and dropped the constant terms. The vector $\mathbf{x}^* \in \mathbb{R}^{3m}$ where the minimum in eq. (14) is attained is an *exact* solution of the implicit Euler timestep.

Object	m	s	h	Iteration Time	Iteration Count	Collision Time	Total Frame Time	Pre-factor Time
Curtain	6561	32158	33 ms	5 ms	10	-	50 ms	113 ms
Hippo	2387	13135	33 ms	1 ms	20	-	20 ms	18 ms
Frog	6834	35261	33 ms	3.1 ms	20	-	62 ms	54 ms
Dog	28390	148047	33 ms	20.3 ms	20	-	406 ms	442 ms
Dress	3505	6811	8.3 ms	0.61 ms	20	476 ms	488 ms	11 ms
Brush	1995	6405	33 ms	0.68 ms	20	0.99 ms	14.6 ms	9.8 ms

Table 1: *Timings for all our examples. m is number of vertices, s is number of springs, and h is the integration timestep.* Iteration Time *is the cost of one iteration of block coordinate descent,* Iteration Count *is number of iterations per frame, and* Collision Time *is collision detection and response cost for one frame.* Total Frame Time *is the total time to simulate one frame, with* (Total Frame Time) = (Iteration Time) * (Iteration Count) + (Collision Time). Pre-factor Time *is the time to pre-compute the sparse Cholesky factorization.*

Numerical solution. The minimization problem eq. (14) can be solved using block coordinate descent [Sorkine and Alexa 2007] (also known as alternating optimization). Starting with an initial guess for $\mathbf{x}$ (we use $\mathbf{y}$), we first fix $\mathbf{x}$ and compute the optimal $\mathbf{d}$ (local step). Second, we fix $\mathbf{d}$ and compute the optimal $\mathbf{x}$ (global step), repeating this process until a maximal number of iterations is reached. In contrast to previous as-rigid-as-possible methods, our local step does not require Singular Value Decompositions, but only vector normalizations (reciprocal square roots). It can also be interpreted as projecting the springs to their rest lengths, but unlike with Position Based Dynamics, spring stiffness are correctly taken into account (they are built into $\mathbf{L}$ and $\mathbf{J}$). In the global step (fixed $\mathbf{d}$), we need to solve a convex quadratic minimization problem. Indeed, because $\mathbf{L}$ is symmetric and positive semi-definite, the system matrix $\mathbf{M} + h^2\mathbf{L}$ is symmetric positive definite. Most importantly, as long as the timestep, particle masses, spring stiffness, and connectivity remain unchanged, the system matrix is constant. Therefore, we pre-compute its sparse Cholesky factorization (guaranteed to exist), which makes the linear system solve very fast. We would like to emphasize that this is *not* an ad-hoc approximation—our method converges to the exact solution of the implicit Euler method with standard Hookean springs.

4.1 Damping and Collisions

A simple method to introduce viscous damping into our formulation is as follows. Recall that the term $\mathbf{y}$ from eq. (8) is simply the result of inertia (Newton's first law) when all forces are ignored, i.e., $\mathbf{y} = \mathbf{q}_n + h\mathbf{v}_n$. Damping can be achieved simply by setting $\mathbf{y}$ to $\mathbf{q}_n + h\tilde{\mathbf{v}}_n$, where we replaced $\mathbf{v}_n$ as defined in eq. (3) with a damped velocity $\tilde{\mathbf{v}}_n$. We use only a very simple damping model—ether drag [Su et al. 2013], which sets $\tilde{\mathbf{v}}_n := \alpha\mathbf{v}_n$, where $\alpha \in [0, 1]$ is a parameter, typically very close to 1. However, any damping model can be used with our method, such as the rigid-body modes preserving drag [Müller et al. 2007] or truly material-only stiffness-proportional damping [Nealen et al. 2005].

Conceptually, collision forces are part of the external force vector $\mathbf{f}_{ext}$. Instead of calculating the collision forces explicitly, we note that in the global step $\mathbf{f}_{ext}$ enters the right-hand-side term, and because $\mathbf{M} + h^2\mathbf{L}$ has full rank, *any* translation of $\mathbf{x}$ can be accomplished by appropriately chosen $\mathbf{f}_{ext}$. Therefore, we can short-circuit this process and instead of computing $\mathbf{f}_{ext}$, we directly move $\mathbf{x}$ to the desired collision-free state, computed by collision response routines. Our method can be coupled with many collision detection and response techniques. For our real-time examples, we use fast approximate collision detection based on spatial subdivision [Müller et al. 2007]. For off-line simulation of character clothing, we rely on accurate ARCSim collision handling routines [Narain et al. 2012]. Specifically, collisions are detected using a bounding volume hierarchy [Tang et al. 2010] and collision response is computed using non-rigid impact zones [Harmon et al. 2008].

5 Results

In real-time simulation, it is desirable to use a constant timestep h chosen according to the target framerate. We compare our method to classical semi-implicit methods [Baraff and Witkin 1998], employing only one iteration of Newton's method; please see the accompanying video. While semi-implicit methods work well for $h = 1/120s$ or $h = 1/60s$, with $h = 1/30s$ we obtain indefinite system matrices. To avoid this issue, Baraff and Witkin [1998] recommend adaptive step size control. Unfortunately, doing so is impractical due to variable run-time cost, and also inconsistent because the amount of artificial damping inherent to implicit Euler depends on the step size. Our method works robustly with fixed timestep even if only a few iterations of the local/global solver are enabled. In most of our examples we use $h = 1/30s$ and compare our technique to a robust numerical implementation of Newton's method that employs a line search scheme and diagonal Hessian correction in the case of indefinite matrices [Martin et al. 2011]. Note that our method does not require any such precautions—both the local and global steps find the exact minimum in their subsets of variables, so no line search is necessary.

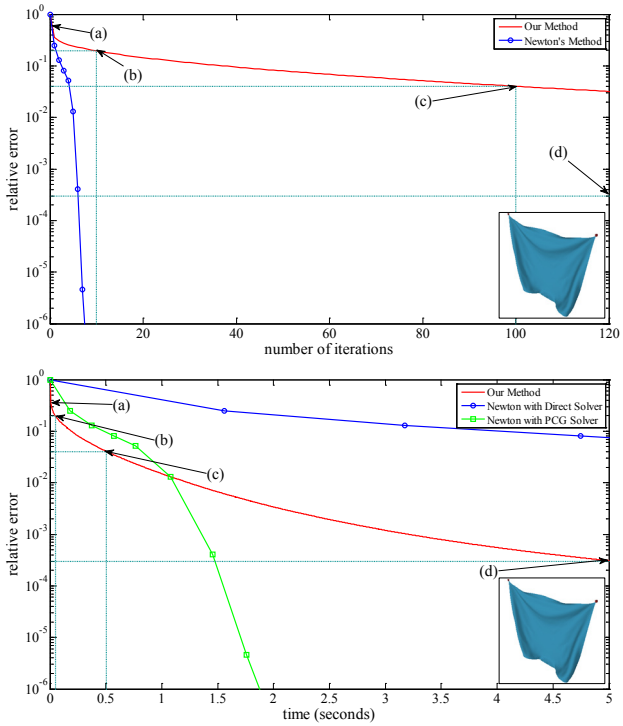
The complexity of our testing models and the performance of our method is summarized in Table 1. In Figure 2, we study the convergence speed on one typical frame of our cloth-swinging animation. The relative error reported in Figure 2 is defined as:

$$\frac{g(\mathbf{x}_i) - g(\mathbf{x}^*)}{g(\mathbf{x}_0) - g(\mathbf{x}^*)} \tag{15}$$

where $\mathbf{x}_0$ is the initial guess, $\mathbf{x}_i$ is the current iterate, and $\mathbf{x}^*$ is the final solution. Our method exhibits a linear convergence rate, whereas Newton's method quickly enters its quadratic convergence phase [Boyd and Vandenberghe 2004]. However, Figure 2 (top) ignores the fact that one iteration of Newton's method is computationally substantially more expensive than one iteration of our method. In Figure 2 (bottom), we therefore plot the relative error with respect to time. We see that Preconditioned Conjugate Gradients runs much faster in this case than a sparse direct solver. For both methods, as well as with our technique, we use the Eigen library [Guennebaud et al. 2013], running on a single core of Intel i7-3720QM CPU at 2.60GHz.

While block coordinate descent cannot compete with the quadratically convergent stage of Newton's method, we notice that our approach outperforms Newton's method in its first (damped) phase. In other words, Newton's method becomes more effective only when the current iterate $\mathbf{x}_i$ is already close to the solution $\mathbf{x}^*$. If an exact solution is desired, our technique can be useful for quickly calculating a good starting point for Newton's method.

The main practical benefit of our method stems from the fact that exact solution is rarely required in physics-based animation. Indeed, previous methods [Baraff and Witkin 1998] limit the number of iterations of Newton's method to one. To experimentally evaluate



	Iteration Count	Time	Relative Error
(a)	1	5.4ms	3.61×10^{-1}
(b)	10	50.6ms	1.96×10^{-1}
(c)	100	501ms	4.02×10^{-2}
(d)	1000	5.05s	2.98×10^{-4}

Figure 2: *Comparison of relative error vs. iteration count (top) does not reflect the cost of each iteration. Below we plot the relative error vs. computation time. In both graphs we focus on one time step of our curtain-swinging animation at the depicted frame.*

the effect of approximate solutions, we tested our method on a simple animation sequence simulated with our method using 1, 10, 100, and 1000 iterations of the local/global solver. One iteration produces a stable and plausible simulation, but the wrinkles look a bit inflexible (Figure 4, and the accompanying video). Ten iterations seem to offer the best trade-off between speed and quality. In our example frame (Figure 2), ten iterations of our method achieve better relative error than one iteration of Newton's method (0.196, vs. 0.2496 for Newton) as well as faster run-time (50.6ms, vs. 181ms for one iteration of Newton with PCG). With a hundred or a thousand iterations it is difficult to tell the difference from an exact solution.

Quick approximate simulation can be achieved also using Position Based Dynamics (PBD) [Müller et al. 2007]. One problem with PBD is that its stiffness parameters are not compatible with the standard Hookean model. We tried to carefully tune the PBD parameters to get behavior as close as possible to our settings. Unfortunately, even though the PBD solver adjusts its parameters according to the number of iterations, increasing the number of iterations still increases the effective stiffness of the system. Our method does not suffer from this problem and converges to the exact implicit Euler solution, as shown in the accompanying video.

We designed the following experiments to analyze the convergence of our method. In the first experiment, we test how spring stiffness affects the convergence, using our curtain model with 441 vertices. Figure 3 reports the relative error from eq. (15) averaged over 50

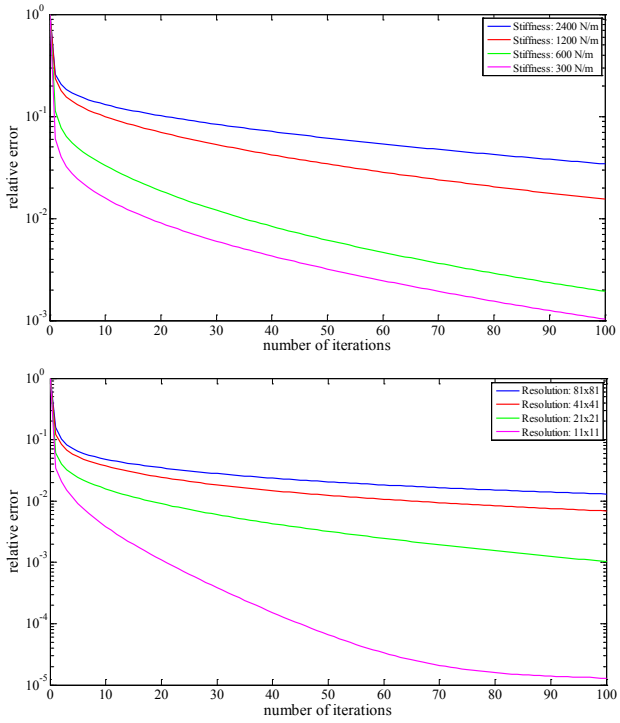


Figure 3: *Convergence of our method for varying spring stiffness coefficients (top) and varying spatial resolution (bottom).*

simulation frames. As expected, higher stiffness leads to slower convergence. In the next experiment, we study the effect of varying spatial resolution using a curtain model with fixed dimension (1m $\times$ 1m) and mass (1kg). To achieve resolution independent material behavior, the spring stiffness is proportional to the resolution, i.e., when we double the resolution, we divide the spring lengths by two and multiply their stiffness by two. In all experiments we observe that while our method proceeds very quickly early on, subsequent iterations are less effective in reducing the error. Therefore, if exact results are desired, we do not advise iterating our method until convergence but instead recommend switching to Newton's method.

Collision handling is an important aspect of physics-based animation. We designed two experiments to test the behavior of our method in scenarios rich in contact and collisions. The first test involves a brush model, where individual strands collide not only with a static rigid object, but also with each other (self-collisions), see Figure 5. Our second example is a dancing clothed character (Figure 6), leveraging the publicly available models and code from ARCSim [Narain et al. 2012]. In the clothing example, 20 iterations of our method take a total time of 12.2ms. ARCSim's collision detection and response, executed once per frame, takes 476ms and therefore presents the bottleneck. We conclude that our method is best suited for real-time applications where approximate collision handling is sufficient. Note that adaptive remeshing would have invalidated our pre-factored system matrices, so we disable the adaptive remeshing functions of ARCSim.

6 Limitations and Future Work

We note that using a fixed number of iterations of our local/global solver produces only approximate results. In some settings, e.g., when strong impact forces are generated by a collision event, we observe lack of detail (Figure 7). The problem is that the limited number of iterations taken by our method does not allow propagation

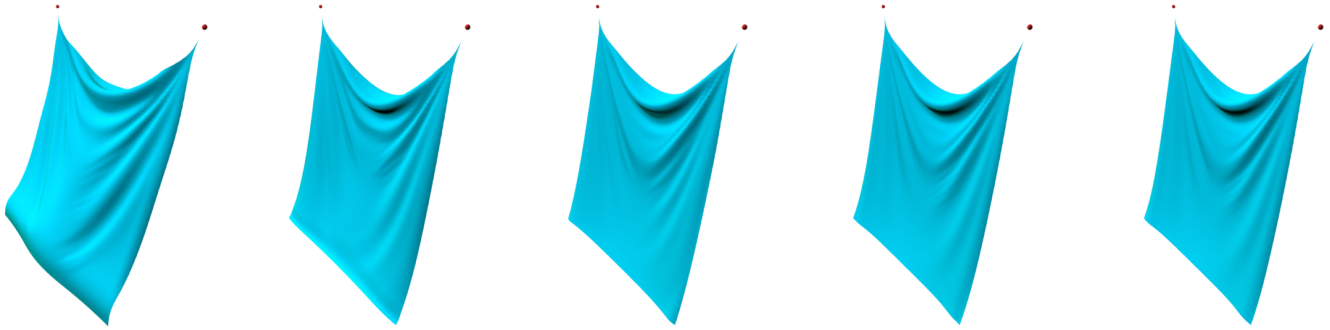


Figure 4: *One example frame from our cloth animation simulated using our method with 1, 10, 100, and 1000 iterations of our local/global solver. Exact solution computed using Newton's method is shown for comparison.*

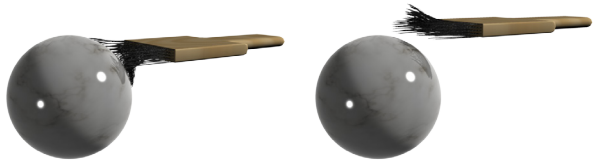


Figure 5: *Bristles of a brush colliding with a rigid object and each other.*

Figure 6: *Character clothing with continuous collision detection, including both cloth-body and cloth-cloth collisions.*

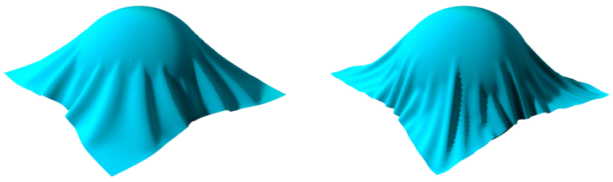


Figure 7: *In challenging situations such as strong impact on collision our approximate solution results in loss of detailed wrinkles.*

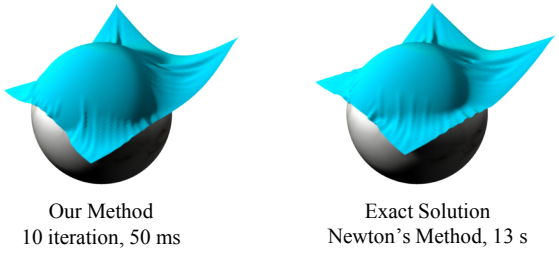


Figure 8: *Our method produces results comparable to the exact solution when only moderate impact forces are involved.*

of the shockwave quickly enough, resulting in artificial damping. This issue does not occur with slower collisions and weaker impact forces, as shown in Figure 8. A trade-off inherent to our method is that changing the mass-spring system parameters (masses or stiffness) or the spring connectivity requires re-computing the Cholesky factorization. This may be an issue if effects such as tearing are required, and one possible solution would be to employ fast Cholesky updates [Hecht et al. 2012].

We only consider mass-spring systems in this paper and currently support only classical linear (Hookean) springs. In the future, we are planning to include more general spring models, e.g., bending and/or area springs, and generalize our approach to thin shells, where the rest-length spring directions in eq. (10) would be replaced by 2×2 SVD, which can be still computed in a closed form. We believe it will also be very fruitful to experiment with different types of numerical techniques, such as nonlinear conjugate gradients and quasi-Newton methods. We also intend to generalize our method to symplectic time integration approaches. Finally, we are interested in the perceptual aspects of time integration and we would like to more formally address the question of how much error is noticeable by the average observer.

7 Conclusions

We presented a novel numerical method for implicit Euler time stepping of mass-spring system dynamics. Our technique is based on block coordinate descent, which gives it different properties than the traditional Newton's method. Our method can approximate the solution in a limited amount of computational time, making it particularly attractive for real-time applications—we demonstrate real-time cloth with quality similar to the exact solution. The proposed algorithm can also be useful for quick simulation preview and for bootstrapping Newton's method. We hope that our method will encourage further investigation of time integration techniques and the underlying nonlinear numerical problems.

Acknowledgements

The authors thank Stelian Coros, Alec Jacobson, Lily Kharevych, Sebastian Martin, Dominik Michels, Rahul Narain, Mark Pauly, Bernhard Thomaszewski, Andreas Weber, and Changxi Zheng for many insightful discussions, Rahul Narain and Armin Samii for help with ARCSim, and the anonymous reviewers for their helpful suggestions. The original ideas for this work were developed at the Bellairs workshop sponsored by McGill University and organized by Paul Kry. This work was supported in part by funding from the Intel Science and Technology Center for Visual Computing, gifts from Pixar Animation Studios, Adobe Systems Incorporated, and National Science Foundation Awards IIS-1249756 and CNS-0855167.

References

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proceedings of SIGGRAPH 1998*, 43–54.

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

BONET, J., AND WOOD, R. D. 1997. *Nonlinear continuum mechanics for finite element analysis*. Cambridge university press.

BOTSCH, M., BOMMES, D., AND KOBBELT, L. 2005. Efficient linear system solvers for mesh processing. In *Mathematics of Surfaces XI*. Springer, 62–83.

BOYD, S., AND VANDENBERGHE, L. 2004. *Convex optimization*. Cambridge university press.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, 28–36.

CHAO, I., PINKALL, U., SANAN, P., AND SCHRÖDER, P. 2010. A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4, 38:1–38:6.

CHOI, K.-J., AND KO, H.-S. 2005. Stable but responsive cloth. In *ACM SIGGRAPH 2005 Courses*, ACM, 1.

GOLDENTHAL, R., HARMON, D., FATTAL, R., BERCOVIER, M., AND GRINSPUN, E. 2007. Efficient simulation of inextensible cloth. *ACM Trans. Graph. 26*, 3, 49:1–49:8.

GUENNEBAUD, G., JACOB, B., ET AL., 2013. Eigen v3. http://eigen.tuxfamily.org.

HAHN, F., MARTIN, S., THOMASZEWSKI, B., SUMNER, R., COROS, S., AND GROSS, M. 2012. Rig-space physics. *ACM Transactions on Graphics (TOG) 31*, 4, 72.

HARMON, D., VOUGA, E., TAMSTORF, R., AND GRINSPUN, E. 2008. Robust treatment of simultaneous collisions. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, 23.

HECHT, F., LEE, Y. J., SHEWCHUK, J. R., AND O'BRIEN, J. F. 2012. Updated sparse cholesky factors for corotational elastodynamics. *ACM Transactions on Graphics (TOG) 31*, 5, 123.

KHAREVYCH, L., YANG, W., TONG, Y., KANSO, E., MARSDEN, J. E., SCHRÖDER, P., AND DESBRUN, M. 2006. Geometric, variational integrators for computer animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, 43–51.

MARTIN, S., THOMASZEWSKI, B., GRINSPUN, E., AND GROSS, M. 2011. Example-based elastic materials. In *ACM Transactions on Graphics (TOG)*, vol. 30, ACM, 72.

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2, 109–118.

NARAIN, R., SAMII, A., AND O'BRIEN, J. F. 2012. Adaptive anisotropic remeshing for cloth simulation. *ACM Transactions on Graphics (TOG) 31*, 6, 152.

NEALEN, A., MÜLLER, M., KEISER, R., BOXERMAN, E., AND CARLSON, M. 2005. Physically based deformable models in computer graphics. *Comput. Graph. Forum 25*, 4, 809–836.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 2007. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

PROVOT, X. 1995. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface 1995*, 147–154.

SELLE, A., LENTINE, M., AND FEDKIW, R. 2008. A mass spring model for hair simulation. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, 64.

SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Pittsburgh, PA, USA.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proc. Symposium on Geometry Processing*, 109–116.

STAM, J. 2009. Nucleus: towards a unified dynamics solver for computer graphics. In *IEEE Int. Conf. on CAD and Comput. Graph.*, 1–11.

STERN, A., AND DESBRUN, M. 2006. Discrete geometric mechanics for variational time integrators. In *ACM SIGGRAPH 2006 Courses*, ACM, 75–80.

STERN, A., AND GRINSPUN, E. 2009. Implicit-explicit variational integration of highly oscillatory problems. *Multiscale Modeling & Simulation 7*, 4, 1779–1794.

SU, J., SHETH, R., AND FEDKIW, R. 2013. Energy conservation for the simulation of deformable bodies. *IEEE Trans. Vis. Comput. Graph. 19*, 2, 189–200.

TANG, M., MANOCHA, D., AND TONG, R. 2010. Fast continuous collision detection using deforming non-penetration filters. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, 7–13.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. *SIGGRAPH Comput. Graph. 21*, 4 (Aug.), 205–214.

TESCHNER, M., HEIDELBERGER, B., MULLER, M., AND GROSS, M. 2004. A versatile and robust model for geometrically complex deformable solids. In *Computer Graphics International, 2004. Proceedings*, IEEE, 312–319.

THOMASZEWSKI, B., PABST, S., AND STRASSER, W. 2009. Continuum-based strain limiting. *Comput. Graph. Forum 28*, 2, 569–576.

WANG, H., O'BRIEN, J., AND RAMAMOORTHI, R. 2010. Multi-resolution isotropic strain limiting. *ACM Transactions on Graphics (TOG) 29*, 6, 156.