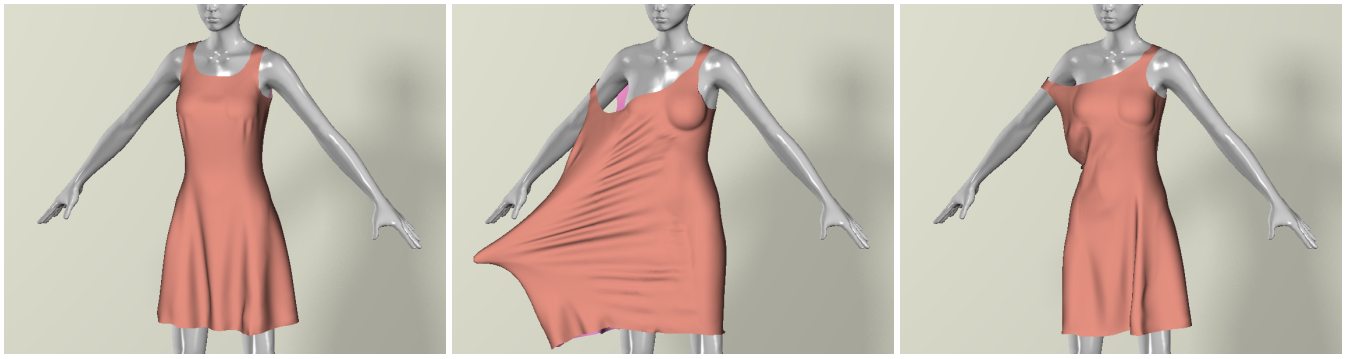
A Chebyshev Semi-Iterative Approach for Accelerating Projective and Position-based Dynamics

Huamin Wang*
The Ohio State University



(a) Before stretching (b) During stretching (c) After stretching

Figure 1: *The dress example. Using the Chebyshev semi-iterative approach to accelerate projective dynamics, our simulator animates this example with 16K vertices and 87K constraints at 37FPS. It uses a 1/30s time step and it handles large deformations, as shown in (b).*

Abstract

In this paper, we study the use of the Chebyshev semi-iterative approach in projective and position-based dynamics. Although projective dynamics is fundamentally nonlinear, its convergence behavior is similar to that of an iterative method solving a linear system. Because of that, we can estimate the "spectral radius" and use it in the Chebyshev approach to accelerate the convergence by at least one order of magnitude, when the global step is handled by the direct solver, the Jacobi solver, or even the Gauss-Seidel solver. Our experiment shows that the combination of the Chebyshev approach and the direct solver runs fastest on CPU, while the combination of the Chebyshev approach and the Jacobi solver outperforms any other combination on GPU, as it is highly compatible with parallel computing. Our experiment further shows position-based dynamics can be accelerated by the Chebyshev approach as well, although the effect is less obvious for tetrahedral meshes. The whole approach is simple, fast, effective, GPU-friendly, and has a small memory cost.

Keywords: Jacobi method, Chebyshev semi-iterative method, position-based dynamics, projective dynamics, parallel computing.

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

*e-mail: {whmin}@cse.ohio-state.edu

1 Introduction

Position-based dynamics is a popular technique for simulating deformable objects, and it has been widely used in many high-end physics engines, such as PhysX, Havok Cloth, and Maya nCloth, thanks to its simplicity. The basic idea behind position-based dynamics is to update vertex positions by enforcing position-based constraints iteratively, rather than by using elastic forces. When a vertex is involved in multiple constraints, its position can be updated either sequentially, known as the *Gauss-Seidel* way, or simultaneously through averaging, known as the *Jacobi* way. To implement position-based dynamics on GPU, the Jacobi way is often preferred so that the constraints can be processed in parallel. Position-based dynamics uses the number of iterations to control how strictly the constraints are enforced and how stiff an object behaves, so it is free of numerical instability. However, it is difficult to formulate the relationship between the mechanical properties of an object and the number of iterations. In fact, position-based dynamics can cause the stiffness behavior of an object to be affected the mesh resolution, since its convergence rate drops as the mesh resolution increases. To speed up the convergence, researchers proposed to enforce the constraints in a multi-resolution fashion [Müller 2008; Wang et al. 2010]. But building a mesh hierarchy is not simple and the result can be even more mesh-dependent.

A new constraint-based simulation technique, known as projective dynamics [Liu et al. 2013; Bouaziz et al. 2014] , emerged recently. Different from position-based dynamics, projective dynamics tries to exactly solve implicit time integration of a dynamical system, formulated under the variational form. Specifically, it iteratively runs two steps: a local step that projects each constraint into an acceptable state, and a global step that transfers these states to vertex positions. While projective dynamics can be considered as a generalized version of position-based dynamics, its converged result is physically plausible and controllable by stiffness variables. Since the linear system involved in the global step has a constant matrix, the original implementation of projective dynamics pre-factors the matrix and solve the global system directly by forward and backward substitutions. Previous research showed that doing this can

achieve visually acceptable results even within a few iterations. The catch is that a direct solver cannot be easily accelerated by GPU. So the whole method becomes less efficient, when more iterations are needed to reduce errors and artifacts of a fast deforming object.

Since projective dynamics and position-based dynamics are similar to linear systems in many ways, a natural question is: *can we borrow ideas from the existing linear system solvers and get these techniques accelerated?* Here we are especially interested in the Chebyshev semi-iterative method [Golub and Van Loan 1996]. Unlike generalized minimal residual or conjugate gradient, the Chebyshev method has a simple recurrence form and uses no inner product, so it is ideal for GPU acceleration. The challenge is that the Chebyshev method needs to know the range of the eigenvalues, which is hard to get in practice. If this range is under- or over-estimated, the method can converge slowly or even diverge. This problem becomes even more sophisticated, after we realize that projective and position-based dynamics are fundamentally nonlinear and there exists no matrix for eigenvalue analysis.

In this work, we demonstrate how the Chebyshev semi-iterative approach can be applied to accelerate both projective dynamics and position-based dynamics. Our contributions and conclusions are:

- **Analysis.** We noticed that the convergence of projective dynamics is highly similar to that of an iterative method solving a linear system, even when projective dynamics uses a direct solver. This is true to position-based dynamics as well, which can be considered as using a trivial global step. Based on these observations, we propose to estimate the "spectral radius" of projective or position-based dynamics from its convergence rate.
- **Implementation.** We show that the Chebyshev approach is easy to implement and compatible with GPU acceleration. Given an existing projective or position-based dynamics simulator, the approach can be implemented in less than five minutes! The Jacobi+Chebyshev combination further allows positional and contact constraints to be easily handled in each iteration. In contrast, the previous implementation [Bouaziz et al. 2014] requires updating the linear system.
- **Evaluation.** We tested our simulator using both triangular and tetrahedral meshes. Our experiment shows that the approach can accelerate projective dynamics by at least one order of magnitude, when the global step uses the direct solver, the Jacobi solver, or even the Gauss-Seidel solver. The Chebyshev approach can effectively accelerate position-based dynamics as well, especially for triangular meshes.

In summary, we present a simple, fast, and effective approach for accelerating projective and position-based dynamics, based on the Chebyshev semi-iterative method. This approach requires a small memory cost and it can handle large time steps and deformations, as shown in Figure 1. It is highly compatible with GPU acceleration and it can work together with other acceleration approaches as well, such as multi-resolution techniques.

2 Related Work

The simulation of deformable objects, such as cloth and soft tissues, has been an important research topic in computer graphics for decades, since the early work by Terzopoulos and colleagues [1987]. The key problem here is how to integrate elastic forces over time. To account for the nonlinearity of the elastic force in real-world deformable objects, researchers have developed various hyperelastic models to derive the force from the gradient of the elastic potential energy. Doing this makes explicit time integrators more vulnerable to numerical instability, especially when the stiffness and the time steps are large. Proposed by Baraff and Witkin [1998], one solution is to use an implicit Euler integrator, by linearizing the dynamical system into a linear system. However, implicit time integrators have the artificial damping issue. To overcome this issue, Kharevych and colleagues [2006] suggested the use of symplectic integrators, and Bridson and collaborators [2003] and Stern and Grinspun [2009] developed hybrid implicit-explicit integrators. Su and colleagues [2013] proposed to track and preserve the total energy explicitly.

Since numerical instability is related to the stiffness and many objects become stiffer under larger deformations, a natural way to address numerical instability is to prevent objects from large deformations by constraints, not forces. This idea was initially explored by Provost [1996] on mass-spring systems, and later extended to handle triangular and tetrahedral elements [Thomaszewski et al. 2009; Wang et al. 2010]. Müller and colleagues [2007; 2008] pushed this strain limiting idea even further, so they can replace elastic forces completely by constraints in dynamic simulation. This technique, known as position-based dynamics, was later used to simulate particle-based fluids [Macklin and Müller 2013; Macklin et al. 2014] and volumetric elements [Müller et al. 2014] as well. To simulate inextensible clothing by position-based dynamics, Kim and colleagues [2012] used unilateral distance constraints to connect cloth vertices with body vertices. Similar to position-based dynamics, the shape matching technique [Müller et al. 2005; Rivers and James 2007], replaces elastic forces by the difference between deformed shapes and goal shapes. If cloth is completely inextensible, it can also be simulated by constrained Lagrangian mechanics, as Goldenthal and collaborators [2007] showed.

One limitation of position-based dynamics is that it controls the stiffness of an object indirectly by the number of iterations, which can vary dramatically from one mesh to another. Under the optimization-based implicit Euler framework developed by Martin and colleagues [2011] for graphics purposes, Liu and collaborators [2013] discovered that implicit time integration of a mass-spring system can be approximated by iteratively solving a local constraint enforcement step and a global linear system step. Bouaziz and colleagues [2014] later formulated this idea into projective dynamics, and they studied its use in the simulation of triangular and tetrahedral elements. While projective dynamics can be treated as a generalized version of position-based dynamics, it has better physical meanings: the converged result is the same as the exact solution to a dynamical system under implicit Euler integration, which is independent of the mesh tessellation and the number of iterations.

3 Background

In this section, we will review some background knowledge about iterative solvers for linear systems, including the Chebyshev semi-iterative method. A linear system can be formulated as: $\mathbf{Ax} = \mathbf{b}$, in which $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a matrix, $\mathbf{b} \in \mathbb{R}^N$ is a given vector, and $\mathbf{x} \in \mathbb{R}^N$ is the unknown vector that needs to be found. When $\mathbf{A}$ is large and sparse, iterative methods are often favored over direct methods, to avoid matrices from being filled by new nonzeros during the solving process. Based on the splitting idea: $\mathbf{A} = \mathbf{B} - \mathbf{C}$, standard iterative methods, such as Jacobi and Gauss-Seidel, have the form:

$$\mathbf{x}^{(k+1)} = \mathbf{B}^{-1}\left(\mathbf{C}\mathbf{x}^{(k)} + \mathbf{b}\right). \tag{1}$$

It is straightforward to see that when these methods converge, they provide the solution to the linear system: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} = \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, since $\mathbf{Bx} = \mathbf{Cx} + \mathbf{b}$. If we split $\mathbf{B}^{-1}\mathbf{b}$ into $\mathbf{B}^{-1}\mathbf{Ax} = \mathbf{x} - \mathbf{B}^{-1}\mathbf{Cx}$, we can convert Equation 1 into:

$$\mathbf{e}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x} = \mathbf{B}^{-1}\mathbf{C}\left(\mathbf{x}^{(k)} - \mathbf{x}\right) = \mathbf{B}^{-1}\mathbf{C}\mathbf{e}^{(k)}, \tag{2}$$

in which $\mathbf{e}^{(k)}$ is the error vector at the k-th iteration. Let the eigenvalue decomposition of $\mathbf{B}^{-1}\mathbf{C}$ be $\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$, where $\mathbf{\Lambda}$ is the eigenvalue matrix. We can reformulate the error vector at the k-th iteration as:

$$\mathbf{e}^{(k)} = \left(\mathbf{B}^{-1}\mathbf{C}\right)^k \mathbf{e}^{(0)} = \mathbf{Q}\mathbf{\Lambda}^k\mathbf{Q}^{-1}\mathbf{e}^{(0)}. \tag{3}$$

This means that these iterative methods converge linearly and their convergence rates depend on the largest eigenvalue magnitude, known as the *spectral radius*: $\rho(\mathbf{B}^{-1}\mathbf{C})$. To ensure the convergence, we must have: $\rho(\mathbf{B}^{-1}\mathbf{C}) < 1$.

3.1 The Chebyshev Semi-Iterative Method

Given the results produced by the iterative formula in Equation 1: $\mathbf{x}^{(0)}$, $\mathbf{x}^{(1)}$, ..., $\mathbf{x}^{(k)}$, we would like to obtain a better result from their linear combinations, which has the following form:

$$\mathbf{y}^{(k)} = \sum_{j=0}^{k} \nu_{j,k}\mathbf{x}^{(j)}, \tag{4}$$

in which $\nu_{j,k}$ are the blending coefficients to be determined. If the results are good already: $\mathbf{x}^{(0)} = \mathbf{x}^{(1)} = ... = \mathbf{x}^{(k)} = \mathbf{x}$, we must have $\mathbf{y}^{(k)} = \mathbf{x}$. So we require the following constraint:

$$\sum_{j=0}^{k} \nu_{j,k} = 1. \tag{5}$$

The question is how to reduce the error of $\mathbf{y}^{(k)}$. Using Equation 5 and 2, we can formulate the error $\mathbf{y}^{(k)} - \mathbf{x}$ into:

$$\sum_{j=0}^{k} \nu_{j,k}\left(\mathbf{x}^{(j)} - \mathbf{x}\right) = \sum_{j=0}^{k} \nu_{j,k}\left(\mathbf{B}^{-1}\mathbf{C}\right)^j \mathbf{e}^{(0)} = p_k(\mathbf{B}^{-1}\mathbf{C})\mathbf{e}^{(0)}, \tag{6}$$

in which $p_k(x) = \sum_{j=0}^{k} \nu_{j,k}x^j$ is a polynomial function. So to reduce the error, we must reduce $\left\|p_k(\mathbf{B}^{-1}\mathbf{C})\right\|_2 = \max_{\lambda_i} |p_k(\lambda_i)|$, in which λ_i can be any eigenvalue of $\mathbf{B}^{-1}\mathbf{C}$. Suppose that all of the eigenvalues are real. If we know all of the eigenvalues and if k is sufficiently large, we can construct the polynomial function in a way that $p_k(\lambda_i) = 0$ for any λ_i. Unfortunately, it is difficult to know the eigenvalues, when the linear system is large and varying. Instead, if we know the spectral radius ρ such that $-1 < -\rho \le \lambda_n \le ... \le \lambda_1 \le \rho < 1$, we can ask $p_k(x)$ to be minimized for all $x \in [-\rho, \rho]$:

$$p_k(x) = \arg\min\left\{\max_{-\rho \le x \le \rho} |p_k(x)|\right\}. \tag{7}$$

The unique solution to Equation 7 is given by:

$$p_k(x) = \frac{C_k(x/\rho)}{C_k(1/\rho)}, \tag{8}$$

in which $C_k(x)$ is the Chebyshev polynomial with the recurrence relation: $C_{k+1}(x) = 2xC_k(x) - C_{k-1}(x)$, with $C_0(x) = 1$ and $C_1(x) = x$. It is trivial to see that $p_k(1) = 1$, satisfying Equation 5. For any $x \in [-1, 1]$, $|C_k(x)| \le 1$, but for any $x \notin [-1, 1]$, $|C_k(x)|$ grows rapidly when $k \to \infty$. So $p_k(x)$ diminishes quickly for any $x \in [-\rho, \rho]$, when $k \to \infty$.

To reduce the computational and memory cost, we can avoid calculating $\mathbf{y}^{(k)}$ by its definition in Equation 4. Instead, we use Equation 8 to formulate the recurrence relation of $p_k(x)$ as:

$$\begin{aligned} p_{k+1}(x)C_{k+1}\left(\tfrac{1}{\rho}\right) &= \tfrac{2x}{\rho}C_k\left(\tfrac{x}{\rho}\right) - C_{k-1}\left(\tfrac{x}{\rho}\right) \\ &= \tfrac{2x}{\rho}p_k(x)C_k\left(\tfrac{1}{\rho}\right) - p_{k-1}(x)\left(\tfrac{2}{\rho}C_k\left(\tfrac{1}{\rho}\right) - C_{k+1}\left(\tfrac{1}{\rho}\right)\right), \end{aligned} \tag{9}$$

which can be reorganized into:

$$C_{k+1}\left(\tfrac{1}{\rho}\right)(p_{k+1}(x) - p_{k-1}(x)) = \tfrac{2C_k\left(\frac{1}{\rho}\right)}{\rho}(xp_k(x) - p_{k-1}(x)). \tag{10}$$

After replacing x by $\mathbf{B}^{-1}\mathbf{C}$ and multiplying both sides of Equation 10 by $\mathbf{e}^{(0)}$, we get:

$$C_{k+1}\left(\tfrac{1}{\rho}\right)\left(\mathbf{y}^{(k+1)} - \mathbf{y}^{(k-1)}\right) = \tfrac{2C_k\left(\frac{1}{\rho}\right)}{\rho}\left(\mathbf{B}^{-1}\mathbf{C}\left(\mathbf{y}^{(k)} - \mathbf{x}\right) - \mathbf{y}^{(k-1)} + \mathbf{x}\right). \tag{11}$$

Using the fact that $-\mathbf{B}^{-1}\mathbf{C}\mathbf{x} + \mathbf{x} = \mathbf{B}^{-1}(\mathbf{B} - \mathbf{C})\mathbf{x} = \mathbf{B}^{-1}\mathbf{b}$, we can now obtain the following update function:

$$\mathbf{y}^{(k+1)} = \omega_{k+1}\left(\mathbf{B}^{-1}\left(\mathbf{C}\mathbf{y}^{(k)} + \mathbf{b}\right) - \mathbf{y}^{(k-1)}\right) + \mathbf{y}^{(k-1)}, \tag{12}$$

where

$$\omega_{k+1} = \frac{2C_k\left(\frac{1}{\rho}\right)}{\rho C_{k+1}\left(\frac{1}{\rho}\right)}, \quad i \ge 1, \quad \omega_1 = 1. \tag{13}$$

In Equation 12, $\mathbf{B}^{-1}\left(\mathbf{C}\mathbf{y}^{(k)} + \mathbf{b}\right)$ is essentially one iterative solve step described in Equation 1. By definition, we can further get:

$$\omega_{k+1} = \frac{2C_k}{\rho\left(\frac{2}{\rho}C_k - C_{k-1}\right)} = \frac{2C_k}{\rho\left(\frac{2}{\rho}C_k - \frac{\rho\omega_k C_k}{2}\right)} = \frac{4}{4 - \rho^2\omega_k}. \tag{14}$$

This allows us to more efficiently compute ω, given the initial conditions $\omega_1 = 1$ and $\omega_2 = \frac{2}{2-\rho^2}$.

Golub and Varga [1961] extensively analyzed the Chebyshev semi-iterative method. They pointed out that although the Chebyshev method looks similar to weighted Jacobi and successive over-relaxation (SOR), it converges much faster. This is because the Chebyshev method changes the factor ω in each iteration and it uses $\mathbf{y}^{(k-1)}$, not $\mathbf{y}^{(k)}$.

Real eigenvalues. The previous analysis is based on the assumption that all eigenvalues of $\mathbf{B}^{-1}\mathbf{C}$ are real. Although this is not true in general, if $\mathbf{A}$ is a symmetric matrix with positive diagonal entries and $\mathbf{B}^{-1}\mathbf{C}$ is created by Jacobi, the eigenvalues must be real. Let λ and $\mathbf{v}$ be an eigenvalue and its eigenvector of $\mathbf{B}^{-1}\mathbf{C}$: $\mathbf{B}^{-1}\mathbf{C}\mathbf{v} = \lambda\mathbf{v}$. We have $\mathbf{B}^{-\frac{1}{2}}\mathbf{C}\mathbf{B}^{-\frac{1}{2}}(\mathbf{B}^{\frac{1}{2}}\mathbf{v}) = \mathbf{B}^{\frac{1}{2}}\lambda\mathbf{B}^{-\frac{1}{2}}(\mathbf{B}^{\frac{1}{2}}\mathbf{v}) = \lambda(\mathbf{B}^{\frac{1}{2}}\mathbf{v})$. So λ is also the eigenvalue of $\mathbf{B}^{-\frac{1}{2}}\mathbf{C}\mathbf{B}^{-\frac{1}{2}}$. Since $\mathbf{B}$ is the diagonal matrix of $\mathbf{A}$ and its diagonal elements are all positive, $\mathbf{B}^{-\frac{1}{2}}$ must be real. So $\mathbf{B}^{-\frac{1}{2}}\mathbf{C}\mathbf{B}^{-\frac{1}{2}}$ is real symmetric and λ is real. Therefore, we can use the Chebyshev method to accelerate Jacobi iterations immediately.

Strengths and weaknesses. The biggest advantage of the Chebyshev semi-iterative method is its simplicity. Compared with Krylov subspace iterative methods, such as generalized minimum residual and conjugate gradient, the Chebyshev method has a short recurrence form and it does not use inner products, so it is ideal for parallel computing. Unfortunately, it is known that the Chebyshev method does not converge as fast as Krylov subspace methods do.

The Chebyshev method also requires[1] the spectral radius ρ, which is often estimated numerically. Let $\hat{\rho}$ be the estimated spectral radius. The Chebyshev method converges differently when $\hat{\rho}$ varies from 0 to 1. When $\hat{\rho} = 0$, the method is reduced to a standard iterative solver and it offers no acceleration. When $\hat{\rho}$ varies from 0 to ρ, the convergence rate is gradually improved. Once $\hat{\rho}$ grows above ρ, the method still converges but oscillation happens. Eventually, the method diverges as $\hat{\rho}$ becomes close to 1. We will explore this convergence property for estimating ρ later in Subsection 5.1.

[1] If we know an even tighter range of the eigenvalues: $\lambda_i \in [\alpha, \beta]$, the Chebyshev method can be adjusted to converge faster, as described in [Golub and Van Loan 1996; Gutknecht and Röllin 2002].

4 Projective Dynamics

Given a 3D dynamical system with N vertices, we can simulate its movement from the t-th time instant to the t+1-th time instant by implicit time integration [Baraff and Witkin 1998]:

$$\mathbf{q}_{t+1} = \mathbf{q}_t + h\mathbf{v}_{t+1}, \quad \mathbf{v}_{t+1} = \mathbf{v}_t + h\mathbf{M}^{-1}\mathbf{f}_{t+1}, \tag{15}$$

where $\mathbf{q} \in \mathbb{R}^{3N}$ and $\mathbf{v} \in \mathbb{R}^{3N}$ are stacked position and velocity vectors, $\mathbf{M} \in \mathbb{R}^{3N\times 3N}$ is the mass matrix, h is the time step, and $\mathbf{f} \in \mathbb{R}^{3N}$ is the total force. Assuming that $\mathbf{f}_{t+1} = \mathbf{f}_{\text{int}}(\mathbf{q}_{t+1}) + \mathbf{f}_{\text{ext}}$, in which the internal elastic force $\mathbf{f}_{\text{int}}$ is a function of $\mathbf{q}$ and the external force $\mathbf{f}_{\text{ext}}$ is constant, we reformulate Equation 15 into a nonlinear system:

$$\mathbf{M}\left(\mathbf{q}_{t+1} - \mathbf{q}_t - h\mathbf{v}_t\right) = h^2\left(\mathbf{f}_{\text{int}}(\mathbf{q}_{t+1}) + \mathbf{f}_{\text{ext}}\right). \tag{16}$$

One solution to Equation 16 is to use the Newton's method:

$$\mathbf{M}\left(\mathbf{q}^{(k+1)} - \mathbf{q}_t - h\mathbf{v}_t\right) = h^2\left(\mathbf{f}_{\text{int}}(\mathbf{q}^{(k)}) + \mathbf{K}(\mathbf{q}^{(k)})(\mathbf{q}^{(k+1)} - \mathbf{q}^{(k)}) + \mathbf{f}_{\text{ext}}\right), \tag{17}$$

in which $\mathbf{q}^{(0)} = \mathbf{q}_t$ and $\mathbf{K}(\mathbf{q}^{(k)}) = \partial\mathbf{f}_{\text{int}}/\partial\mathbf{x}$ is the stiffness matrix evaluated at $\mathbf{q}^{(k)}$. Although the Newton's method converges fast, its iterations are computationally expensive and its overall performance is often far from satisfactory. If we just use a single iteration as proposed by Baraff and Witkin [1998], the result will not be accurate, but at least visually plausible for some applications.

A different way of handling implicit time integration is to convert it into an energy minimization problem:

$$\mathbf{q}_{t+1} = \arg\min_{\mathbf{q}} \epsilon(\mathbf{q}) = \arg\min_{\mathbf{q}} \frac{1}{2h^2}\|\mathbf{M}^{\frac{1}{2}}(\mathbf{q} - \mathbf{s}_t)\|^2 + E(\mathbf{q}), \tag{18}$$

in which $E(\mathbf{q})$ is the internal potential energy at $\mathbf{q}$, and $\mathbf{s}_t = \mathbf{q}_t + h\mathbf{v}_t + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$ is the expected position vector without internal forces. It is straightforward to see that the critical point of Equation 18 is identical to Equation 16. In other words, solving Equation 18 is equivalent to solving Equation 16. Projective dynamics [Liu et al. 2013; Bouaziz et al. 2014] assumes that the internal energy is the sum of the constraint energies and each constraint energy has a quadratic form:

$$\epsilon(\mathbf{q}) = \frac{1}{2h^2}\|\mathbf{M}^{\frac{1}{2}}(\mathbf{q} - \mathbf{s}_t)\|^2 + \sum\nolimits_c \frac{w_c}{2}\left\|\mathbf{A}_c\mathbf{q} - \hat{\mathbf{A}}_c\mathbf{p}_c\right\|^2, \tag{19}$$

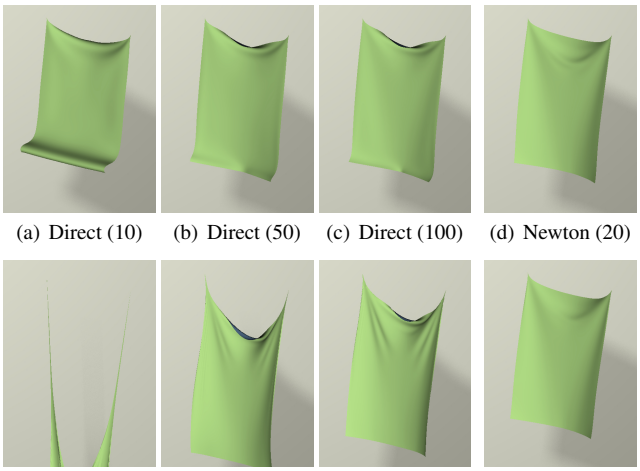
where w_c is a positive weight of constraint c, $\mathbf{p}_c$ is the projection of $\mathbf{q}$ into the energy-free space defined by c, and $\mathbf{A}_c$ and $\hat{\mathbf{A}}_c$ are two constant matrices specifying the relationship between $\mathbf{q}$ and $\mathbf{p}_c$. For example, if c is a spring, $\mathbf{A}_c\mathbf{q}$ and $\hat{\mathbf{A}}_c\mathbf{p}_c$ give the displacement vectors of the two vertices, and the internal energy is equivalent to the spring potential energy. Based on this model, projective dynamics iteratively minimizes $\epsilon(\mathbf{q})$ in two steps: in a local step, it projects the current $\mathbf{q}$ into $\mathbf{p}_c$ for every constraint c; in a global step, it treats $\mathbf{p}_c$ as constant and finds the next $\mathbf{q}$ that minimizes $\epsilon(\mathbf{q})$. Liu and colleagues [2013] demonstrates that projective dynamics using spring constraints is equivalent to an implicit mass-spring system.

4.1 The Global Solve

An important question regarding projective dynamics is how to solve the global step. Liu and colleagues [2013] and Bouaziz and collaborators [2014] found that the linear system resulted from $\nabla\epsilon(\mathbf{q}) = 0$ has a constant matrix $\mathbf{A}$:

$$\mathbf{A}\mathbf{q} = \left(\frac{\mathbf{M}}{h^2} + \sum_c w_c\mathbf{A}_c^{\mathsf{T}}\mathbf{A}_c\right)\mathbf{q} = \frac{\mathbf{M}}{h^2}\mathbf{s}_t + \sum_c w_c\mathbf{A}_c^{\mathsf{T}}\hat{\mathbf{A}}_c\mathbf{p}_c, \tag{20}$$

which means $\mathbf{A}$ can be pre-factored for fast direct solve in every iteration. If each time step contains only a small number of iterations, we can solve projective dynamics efficiently in real time. However, a few iterations are often insufficient for the local step to stabilize its results, when an object undergoes large and fast deformation. This issue is especially noticeable in mass-spring systems, where the displacement vectors change dramatically within one time step. For example, Figure 2a shows that 10 iterations are insufficient to remove the curly edge artifact near the bottom of a swinging tablecloth, which does not exist in the ground truth provided by the Newton's method in Figure 2d. This artifact becomes less noticeable, but still exists after we increase the number of iterations from 10 to 100, as Figure 2c shows. Unfortunately, we cannot afford running the direct solve too many times, since it is still relatively expensive and it cannot be easily accelerated by GPU.



(a) Direct (10) (b) Direct (50) (c) Direct (100) (d) Newton (20)

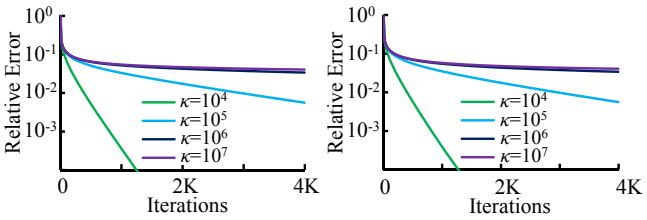
(e) Jacobi (400) (f) Jacobi (4000) (g) OR (4000) (h) Ours (400)

Figure 2: *Projective dynamics results produced by using different solvers. The numbers in the brackets are the numbers of iterations. The direct solver can still cause curly edge artifacts after 100 iterations, as (c) shows. Meanwhile, the Jacobi solver converges slowly, even when it is accelerated by over-relaxations in (g). In contrast, the result generated by our Jacobi+Chebyshev method after 400 iterations in (h) is highly similar to the ground truth, which is simulated by the Newton's method shown in (d).*

From a different perspective, we may consider the cause of the aforementioned issue to be the unneeded accuracy of the direct solver. Without rounding errors, the direct solver finds the exact solution to the linear system in Equation 20. But such accuracy is unnecessary and the computational cost is wasted, once the local step modifies the linear system in the next iteration. One idea is to replace the direct solver by one Jacobi iteration, which is cheap and compatible with GPU acceleration. But projective dynamics will converge slowly as shown in Figure 2e and 2f. The over-relaxation method cannot make much difference, as Figure 2g shows. So to achieve the same accuracy, the Jacobi solver often ends up with spending even more computational time.

4.2 Position-based Dynamics

Liu and colleagues [2013] and Bouaziz and collaborators [2014] pointed out that the original implementation of position-based dynamics [Müller et al. 2007] can be considered as a simplified version of projective dynamics, by setting $\mathbf{A}_c^{\mathsf{T}}\mathbf{A}_c = \mathbf{A}_c^{\mathsf{T}}\hat{\mathbf{A}}_c = \mathbf{I}$, ignoring the first term in $\epsilon(\mathbf{q})$, and using one Gauss-Seidel iteration to solve the global step. If we replace the Gauss-Seidel iteration by a Jacobi iteration, the local step calculates the expected positions of a vertex according to the constraints, and the global step uses their weighted average to update the vertex position. This is basically equivalent



(a) The Jacobi method (b) Projective dynamics

Figure 3: *The convergence behaviors of the Jacobi method and projective dynamics, without Chebyshev acceleration. This example shows that the convergence of projective dynamics behaves similarly to that of the Jacobi method solving the linear system in one global step. Here κ is the stiffness variable controlling the weight w_c in Equation 20. The error in (a) is the error magnitude of the linear system, and the error in (b) is the error magnitude of the dynamical system in Equation 16.*

to the Jacobi implementation of position-based dynamics, proposed by Macklin and colleagues [2014]. In general, position-based dynamics converges slower than projective dynamics, even when both of them use Jacobi iterations. This is because projective dynamics can use non-diagonal entries in $\mathbf{A}_c$ and $\hat{\mathbf{A}}_c$ to propagate the influence of the constraints faster.

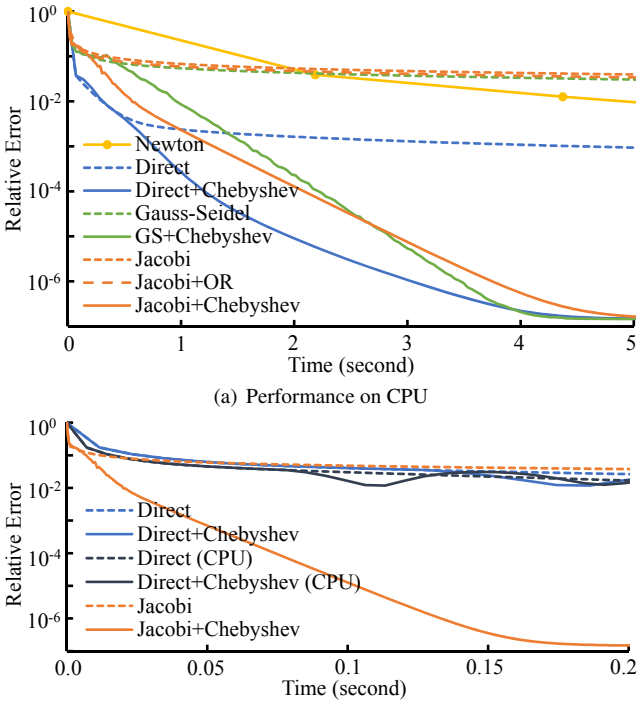
5 The Chebyshev Approach

In this section, we will investigate the extension of the Chebyshev method from linear systems to projective dynamics. We will also discuss the use of the Chebyshev approach in position-based dynamics in Subsection 5.3.

5.1 Chebyshev for Projective Dynamics

Let us first reconsider the idea of replacing one direct solve by one Jacobi iteration in each global step of projective dynamics, as proposed in Subsection 4.1. In that case, each projective dynamics iteration contains a local step and a Jacobi iteration. Since the result of the local step is typically stabilized within a few iterations, the linear system in the global step is almost unchanged afterwards. So it is not surprising to see that the convergence of projective dynamics is similar to that of the Jacobi method solving the linear system in a single global step, as shown in Figure 3. Based on this observation, we propose a Chebyshev semi-iterative approach for projective dynamics, which simply replaces one Jacobi iteration in Equation 12 by one projective dynamics iteration. Figure 4a shows this approach effectively accelerates the convergence of projective dynamics, when the global step is solved by one Jacobi iteration.

Interestingly, Figure 4a shows the approach also works when the global step is solved by the direct method or the Gauss-Seidel method. We believe this is because projective dynamics converges in similar patterns, regardless of the methods used to solve the global step. Note that divergence can occur when using the Chebyshev method to immediately accelerate Gauss-Seidel iterations, because the iterative matrix $\mathbf{B}^{-1}\mathbf{C}$ may have complex eigenvalues as Golub and Van Loan [1996] pointed out. We found that the same issue happens to projective dynamics, when the global step is solved by one Gauss-Seidel iteration. Our solution is to solve the global step by two Gauss-Seidel iterations, once in the forward order and once in the backward order. By doing so, the joint iterative matrix has real eigenvalues and our Chebyshev approach is effective again.



(a) Performance on CPU

(b) Performance on GPU

Figure 4: *The performance of projective dynamics on the tablecloth example, using different methods. The Chebyshev approach can accelerate projective dynamics that uses any of the three methods: the direct method, the Gauss-Seidel method, or the Jacobi method. Among them, the Jacobi method benefits most from the Chebyshev approach, since it is more compatible with GPU acceleration as shown in (b).*

The choice of ρ. The coefficient ρ was originally defined in Section 3.1 as the spectral radius of the iterative matrix $\mathbf{B}^{-1}\mathbf{C}$, when we use the Chebyshev method to solve a linear system. But for projective dynamics, we cannot define ρ in this way, since the combination of the local step and the global step is fundamentally nonlinear. One may think that ρ is related to the iterative method solving the linear system in the global step. But that cannot explain why the Chebyshev approach still works when the global step is solved by the direct method.

Fortunately, due to the similarity between the convergence of projective dynamics and that of a linear solver, we treat ρ as constant for each simulation problem and we estimate ρ by pre-simulation in two steps. Let K be the total number of projective dynamics iterations. We first initialize ρ by $\left\|\mathbf{e}^{(K)}\right\|_2 / \left\|\mathbf{e}^{(K-1)}\right\|_2$, where the error is defined[2] as $\mathbf{e}^{(k)} = \nabla\epsilon(\mathbf{q}^{(k)})$. This is similar to how the power method estimates the spectral radius for a standard iterative method. After that, we manually adjust ρ and run the simulation by the Chebyshev approach multiple times, to find the optimal ρ that maximizes the convergence rate. Similar to the Chebyshev method for linear systems, our Chebyshev approach causes oscillation when ρ is overestimated. So if oscillation occurs, we know ρ is too large and we reduce it accordingly.

We can safely assume that ρ is constant through the whole simulation process, since the choice of ρ is insensitive to the simulation state $\mathbf{q}$ as shown in all of our simulation examples. In fact,

[2] Our error definition is different from the one used in [Bouaziz et al. 2014], which measures the energy difference from the ground truth $\mathbf{q}^*$: $\epsilon(\mathbf{q}) - \epsilon(\mathbf{q}^*)$. We will use our definition in the rest of this paper.

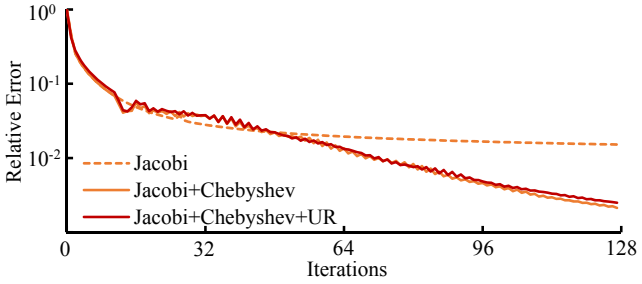
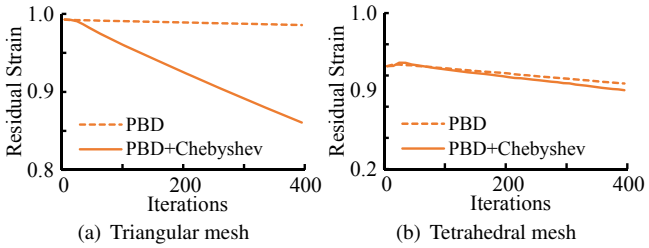


Figure 5: *The effect of under-relaxation (UR) on the armadillo example. This plot shows that under-relaxation has limited influence on the convergence of the Chebyshev approach, when $\gamma = 0.9$.*



(a) Triangular mesh (b) Tetrahedral mesh

Figure 6: *The convergence of position-based dynamics, using Jacobi iterations. This example shows that the acceleration effect of the Chebyshev approach is more significant on triangular meshes as shown in (a) than on tetrahedral meshes as shown in (b).*

ρ is relatively insensitive to small changes made to the system as well. These changes are typically needed by positional or collision constraints, or local remeshing processes. Therefore, we can make these changes immediately during simulation, without changing ρ. Our experiment does demonstrate a strong relationship between ρ and the stiffness of an object. When the stiffness increases, projective dynamics converges more slowly as shown in Figure 3b. So ρ needs to be larger to make the Chebyshev approach more effective. Our experiment also reveals that ρ depends on the total number of iterations K. Projective dynamics typically converges fast in the first few iterations. After that, the convergence rate gradually drops as Figure 4 shows. So we must make ρ larger, when K increases.

Performance evaluation. Figure 4 compares the performance of projective dynamics using different methods. The Newton's method converges fastest, but its iterations are too expensive to make its overall performance attractive, as Figure 4a shows. On CPU, projective dynamics using the direct method runs fastest. The use of Gauss-Seidel iterations allows projective dynamics to run slightly faster than the use of Jacobi iterations as expected, and over-relaxation (OR) has little effect on the performance. The Chebyshev approach accelerates projective dynamics that uses any of the three methods, among which the direct+Chebyshev method is the fastest.

On GPU, the Jacobi+Chebyshev method outperforms any other method as Figure 4b shows, since it is naturally compatible with parallel computing. In contrast, forward and backward substitutions involved in the direct method are more expensive to run on GPU than on CPU, as they are largely sequential. To implement the direct method, we use the eigen library on CPU and the cuSOLVER library on GPU. The pre-computed factorization is done by Cholesky decomposition with permutation. The GPU implementation of the direct method was tested on NVIDIA Tesla K40c.

5.2 Convergence and Robustness

It is uncommon to see divergence caused solely by the Chebyshev approach in practice, since we would notice oscillations first during the parameter tuning process and we can always lower ρ to fix the issue. However, there is one small problem due to the use of a constant ρ. Projective dynamics typically converges fast in the first few iterations, and its convergence rate drops after that. This means when the total number of iterations is large, ρ must be smaller than it should be, to avoid oscillation or even divergence in the first few iterations. To address this problem, we simply delay the start of the Chebyshev approach, by setting $\omega_1 = \omega_2 = ... = \omega_S = 1$. In our experiment, we typically use $S = 10$ for the Jacobi+Chebyshev method. A more suitable strategy is to use different ρ for different iterations, but doing so makes ρ difficult to estimate and we need to study this further in the future.

To ensure the convergence of the Jacobi method, we must have: $\rho(\mathbf{B}^{-1}\mathbf{C}) < 1$. This is automatically satisfied by spring constraints, since their matrix $\mathbf{A}$ is diagonally dominant. Unfortunately, this may not be true for other constraints, and the Jacobi method will not be able to solve the linear system involved in the global step. Interestingly, this does not immediately lead to the failure of projective dynamics, when it solves the global step by only one Jacobi iteration. In fact, our experiment shows the divergence issue occurs, only when the mesh contains small elements or experiences very large stress. We also found that the divergence issue can be reduced by inserting an under-relaxation step before the Chebyshev update:

$$\mathbf{q}^{(k+1)} = \omega_{k+1}\left(\gamma\left(\hat{\mathbf{q}}^{(k+1)} - \mathbf{q}^{(k)}\right) + \mathbf{q}^{(k)} - \mathbf{q}^{(k-1)}\right) + \mathbf{q}^{(k-1)}, \qquad (21)$$

in which $\hat{\mathbf{q}}^{(k+1)}$ is the result produced by one Jacobi iteration and $\gamma \in [0.6, 0.9]$ is the under-relaxation coefficient. This under-relaxation step has little influence on the convergence rate as shown in Figure 5. Note that under-relaxation cannot fully eliminate the divergence issue. An ultimate solution is to make the matrix diagonally dominant by using a smaller time step, but that increases the computational cost of the whole system. It is possible that strain limiting can address this issue, by preventing elements from being significantly deformed.

5.3 Chebyshev for Position-based Dynamics

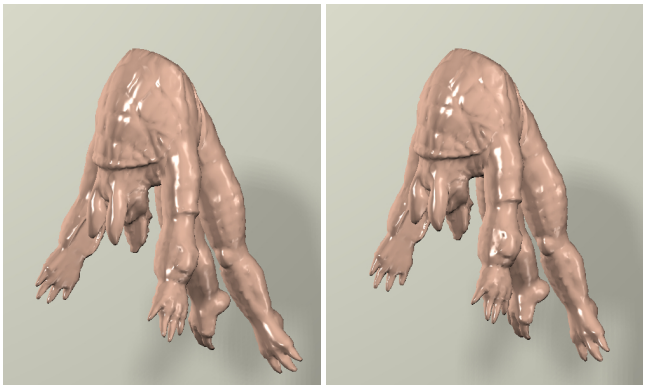
Since position-based dynamics can be considered as a simplified version of projective dynamics as shown in Subsection 4.2, one may think that it can be naturally accelerated by the Chebyshev approach as well. Our research shows the actual situation is complicated.

Let i be a vertex and $\Delta\mathbf{q}_{i,c}$ be its position movement suggested by constraint c. We calculate its actual position update in each iteration by a relaxation coefficient α:

$$\mathbf{q}_i^{(k+1)} = \mathbf{q}_i^{(k)} + \alpha \sum_c \Delta\mathbf{q}_{i,c}. \qquad (22)$$

We prefer using Equation 22 over the averaging scheme proposed by Macklin and colleagues [2014], since it is momentum preserving and slightly more robust against divergence. Given the same α, if no divergence occurs, using the Chebyshev approach always results in faster convergence than not using the Chebyshev approach, as our experiment shows. The problem is that the Chebyshev approach makes the algorithm more vulnerable to divergence, so smaller α is needed for better stability. When simulating triangular meshes, we found the difference in α is not significant: $\alpha = 0.3$ for not using the Chebyshev approach[3], and $\alpha = 0.25$ for using the Chebyshev approach. So the acceleration provided by the Chebyshev approach is still obvious, as Figure 6a shows. But when simulating tetrahedral

[3] If we assume that a vertex has six neighbors, $\alpha = 0.3$ is equivalent to setting the over-relaxation coefficient to 1.8 in [Macklin et al. 2014].



(a) Before the Chebyshev approach (b) After the Chebyshev approach

Figure 7: *The armadillo example simulated by position-based dynamics. In this example, position-based dynamics is more unstable after it gets accelerated by the Chebyshev approach. To avoid divergence, a smaller under-relaxation coefficient must be used and the resulting acceleration becomes less noticeable.*

Algorithm 1 `Chebyshev_PD_Solve`($\mathbf{q}_t$, $\mathbf{v}_t$, ρ, h, K)

$\mathbf{q}^{(0)} \leftarrow \mathbf{s}_t \leftarrow \mathbf{q}_t + h\mathbf{v}_t + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$;
for $k = 0...K-1$ **do**
  for each constraint c **do**
    $\mathbf{p}_c$ = `Local_Solve`$(c, \mathbf{q}^{(k)})$;
  $\hat{\mathbf{q}}^{(k+1)} \leftarrow$ `Jacobi_Solve`$(\mathbf{s}_t, \mathbf{q}^{(k)}, \mathbf{p}_1, \mathbf{p}_2, ...)$;
  if $k < S$ **then** $\omega_{k+1} \leftarrow 1$;
  if $k = S$ **then** $\omega_{k+1} \leftarrow 2/(2-\rho^2)$;
  if $k > S$ **then** $\omega_{k+1} \leftarrow 4/(4-\rho^2\omega_k)$;
  $\mathbf{q}^{(k+1)} = \omega_{k+1}\left(\gamma\left(\hat{\mathbf{q}}^{(k+1)} - \mathbf{q}^{(k)}\right) + \mathbf{q}^{(k)} - \mathbf{q}^{(k-1)}\right) + \mathbf{q}^{(k-1)}$;
$\mathbf{q}_{t+1} \leftarrow \mathbf{q}^{(K)}$;
$\mathbf{v}_{t+1} \leftarrow (\mathbf{q}_{t+1} - \mathbf{q}_t)/h$;

meshes, α must be much smaller to avoid divergence: $\alpha = 0.045$ for not using the Chebyshev approach and $\alpha = 0.0025$ for using the Chebyshev approach. Overall, the simulation is still accelerated, but the effect is less evident as shown in Figure 6b and 7.

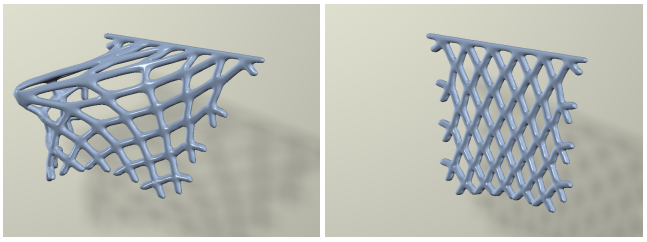
We believe these phenomena are largely due to the fact that the linear system in the global step is trivially solved in an iteration, when position-based dynamics assumes $\mathbf{A}_c^\mathsf{T}\mathbf{A}_c = \mathbf{A}_c^\mathsf{T}\hat{\mathbf{A}}_c = \mathbf{I}$. As a result, the convergence of position-based dynamics does not behave so closely to the convergence of an iterative method, and it cannot always benefit a lot from the Chebyshev approach.

6 Implementation

The pseudo code of our system is given in Algorithm 1.

Constraints and their energies. We use two types of constraints to simulate triangular meshes: the spring constraint for every edge, and the hinge-edge constraint for every edge adjacent to two triangles. We define the energy of a spring constraint by the spring potential energy and the energy of a hinge-edge constraint by the quadratic bending energy [Bergou et al. 2006], assuming that the mesh is initially flat. Since the bending energy is a quadratic function of $\mathbf{q}$, its energy Hessian matrix can be directly inserted into the linear system and it needs no local step.

We use tetrahedral constraints to simulate tetrahedral meshes and we define the elastic energy of a tetrahedron as: $\kappa \text{vol}(c)\,\|\mathbf{F}_c - \mathbf{R}_c\|_\text{F}^2$,



(a) Deformed shape (b) Recovered shape

Figure 8: *The fishnet example. Our simulator can robustly recover the rest shape of an elastic fishnet in (b), after it got deformed by user interaction shown in (a).*

where κ is the stiffness, $\text{vol}(c)$ is the volume before deformation, $\mathbf{F}_c$ is the deformation gradient, and $\mathbf{R}_c$ is the rotational component of $\mathbf{F}_c$. To implement this energy in projective dynamics, we define $\mathbf{A}_c$ as the matrix that converts $\mathbf{q}$ into the deformation gradient $\mathbf{F}_c$ in the Voigt form. We directly formulate $\hat{\mathbf{A}}_c\mathbf{p}_c$ as the Voigt form of $\mathbf{R}_c$, so we do not define $\hat{\mathbf{A}}_c$ explicitly. Let $\mathbf{F}_c = \mathbf{R}_c\mathbf{Q}_c\mathbf{\Lambda}_c\mathbf{Q}_c^\mathsf{T}$ be the decomposition of $\mathbf{F}_c$, such that $\mathbf{Q}_c$ is an orthogonal matrix and $\mathbf{\Lambda}_c$ is a diagonal matrix containing the principal stretches λ_1, λ_2, and λ_3. The elastic energy density function is proportional to:

$$\left\|\mathbf{A}_c\mathbf{q} - \hat{\mathbf{A}}_c\mathbf{p}_c\right\|^2 = \|\mathbf{F}_c - \mathbf{R}_c\|_\text{F}^2 = \|\mathbf{\Lambda}_c - \mathbf{I}\|_\text{F}^2 = \sum_i (\lambda_i - 1)^2. \quad (23)$$

Our experiment shows that this hyperelastic model is sufficient to produce many interesting elastic behaviors.

Polar decomposition. To simulate tetrahedral meshes, we need polar decomposition to extract the rotational component $\mathbf{R}$ from the deformation gradient $\mathbf{F}$: $\mathbf{F} = \mathbf{RS}$, in which $\mathbf{S}$ is a symmetric matrix known as the *stretch tensor*. A typical way of obtaining $\mathbf{R}$ is to perform singular value decomposition (SVD) on $\mathbf{F} = \mathbf{U\Sigma V}^\mathsf{T}$ and calculate $\mathbf{R}$ as: $\mathbf{R} = \mathbf{UV}^\mathsf{T}$. Unfortunately, SVD is too computationally expensive and it can easily become the bottleneck of our algorithm. We also tested several iterative methods [Bouby et al. 2005; Rivers and James 2007], which turned out to be either too expensive or too unstable. Our final implementation is based on the direct approach proposed by Franca [1989]. Their idea is to calculate the three principal invariants of $\mathbf{S}$ from $\mathbf{F}^\mathsf{T}\mathbf{F}$ first, and then use them to derive $\mathbf{S}$. We note that their original method can be easily modified to handle inverted elements as well, by using the determinant of $\mathbf{F}$ to decide the sign of the third invariant.

Positional and contact constraints. Similar to [Bouaziz et al. 2014], we handle positional constraints by using stiff springs to connect vertices and their desired positions. Although we can introduce collision constraints into the system as Bouaziz and colleagues [2014] did, we found it is easier to simply enforce them at the end of each iteration. To efficiently detect cloth-body collisions in the dress example shown in Figure 1, we model the female body by a signed distance field. We found that it is difficult to handle static friction correctly in each iteration, since the vertex position change does not have sufficient physical meaning. So in this example, we break each time step into eight sub-steps and handle collisions and frictions at the end of each sub-step.

7 Results and Discussions

(Please see the supplementary video for animation examples.) We integrated the Chebyshev approach into our simulators and tested their performances on both CPU and GPU. Our CPU test runs on an Intel i5-2500K processor using a single core. Our GPU test runs

Name	#vert	#ele	# constraints	Direct (CPU)			ρ	Jacobi (CPU)			Jacobi (GPU)		
				#iter	Cost	FPS		#iter	Cost	FPS	#iter	Cost	FPS
Tablecloth (Fig. 2)	10K	20K	30K+30K	10	60.9ms	16	0.9999	400	760ms	1.3	400	26.3ms	38
Dress (Fig. 1)	16K	29K	43K+44K	10	151ms	7	0.9992	192	646ms	1.5	192	27.0ms	37
Armadillo (Fig. 7)	15K	55K	55K	10	76.8ms	13	0.9992	64	1.34s	0.7	64	20.8ms	48
Fishnet (Fig. 8)	20K	64K	64K	10	92.9ms	11	0.9996	64	1.53s	0.7	64	26.7ms	38

Table 1: *Statistics and timings of the examples. We did not test the use of the direct method on GPU for all of the examples, since the direct solve is typically slower on GPU than on CPU. To make our comparisons fair, the CPU timings of the direct method do not contain the costs of the local steps, which can be accelerated by GPU.*

on an NVIDIA GeForce GTX 970 card, except for the timings in Figure 4. The computational cost of the system depends on the number of constraints and the number of vertices involved in each constraint, rather than the total number of vertices. For example, tetrahedral constraints are more expensive than spring constraints, as each tetrahedral constraint needs four vertices. In average, the local step that enforces all of the constraints consumes 20 to 50 percent of the dynamic simulation cost on GPU. Note that our GPU implementation does not use atomic operations to transfer results back to vertices in the local step, as did in [Macklin et al. 2014]. Instead, it collects the enforced results for every vertex in the global step. So the cost of the global step is dominated by memory access. Table 1 lists the statistics and the timings of our examples. All of the examples use $h = 1/30$s as the time step. Figure 8 shows a fishnet example simulated by our Chebyshev approach.

In Table 1, we use 10 iterations to solve projective dynamics that uses the direct method, as suggested in [Bouaziz et al. 2014]. But 10 iterations is typically not enough for projective dynamics to reach a small error, especially if an object undergoes large and fast deformation. We do not recommend to couple the direct method with the Chebyshev approach in the first 10 iterations, for robustness reasons discussed in Subsection 5.2. Therefore, we need more iterations and computational time, if we want more accurate results from the use of the direct method.

Comparisons to conjugate gradient. An interesting question is whether projective dynamics can be accelerated by conjugate gradient as well. So we implement the nonlinear preconditioned conjugate gradient method as Algorithm 2 shows. Here we just use one sub-iteration to solve the linear search step, since the result often becomes worse when more sub-iterations are used. Figure 9 shows that regardless of the preconditioner, the convergence rate of the conjugate gradient approach cannot be higher than that of the Chebyshev approach in both examples. Since one conjugate gradient iteration is more computationally expensive than one Chebyshev iteration on both CPU and GPU, the overall performance of the conjugate gradient approach must be lower and we do not recommend its use in practice.

Strengths and weaknesses. While the Chebyshev approach effectively accelerates projective dynamics that uses a variety of linear system solvers, we would like to advocate the combination of the Chebyshev approach and Jacobi iterations for several reasons. First, it is straightforward to implement and it does not require additional linear solver libraries. Second, it is compatible with GPU acceleration and it has a small memory cost. Finally, the Chebyshev approach is relatively insensitive to small system changes. In contrast, the direct method needs to re-factorize the matrix every time the system gets changed, which requires a large computational cost.

That being said, the convergence criterion of the Jacobi method compromises the robustness of projective dynamics for tetrahedral meshes. Since divergence happens only when the mesh is in low quality or under large stress as our experiment shows, it would be

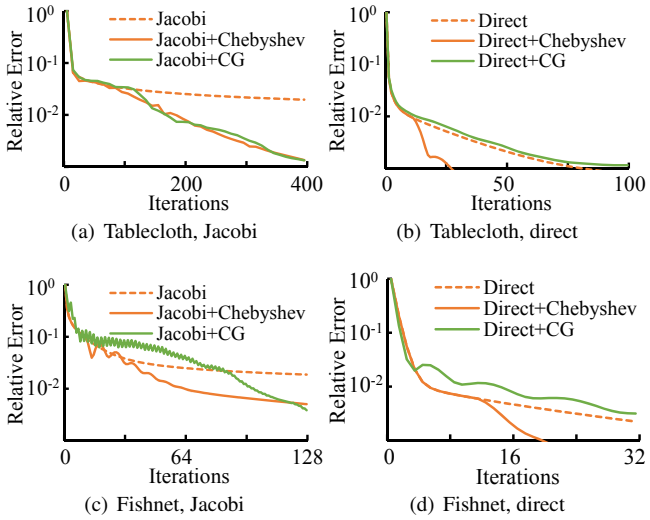


Figure 9: *The comparisons between the Chebyshev approach and the conjugate gradient approach. In both examples, the conjugate gradient approach using the Jacobi preconditioner cannot converge faster than the Chebyshev approach, as shown in (a) and (c). This is also true when the conjugate gradient approach uses the direct solver as the preconditioner, as shown in (b) and (d).*

interesting to know whether we can use failsafe methods (such as strain limiting) to avoid divergence in these cases accordingly. While we can now simulate the dynamics fast, we still cannot efficiently handle self-collisions on GPU yet. In our current implementation, the computational cost of self-collision handling is nearly twice of the dynamic simulation cost.

8 Conclusions and Future Work

In this work, we show that the convergence of projective dynamics is similar to that of an iterative method solving a linear system, even when projective dynamics solves its global step by the direct method. Because of this, projective dynamics can be efficiently accelerated by the Chebyshev approach. Our experiment indicates that it is a good practice to combine the Chebyshev approach with Jacobi iterations for projective dynamics running on GPU. Position-based dynamics can be accelerated by the Chebyshev approach as well, although the effect is less significant for tetrahedral meshes.

Our immediate plan is to make the system more robust, by finding better ways to address the divergence issue caused by the use of Jacobi iterations. We are also interested in making projective dynamics suitable for simulating hyperelastic models, and we would like to know how the Chebyshev approach behaves if the dynamical system becomes even more nonlinear. So far our research considered deformable bodies only. We plan to further explore the use of

Algorithm 2 PCG_PD_Solve($\mathbf{q}_t$, $\mathbf{v}_t$, ρ, h, K)

$\mathbf{q}^{(0)} \leftarrow \mathbf{s}_t \leftarrow \mathbf{q}_t + h\mathbf{v}_t + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$;
for $k = 0...K-1$ **do**
  for each constraint c **do**
    $\mathbf{p}_c$ = Local_Solve($c, \mathbf{q}^{(k)}$);
  $\mathbf{r}^{(k+1)} \leftarrow -\nabla\epsilon(\mathbf{q}^{(k)}, \mathbf{p}_c)$;
  If $\left\|\mathbf{r}^{(k+1)}\right\|_2^2 < \varepsilon$ **then break**;
  $\mathbf{z}^{(k+1)} \leftarrow$ Precondition_Solver($\mathbf{r}^{(k+1)}$);
  if k=0 **then**
    $\mathbf{p}^{(k+1)} \leftarrow \mathbf{z}^{(k+1)}$;
  else
    $\beta \leftarrow (\mathbf{z}^{(k+1)} \cdot \mathbf{r}^{(k+1)})/(\mathbf{z}^{(k)} \cdot \mathbf{r}^{(k)})$;
    $\mathbf{p}^{(k+1)} \leftarrow \mathbf{z}^{(k+1)} + \beta\mathbf{p}^{(k)}$;
  $\alpha \leftarrow (\mathbf{p}^{(k+1)} \cdot \mathbf{r}^{(k+1)})/(\mathbf{p}^{(k+1)} \cdot \mathbf{A}\mathbf{p}^{(k+1)})$; ▷ Line search
  $\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha\mathbf{p}^{(k+1)}$;
$\mathbf{q}_{t+1} \leftarrow \mathbf{q}^{(K)}$;
$\mathbf{v}_{t+1} \leftarrow (\mathbf{q}_{t+1} - \mathbf{q}_t)/h$;

projective dynamics and the Chebyshev approach in other simulation problems, such as particle-based fluid simulation.

9 Acknowledgments

The author thank Nvidia and Adobe for their funding and equipment support. The author especially thank the Nvidia CUDA team (Joe Eaton and Lung Sheng Chien) for testing and suggestions. This work was funded in part by the open project program of the State Key Lab of CAD&CG at Zhejiang University.

References

Baraff, D., and Witkin, A. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 43–54.

Bergou, M., Wardetzky, M., Harmon, D., Zorin, D., and Grinspun, E. 2006. A quadratic bending model for inextensible surfaces. In *Proc. of SGP*, 227–230.

Bouaziz, S., Martin, S., Liu, T., Kavan, L., and Pauly, M. 2014. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph. (SIGGRAPH) 33*, 4 (July), 154:1–154:11.

Bouby, C., Fortuné, D., Pietraszkiewicz, W., and Vallée, C. 2005. Direct determination of the rotation in the polar decomposition of the deformation gradient by maximizing a Rayleigh quotient. *Journal of Applied Mathematics and Mechanics 85*, 3 (Mar.), 155–162.

Bridson, R., Marino, S., and Fedkiw, R. 2003. Simulation of clothing with folds and wrinkles. In *Proceedings of SCA*, 28–36.

Franca, L. 1989. An algorithm to compute the square root of 3x3 positive definite matrix. *Computers. Math. Applic. 18*, 5, 459–466.

Goldenthal, R., Harmon, D., Fattal, R., Bercovier, M., and Grinspun, E. 2007. Efficient simulation of inextensible cloth. *ACM Trans. Graph. (SIGGRAPH) 26*, 3 (July).

Golub, G. H., and Van Loan, C. F. 1996. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

Golub, G. H., and Varga, R. S. 1961. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods. *Numerische Mathematik 3*, 1, 157–168.

Gutknecht, M. H., and Röllin, S. 2002. The Chebyshev iteration revisited. *Parallel Comput. 28*, 2 (Feb.), 263–283.

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 SCA*, 43–51.

Kim, T.-Y., Chentanez, N., and Müller-Fischer, M. 2012. Long range attachments - A method to simulate inextensible clothing in computer games. In *Proceedings of SCA*, 305–310.

Liu, T., Bargteil, A. W., O'Brien, J. F., and Kavan, L. 2013. Fast simulation of mass-spring systems. *ACM Trans. Graph. (SIGGRAPH Asia) 32*, 6 (Nov.), 214:1–214:7.

Macklin, M., and Müller, M. 2013. Position based fluids. *ACM Trans. Graph. (SIGGRAPH) 32*, 4 (July), 104:1–104:12.

Macklin, M., Müller, M., Chentanez, N., and Kim, T.-Y. 2014. Unified particle physics for real-time applications. *ACM Trans. Graph. (SIGGRAPH) 33*, 4 (July), 153:1–153:12.

Martin, S., Thomaszewski, B., Grinspun, E., and Gross, M. 2011. Example-based elastic materials. *ACM Trans. Graph. (SIGGRAPH) 30*, 4 (July), 72:1–72:8.

Müller, M., Heidelberger, B., Teschner, M., and Gross, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. (SIGGRAPH) 24*, 3 (July), 471–478.

Müller, M., Heidelberger, B., Hennix, M., and Ratcliff, J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2 (Apr.), 109–118.

Müller, M., Chentanez, N., Kim, T., and Macklin, M. 2014. Strain based dynamics. In *Proceedings of SCA*, 21–23.

Müller, M. 2008. Hierarchical position based dynamics. In *Proceedings of VRIPHYS*, 1–10.

Provot, X. 1996. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Proceedings of Graphics Interface*, 147–154.

Rivers, A. R., and James, D. L. 2007. FastLSM: Fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph. (SIGGRAPH) 26*, 3 (July).

Stern, A., and Grinspun, E. 2009. Implicit-explicit variational integration of highly oscillatory problems. *Multiscale Model. Simul. 7*, 4, 1779–1794.

Su, J., Sheth, R., and Fedkiw, R. 2013. Energy conservation for the simulation of deformable bodies. *IEEE Transactions on Visualization and Computer Graphics 19*, 2 (Feb.), 189–200.

Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. 1987. Elastically deformable models. *SIGGRAPH Comput. Graph. 21*, 4 (Aug.), 205–214.

Thomaszewski, B., Pabst, S., and Strasser, W. 2009. Continuum-based strain limiting. *Computer Graphics Forum (Eurographics) 28*, 2, 569–576.

Wang, H., O'Brien, J., and Ramamoorthi, R. 2010. Multi-resolution isotropic strain limiting. *ACM Trans. Graph. (SIGGRAPH Asia) 29*, 6 (Dec.), 156:1–156:10.