The Visual Computer manuscript No.
(will be inserted by the editor)

Juntao Ye · Robert E. Webber · Yangsheng Wang

A Reduced Unconstrained System for the Cloth Dynamics Solver

Abstract Modern direct solvers have been more and more widely used by computer graphics community for solving sparse linear systems, such as those that arise in cloth simulation. However, external constraints usually prevent a direct method from being used for cloth simulation due to the singularity of the constrained system. This paper makes two major contributions towards the re-introduction of direct methods for cloth dynamics solvers. The first one is an approach which eliminates all the constrained variables from the system so that we obtain a reduced, non-singular and unconstrained system. As alternatives to the well known MPCG algorithm, not only the original, unmodified PCG method, but also any direct method can be used to solve the reduced system at a lower cost. Our second contribution is a novel *direct-iterative* scheme applied for the reduced system, which is basically the conjugate gradient method using a special preconditioner. Specifically, we use the stiff part of the coefficient matrix, which we call the *matrix core*, as the preconditioner for the PCG. The inverse of this preconditioner is computed by any eligible direct solver. The direct-iterative method has proved to be more efficient than both direct and iterative methods. Our experiments show a factor of two speedup over direct methods when stiff springs are used, even greater improvements over the MPCG iterative method.

Juntao Ye
Institute of Automation, Chinese Academy of Sciences, 95 Zhongguancun East Road, Beijing, China, 100190
E-mail: yejuntao@gmail.com

Robert E. Webber
Department of Computer Science, University of Western Ontario, London, Ontario, Canada N6A 5B7
Tel.: +1-519-6612111 ext. 86916
E-mail: webber@csd.uwo.ca

Yangsheng Wang
Institute of Automation, Chinese Academy of Sciences, 95 Zhongguancun East Road, Beijing, China, 100080
Tel.: +86-10-62542942
E-mail: yangsheng.wang@ia.ac.cn

Fig. 1 Using the direct-iterative (corePCG) method with solid/cloth and cloth/cloth collision. The T-shirt is modeled by 2,792 nodes and the square sheet has 1,681 nodes.

Keywords Direct-iterative Method · Cloth Simulation · Physically-based Modeling · Matrix Reordering · Direct Method · Preconditioner

1 Introduction

In computer graphics, a mass-spring model or any of its variants has proved to be an effective model for cloth simulation. Simulating cloth usually involves integrating an *ordinary differential equation* (ODE) over time. Explicit integration methods served researchers in the early days but are seldom used now due to their instability. Implicit methods have been widely adopted ever since the breakthrough work of Baraff and Witkin [5]. Among them are semi-implicit backward Euler [5], backward differential formula (BDF2) [13] etc. Although more stable than explicit solvers, these methods still don't allow for arbitrary step sizes because they approximate a nonlinear system with a linear one. Sometimes adaptive stepsizing was adopted to ensure a stable solution. Hauth [24] presented a more accurate approach, the Inexact Simplified Newton's method, which is actually a nonlinear solver with an embedded linear solver. The linear solver is executed multiple times within one time step until certain residual criterion is satisfied. Although this method

allows much larger step sizes, handling collisions often requires small step sizes during the solution, limiting the usefulness of being able to handle large step sizes.

How to solve the linear system – either stand-alone or embedded – efficiently has become inevitably important. Various techniques have been put forward to speed up the linear system solver. These include the implicit-explicit (IMEX) of Ascher *et al.* [3] by Eberhardt *et al.* [18] and Boxerman and Ascher [9]. And also a simplified system by Desbrun *et al.* [17] and a mixed explicit/implicit by Bridson *et al.* [11]. Many of the existing cloth dynamics solvers suffer performance loss as higher stretching coefficients are applied to overcome the excessive elongation problem. This is because the condition number of the system grows with the material stiffness, forcing an iterative solver to perform many steps. Low coefficients, on the other hand, result in stretchy and less appealing cloth behavior.

Many researchers tried to alleviate the over-stretching problem by a strain limiting post-process [10] [17] [20] [26] [29] [32]. Among these attempts, [20] works very well for quadrilateral meshes, while others have poor convergence properties. Most recently, English and Bridson [19] presented a method that introduce no in-plane deformation at all and it works for arbitrary triangular meshes. However, all methods fail to integrate ideal solid/cloth collision handling into one pass.

We realized that being able to efficiently handle systems with very high spring coefficients is of great importance. This problem is addressed in this paper from a numerical analysis point-of-view. We put forward a *direct-iterative* framework, called *corePCG*, for solving the linear system in semi-implicit setup. With this method, the system is solved by an iterative method – the preconditioned conjugate gradient (PCG) method – while inside the PCG, a direct method is employed to compute the inverse of the preconditioner. The preconditioner, which we call *matrix core*, is the "stiff" part of the matrix and it makes the iterative solver converge more quickly than other preconditioners, including the widely adopted diagonal preconditioner and the Incomplete Cholesky (IC) factor, etc. The computation time of corePCG is almost a constant, independent of spring stiffness. Thus for very stiff systems, corePCG is an order of magnitude faster than the regular PCG where the matrix diagonal is used as the preconditioner.

In computer animation, collisions and user interventions often add constraints to the simulation system. For example, Baraff and Witkin [5] presented an approach of applying constraints on the simulation. These constraints could easily cause the system to become singular and thus cause the solver to fail. Baraff and Witkin modified the preconditioned conjugate gradient method to make it handle the constrained system gracefully. For the direct-iterative method to be applied for the constrained case, our solution is to first transform the singular system into a non-singular one. Moreover, if the original system is symmetric and positive definite (such as the one built by Choi and Ko [13]), our reduction transformation preserves both the symmetry and positive definiteness.

The paper is organized as follows: in §2, we give an analysis of existing numerical techniques, including the MPCG algorithm; in §3, we present how to reduce constraints so that direct methods can be used for the constrained situations; in §4 and §5, we discuss and evaluate direct methods for the linear systems assembled in the semi-implicit and IMEX integrations; our corePCG, the direct-iterative method, is presented in §6; we discuss a little about how the collision handling could affect the direct-iterative solver in §7 and draw the conclusion in §8.

2 Related Work

There are a number of ways of approaching the mathematical modeling of the physics of cloth. Adopting a mass-spring system, our work follows one of the most common approaches that deals with integrating ordinary differential equations and solving a linear system of the form

$$(\mathbf{I} - \mathcal{M}_1\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - \mathcal{M}_2\frac{\partial \mathbf{f}}{\partial \mathbf{x}})\mathbf{u} = \mathbf{b}, \tag{1}$$

where $\mathbf{f}$, $\mathbf{x}$ and $\mathbf{v}$ are vectors of force, position and velocity, respectively. Furthermore, matrices $\mathcal{M}_1$, $\mathcal{M}_2$ and the right-hand-side vector $\mathbf{b}$ are assembled differently according to different integration methods. The unknown vector $\mathbf{u}$ could be either the position change or velocity change according to different methods. For example, in the semi-implicit backward Euler [5], $\mathcal{M}_1 = h\mathbf{M}^{-1}$, $\mathcal{M}_2 = h^2\mathbf{M}^{-1}$ and $\mathbf{u} = \Delta\mathbf{v}$, where $\mathbf{M}$ is the diagonal mass matrix, h is the time step. In the second order BDF method used by Choi and Ko [13], $\mathcal{M}_1 = \frac{2}{3}h\mathbf{M}^{-1}$, $\mathcal{M}_2 = \frac{4}{9}h^2\mathbf{M}^{-1}$ and $\mathbf{u} = \Delta\mathbf{x}$. The linear system in our work is assembled according the first approach (i.e.,the semi-implicit backward Euler) and we denote the system as $\mathbf{A}\Delta\mathbf{v} = \mathbf{b}$ by defining

$$\mathbf{A} = \mathbf{I} - h\mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2\mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}, \quad \mathbf{b} = h\mathbf{M}^{-1}(\mathbf{f} + h\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{v}) \ . \tag{2}$$

If the above system is symmetric positive-definite, it usually can be efficiently solved by PCG method. The PCG method uses a preconditioner $\mathbf{M}_p$ such that $\mathbf{M}_p^{-1}$ is close to $\mathbf{A}^{-1}$. It then addresses the solution of $\mathbf{M}_p^{-1}\mathbf{A}\Delta\mathbf{v} = \mathbf{M}_p^{-1}\mathbf{b}$ which should require fewer iterations than the original equation since $\mathbf{M}_p^{-1}\mathbf{A}$ is closer to the identity matrix than $\mathbf{A}$ was. The key to the method is finding a matrix $\mathbf{M}_p$ that is similar to $\mathbf{A}$ but easier to invert than $\mathbf{A}$ is. One of the simplest such matrices is formed from the diagonal elements of $\mathbf{A}$, which is trivial to invert, but

may or may not be sufficiently similar to $\mathbf{A}$ to be useful (this approach is sometimes called *diagonal scaling*).

In the absence of constraints, Baraff and Witkin tried to make matrix $\mathbf{A}$ be symmetric positive-definite by dropping some Jacobian terms in Equ 2, so that the system can be solved by the PCG method. Choi and Ko [13] pointed out that dropping certain terms is not enough to guarantee a symmetric positive-definite system, thus the solver could fail. Based on a mass-spring model defined over a rectangular mesh, they found the indefiniteness is more likely to happen when springs are excessively compressed. In this case, the term $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ turns to be negative definite, causing the whole system to be indefinite. One solution to this problem is to use adaptive stepsizing, i.e., decrease the time step h when the system is hard to converge. Since $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ is scaled by $h^2 (h < 1)$ in Equ 2, decreasing h will cause this term to diminish — at a quadratic rate. Choi and Ko gave a different solution by using a specially designed compression model for the post-buckling response. With their model, $\mathbf{A}$ is guaranteed to be positive definite. We adopted their buckling model thus the linear systems in our experiments are always positive definite.

However, in the context of solid/cloth collision (such as cylinder/cloth interaction as shown in Figure 1 and sphere/cloth interaction as shown in Figure 3), things are more complicated due to constraints that represent restrictions on the movement of nodes at the point of impact. Integrating those constraints into the system makes us face a singular system. Baraff and Witkin [5] customized the conjugate gradient method to handle constraints. They called their method modified-pcg. Ascher and Boxerman [2] improved the modified-pcg algorithm (calling their method MPCG) and gave a proof of its convergence. We present the pseudo code of the original, unmodified PCG algorithm [6] [21] [31] (we call it *uPCG* in this paper) and the MPCG algorithm, where the matrix diagonal is adopted as the preconditioner $\mathbf{M}_p$ because it is easily invertible.

```
1.  function uPCG()
2.     initial guess for Δv
3.     δ0 = b^T M_p^{-1} b
4.     r = b − AΔv
5.     c = M_p^{-1} r
6.     δ = r^T c
7.     while (δ > tol^2 δ0)
8.        q = Ac
9.        α = δ/(c^T q)
10.       Δv = Δv + αc
11.       r = r − αq
12.       s = M_p^{-1} r
13.       δ̃ = δ
14.       δ = r^T s
15.       c = s + (δ/δ̃)c
16.    end
```

The above algorithm is only good for unconstrained systems. When constraints (e.g. those caused by solid/cloth interactions) exist in applications, Baraff and Witkin defined a 3×3 constraint matrix $\mathbf{S}_i$ for each particle to be

$$\mathbf{S}_i = \begin{cases} \mathbf{I} & \text{if ndof}(x_i) = 3 \\ \mathbf{I} - \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^T & \text{if ndof}(x_i) = 2 \\ \mathbf{I} - \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^T - \hat{\mathbf{q}}_i \hat{\mathbf{q}}_i^T & \text{if ndof}(x_i) = 1 \\ \mathbf{0} & \text{if ndof}(x_i) = 0 \end{cases} \tag{3}$$

where $\hat{\mathbf{p}}_i$ and $\hat{\mathbf{q}}_i$ are unit vectors along which accelerating is prohibited. Note that the name $\mathbf{q}_i$ is unrelated to name $\mathbf{q}$ in the uPCG pseudo code. A global constraint matrix $\mathbf{S}$ for a system of N particles can be formed from $\mathbf{S}_i$

$$\mathbf{S} = diag\{\mathbf{S}_1, \cdots, \mathbf{S}_N\} \tag{4}$$

The linear system for the constrained case is

$$\mathbf{SA}\Delta\mathbf{v} = \mathbf{Sb} \ . \tag{5}$$

And a further restriction is

$$(\mathbf{I} - \mathbf{S})\Delta\mathbf{v} = (\mathbf{I} - \mathbf{S})\mathbf{z}, \tag{6}$$

where $\mathbf{z}$ is the desired velocity change due to constraints (In the experiment of ball hitting cloth, the cloth node being hit will assume velocity of the ball in the direction of the colliding normal. Thus the corresponding x, y, z components in $\mathbf{z}$ is set to that velocity. For unconstrained nodes, their components in $\mathbf{z}$ are zero). This equation says that the constrained coordinates of $\Delta\mathbf{v}$ are set to be equal to those of $\mathbf{z}$. The matrix $\mathbf{S}$ of Equ. 5 guarantees that the values of the constrained coordinates not to change while $\Delta\mathbf{v}$ is being solved.

With constraints, although $\mathbf{A}$ is still positive definite, the system of Equ. 5 is singular because $\mathbf{S}_i$ is singular for $\text{ndof}(x_i) \neq 3$. In MPCG, the rows and columns in $\mathbf{SA}$ that cause the singularity are filtered out in the computation and thus the iteration proceeds successfully. Ascher and Boxerman [2] gave the proof of convergence of the MPCG method and accelerated the convergence of MPCG by using a corrected stop criterion and a better initial guess. The following is their MPCG algorithm, which is slightly different from Baraff's. We followed this algorithm in our experiments.

```
1.  function MPCG()
2.     initial guess for Δv
3.     b̂ = S(b − A(I − S)z)
4.     δ0 = (Sb̂)^T M_p^{-1} (Sb̂)
5.     r = S(b − AΔv)
6.     c = SM_p^{-1} r
7.     δ = r^T c
8.     while δ > tol^2 δ0
9.        q = S(Ac)
10.       α = δ/(c^T q)
11.       Δv = Δv + αc
12.       r = r − αq
13.       s = M_p^{-1} r
```

14. $\tilde{\delta} = \delta$
15. $\delta = \mathbf{r}^T\mathbf{s}$
16. $\mathbf{c} = \mathbf{S}(\mathbf{s} + (\delta/\tilde{\delta})\mathbf{c})$
17. end

Ascher and Boxerman pointed out that their improved MPCG method is equivalent to the uPCG method applied to a reduced system, and also proved the existence of the reduced system. A system of N cloth particles determines matrix $\mathbf{A}$ to be of size $n \times n$, where $n = 3N$. If d DOFs are constrained, the system has $(n-d)$ DOFs. Matrix $\mathbf{S}$ is an orthogonal projection which defines a mapping from the n-dimensional space to the $(n-d)$-dimensional space. This fact allows us to write $\mathbf{S} = \mathbf{U}\mathbf{U}^T$, where $\mathbf{U} \in \mathbb{R}^{n\times(n-d)}$ has $(n-d)$ orthogonal columns. Note that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $rank(\mathbf{S}) = n-d$. From Equ. 6, there is

$$\Delta\mathbf{v} = \mathbf{S}\Delta\mathbf{v} + (\mathbf{I}-\mathbf{S})\mathbf{z} \tag{7}$$

Substitute it into Equ. 5 and left-multiplying the equation by $\mathbf{U}^T$ yields the unconstrained, positive definite system.

$$\tilde{\mathbf{A}}\Delta\tilde{\mathbf{v}} = \tilde{\mathbf{b}} \tag{8}$$

where

$$\tilde{\mathbf{A}} = \mathbf{U}^T\mathbf{A}\mathbf{U} \tag{9a}$$

$$\tilde{\mathbf{b}} = \mathbf{U}^T(\mathbf{b} - \mathbf{A}(\mathbf{I}-\mathbf{S})\mathbf{z}) \tag{9b}$$

$$\Delta\tilde{\mathbf{v}} = \mathbf{U}^T\Delta\mathbf{v} \tag{9c}$$

However, Ascher and Boxerman never formed the transformation matrix $\mathbf{U}$ explicitly.

Interestingly, Ascher also suggested another way of solving the constrained system, but they seemed never implement it. Their idea was to combine Equ. 5 and 6 into one system

$$(\mathbf{S}\mathbf{A} + \mathbf{I} - \mathbf{S})\Delta\mathbf{v} = \mathbf{S}\mathbf{b} + (\mathbf{I}-\mathbf{S})\mathbf{z} \tag{10}$$

and solve it using a preconditioned Krylov-space method.

However, this will not work. The two systems, Equ. 5 and Equ. 10, are not equivalent, as the solution of the latter does not satisfy Equ. 6. While solving a constrained system with $(n-d)$ DOFs, we are in fact solving for the $(n-d)$ components of a n dimensional vector. But Equ. 10 gives a solution where the desired $(n-d)$ numerical values are "diluted" to all n components. To make it work, a transformation matrix, which should be similar to the matrix $\mathbf{U}$ given below, is needed to turn the solution vector for Equ. 10 from n dimensional into $(n-d)$ dimensional. Thus finding matrix $\mathbf{U}$ is the key point.

In the next section, we will show how to turn the constrained system into an unconstrained one by eliminating all constraints, so that a direct solver, as well as the direct-iterative solver presented in this paper, can be used.

3 Reducing a Constrained System into an Unconstrained One

In this section, we present our solution which makes using a direct method to solve constrained systems possible. Our work is an extension of the above Ascher and Boxerman approach. The constraints are eliminated and the original system is reduced to a lower dimensional and non-singular system.

Our solution is still based on Equ. 8 to 9c. We observe that if such a matrix $\mathbf{U}$ can be found, Equ. 5 can be transformed into Equ. 8. In fact Equ. 9a turns an $n \times n$ singular matrix $\mathbf{S}\mathbf{A}$ into an $(n-d)\times(n-d)$ full-rank matrix $\tilde{\mathbf{A}}$. Similarly, Equ. 9b turns an n-dimensional vector into an $(n-d)$-dimensional vector. Once we solve $\Delta\tilde{\mathbf{v}}$ out of Equ. 8, we can get $\Delta\mathbf{v}$ from Equ. 9c: multiplying both sides with $\mathbf{U}$ yields

$$\mathbf{U}\Delta\tilde{\mathbf{v}} = \mathbf{U}\mathbf{U}^T\Delta\mathbf{v} = \mathbf{S}\Delta\mathbf{v},$$

and together with Equ. 7, we have

$$\Delta\mathbf{v} = \mathbf{U}\Delta\tilde{\mathbf{v}} + (\mathbf{I}-\mathbf{S})\mathbf{z}\ . \tag{11}$$

Now our task is to find the transformation matrix $\mathbf{U}$. A fact is such a matrix $\mathbf{U}$ is not unique. As long as n-dimensional column vectors $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_r$ form a set of orthogonal basis for the subspace defined by $\mathbf{S}$, matrix $\mathbf{U} = (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_r)$ is one solution. However, not all candidates perform equally well.

Re-writing $\mathbf{S} = \mathbf{U}\mathbf{U}^T$ into block form

$$\begin{pmatrix} \mathbf{S}_1 & & \\ & \ddots & \\ & & \mathbf{S}_N \end{pmatrix} = \begin{pmatrix} \mathbf{U}_1\mathbf{U}_1^T & & \\ & \ddots & \\ & & \mathbf{U}_N\mathbf{U}_N^T \end{pmatrix} \tag{12}$$

we get $\mathbf{S}_i = \mathbf{U}_i\mathbf{U}_i^T$. Recall Equ. 3, the dimension of $\mathbf{U}_i$ varies according to the number of DOFs a cloth particle has. When $\text{ndof}(x_i) = 3, 1, 0$, $\mathbf{U}_i$ is uniquely defined:

- If $\text{ndof}(x_i) = 3$, $\mathbf{U}_i = \mathbf{I} \in \mathbb{R}^{3\times3}$;
- If $\text{ndof}(x_i) = 1$, $\mathbf{U}_i = \frac{\mathbf{p}_i\times\mathbf{q}_i}{|\mathbf{p}_i\times\mathbf{q}_i|} \in \mathbb{R}^{3\times1}$, where $\mathbf{p}_i$ and $\mathbf{q}_i$ are the same as in Equ. 3;
- If $\text{ndof}(x_i) = 0$, $\mathbf{U}_i = \mathbf{0}$;

When $\text{ndof}(x_i) = 2$, there are infinitely many $\mathbf{U}_i \in \mathbb{R}^{3\times2}$ satisfying $\mathbf{S}_i = \mathbf{U}_i\mathbf{U}_i^T$. If we consider $\hat{\mathbf{p}}_i$ as a normal vector which defines a plane, we could find infinitely many pairs of mutual orthogonal unit vectors $\hat{\mathbf{u}}_1$ and $\hat{\mathbf{u}}_2$ in that plane. Juxtaposing any pair of column vectors $\hat{\mathbf{u}}_1$ and $\hat{\mathbf{u}}_2$ forms one of the candidate matrices $\mathbf{U}_i$. However, when using such $\mathbf{U}_i$ to eliminate the constraints, we find sometimes the average number of iterations of uPCG is bigger than MPCG. We conclude that improper $\mathbf{U}$ causes an increment of the conditional number of $\mathbf{A}$.

We then tried QR decomposition [21] to get a pair of orthogonal vectors and it works well. Actually, given the specialty of the matrix $\mathbf{S}_i = \mathbf{I} - \hat{\mathbf{p}}_i\hat{\mathbf{p}}_i^T$, We don't have to use the generic QR algorithm which is intended

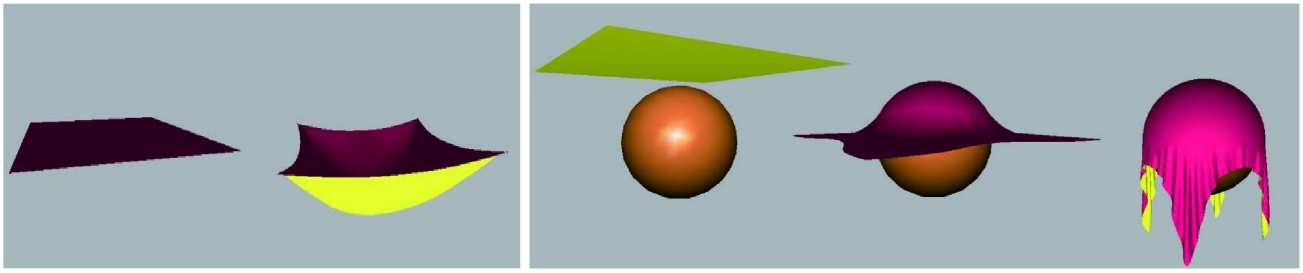


Fig. 2 Experiments of pinned cloth and draping cloth in order to compare uPCG on the reduced system with MPCG on the original system.

for rectangle matrices. Recall that $\hat{\mathbf{p}}_i = (p_1, p_2, p_3)^T$ and $p_1^2 + p_2^2 + p_3^2 = 1$. We give a close form expression for $\mathbf{U}_i$. We define

$$\mathbf{G}_i = \begin{cases} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} & \text{if}(p_1 = 1) \\ \begin{pmatrix} \sqrt{1-p_1^2} & 0 & 0 \\ -\frac{p_1 p_2}{\sqrt{1-p_1^2}} & 0 & 0 \\ -\frac{p_1 p_3}{\sqrt{1-p_1^2}} & 0 & 1 \end{pmatrix} & \text{if}(p_1 \neq 1)\text{and}(p_3 = 0) \\ \begin{pmatrix} \sqrt{1-p_1^2} & 0 & 0 \\ -\frac{p_1 p_2}{\sqrt{1-p_1^2}} & \frac{|p_3|}{\sqrt{1-p_1^2}} & 0 \\ -\frac{p_1 p_3}{\sqrt{1-p_1^2}} & -\frac{p_2 p_3}{|p_3|\sqrt{1-p_1^2}} & 0 \end{pmatrix} & \text{otherwise} \end{cases} \tag{13}$$

Eliminating the zero column from $\mathbf{G}_i$ gives $\mathbf{U}_i \in \mathbb{R}^{3\times 2}$.

Once matrix $\mathbf{U}$ is determined, all constraints can be eliminated according to Equ. 9a and Equ. 9b. A nice thing about the matrix $\mathbf{U}$ is it does not introduce any new non-zero to the sparse system. In the implementation, we adopted the widely used *block compressed column storage* (BCCS) [6] for $\mathbf{A}$ in MPCG, and *compressed column storage* (CCS) for $\tilde{\mathbf{A}}$ in uPCG.

Please note that Ascher and Boxerman have proved that the convergence performance of the uPCG algorithm for the reduced Equ. 8 is equivalent to that of the MPCG algorithm. From [31] we can conclude the solution searching directions (vector $\mathbf{c}$ in the pseudo-code) of the MPCG are restricted to the space spanned by the column vectors of $\mathbf{SA}$, which is a subspace of the space spanned by $\mathbf{A}$. This subspace is actually the same space spanned by $\tilde{\mathbf{A}}$ in the uPCG. Moreover, the values of δ_0 and δ in the uPCG are identical to their counterparts in the MPCG, thus the two algorithms terminate the iteration according to the same criteria. Our experiments have verified this – the number of iterations of the uPCG and the MPCG are almost equal (roundoff error makes them differ slightly).

Due to the constraints in the system, the MPCG is different from uPCG in that matrix $\mathbf{S}$ is applied twice in each *while*-loop, at line 9 and line 16 of the pseudo code. The reduced system no longer suffers such issue.

How is the performance of the uPCG for a reduced system compared to the MPCG for a constrained system? We set up two experiments: (1) a piece of cloth with four corners pinned subjected to gravity; (2) a piece of cloth draped on a sphere. We try to introduce as few constraints as possible in the first experiment, and introduce as many constraints as possible in the second. The matrix main diagonal is used as the preconditioner for both MPCG and uPCG. The initial guess was chosen the same way as Ascher and Boxerman did.

	k_{strh}	MPCG		uPCG	
		#iter	time	#iter	time
pinned	60	#35	0.25s	#35	0.22s
	600	#116	0.65s	#115	0.50s
	6,000	#413	2.03s	#414	1.53s
draping	60	#26	0.23s	#24	0.11s
	600	#78	0.49s	#77	0.17s
	6,000	#267	1.54s	#270	0.42s

Table 1 CPU time of one step ODE integration with $\Delta t = 3.67ms$ for experiments of Figure 2. The mesh has 6,561 nodes.

All the experiments mentioned in this paper ran on a system with Intel Core 2 Duo 2.13GHz CPU and 2 Gig memory. There was no multi-core programming involved. The computation time in the tables is for ODE integration, including matrix assembling and linear system solving but excluding the self-collision handling. The cloth is based on Provot's rectangular model [29], each node is connected to 12 neighbors via linear stretch, shear and bend springs.

Table 1 shows the result of our experiments. The cloth mesh has 6,561 nodes, making the matrix $\mathbf{A}$ to be of size 19,683. In the first case, four nodes are completely constrained, decreasing the DOFs of the whole system only by 12. After eliminating the constraints, the size of matrix $\tilde{\mathbf{A}}$ in uPCG is of size 19,671, which is almost of no change to size of $\mathbf{A}$. The experiment showed that the number of iterations in the MPCG is almost equal to that in the uPCG, which has been predicted by Ascher and Boxerman. The CPU time of uPCG is

10% to 25% less than MPCG, because no application of matrix $\mathbf{S}$ and transposing block matrices is needed. In the second case, the cloth falls down onto a sphere and strong static friction makes each colliding node to stick with the sphere. Thus many nodes have DOF = 0 and the matrix size is greatly reduced. Due to a much smaller matrix $\tilde{\mathbf{A}}$, the CPU time of uPCG cuts the running time by 50% to 75% off MPCG. Note although many nodes are constrained, setting up the matrix $\mathbf{A}$ (evaluation of forces and force Jacobians) does not take advantage of the constraints, thus no saving happens at this stage for uPCG. We did a series of experiments, varying the size of the mesh and stiffness of the spring. All experiments showed that solving reduced systems by uPCG is more efficient than solving the original system by MPCG.

The reduced system actually offers us more choices on how to solve it. Previous work has shown that the uPCG solver can be performed on a GPU [8] [28]. Doing general purpose GPU computation is not an interest of this paper. However, we would like to explore other methods for solving the reduced system.

4 Direct Solvers for Linear Systems

Although iterative methods, particularly the PCG method, are widely used for sparse linear systems, direct methods should not be neglected. A direct method is either the Gaussian elimination or its variant tailored to exploit the special structure of the matrix. For symmetric systems there are LDL^T and LL^T factorizations. For example, applying the LDL^T factorization to a symmetric matrix $\mathbf{A}$ yields

$$\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T, \tag{14}$$

where $\mathbf{L}$ is the lower triangular matrix with all diagonal elements being one, and $\mathbf{D}$ is a diagonal matrix. Thus solving the system $\mathbf{Ax} = \mathbf{b}$ turns into solving three subsystems $\mathbf{Ly} = \mathbf{b}$, $\mathbf{Dz} = \mathbf{y}$ and $\mathbf{L}^T\mathbf{x} = \mathbf{z}$.

One issue with direct methods is that when a sparse matrix $\mathbf{A}$ is factored, it typically suffers some *fill-in* and becomes less sparse. That is, $\mathbf{L}$ has nonzeros in positions which are zero in the lower triangular part of $\mathbf{A}$. Fill-in not only adds load to forward substitution $\mathbf{Ly} = \mathbf{b}$ and backward substitution $\mathbf{L}^T\mathbf{x} = \mathbf{z}$, but also increases the round-off error as it causes more arithmetic to be done. However, it is possible to reduce the amount of fill-in by reordering the matrix prior to the factorization, thus saving computer execution time and storage. A reordering means a symmetric permutation of the rows and columns of $\mathbf{A}$, and the same permutation applied to $\mathbf{b}$. Permutation of the rows is done by left-multiplying $\mathbf{A}$ with a permutation matrix $\mathbf{P}$, and permutation of the columns is done by right-multiplying $\mathbf{A}$ with $\mathbf{P}^T$. Once the reordering is done, factorizing $\mathbf{PAP}^T$, instead of $\mathbf{A}$, into $\mathbf{LDL}^T$ requires less calculation and leads to much less fill-in in $\mathbf{L}$. Thus instead of solving $\mathbf{Ax} = \mathbf{b}$, we solve

$$(\mathbf{PAP}^T)(\mathbf{Px}) = \mathbf{Pb} \ , \tag{15}$$

and then do a inverse permutation to the solution vector.

Which is better, the iterative method or the direct factorization, is really application-dependent. With an iterative method, the number of iterations required varies depending on the condition number of the matrix, or the stiffness of the linear system in cloth simulation. Quite a few researchers [33] [18] have observed that stiffer systems converge more slowly. This is not a surprise because δ_0 in the uPCG depends on $\mathbf{M}_p$ and $\mathbf{b}$, so *tol* needs to be adjusted whenever $\mathbf{M}_p$ or $\mathbf{b}$ changes. Several factors affect the stiffness of a system, such as the spring coefficients, the time step size and the velocities of the constrained particles. The direct method, however, does not suffer such drawbacks, as its running time does not change along the stiffness. For the direct factorization, certain matrices have sparsity patterns that the reordering algorithm results no fill-in (Baraff [4] presented such a case) or little fill-in. But for most matrices, we are not so lucky. Actually, finding an optimal permutation that results in the minimum amount of fill-in is NP-hard [25]. Heuristics are usually used to find a decent permutation.

Since the early 1990s, many new direct algorithms and a number of new software packages for sparse systems have been developed. Using sophisticated direct methods in computer graphics, even in the field of cloth simulation, is not new. In addition to [4], Hauth used and compared several direct solvers for cloth simulation in his Ph.D thesis [23], and he found direct methods were comparable to iterative methods. More recently, the direct Pardiso solver [30] has been employed in cloth simulation [19] [20].

Gould *et al.* [22] did an evaluation of the state-of-the-art solvers, including BCSLIB-EXT, CHOLMOD, MA57, MUMPS, Oblio, PARDISO, SPOOLES, SPRSBLKLLT, UMFPACK, TAUCS and WSMP (please refer to [22] for references to each individual solver). Among them, we are particularly interested in Cholmod [12] [14] [15] [16], designed for solving positive definite systems, and Pardiso [30], designed for both definite and indefinite systems. We have benefitted from the early version of the Cholmod package in our previous work [35] [36]. The latest Cholmod package automatically choose the best reordering between *approximate minimum degree* (AMD) [1] and *graph-partitioning-based nested dissection* (METIS) [27]. The evaluation work [22] shows that the Cholmod gives the best overall performance in 88 positive definite problems, and the Pardiso takes the second place. Therefore, we were very interested in applying the Cholmod to the linear system assembled in our cloth simulation, which happens to be positive definite. More importantly, the source code of the Cholmod package is available to the public domain, and this enable us to acquire the permutation matrix so that the direct-iterative

solver (see §6) can be carried out. Since the Pardiso package is among the most efficient solvers and has been favored by the computer graphics community, we integrated it into our system to do a comparison against the Cholmod side-by-side. It is unfortunate that the latest version Pardiso package comes only in the format of binary library file and it does not return the permutation matrix, so we are unable to use it in our direct-iterative algorithm at this moment.

5 Experiments of Using the Direct Solvers

In the semi-implicit Euler integration setup of Equ 2, the stretching forces, the shearing forces and the bending forces and their damping are all treated implicitly. In addition to the semi-implicit setup, another popular ODE integration method used in cloth simulation is the implicit-explicit method (IMEX) developed by Ascher *et al.* [3]. The IMEX method is a splitting method that consists of two portions, an implicit and an explicit one, the first one being applied to the stiff part of the ODE and the second one to the nonstiff part. Splitting can be done differently depending on the applications. For example, Eberhardt *et al.* [18] split the stretching force into a linear portion and a nonlinear portion; Boxerman and Ascher [9] treated all stretching springs implicitly, and at the same time adopted a stability criterion to deal with shear springs, thus resulted in an adaptive IMEX scheme. IMEX leads to matrices sparser than those assembled in either the implicit or semi-implicit integration, and we will occasionally call them *thin-matrix* and *full-matrix* hereafter, respectively. In our IMEX setup, the stretch force is treated implicitly while the shear and the bend forces are treated explicitly. Handling a spring connection explicitly is as simple as dropping its contribution to the Jacobian matrix.

Our next experiment [1] simulates a piece of cloth with two corners pinned being hit by a flying ball (Figure 3). In the semi-implicit setup, each row of matrix has 13 nonzero 3×3 blocks, equivalent to around 40 non-zero floats per row/column. The thin-matrix from IMEX is much sparser, with each row/column having only 15 nonzeros. The linear force model follows Choi and Ko's post-buckling model [13] so that the matrices (thin or full) are positive definite.

Choi and Ko's model results in a stable simulation in which high spring coefficient and large time step can be used. It is known that the stiffness of the system does not depend on the spring coefficient k but on $\frac{k}{m}$ [29] [18]. We simulates three meshes of different resolutions, varying the stretch coefficient k_{strh} to be 60, 600 and 6000 for each mesh. Fixed mass of 9.5e-06 kg is used for each cloth node, making $\frac{k_{strh}}{m}$ to be as high as 6.3e+08 $\frac{N}{m \cdot kg}$. The shear and bend coefficients, while unchanged, are set as small as 0.3 and 0.05, respectively, making the ratio $\frac{k_{strh}}{k_{bend}}$ as high as 1.2e+05. With such a $\frac{k_{strh}}{k_{bend}}$ ratio that the stretching stiffness is greater than the bending stiffness by four or more orders, it is enough simulate inextensible plates and shells [7]. The time steps used for the three different sized meshes are 6.67ms, 5.55ms and 3.67ms, respectively. In fact, a larger time step is allowed if we do any of the following adjustments: (1) fix more nodes on the top row of the mesh; (2) slow down the motion of the ball; (3) make the cloth to fit less tightly on the surface of the ball.

We applied MPCG, Cholmod and Pardiso to linear systems formed by the semi-implicit and the IMEX integrations. The result is shown in Table 2. As previous experiments, we only record the computation time for ODE integration, excluding the time for self-collision handling. The value of *tol* is carefully chosen to make the computation barely converge so that the iteration terminates as early as possible. Both of the two direct solvers in our implementation use Level-3 BLAS supernode techniques. A flag can be set for the Pardiso solver so that it is optimized to handle symmetric positive definite systems more efficiently than indefinite systems (data is shown in column "Pardiso(1)" of Table 2). An interesting point about the Pardiso solver is it allows a combination of direct and iterative methods in order to accelerate the solution process. This mode is referred as the *preconditioned CSG*, and it works by applying the L and U factors from the previous step to a preconditioned Krylov-Subspace iteration. If that attempt fails to converge, the solver will automatically switch back to the direct factorization. Experimental data corresponding to this mode is shown in the column "Pardiso(2)" of Table 2.

Our experiment shows that MPCG performs better than direct methods only when the spring stiffness is very low, and this is true for both the semi-implicit and IMEX cases. Comparing Cholmod with Pardiso(1), the former is more efficient in all experiments, and this is consistent with the evaluation result of Gould *et al.* [22]. Using the preconditioned CSG mode (Pardiso(2)), we gained no benefit in terms of running time. However, we did monitor that many steps of preconditioned CSG attempts were successful during the simulation. We guess the preconditioned CSG mode works best in cases that identical sparsity pattern is maintained for many consecutive steps and only the numerical values change. Our matrices are reduced from constrained ones, and their sparsity pattern is not fixed.

The key insight of the experimental data is that the execution time for the thin-matrix of the IMEX setup is 1/3 of that for the full-matrix of the semi-implicit setup, and that the larger the mesh size, the more IMEX out-

[1] This experiment is used in the next section as well. One difficulty in cloth simulation is overstretching. To explore this problem, this experiment involves a ball applying force on the surface of a hanging cloth. The proper behavior is for the cloth to get out of the way of the ball rather than let the ball stretch the cloth like the cloth were made of rubber.

#nodes in mesh	Δt (ms)	Integration	k_{strh}	MPCG #iter	MPCG time	Direct method Cholmod	Direct method Pardiso(1)	Direct method Pardiso(2)	corePCG #iter	corePCG time
1,681	6.67	semi-implicit	60	#119	0.16s	0.11s	0.15s	0.15s	#2	0.07s
			600	#314	0.38s				#4	0.07s
			6,000	#722	0.84s				#3	0.07s
		IMEX	60	#131	0.10s	0.05s	0.06s	N/A	N/A	N/A
			600	#305	0.22s				N/A	N/A
			6,000	#681	0.46s				N/A	N/A
6,561	5.55	semi-implicit	60	#50	0.32s	0.65s	0.78s	0.76s	#3	0.35s
			600	#123	0.66s				#4	0.35s
			6,000	#511	2.43s				#5	0.37s
		IMEX	60	#53	0.21s	0.23s	0.29s	N/A	N/A	N/A
			600	#114	0.36s				N/A	N/A
			6,000	#442	1.22s				N/A	N/A
25,921	3.67	semi-implicit	60	#88	2.24s	3.88s	4.29s	4.40s	#4	1.79s
			600	#256	5.77s				#4	1.81s
			6,000	#574	12.42s				#4	1.80s
		IMEX	60	#88	1.32s	1.16s	1.42s	N/A	N/A	N/A
			600	#258	3.35s				N/A	N/A
			6,000	#575	7.22s				N/A	N/A

Table 2 CPU time of one step ODE integration while simulating a piece of cloth being hit by a sphere using the semi-implicit and the IMEX integration to assemble the linear system. Cholmod, Pardiso(1), Pardiso(2) and corePCG are applied to reduced systems. Pardiso(1) is the case that the solver is set to handle symmetric positive systems, while Pardiso(2) is the preconditioned CSG mode.

performs semi-implicit. This observation inspired us to develop the corePCG method of the next section.

6 corePCG: a Direct-iterative Framework for Linear Systems

Although the IMEX typically gives savings on execution time compared to the semi-implicit method, it is not always preferable for time-dependent PDE/ODE, as mentioned by Ascher *et al.* [3]. This is also true for the ODE arising in cloth simulation. In the above experiments where all shear forces were treated explicitly, we noticed that increasing the shear coefficient makes the IMEX setup become unstable. Stability can be increased if the shear forces are treated implicitly. This, however, will increase the density of the IMEX matrix, thus increase the computation cost of the linear solver. In such a case, it is often a tough call in deciding whether shear forces should be treated explicitly or implicitly. This is the main reason behind the adaptive IMEX proposed by Boxerman and Ascher [9]. And also, the IMEX solver is usually less accurate than implicit or semi-implicit solvers. For example, Eberhardt *al.* [18] had to update the right-side-hand within the PCG while-loop, otherwise the system does not generate enough motion to the cloth mesh. Therefore, in certain situations, dealing with a full-matrix is preferred rather than a thin-matrix. In this section, we focus on full-matrices formed in the semi-implicit setup, and present a technique to speed up the iterative solver. The efficiency of direct methods in IMEX presented in the previous section inspired us in finding a new preconditioner, which we will call matrix core, for the iterative solver.

A good preconditioner is very helpful in making the PCG converge quickly. Some people have spent much effort in finding a better preconditioner. In addition to the diagonal preconditioner used by Baraff and Witkin [5], Choi and Ko [13] tried using 3×3 block diagonal preconditioner, Incomplete Cholesky (IC) and Incomplete LU (ILU), and got an improvement of approximately 20% from the block diagonal but no gain from the other two. Hauth et al. [24] experimented with IC and Symmetric Successive Overrelaxation (SSOR) preconditioners, and both gave a performance improvement of approximate 20%. Boxerman and Ascher [9] also tried the 3×3 block diagonal preconditioner, but taking consideration of the constraints. They reported a performance improvement of 30% when a significant number of constraints exist. Their preconditioner, constructed by projecting the tri-diagonal of unconstrained matrix $\mathbf{A}$ onto the constrained space, is equivalent to the tri-diagonal of our reduced matrix $\tilde{\mathbf{A}}$ (discussed in §3), so the improvement brought by our new algorithm is indeed on top of their work. All above preconditioning techniques, although offer certain improvement, are still not suitable for handling very stiff systems because the number of iterations still increases as the stiffness increases.

Different from existing techniques, we select the preconditioner by taking consideration of not only the matrix pattern but also the numerical values of matrix entries. This is achieved by a careful analysis of how the matrix $\mathbf{A}$ for cloth simulation is assembled. In Equ. 2, the term $\mathbf{f}$ includes internal forces ($\mathbf{f}_{strh}$, $\mathbf{f}_{shear}$ and $\mathbf{f}_{bend}$) and external forces (gravity, air friction etc.). The Jaco-

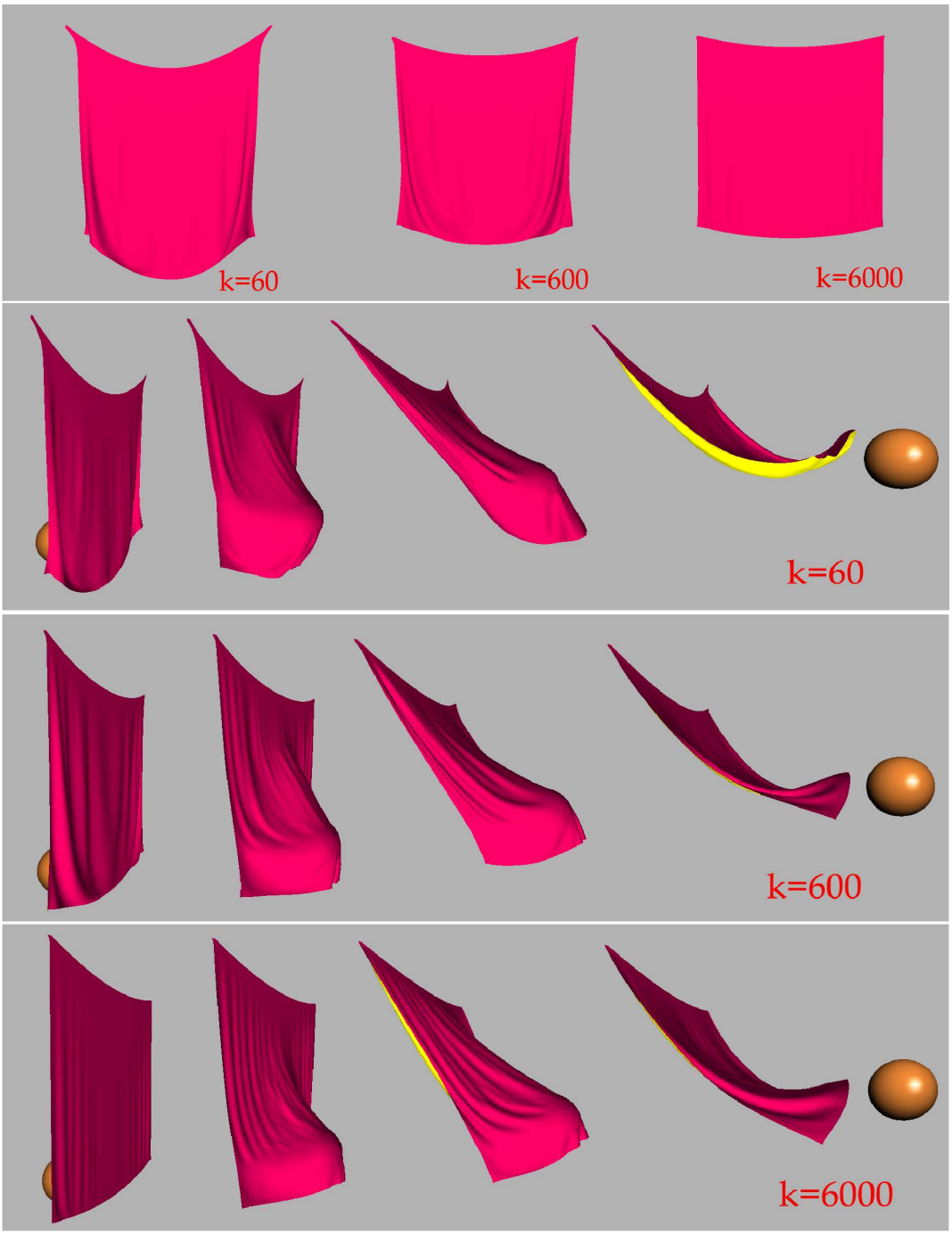


Fig. 3 A piece of cloth modeled by 25,921 nodes is being hit by a flying ball. The stretch coefficient varies from 60 to 6000.

bian of external forces is zero. We write $\mathbf{A}$ into the form of $\mathbf{A} = \mathbf{A}_{strh} + \mathbf{A}_{shear} + \mathbf{A}_{bend} + \mathbf{D}_{other}$. Sparse matrix $\mathbf{A}_{strh}$ is due to stretching forces and their Jacobian,and their corresponding damping terms. ($\mathbf{A}_{shear}$ and $\mathbf{A}_{bend}$ can be interpreted similarly.) Each sparse matrix has a different sparsity pattern, which is determined by the internal dynamics model. The diagonal matrix $\mathbf{D}_{other}$ can be considered as the identity matrix in Equ. 2. All four matrices contribute to the main diagonal of $\mathbf{A}$, making the main diagonal somewhat "heavier" than other subdiagonals. Using the diagonal of $\mathbf{A}$ as the preconditioner works well when the particle mass is relatively high and the spring stiffness, particularly the stretching stiffness, is relatively low, as in this case $\mathbf{A}$ numerically approaches the $\mathbf{D}_{other}$. We can see this in Table 2 where the diagonal of A is the preconditioner being used for the MPCG

results. This was also observed and pointed out by Ascher [2]. As the stiffness increases, the matrix diagonal becomes less good as a preconditioner. Due to the fact that the cloth model has a large stretch stiffness but relatively small shear and bend stiffness, the numerical values in $\mathbf{A}_{strh}$ are greater in magnitude (the absolute values) than those in $\mathbf{A}_{shear}$ and $\mathbf{A}_{bend}$. Increasing the stretching stiffness causes $\mathbf{A}_{strh}$ to have more "weight" in $\mathbf{A}$.

We derive a special matrix, which we call matrix core, as the preconditioner: $\mathbf{A}_{core} = \mathbf{A}_{strh} + diag(\mathbf{A}_{shear}) + diag(\mathbf{A}_{bend}) + \mathbf{D}_{other}$. Matrix $\mathbf{A}_{core}$ has the same sparsity pattern as the thin-matrix used in IMEX integration but different numerical values, as the IMEX thin-matrix does not contain $diag(\mathbf{A}_{shear})$ and $diag(\mathbf{A}_{bend})$. In implementation, we extract $\mathbf{A}_{core}$ from $\mathbf{A}$ by deleting those nonzero blocks which occur in $\mathbf{A}_{shear}$ or $\mathbf{A}_{bend}$ but not in $\mathbf{A}_{strh}$.

Using the matrix core as the preconditioner, we call our method *corePCG()* algorithm. A preconditioner matrix should be easily invertible. Actually, the inverse of the preconditioner $\mathbf{A}_{core}$ does not have to be explicitly computed. Applying inverse matrix $\mathbf{s} = \mathbf{M}_p^{-1}\mathbf{r}$ (line 12 of the uPCG() algorithm) is equivalent to solving $\mathbf{M}_p\mathbf{s} = \mathbf{r}$ (and the same rule applies to the calculation of δ_0 at line 3). To solve this system, we use a direct method – Cholmod in our experiment – with matrix reordering. In corePCG(), the LDL^T factorization, applied to the reordered preconditioner $\check{\mathbf{A}}_{core}(= \mathbf{P}\mathbf{A}_{core}\mathbf{P}^T)$, should be done once outside of the while-loop (line 5 of corePCG()) so that the system can be backsolved for multiple values of RHS inside the loop (line 17). $\mathbf{A}$, $\mathbf{b}$ and potentially the initial guess $\Delta\mathbf{v}$ are reordered before the while-loop starts so that their orders are consistent with the reordered preconditioner.

We present the corePCG algorithm as follows, and note lines 8 to 20 is the original uPCG pseudo code.

1. **function corePCG()**
2. extract $\mathbf{A}_{core}$ from $\mathbf{A}$ as the preconditioner
3. analyze $\mathbf{A}_{core}$ to find the permutation matrix $\mathbf{P}$
4. reorder $\mathbf{A}_{core}$: $\check{\mathbf{A}}_{core} = \mathbf{P}\mathbf{A}_{core}\mathbf{P}^T$
5. do factorization: $\check{\mathbf{A}}_{core} = \mathbf{L}\mathbf{D}\mathbf{L}^T$
6. reorder $\mathbf{A}$ and $\mathbf{b}$: $\check{\mathbf{A}} = \mathbf{P}\mathbf{A}\mathbf{P}^T, \check{\mathbf{b}} = \mathbf{P}\mathbf{b}$
7. reorder the initial guess: $\Delta\check{\mathbf{v}} = \mathbf{P}\Delta\mathbf{v}$
8. Solve $(\mathbf{L}\mathbf{D}\mathbf{L}^T)\mathbf{t} = \mathbf{b}$ for $\mathbf{t}$, so $\delta_0 = \check{\mathbf{b}}^T\check{\mathbf{A}}_{core}^{-1}\check{\mathbf{b}} = \check{\mathbf{b}}^T\mathbf{t}$
9. $\mathbf{r} = \check{\mathbf{b}} - \check{\mathbf{A}}\Delta\check{\mathbf{v}}$
10. solve $(\mathbf{L}\mathbf{D}\mathbf{L}^T)\mathbf{c} = \mathbf{r}$ for $\mathbf{c}$
11. $\delta = \mathbf{r}^T\mathbf{c}$
12. while $\delta > tol^2\delta_0$
13. $\mathbf{q} = \check{\mathbf{A}}\mathbf{c}$
14. $\alpha = \delta/(\mathbf{c}^T\mathbf{q})$
15. $\Delta\check{\mathbf{v}} = \Delta\check{\mathbf{v}} + \alpha\mathbf{c}$
16. $\mathbf{r} = \mathbf{r} - \alpha\mathbf{q}$
17. solve $(\mathbf{L}\mathbf{D}\mathbf{L}^T)\mathbf{s} = \mathbf{r}$ for $\mathbf{s}$
18. $\tilde{\delta} = \delta$
19. $\delta = \mathbf{r}^T\mathbf{s}$
20. $\mathbf{c} = \mathbf{s} + (\delta/\tilde{\delta})\mathbf{c}$
21. end
22. inversely reorder the solution: $\Delta\mathbf{v} = \mathbf{P}^T\Delta\check{\mathbf{v}}$

One might want to avoid reordering matrix $\mathbf{A}$ before the while-loop, as reordering a matrix is more costly than reordering a vector. An alternative to reordering $\mathbf{A}$, $\mathbf{b}$ and $\Delta\mathbf{v}$ at line 6 and 7 is to apply reordering and the direct solver at line 17 to solve $\check{\mathbf{A}}_{core}\check{\mathbf{s}} = \check{\mathbf{r}}$. This way, $\mathbf{r}$ is reordered a priori to form a proper $\check{\mathbf{r}}$, and solution $\check{\mathbf{s}}$ is inversely-reordered right after to make it consistent with vector $\mathbf{c}$. Since the while-loop will be executed multiple times, vector reordering will occur multiple times. In our experiment, the while-loop is executed only a few times, and there is no obvious performance difference between these two choices. We guess that in applications where the number of iterations is high, this alternative choice is not favorable.

Our implementation of corePCG uses the Cholmod package, not only because Cholmod is particularly efficient for positive definite systems. More importantly, the open source Cholmod makes the permutation matrix $\mathbf{P}$ available to us, while the Pardiso package (available in binary library file only) does not. The permutation matrix is used in line 4, 6, 7 and 22 of corePCG algorithm. If the Pardiso could be used in corePCG, we believe, it will not change the conclusion of this paper. We are looking forward to a new version Pardiso that is more friendly to our corePCG algorithm, thus the system to be solved does not have to be positive definite.

The corePCG method gives exciting results as shown in Table 2. Comparing to the Cholmod for the semi-implicit setup in solving a full-matrix, the corePCG cuts the execution time by 46% for the medium mesh, and 54% for the large mesh. An interesting point here is the number of iterations in corePCG is quite low and it does not change significantly as the stretch coefficient changes. As a result, the execution time of corePCG is almost a constant, no longer sensitive to the stiffness of the system. These justify the observation that the matrix core preconditioner is such a good approximation to the original matrix, and calculating the inverse of the preconditioner is the most time-consuming part in each simulation step.

In conclusion, for systems of high stiffness, modern direct solvers are in general more efficient than iterative methods, although the performance of an iterative method is often implementation-dependent, influenced by the stop criterion and the time step (which affects the initial guess for the solution). Solving a full-matrix system often gives more accurate result than solving a thin-matrix. If the stiff part of the full-matrix can be easily extracted, our direct-iterative method can be applied and it is superior to direct methods. Essentially as an iterative method, the cost of corePCG consists of applying a direct method to the thin-matrix plus a very few number of matrix-vector multiplications and forward/backward substitutions. Compared to the IMEX direct solver for a thin-matrix, the corePCG handles a full-matrix with less

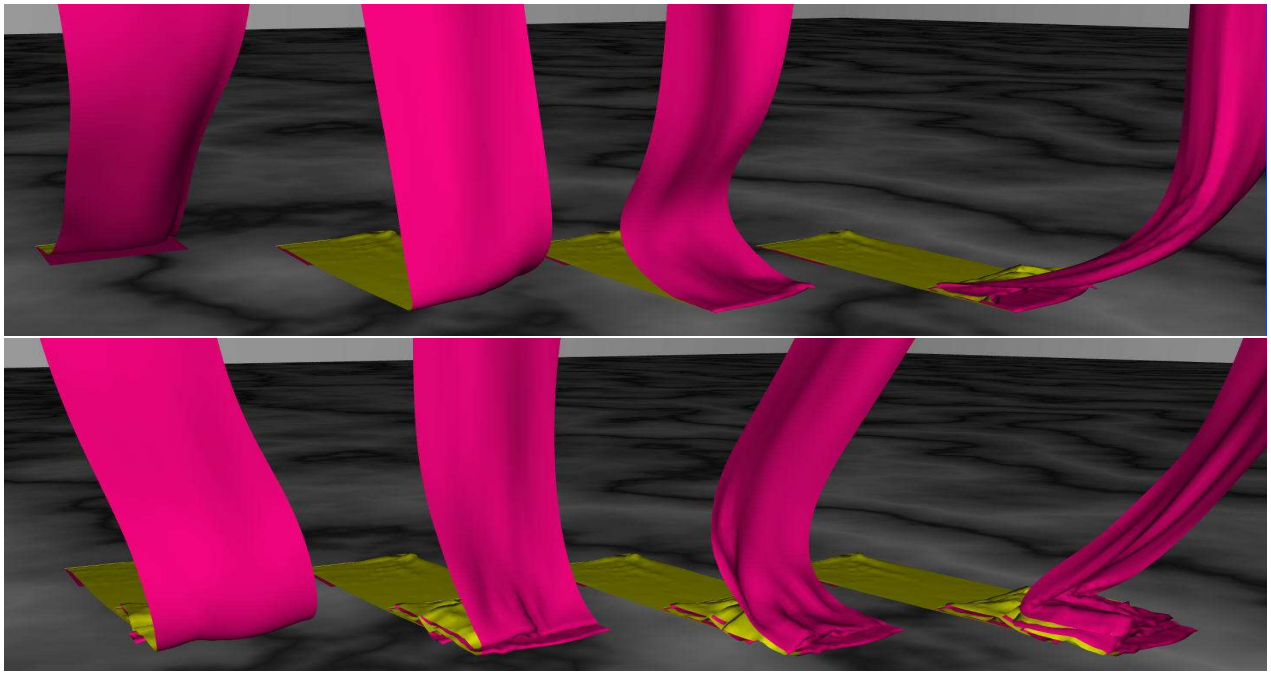


Fig. 4 Frame images for the simulation of clumping cloth with excessive self-collisions. A relatively small $\Delta t = 0.0022$ sec is adopted so that the collisions are resolved at a fine level. When using MPCG, the average running time is 1.83s per step (0.75s for dynamics simulation and 1.08s for collision handling). When using Cholmod, the running time is 1.44s per step (0.36s for dynamics simulation and 1.08s for collision handling). When using corePCG, the running time is 1.29s per step (0.21s for dynamics simulation and 1.08s for collision handling).

than 50% of extra cost. The corePCG intends to bring performance improvement per step basis, and it has no restriction on the step size. None of the existing preconditioning techniques used in cloth simulation, including IC, SSOR and block diagonal, offers a performance improvement as much as corePCG does. That the computation cost of corePCG does not change along the system stiffness could be a good news for real-time applications, since the running time can be predictable.

7 Collision Response

The performance boost offered by our corePCG algorithm is preserved along with self-collision handling (see Figure 1 and 4). We followed Bridson *et al.* [10] to resolve self-collision by doing velocity adjustment, and form *Rigid Impact Zone* (RIZ) if excessive collisions happen in a certain region. All nodes in a RIZ assume the identical linear and angular velocity, thus can be treated as DOF $= 0$ and their corresponding matrix entries are eliminated by the dynamics solver.

Eberhardt [18] reported a performance loss of the regular CG method when the repositioning suggested in [5] was used. We believe the main reason behind this can still be explained by Choi and Ko's analysis [13] (also see §2): arbitrarily repositioning cloth nodes would cause some springs to be over-compressed, turning the matrix towards indefiniteness. This should not be an issue with Choi and Ko's post-buckling model, as it always produces positive definite systems. Therefore, the corePCG would just work well with repositioning collision handling.

Even if our corePCG algorithm is used in a system that is not always positive definite, repositioning will cause it to suffer less performance loss than regular CG, because the major computation burden is in computing the inverse of the preconditioner by a direct solver, whose performance is only sensitive to the nonzero pattern of the matrix, not the numerical values. We believe the corePCG will work well with other robust collision resolution strategies, such as [34].

8 Conclusion

Following the ideas in the convergence proof of Ascher and Boxerman [2], we have presented a technique that reduces a constrained system to an unconstrained one, thus allowing uPCG, direct solvers and our direct-iterative solver to work for cloth simulations. The reduction eliminates two matrix-vector multiplication from the inner loop of the MPCG algorithm and resulted in substantial time savings in the presence of many constraints.

With our constraints reduction approach, we are able to apply direct solvers for linear systems in cloth simulation. The popular numerical methods for integrating

ODEs are IMEX and semi-implicit integrations. We explore two direct methods, Cholmod and Pardiso, for solving linear systems assembled in these two different integrations. While the performance of an iterative method varies greatly depending on the stiffness of the system, the direct method outperforms the iterative method when the system stiffness is above certain level. In IMEX setup, we found direct solvers are particularly efficient. For semi-implicit setup, we proposed a direct-iterative method, corePCG, by developing a new preconditioner that outperforms existing preconditioners. Although inverting the preconditioner is more costly than inverting a traditional diagonal, its inverse is much closer to the inverse of the coefficient matrix than the inverse of the diagonal preconditioner. Thus the extra inversion cost is balanced off by the low number of iterations needed. Our experiments showed its performance relative to the popular diagonal preconditioner improves both as stiffer springs are used to model the cloth and as more nodes are used to model the cloth at higher resolution. By the time we are using 25,921 nodes with a stretch coefficient of 6,000, our corePCG method (uPCG using the matrix core preconditioner) executes at only 50% of the time it takes to run the already efficient Cholmod solver, and it takes only 4 iterations to converge. The iterative method is even far from comparable to corePCG.

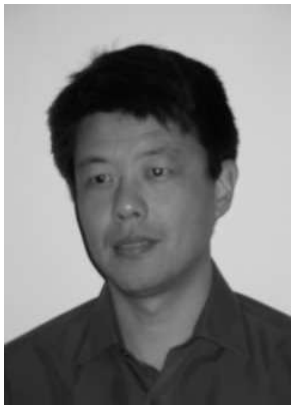
Although all experiments shown in this paper are based on rectangular meshes, our theory is developed independent of mesh geometry. In particular, all algorithms should work on triangular cloth meshes as well. Since triangular meshes are more flexible for modeling complicated cloth structures such as shirts, we are interested in continuing to develop this work in that domain.

Acknowledgements The authors would like to sincerely thank the anonymous reviewers for their suggestions that make this paper better.

References

1. Amestoy, P.R., Davis, T.A., Duff, I.S.: Algorithm 837: AMD, an approximate minimum degree ordering algorithm. ACM Transactions on Mathematical Software **30**(3), 381–388 (2004)
2. Ascher, U.M., Boxerman, E.: On the modified conjugate gradient method in cloth simulation. The Visual Computer **19**(7-8), 526–531 (2003)
3. Ascher, U.M., Ruuth, S.J., Wetton, B.: Implicit-explicit methods for time-dependent partial differential equations. SIAM J. Numer. Anal. **32**(3), 797–823 (1995)
4. Baraff, D.: Linear-time dynamics using Lagrange multipliers. In: Proceedings of SIGGRAPH '96, pp. 137–146 (1996)
5. Baraff, D., Witkin, A.: Large steps in cloth simulation. In: Proceedings of SIGGRAPH '98, pp. 43–54 (1998)
6. Barrett, R.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. Philadelphia : SIAM (1994)
7. Bergou, M., Wardetzky, M., Harmon, D., Zorin, D., Grinspun, E.: A quadratic bending model for inextensible surfaces. In: Eurographics Symposium on Geometry Processing, pp. 227–230 (2006)
8. Bolz, J., Farmer, I., Grinspun, E., Schröder, P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. ACM Transactions on Graphics (SIGGRAPH '03) **22**(3), 917–924 (2003)
9. Boxerman, E., Ascher, U.: Decomposing cloth. In: Proceedings of SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2004), pp. 153–161 (2004)
10. Bridson, R., Fedkiw, R., Anderson, J.: Robust treatment of collisions, contact and friction for cloth animation. ACM Transactions on Graphics (SIGGRAPH '02) **21**(3), 594–603 (2002)
11. Bridson, R., Marino, S., Fedkiw, R.: Simulation of clothing with folds and wrinkles. In: Proceedings of SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2003), pp. 28–36 (2003). URL http://www.cs.ubc.ca/ rbridson/docs/cloth2003.pdf
12. Chen, Y., Davis, T.A., Hager, W.W., Rajamanickam, S.: Algorithm 8xx: Cholmod, supernodal sparse Cholesky factorization and update/downdate. Unpublished Technical Report TR-2006-005, University of Florida (2006). Http://www.cise.ufl.edu/research/sparse/, see also accompanying software
13. Choi, K.J., Ko, H.S.: Stable but responsive cloth. ACM Transactions on Graphics (SIGGRAPH '02) **21**(3), 604–611 (2002)
14. Davis, T.A., Hager, W.W.: Modifying a sparse Cholesky factorization. SIAM Journal on Matrix Analysis and Applications **20**(3), 606–627 (1999)
15. Davis, T.A., Hager, W.W.: Multiple-rank modifications of a sparse Cholesky factorization. SIAM Journal on Matrix Analysis and Applications **22**(4), 997–1013 (2001)
16. Davis, T.A., Hager, W.W.: Row modifications of a sparse Cholesky factorization. SIAM Journal on Matrix Analysis and Applications **26**(3), 621–639 (2005)
17. Desbrun, M., Schröder, P., Barr, A.: Interactive animation of structured deformable objects. In: Proceedings of Graphics Interface '99, pp. 1–8 (1999)
18. Eberhardt, B., Etzmuß, O., Hauth, M.: Implicit-explicit schemes for fast animation with particle systems. In: Eurographics Workshop on Computer Animation and Simulation, vol. 11, pp. 137–151 (2000)
19. English, E., Bridson, R.: Animating developable surfaces using nonconforming elements. ACM Transactions on Graphics (SIGGRAPH '08) **27**(3), to appear (2008)
20. Goldenthal, R., Harmon, D., Fattal, R., Bercovier, M., Grinspun, E.: Efficient simulation of inextensible cloth. ACM Transactions on Graphics (SIGGRAPH '07) **26**(3) (2007)
21. Golub, G.H., Loan, C.F.V.: Matrix Computations. Johns Hopkins University Press (1996)
22. Gould, N.I., Scott, J.A.: A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations. ACM Trans. Math. Software **33**(2) (2007)
23. Hauth, M.: Visual simulation of deformable models. Ph.D. thesis, Department of Computer Science, University of Tubingen (2004)
24. Hauth, M., Etzmuß, O., Straßer, W.: Analysis of numerical methods for the simulation of deformable models. The Visual Computer **19**(7–8), 581–600 (2003)
25. Heggernes, P., Eisenstatz, S.C., Kumfert, G., Pothen, A.: The computational complexity of the minimum degree algorithm. In: Proceedings of 14th Norwegian Computer Science Conference, NIK 2001, University of Troms, Norway, pp. 98 – 109 (2001)
26. Hong, M., Choi, M.H., Jung, S., Welch, S., Trapp, J.: Effective constrained dynamic simulation using implicit constraint enforcement. In: International Conference on Robotics and Automation, pp. 4520–4525 (2005)

27. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing **20**(1), 359–392 (1998)
28. Krüger, J., Westermann, R.: Linear algebra operators for GPU implementation of numerical algorithms. ACM Transactions on Graphics (SIGGRAPH '03) **22**(3), 908–916 (2003)
29. Provot, X.: Deformation constraints in a mass-spring model to describe rigid cloth behavior. In: Graphics Interface '95, pp. 147–154 (1995)
30. Schenk, O., Gärtner, K.: On fast factorization pivoting methods for symmetric indefinite systems. Elec. Trans. Numer. Anal. **23**, 158–179 (2006)
31. Shewchuk, J.: An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-TR-94-125, Carnegie Mellon University (1994). Http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.ps
32. Tsiknis, D., Bridson, R.: Cloth animation through unbiased strain limiting and physics-aware subdivision. In: Proceedings of SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2006), poster session (2006)
33. Volino, P., Magnenat Thalmann, N.: Comparing efficiency of integration methods for cloth simulation. In: Computer Graphics International '01, pp. 265–272 (2001)
34. Volino, P., Magnenat-Thalmann, N.: Resolving surface collisions through intersection contour minimization. ACM Trans. Graph. **25**(3), 1154–1159 (2006). DOI http://doi.acm.org/10.1145/1141911.1142007
35. Ye, J.: Simulating inextensible cloth using impulses. Computer Graphics Forum (Special issue for Pacific Graphics 2008) **27**(7), 1901–1907 (2008)
36. Ye, J., Webber, R.E., Gargantini, I.: On the impulse method for cloth animation. In: International Conference on Computational Science (2), *Lecture Notes in Computer Science*, vol. 3515, pp. 331–334. Springer (2005)



Juntao Ye was awarded his MSc in Applied Mathematics from Institute of Computational Mathematics and Sci/Eng Computing, Chinese Academy of Sciences in 2000, and his PhD in Computer Science from University of Western Ontario, Canada, in 2005. He had worked for a video game studio in Ontario for two years from 2005 to 2007. He is currently a researcher at Institute of Automation, Chinese Academy of Sciences.

Robert E. Webber was awarded his PhD in Computer Science at the University of Maryland, College Park in 1983. He is currently an Associate Professor at the University of Western Ontario, Canada.

Yangsheng Wang was awarded his M.Sc in Information Technology and PhD in Computer Vision from Huazhong University of Science and Technology in 1989. He had been a senior academic visitor in Oxford of UK for one year since 1987, and moved to the Department of Artificial Intelligence, Edinburgh University of UK as a postdoctoral fellow for another year. He had been a Senior Research fellow in Brunel University from 1990 to 1994 and then worked in Active Imaging Ltd of UK and Nortel Networks of UK from 1994 to 2000. He has been with Institute of Automation, Chinese Academy of Sciences as a professor since 2000.