Parallelized Incomplete Poisson Preconditioner in Cloth Simulation

Costas Sideris[1], Mubbasir Kapadia[1,2], Petros Faloutsos[1,3]

[1]University of California Los Angeles
[2]University of Pennsylvania
[3]York University.

Abstract. Efficient cloth simulation is an important problem for interactive applications that involve virtual humans, such as computer games. A common aspect of many methods that have been developed to simulate cloth is a linear system of equations, which is commonly solved using conjugate gradient or multi-grid approaches. In this paper, we introduce to the computer gaming community a recently proposed preconditioner, the *incomplete Poisson* preconditioner (IPP), for conjugate gradient solvers. We show that IPP performs as well as the current state-of-the-art preconditioners, while being much more amenable to standard thread-level parallelism. We demonstrate our results on an 8-core Mac Pro and a 32-core Emerald Rigde system.

1 Introduction

Simulating flexible materials, such as cloth, is an important task for applications involving virtual humans such as computer games and visual effects. High quality offline simulations are achieved by using implicit methods for simulating cloth [20–22]. Real-time applications on the other hand use explicit or semi-explicit methods for cloth simulation in order to meet time constraints [5, 17]. Despite the decades of research on simulating flexible materials, the efficient simulation of cloth remains an important challenge for computer animation.

There exists a large amount of research that addresses algorithmic optimizations for speeding up implicit integration methods for simulating cloth. The use of preconditioners [4, 9, 13] have been shown to greatly reduce the number of iterations of the conjugate gradient method in an effort to achieve convergence. In this paper, we explore the use of a novel preconditioning scheme – *the incomplete poisson preconditioner* – that has not been used before in clothing simulation. Using a variety of standard benchmarks, we first demonstrate that this preconditioner is just as good, if not better than currently used methods. A major advantage of this method is that it is extremely easy to parallelize and can take advantage of the processing power available in current and next generation multi-core hardware. Current state of the art preconditioners [13] are not as suitable for parallelization and do not scale well with increase in computational resources. This paper makes the following contributions:

1. To our knowledge, we propose for the first time the use of the incomplete poisson preconditioner (IPP) for clothing simulation.
2. We compare the IPP to the most commonly used preconditioning methods in terms of efficiency, quality and ease of parallelization.
3. We demonstrate that a parallel implementation of the IPP achieves significant performance improvement on multi-core computers.
4. We demonstrate the scalability of the IPP on a state of the art 32-core compute server and show that it is ready for the next generation of hardware resources.

The rest of this document is organized as follows. Section 2 reviews related work. Section 3 presents an overview of the method we use for simulating cloth. We describe the Jacobi preconditioner, the Symmetric Successive over Relaxation, and the incomplete Cholesky preconditioner which are commonly used to accelerate convergence. In addition, we propose the use of the incomplete poisson preconditioner for cloth simulation. Section 4 compares the four preconditioning methods on four standard benchmarks and also demonstrates the effectiveness of parallelizing the incomplete poisson preconditioning scheme. Finally, Section 5 concludes with a discussion of future work.

2 Related Work

Early work by [20–22] has applied techniques from mechanical engineering and finite element communities to cloth simulation. Since then, there has been an extensive amount of work by different research groups [5, 7, 10, 23] that have addressed several aspects of simulating cloth. An extensive overview of cloth simulation techniques can be found in two survey papers [8, 16].

Preconditioners play a very important part in implicit cloth simulation as they can greatly speed up convergence of numerical methods. The work in [3] used a simple diagonal preconditioner for the modified preconditioned conjugate gradient method (MPCG). The work in [9] demonstrated 20% speedup by using a 3×3 block diagonal preconditioner. This work was extended in [4] by proposing an approximation of the filter matrix A of the MPCG. The work in [13] demonstrates the effectiveness of the incomplete Cholesky and Successive Symmetric over Relaxation (SSOR) preconditioning schemes by reducing the number of iterations by 20%.

Relation to Prior Work. In this paper, we first examine the fitness of three commonly used preconditioning schemes [3, 13] in comparison to the proposed incomplete poisson preconditioner. Our simulation method is similar to the implicit simulation method described in [2,3]. We perform collision detection using distance fields [11]. Collision resolution is performed using the techniques described in [6] and [18].

3 Cloth Simulation Overview

There are many aspects to cloth simulation. A cloth simulator is required to solve a linear system of equations which is used to step the simulator forward by one time step. This system of equations is derived taking into account the specifics of the internal forces and their derivatives. Different soft and hard constraints are imposed on the simulation which must be met. Collision detection and resolution is another area of research that has many contributions. We refer the reader to excellent works in cloth simulation research [3,12,14,15] for more information. In this paper, we focus on the methods of preconditioning that are used to accelerate the preconditioned conjugate gradient solver. Section 3.1 presents an overview of the preconditioned conjugate gradient solver and Section 3.2 describes the different methods of preconditioning for cloth simulation.

3.1 Preconditioned Conjugate Gradient Solver

An overview of the preconditioned conjugate gradient solver is shown in Algorithm 1. A detailed description of the algorithm can be found here [19]. The preconditioned conjugate gradient method takes as input the following: (a) a symmetric positive semi-definite matrix $\mathbf{A}$, (b) a symmetric positive definite preconditioning matrix $\mathbf{P}$ of the same dimension as $\mathbf{A}$, and (c) a vector $\mathbf{b}$. The algorithm iteratively solves the linear system of equations, $\mathbf{Ax} = \mathbf{b}$ and the iterations stop when $|\mathbf{b} - \mathbf{Ax}| < \epsilon|\mathbf{b}|$, where ϵ is a user-defined tolerance value. The preconditioning matrix $\mathbf{P}$, which must be easily invertible, speeds convergence to the extent that $\mathbf{P}^{-1}$ approximates $\mathbf{A}$.

3.2 Preconditioning Methods

We examine the performance of three commonly used preconditioning methods: (1) `diagonal`, (2) `symmetric successive over relaxation (SSOR)`, and (3) `incomplete cholesky` against the unconditioned conjugate gradient method. We also examine a new preconditioning scheme, the incomplete Poisson preconditioner, proposed by Ament et al. [1]for the Poisson problem. Their motivation was to find an easily parallelizable preconditioner for simulations on multi-gpu systems. To the best of our knowledge, this is the first time this preconditioning scheme has been applied to cloth simulation. The mathematical formulation of these preconditioners is as follows.

Diagonal (Jacobi) Preconditioner :

$$P = \text{diag}\{\mathbf{A}\}^{-1} \text{ or } P_{i,i} = \frac{1}{A_{i,i}} \tag{1}$$

This simple preconditioning scheme approximates the inverse of a diagonal matrix. Although lacking in quality, it can be computed quickly and provides increase in performance in many cases. The computation of P^{-1} is relatively simple and this preconditioner can be subsequently applied using SpMV.

Procedure **Preconditioned Conjugate Gradient Solver**$(\mathbf{A}, \mathbf{x}, \mathbf{b}, \mathbf{P}, \epsilon)$
Input: $\mathbf{A}$: Left hand side of linear system of equations $\mathbf{Ax} = \mathbf{b}$.
Input: $\mathbf{x}$: Input constraint.
Input: $\mathbf{b}$: Right hand side of linear system of equations $\mathbf{Ax} = \mathbf{b}$.
Input: $\mathbf{P}$: Preconditioner
Input: ϵ: Maximum tolerance
Output: $\mathbf{x}$: Result.
// Initialization
$\mathbf{r} = \mathbf{b} - \mathbf{Ax}$; // residual
$\mathbf{d} = \mathbf{P}^{-1} \cdot \mathbf{r}$;
$d_{new} = \mathbf{r} \cdot \mathbf{d}$;
while $i < MAX \wedge d_{new} > \epsilon^2$ **do**
    $\mathbf{q} = \mathbf{A} \cdot \mathbf{d}$;
    $c = \mathbf{d} \cdot \mathbf{q}$; // curvature
    if $c < 0$ **then**
        return FAIL;
    else if $c == 0$ **then**
        break;
    end
    $\alpha = \frac{d_{new}}{c}$;
    $\mathbf{x} = \mathbf{x} + \alpha \cdot \mathbf{d}$;
    $\mathbf{r} = \mathbf{r} - \alpha \cdot \mathbf{q}$;
    $\mathbf{s} = \mathbf{P}^{-1} \cdot \mathbf{r}$;
    $d_{old} = d_{new}$;
    $d_{new} = \mathbf{r} \cdot \mathbf{s}$;
    if $d_{new} < 0$ **then**
        break;
    end
    $\beta = \frac{d_{old}}{d_{new}}$;
    $\mathbf{d} = \mathbf{s} + \beta \cdot \mathbf{d}$;
    $i = i + 1$;
end
if $d_{new} < 0 \vee i == MAX$ **then**
    return FAIL;
else
    return SUCCESS;
end

Algorithm 1: Preconditioned Conjugate Gradient Solver

Incomplete Cholesky Preconditioner :

$$P = \left(LL^T\right)^{-1}, \tag{2}$$

where L is the Choleshy factorization defined as follows:

$$L_{i,i} = \sqrt{A_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2}, \tag{3}$$

$$L_{i,j} = \frac{1}{L_{i,i}}(A_{i,j} - \sum_{k=1}^{i-1} L_{i,k}L_{j,k}), \quad i > j, \tag{4}$$

with the additional constraint to keep the original sparsity pattern of A. The Incomplete Cholesky is derived from the Cholesky decomposition method. A symmetric positive-definite matrix can be decomposed into the product of a lower triangular matrix and its conjugate transpose. These triangular matrices can quickly be inverted in order to solve linear systems. In that sense , the Incomplete Cholesky preconditioner approximates the full inverse of A without

incurring the cost of actually inverting it. It should be noted that P^{-1} is calculated using expensive forward and backward substitutions, which are inherent serial processes because of the triangular structures of L and L^T.

Incomplete Poisson Preconditioner :

$$P = HH^T \tag{5}$$

where

$$H = I - L\text{diag}\{A\}^{-1}. \tag{6}$$

and L is the strictly lower triangular matrix of A. This novel preconditioner has a simple structure and is kind of an approximate inverse. As a result, no substitutions are required and this preconditioner can be applied efficiently with SpMV and thread-level parallelism.

Symmetric Successive Over Relaxation :

$$P = (M1 * M2)^{-1}, \tag{7}$$

where

$$M1 = \frac{1}{\omega} * D + L, \tag{8}$$

$$M2 = \frac{1}{(2-\omega)} * (I + \omega * D^{-1} * U) \tag{9}$$

and L,U,D are the strictly lower triangular,the strictly upper triangular and the diagonal matrix of A respectively. Symmetric successive over-relaxation is a variant of the Gauss-Seidel method but with improved convergence speed. As with Incomplete Cholesky, a relatively expensive forward and backward substitution step occurs to calculate P^{-1}. It should also be noted that the choice of ω influences convergence. We use the following ω:

$$\omega = \frac{1}{max([1, max(L), max(U)])}. \tag{10}$$

4 Evaluation Results

In this section we compare the proposed incomplete poisson preconditioner to the most commonly used preconditioners. Section 4.1 describes the test cases we use for the comparison. Section 4.2 evaluates the fitness of each of the preconditioning methods and Section 4.3 provides the results of parallelizing the incomplete poisson preconditioner on next generation multi-core hardware. A visual comparison of using each of the preconditioners on the benchmarks can be seen in the accompanying video. All preconditioners seem to produce results of similar quality.

4.1 Benchmarks

We use four benchmarks for the purpose of exercising the preconditioners on a variety of challenging scenarios that are frequently encountered in simulating cloth. The three benchmarks are described below.

1. **Free Fall.** This is more of baseline case, where a piece of cloth falls under gravity and come to rest on a static sphere with no tangling (Figure 2(a)).
2. **Curtain.** This case extends the previous benchmark by including fixed point constraints (Figure 2(b)).
3. **Moving Collider.** Further extending the previous case, a cloth patch hung as a curtain interacts with a moving spherical collider (Figure 2(c)). This benchmark is used to test the behavior of the simulator in a dynamic environment.
4. **Tangling.** Tangling is one of the toughest cases for cloth simulators to handle because of the complexities introduced by the multiple self-collisions (Figure 2(d)). As far as the conjugate gradient solver is concerned, for tangled states the number of nonzeros (thus the stiffness) of the matrix $A(Ax = b)$ increases significantly. The increased matrix density can significantly affect performance.

4.2 Preconditioner Evaluation

We test the performance of the preconditioners by simulating 200 frames for each of the benchmarks described above. The parameters used for the cloth simulator are described in Table 1. The inter-particle forces were shear, bend and stretch. The evaluation results are illustrated in Figure 3. From the results, it is apparent that the Incomplete Poisson preconditioner performs on par with Incomplete Cholesky for cloth simulation. Table 2 demonstrates the performance results of all preconditioning schemes with increase in number of nodes on cloth patch. Here, we see that IPP scales well with increase in resolution of cloth patch, but the incomplete Chokesly preconditioner does not. The main advantage of this novel preconditioner is that it can be easily parallelized whereas Incomplete Cholesky is inherently a serial algorithm.

Simulation Parameter	**Value**
Grid resolution	50×50
Spring Constant	1000
Inter-particle distance	0.005
Damping Factor	2
Time step	0.01
Error Threshold	10^{-15}
Mass of particle	1

Table 1: Simulation Parameters.

#Nodes	Cholesky		Poisson		SSOR		Jacobi		None	
	#Iter	Time(s)	#Iter	Time(s)	#Iter	Time(s)	#Iter	Time (s)	#Iter	Time (s)
2500	10	0.079	11	0.032	11	0.035	18	0.062	13	0.028
3600	10	0.099	11	0.046	11	0.046	18	0.080	13	0.049
4900	10	0.160	11	0.054	11	0.064	18	0.090	13	0.058
6400	10	0.133	11	0.062	11	0.078	18	0.101	13	0.064
8100	9	0.150	11	0.076	11	0.096	18	0.121	13	0.083
10000	9	0.151	11	0.084	11	0.110	18	0.138	13	0.091
19600	9	0.241	11	0.157	11	0.217	18	0.241	13	0.182
30625	9	0.482	12	0.326	11	0.359	18	0.388	13	0.337
40000	9	1.649	12	0.396	11	0.490	18	0.567	14	0.542

Table 2: Performance results (number of iterations and simulation time in seconds) for all preconditioning schemes with increase in number of nodes.

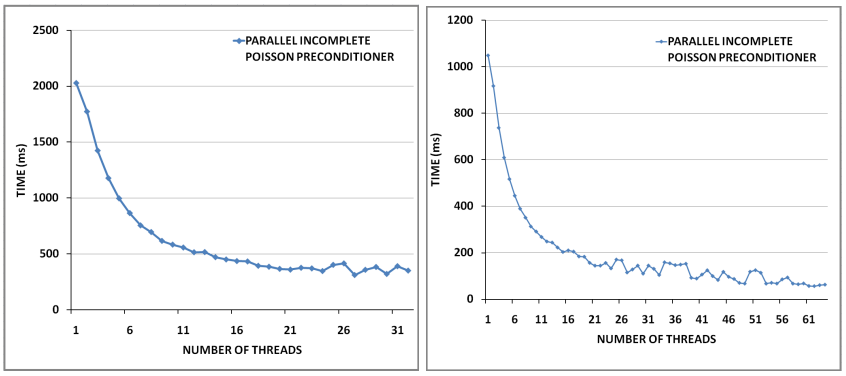
4.3 Parallelization Results

In order to evaluate parallelization options for the Incomplete Poisson preconditioner, we implemented a parallel version using pthreads.This parallel version assigns to each thread an equal number of columns of the matrix as a workload. This number is calculated as $No.Columns\ /\ No.threads$. The input of the parallelization function is A and the output $P = HH^T$, where $H = I - L\text{diag}\{A\}^{-1}$. We tested for different numbers of threads and the times we report include construction of the threads as well as thread synchronization. Our tests were performed on an 8 core Mac Pro running OS X 10.6 with 12GBs of RAM(Figure 1(a)) and a 32 core Emerald Ridge server with Intel Xeon X7560 processors and 32GMBs of RAM running OpenSuse Linux 11.3 (Figure 1(b)). Both systems are hyper-threaded. To compute execution time we used system specific high resolution timers: `mach_absolute_time()` on OS X and `clock_gettime()` on Linux. We further refer the reader to Ament et al [1] for GPU parallel implementations of the Incomplete Poisson preconditioner.

Figure 1 shows that the parallel implementation of the IPP scales very well with the number of available cores. For both systems the performance of the IPP increases significantly and reaches saturation after the number of threads equals the number of available hyper-cores. In the case of the 8-core system the performance of a single thread is about 2 seconds while the performance of 16 threads is about 0.5 seconds. Similarly, in the case of the 32-core system a single thread takes more than second while 64 threads run at about 0.06 seconds.

5 Conclusion and Future Work

We have presented a recently proposed preconditioner, the *incomplete Poisson* preconditioner (IPP), for conjugate gradient solvers. We have analyzed the fitness of the proposed preconditioning scheme on several benchmarks and compared its performance to commonly used methods. We have showed that IPP performs as well as the current state-of-the-art preconditioners, while being much



(a) (b)

Fig. 1: Parallelization results on multi-core hardware. (a) 8 core machine. (b) 32 core machine.

more amenable to standard thread-level parallelism. Our experiments on two multi-core systems show that a parallel implementation of IPP scales very well with the number of available processing cores.

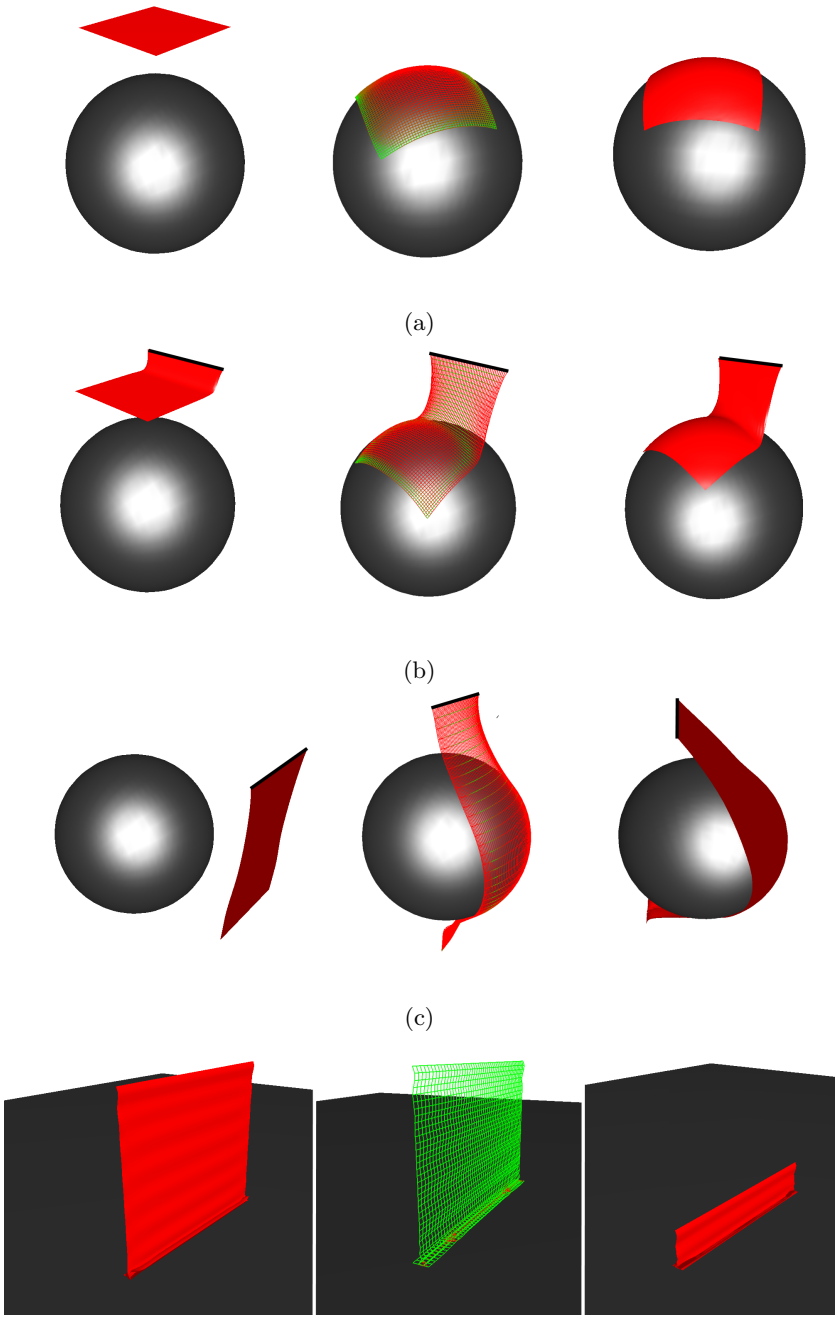
6 Acknowledgements

The work in this paper was partially supported by Intel through a Visual Computing grant, and the donation of the 32-core Emerald Ridge system with Xeon processors X7560. In particular we would like to thank Randi Rost, and Scott Buck from Intel for their support. We would like to thank Rhythm&Hues Studios and in particular Peter Huang and Tae-Yong Kim for their their support through grants and software donations. We would also like to extend our gratitude to Thanasis Vogiannou for providing an open source cloth simulation engine which was used in part to generate the results for this paper.

References

1. Ament, M., Knittel, G., Weiskopf, D., Strasser, W.: A parallel preconditioned conjugate gradient solver for the poisson problem on a multi-gpu platform. In: Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing. pp. 583–592. PDP '10, IEEE Computer Society (2010)
2. Ascher, U., Boxerman, E.: On the modified conjugate gradient method in cloth simulation. The Visual Computer 19, 526–531 (2003)
3. Baraff, D., Witkin, A.: Large steps in cloth simulation. In: Proceedings of ACM SIGGRAPH. pp. 43–54 (1998)
4. Boxerman, E.: Speeding up cloth simulation. Ph.D. thesis, The University of British Columbian, BC, Canada (2003)
5. Breen, D.E., House, D.H., Wozny, M.J., Breen, D.E.: Predicting the drape of woven cloth using interacting particles (1994)

6. Bridson, R., Fedkiw, R., Anderson, J.: Robust treatment of collisions, contact and friction for cloth animation. In: ACM SIGGRAPH 2005 Courses. SIGGRAPH '05, ACM, New York, NY, USA (2005)
7. Carignan, M., Yang, Y., Thalrnann, N.M., Thalrnanrp, D.: Dressing animated synthetic actors with complex deformable clothes. In: Computer Graphics (Proc. SIGGRAPH. pp. 99–104 (1992)
8. Choi, K., Ko, H.: Research problems in clothing simulation. Computer-Aided Design 37(6), 585–592 (2005)
9. Choi, K.J., Ko, H.S.: Stable but responsive cloth. In: Proceedings of ACM SIGGRAPH. pp. 604–611 (2002)
10. Eberhardt, B., Weber, A., Strasser, W.: A fast, flexible, particle-system model for cloth draping. IEEE Comput. Graph. Appl. 16, 52–59 (September 1996)
11. Fuhrmann, A., Sobottka, G., Grob, C.: Distance fields for rapid collision detection in physically based modeling. In: GRAPHICON (2003)
12. Goldenthal, R., Harmon, D., Fattal, R., Bercovier, M., Grinspun, E.: Efficient Simulation of Inextensible Cloth. SIGGRAPH (ACM Transactions on Graphics) 26(3) (2007)
13. Hauth, M., Etzmuss, O., Strasser, W.: Analysis of numerical methods for the simulation of deformable models. The Visual Computer 19, 581–600 (2003)
14. Müller, M.: Hierarchical position based dynamics. In: Proceedings of Virtual Reality Interactions and Physical Simulations (VRIPhys2008). pp. 13–14 (2008)
15. Müller, M., Heidelberger, B., Hennix, M., Ratcliff, J.: Position based dynamics. J. Vis. Comun. Image Represent. 18, 109–118 (April 2007)
16. Nealen, A., Müller, M., Keiser, R., Boxerman, E., Carlson, M.: Physically based deformable models in computer graphics. Computer Graphics Forum 25, 809–836 (2006)
17. Okabe, H., Imaoka, H., Tomiha, T., Niwaya, H.: Three dimensional apparel cad system. In: Proceedings of the 19th annual conference on Computer graphics and interactive techniques. pp. 105–110. SIGGRAPH '92, ACM, New York, NY, USA (1992)
18. Selle, A., Su, J., Irving, G., Fedkiw, R.: Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. IEEE Transactions on Visualization and Computer Graphics 15, 339–350 (March 2009)
19. Shewchuk, J.R.: An introduction to the conjugate gradient method without the agonizing pain. Tech. rep. (1994)
20. Terzopoulos, D., Fleischer, K.: Deformable models. The Visual Computer 4(6), 306–331 (1988)
21. Terzopoulos, D., Fleischer, K.: Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. Computer Graphics (Proc. SIGGRAPH'88) 22(4), 269–278 (1988)
22. Terzopoulos, D., Platt, J., Barr, A., Fleischer, K.: Elastically deformable models. Computer Graphics (Proc. SIGGRAPH'87) 21(4), 205–214 (1987)
23. Volino, P., Courchesne, M., Magnenat Thalmann, N.: Versatile and efficient techniques for simulating cloth and other deformable objects. In: Proceedings of ACM SIGGRAPH. pp. 137–144 (1995)



(a)

(b)

(c)

(d)

Fig. 2: Benchmark Scenes. (a) Cloth falling on a sphere. (b) Cloth hanging as a curtain colliding with sphere. (c) Cloth patch colliding with moving spherical object. (d)Tangling cloth. In the middle frames the red color indicates edges under stress.

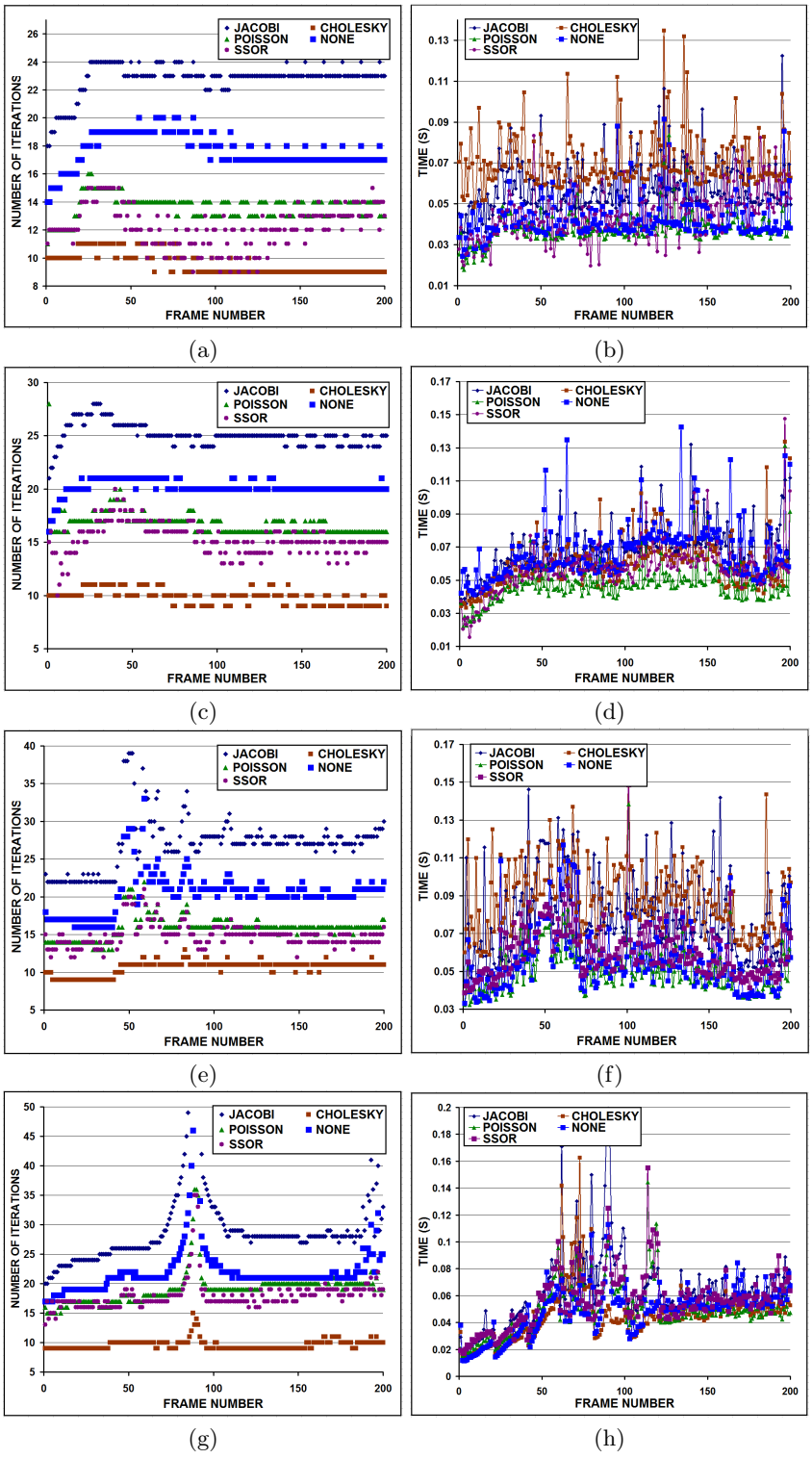


Fig. 3: Performance results of different preconditioning schemes on all benchmarks. (a),(b): Number of iterations and simulation time for Free fall benchmark. (c),(d): Curtain benchmark. (e),(f) Tangling benchmark. (g),(h) Moving collider benchmark.