To appear in ACM TOG ().

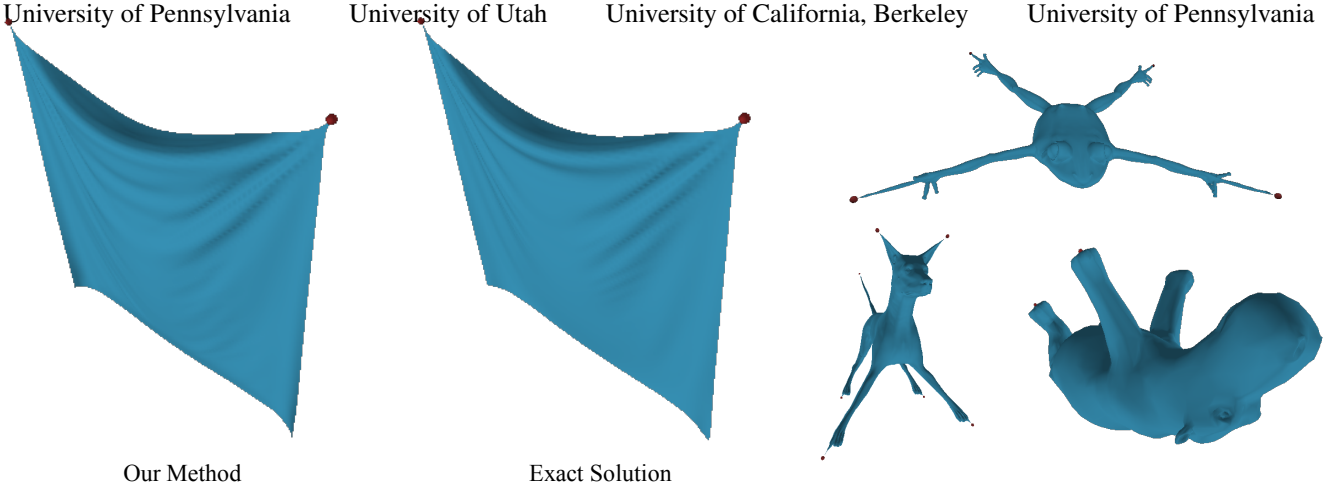
Fast Simulation of Mass-Spring Systems

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: *We propose a method for fast approximate time integration of dynamic mass-spring systems. For example, our cloth model with 6561 vertices simulates in real-time on a single CPU core with quality comparable to off-line techniques.*

Abstract

Mass-spring systems are a classical and popular approach to model the dynamics of deformable objects. A common approach to time integration is implicit Euler method, with numerical solution computed using Newton's iteration. We propose a different numerical procedure to solve implicit Euler time stepping. Our method converges to the same result as Newton's method, but the involved linear systems do not depend on the current state and therefore permit pre-factorization, resulting in very fast iterations. Our method can produce visually pleasing simulations at a fraction of the time required by a single iteration of Newton's method, which is attractive especially for real-time applications. As an example, we demonstrate real-time cloth with quality comparable to off-line simulations. Our technique can also be beneficial in pre-production by providing quick simulation previews before committing resources to a fully accurate simulation. When exact results are required, our algorithm is useful to compute a good starting point for Newton's iteration.

CR Categories: I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism—Animation

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

1 Introduction

Numerical time integration of stiff deformable objects is a challenging problem. Mass-spring systems are a simple yet practical tool to model a large variety of objects, such as cloth, hair, or deformable solids. However, even very simple mass-spring systems often result in difficult numerical problems, because the stiffness of each spring typically needs to be relatively high to avoid "bouncy" behavior. Explicit time integration methods are fast but not sufficiently robust. Traditional methods for implicit integration [**?**] require solving sparse linear systems. This limits their applicability in real-time applications (e.g., games) and slows down production workflows in off-line settings (e.g., film and visual effects).

In this paper, we consider a standard mass-spring system with spring forces governed by the Hooke's law. We consider the optimization formulation of implicit Euler [**?**], where time-stepping is cast as a minimization problem. Our method works very well with large timesteps, all our examples assume a fixed timestep corresponding to the framerate, i.e., $h = 0.033s$. In contrast to the traditional solution 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 stitching the springs together (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 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, unlike block coordinate descent. However, each iteration of Newton's method requires solving a linear system which changes from one timestep to another. This is computationally expensive, and therefore previous techniques [**?**] do not iterate until convergence, but apply only *one* iteration of Newton's method. This is justified by the fact that computer animation does not demand physically accurate, but only physically plausible results. Our goal is to provide artists with fast and practical tools to achieve the desired effects, not to resolve time integration to machine-precision accuracy.

In this light, we believe that our method will be appreciated in physics-based animation, because it can compute visually plausible results very quickly. In fact, our technique obtains useful approximate solutions in a fraction of the time required even for one iteration of Newton's method. 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 [**?**]. The ability to compute fast yet stable simulations is critical in real-time applications, where

speed is much more important than accuracy. Our method can be also practical in pre-production for film and visual effects, where it would enable fast simulation preview without relying on simplified deformation models or coarse resolution. If exact solution is desired, our method can also be useful as a way to produce good starting point for Newton's method, because its initial convergence speed often outperforms the damped Newton phase.

2 Related Work

Mass-spring systems are conceptually simpler and easier to deploy than more rigorous models derived from continuum mechanics using the finite element method [**?**]. In contrast to scientific computing, highly accurate material modeling is typically not necessary in physics-based animation. Mass-spring systems are widely used especially with one and two-dimensional structures, such as hair [**?**] and cloth models [**?**], but find their applications also in the realm of elastic solids [**?**]. For a more detailed discussion we refer to the survey by Nealen et al. [**?**].

Regardless of whether we choose mass-spring systems or continuum mechanics-based models, a numerical time integration technique is necessary to simulate the system dynamics. The most straightforward integration methods are explicit, such as explicit Euler [**?**]. For the purposes of physics-based animation, explicit methods are often not sufficiently robust. Seminal works [**?**; **?**] 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 timestep. The system matrix changes as the system evolves, which typically precludes pre-factorization in direct linear solvers [**?**]. Recently, Hecht et al. [**?**] proposed scheduled updates of Cholesky factors, trading off accuracy of the Hessian for its more efficient amortized evaluation. In contrast to [**?**], 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.

Symplectic integrators [**?**; **?**] are known for their superior energy conservation properties. A related time-stepping strategy involves the combination of implicit and explicit methods (IMEX) [**?**; **?**]. However, damping is often desirable for visually pleasing simulations and many recent methods continue to rely on the implicit Euler method [**?**; **?**]. A recently proposed technique that allows for a more direct control of damping is *energy budgeting* [**?**]. Energy budgeting can be applied on top of any numerical time integration method – ours is no exception.

An interesting alternative to classical force-based physics is Position Based Dynamics (PBD) [**?**]. Due to its robustness, speed, and simplicity, PBD became very popular in the game and visual effects industries. The spring projection concept of PBD is found also in the Nucleus system [**?**] and is also closely related to strain limiting [**?**; **?**; **?**; **?**]. 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. This complicates workflows e.g. in feature film and visual effects – achieving the desired behavior may be tedious, because changing parameters and re-simulating is time consuming. This problem was addressed by Bergou et al. [**?**], 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. The trade-offs are that 1) an auxiliary coarse version of the model has to be developed and tuned and 2) the tracking process may compromise the visual fidelity of the final high-resolution simulation. Our method enables fast previews directly with the high-resolution models.

3 Background and Notation

For didactic purposes, this section recapitulates the basic principles and derives the optimization formulation of implicit Euler. We assume a mechanical system with m points in 3D, evolving through a discrete set of time samples $t_1, t_2, \dots$ with constant time step h (we use $h = 0.033s$). 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 $\mathbf{f} : \mathbb{R}^{3m} \to \mathbb{R}^{3m}$, i.e. $\mathbf{f}(\mathbf{q}_n)$ 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 Sec. 4.1. We assume the forces are conservative, i.e., $\mathbf{f} = -\nabla E$, where $E : \mathbb{R}^{3m} \to \mathbb{R}$ is a potential function (often non-linear and non-convex), encompassing both internal and external forces. The task is to calculate system states $\mathbf{q}_1, \mathbf{q}_2, \dots$ according to the laws of motion.

Given the diagonal mass-matrix as $\mathbf{M} \in \mathbb{R}^{3m \times 3m}$, the implicit Euler time integration method works according to the following update rules [**?**]:

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

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

where $\mathbf{v}_n$ represents velocity at time t_n. Using the integration rule (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 equation (2) by multiplying it with h and substituting (3) and (4):

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

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 (5) to obtain $\mathbf{q}_{n+1}$ (new state).

The classical recipe for solving the nonlinear system (5) involves linearization of the forces in a known state [**?**]:

$$\mathbf{f}(\mathbf{q}_{n+1}) \approx \mathbf{f}(\mathbf{q}_n) + \nabla\mathbf{f}(\mathbf{q}_n)(\mathbf{q}_{n+1} - \mathbf{q}_n) \quad (6)$$

where $\nabla\mathbf{f} = -\nabla^2 E \in \mathbb{R}^{3m \times 3m}$ is the Hessian, i.e., matrix of second derivatives. Equation (6) is then substituted into (5), reducing the problem to a linear system which is solved using e.g. preconditioned conjugate gradients. Alternatively, the system of nonlinear equations (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 (5) succinctly:

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

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

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

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

The optimization problem $\min_{\mathbf{x}} g(\mathbf{x})$ offers an interesting insight into Position Based Dynamics [**?**]. 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. The problem is that the PBD solver 1) does not take the inertial term $(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y})$ explicitly into account and 2) 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 in 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 component 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 (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* $r \geq 0$*:*

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

Proof. We directly minimize over the auxiliary variable $\mathbf{d} \in \mathbb{R}^3$. For brevity we define $\mathbf{p}_{12} := \mathbf{p}_1 - \mathbf{p}_2$ and rewrite:

$$||\mathbf{p}_{12} - \mathbf{d}||^2 = ||\mathbf{p}_{12}|| + ||\mathbf{d}||^2 - 2\mathbf{p}_{12}^\mathsf{T}\mathbf{d}$$

Due to the constraint on $\mathbf{d}$, we note that $||\mathbf{d}||^2 = r^2$ and therefore:

$$\min_{||\mathbf{d}||=r} \; ||\mathbf{p}_{12} - \mathbf{d}||^2 = \min_{||\mathbf{d}||=r} \; -2\mathbf{p}_{12}^\mathsf{T}\mathbf{d} = \max_{||\mathbf{d}||=r} \; \mathbf{p}_{12}^\mathsf{T}\mathbf{d}$$

The solution is obviously $\mathbf{d} = r\,(\mathbf{p}_{12}/||\mathbf{p}_{12}||)$. If we substitute this into $||\mathbf{p}_{12} - \mathbf{d}||^2$, we obtain:

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

□

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 [**?**; **?**], because $\mathbf{d} \in \mathbb{R}^3$ 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 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 $U = \{(\mathbf{d}_1, \ldots, \mathbf{d}_s) \in \mathbb{R}^{3s} : ||\mathbf{d}_i|| = r_i\}$ is the set of rest-length spring directions. We plug this into the minimization objective (8), arriving at the final optimization problem:

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

where we have aggregated the external forces and inertia ($\mathbf{y}$ in Sec. 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 (14) is attained is an *exact* solution of the implicit Euler timestep.

Numerical solution. The minimization problem (14) can be solved using block coordinate descent [**?**] (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 be also 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 damping into our formulation is as follows. Recall that the term $\mathbf{y}$ from equation (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 equation (3) with a damped velocity $\tilde{\mathbf{v}}_n$. We use only a very simple damping model – ether drag [**?**], 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 [**?**] or truly material-only stiffness-proportional damping [**?**].

Object	m	s	time/iteration	Pre-factorization
Cloth	6561	32158	5ms	113ms
Hippo	2387	13135	1ms	18ms
Frog	6834	35261	3.1ms	54ms
Dog	28390	148047	20.3ms	442ms

Table 1: *Our example models: number of vertices (m), springs (s), run-time of our method per one iteration, and time to pre-compute sparse Cholesky factorization.*

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. These methods for handling damping and collisions are arguably basic, but effective in achieving the desired behavior.

5 Results

In real-time simulation, it is desirable to use a constant timestep h chosen according to the target framerate. We use $h = 0.033s$ in all our examples. Our method works robustly with fixed timestep even if only few iterations of the local/global solver are enabled. Semi-implicit methods [**?**] rely on adaptive step size control to achieve robustness which is 1) impractical due to variable run-time cost and 2) inconsistent, because the amount of artificial damping inherent to implicit Euler depends on the step size. We compare our technique to a state-of-the-art numerical implementation of Newton's method which employs a line search scheme and diagonal Hessian correction in case of indefinite matrices [**?**], which enables constant step size. 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.

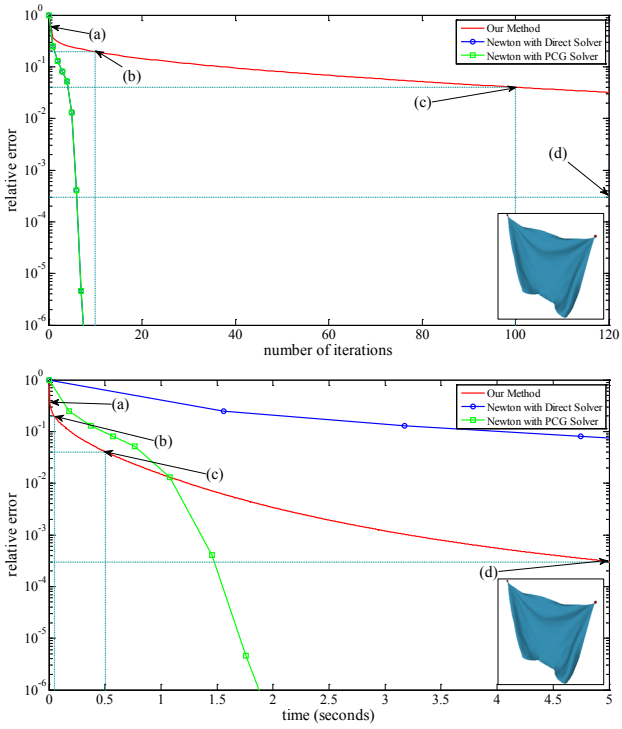
The complexity of our testing models and the performance of our method is reported in Tab. 1. We study the convergence speed on one typical frame of our cloth-swinging animation (Fig. 2). The relative error reported in Fig. 2 is defined as:

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

where $\mathbf{x}_0$ is initial guess, $\mathbf{x}_i$ is the current iterate, and $\mathbf{x}^*$ is the final solution. Our method exhibits linear convergence rate, whereas Newton's method quickly enters its quadratic convergence phase [**?**]. However, Fig. 2 (top) ignores the fact that one iteration of Newton's method is much more computationally expensive than one iteration of our method. In Fig. 2 (bottom), we therefore plot the relative error with respect to time. We see that Preconditioned Conjugate Gradients run much faster in this case than a sparse direct solver. For both methods, as well as with our technique, we use the Eigen library [**?**], 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 enough to the solution $\mathbf{x}^*$. If exact solution is desired, our technique can be useful to quickly calculate a good starting point for Newton's method.

The main practical implication of our method stems from the fact that

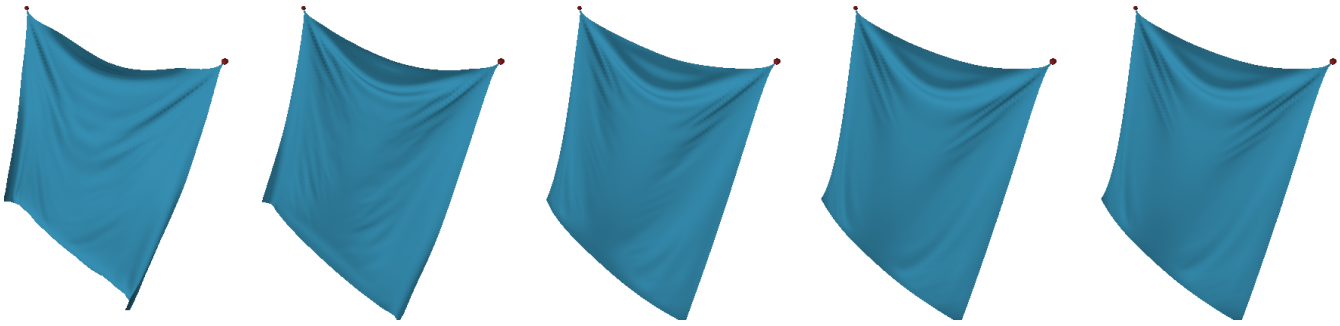


	Number of Iterations	Time	Relative Error
(a)	1	5.4 ms	$3.61 * 10^{-1}$
(b)	10	50.6 ms	$1.96 * 10^{-1}$
(c)	100	501 ms	$4.02 * 10^{-2}$
(d)	1000	5.05 s	$2.98 * 10^{-4}$

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 cloth-swinging animation at the depicted frame.*

exact solution is rarely required in physics-based animation. Indeed, previous methods [**?**] limit the number of iterations of Newton's method to one. To experimentally evaluate 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 (Fig. 3, please see also the accompanying video). Ten iterations seem to offer the best trade-off between speed and quality. In our example frame (Fig. 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 hundred or thousand iterations it is hard or even impossible to tell the difference from an exact solution.

Quick approximate simulation can be achieved also using Position Based Dynamics (PBD) [**?**]. 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 behavior 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 stiffness of the system. Our method does not suffer from this problem and converges to exact implicit Euler solution, as shown in



Our Method
1 iteration, 5 ms

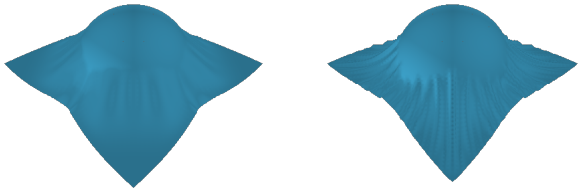
Our Method
10 iterations, 50 ms

Our Method
100 iterations, 500 ms

Our Method
1000 iterations, 5 s

Exact Solution
Newton's Method, 13 s

Figure 3: *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.*



Our Method
10 iteration, 50 ms

Exact Solution
Newton's Method, 13 s

Figure 4: *In challenging situations such as impact on collision our approximate solution results in loss of detailed wrinkles.*

the accompanying video.

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. with strong external forces due to collisions (Fig. 4), this causes obvious loss of detail. We implemented only a simple collision detection technique and our prototype currently does not address self-collisions. 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 [**?**].

We only consider mass-spring systems in this paper. In the future, we are planning to generalize our approach to thin shells, where the rest-length spring directions in equation (10) would be replaced by 2×2 SVD, which can be still computed in a closed form. We believe it will be also 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.