Decomposing Cloth

Eddy Boxerman and Uri Ascher [1]

[1] Department of Computer Science, University of British Columbia, Vancouver, Canada

Abstract

Implicit schemes have become the standard for integrating the equations of motion in cloth simulation. These schemes, however, require the solution of a system representing the entire, fully connected cloth mesh at each time step. In this paper we present techniques that dynamically improve the sparsity of the underlying system, ultimately allowing the mesh to be decomposed into multiple components which can then be solved more efficiently and in parallel.

Our techniques include a novel adaptive implicit-explicit (IMEX) scheme which takes advantage of simulation parameters, locally in both space and time, to minimize the coupling of the system. This scheme further directly improves the efficiency of the computation at each time step. Other sparsity improvements are obtained by exploiting the physical model of Choi and Ko (2002), as well as static constraints in the system.

In addition, we present a modified *preconditioner for the modified preconditioned conjugate gradient (MPCG) technique of Baraff and Witkin (1998), improving its performance by taking constraints into account.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and RealismAnimation I.6.3 [Simulation and Modeling]: Applications

1. Introduction

For nearly two decades, researchers have sought techniques that can efficiently and realistically simulate the motion of cloth. To this end, considerable progress has been made [TPBF87, TW88, CYMTT92, BHW94, Pro95], [VCT95, EWS96, BW98, BFA02, CK02, HES02], providing a variety of viable physical models and numerical techniques. Believable animations are now expected in feature films. Games and virtual reality are next, but the computational costs are still high. In this paper we build upon certain popular cloth simulation techniques, improving their performance without sacrificing accuracy or generality. Our starting point consists of the numerical techniques of Baraff and Witkin [BW98] and the physical model of Choi and Ko [CK02].

1.1. Contributions

To date, cloth simulation methods that employ partially implicit schemes have required the solution of a system representing the entire cloth mesh at each time step. Solving

this system using the modified conjugate gradient (CG) algorithm presented in [BW98] has a computational cost of $O(n^{1.5})$, where n is the number of particles in the mesh. However, this isn't always necessary. Imagine a tablecloth draped over a square table. If we were to manipulate one corner of the cloth (assuming it does not slip with respect to the table) we would not immediately affect the opposite corner. The same applies to the case of a virtual character tapping its foot, or moving its hand — local motion often affects distant regions of the cloth only weakly and not stiffly. Since the computational cost grows faster than n it would be better if we decomposed the mesh into subsections to be solved independently.

In this paper, we employ a number of methods which increase the sparsity of the system to be solved. As a result, the mesh can often be decomposed into multiple independent components which can then be solved more efficiently and in parallel. We present the mechanisms by which the system sparsity is increased, and show how to decompose and solve the separated components (§5).

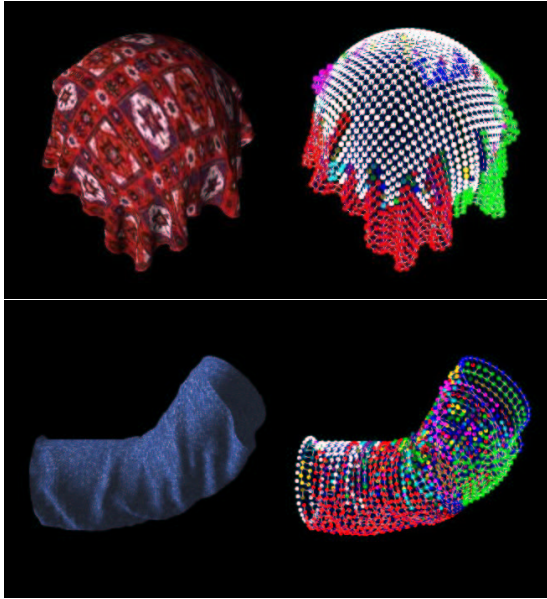Until [BW98], explicit time-stepping techniques were the

Figure 1: *Cloth draping over a sphere and "arm bend" snapshots: textured and decomposed-wireframe renderings. Each particle colour represents a separate component. White particles are fully constrained.*

norm in cloth simulation. Since then, semi-implicit techniques have dominated the field. Recently, implicit-explicit (IMEX) methods have seen use [BMF03, EEH00, HES02]. In §3 we introduce a novel "adaptive IMEX" scheme which takes advantage of simulation parameters, locally in both space and time, to improve the efficiency of the computation. Moreover, this scheme increases the sparsity of the system, making it easier to decompose the mesh. Although only first order accurate, our scheme can be readily extended to yield higher order techniques.

The modified preconditioned conjugate gradient (MPCG) technique [BW98] is widely used in the cloth simulation community. Researchers [CK02, HES02] have experimented with various preconditioners to improve the efficiency of the technique. However, these preconditioners are based on the *unconstrained* problem, thereby neglecting the *modified* nature of this CG algorithm. In §4 we design a preconditioner for the *constrained* problem and demonstrate its improved performance.

2. Cloth Model

A variety of models have been proposed in the cloth simulation literature. These can be broadly categorized into particle system and continuum formulations, although there are many differences within each type.

We have chosen to use the particle system model pre-

sented by Choi and Ko [CK02]. A distinguishing feature of this model is its unified treatment of compression and bending, thereby avoiding the "post-buckling instability" problem. This model provides attractive and stable results. Moreover, as will be seen, this model improves the potential for our decomposition method.

2.1. Model Topology

The connectivity structure is depicted in Figure 2. Each particle in the grid is connected to its four nearest neighbours by stiff *stretch* springs. Each particle is also connected to its four diagonal neighbours by (usually less stiff) *shear* springs. Finally, each particle is connected to its eight next-nearest neighbours by weak, nonlinear *bend* springs. See [CK02] for details.
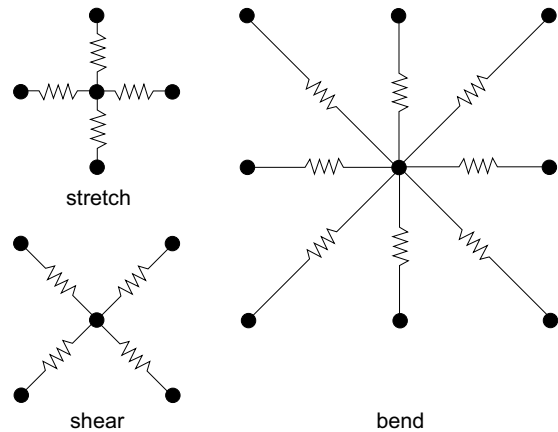

Figure 2: *model connectivity structure for* stretch, shear *and* bend *springs.*

Stretch and shear springs act only in extension. This is of relevance to our decomposition technique, as in regions of compression springs become inactive and the mesh connectivity becomes sparser.

2.2. Aerodynamics and Collision Handling

The modelling of external forces such as aerodynamics, collisions and friction are necessary for producing realistic cloth simulations. We do not contribute to these areas of research, but have implemented them in our simulator.

Aerodynamic forces are computed using the method presented in [EWS96]. We have used a voxel-based technique for cloth-cloth collision detection, similar to that proposed by [ZY00] (and also used by [CK02]). When pairs of particles are too close, a stiff, damped spring is temporarily inserted to separate them. For cloth-solid collision response, we have used the method presented in [BW98]. As for solids,

our implementation is restricted to collections of simple implicit surfaces (boxes, spheres, cylinders, etc.). Detection is thus easily performed.

3. Time Integration and the Adaptive IMEX scheme

In this section we first recall existing time-stepping schemes for cloth simulation. See Hauth et al. [HES02]. We then present a new technique, called "adaptive IMEX", which adaptively applies explicit and implicit schemes locally in both space and time to improve the efficiency of the computation.

3.1. An ODE system

Given some initial configuration of the cloth model, along with external forces, its unconstrained motion in time is generated by the ODE system

$$M\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}), \tag{1}$$

where $\ddot{\mathbf{x}}$ is the vector of particle accelerations, $\mathbf{f}$ is the force vector, and M is the mass matrix. For a cloth mesh consisting of n particles, $\ddot{\mathbf{x}}$ and $\mathbf{f}$ are vectors of size $3n$, and M is a $3n \times 3n$ matrix defined as $M = diag(m_1, m_1, m_1, m_2, m_2, m_2, \ldots, m_n, m_n, m_n)$.

Defining $\mathbf{v} \equiv \dot{\mathbf{x}}$, we rewrite (1) as

$$\frac{d}{dt} \begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ M^{-1}\mathbf{f}(\mathbf{x}, \mathbf{v}) \end{bmatrix}. \tag{2}$$

We now discuss the application of a variety of integration schemes to (2).

3.2. Explicit Integration

Almost all explicit schemes used in the cloth simulation literature are of the one-step, possibly partitioned, Runge-Kutta type [AP98]. The simplest of these schemes is the familiar forward Euler; however, for systems such as (2), a better choice is the forward-backward (FB) Euler scheme [ARS97]:

$$\begin{bmatrix} \Delta\mathbf{x}_n \\ \Delta\mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{n+1} - \mathbf{x}_n \\ \mathbf{v}_{n+1} - \mathbf{v}_n \end{bmatrix} = h \begin{bmatrix} \mathbf{v}_{n+1} \\ M^{-1}\mathbf{f}(\mathbf{x}_n, \mathbf{v}_n) \end{bmatrix}. \tag{3}$$

Here $\mathbf{x}_n$ and $\mathbf{v}_n$ denote the approximate solution at time $t = t_n$, and $t_{n+1} = t_n + h$. The update to $\mathbf{v}$ uses forward Euler, while the update to $\mathbf{x}$ uses backward Euler. Note that the method is still explicit ($\mathbf{v}_{n+1}$ is simply evaluated first). Unlike forward Euler, the FB version does *not* require the addition of damping to maintain stability. And as will be seen in §3.4, it can be incorporated more naturally within an IMEX scheme.

3.3. Implicit Integration

Almost all implicit schemes used in the cloth simulation literature are of the multi-step, BDF type. These methods are popular for solving *stiff* problems such as cloth due to their favourable stability properties, allowing for large simulation time steps to be taken. However, these methods also "smooth over" the details of the solution that they cannot capture. They also require the evaluation of $\mathbf{f}(\mathbf{x}_{n+1}, \mathbf{v}_{n+1})$ at each step n, thus requiring the solution of a nonlinear system at each time step.

Two such schemes that have been used in cloth simulation are (the first order accurate) backward Euler and (the second order) BDF2.

Applying a backward Euler scheme to (2) results in

$$\begin{bmatrix} \Delta\mathbf{x}_n \\ \Delta\mathbf{v}_n \end{bmatrix} = h \begin{bmatrix} \mathbf{v}_n + \Delta\mathbf{v}_n \\ M^{-1}\mathbf{f}(\mathbf{x}_n + \Delta\mathbf{x}_n, \mathbf{v}_n + \Delta\mathbf{v}_n) \end{bmatrix} \tag{4}$$

which is a nonlinear equation in $\Delta\mathbf{x}_n$ and $\Delta\mathbf{v}_n$. A semi-implicit version of (4), adopted in [BW98], is obtained by using a first order Taylor series epproximation, replacing $\mathbf{f}(\mathbf{x}_n + \Delta\mathbf{x}_n, \mathbf{v}_n + \Delta\mathbf{v}_n)$ by $\mathbf{f}_n + \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\Delta\mathbf{x}_n + \frac{\partial\mathbf{f}}{\partial\mathbf{v}}\Delta\mathbf{v}_n$, where $\frac{\partial\mathbf{f}}{\partial\mathbf{x}}$ and $\frac{\partial\mathbf{f}}{\partial\mathbf{v}}$ are the Jacobian matrices of the particle forces with respect to position and velocity, respectively. Substituting this in (4) and rearranging, we have

$$A\Delta\mathbf{v} \equiv (I - hM^{-1}\frac{\partial\mathbf{f}}{\partial\mathbf{v}} - h^2 M^{-1}\frac{\partial\mathbf{f}}{\partial\mathbf{x}})\Delta\mathbf{v} = hM^{-1}(\mathbf{f}_n + h\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{v}_n). \tag{5}$$

This is equivalent to applying one Newton iteration for (4). Due to the local connectivity structure of the mesh, A is a sparse matrix, which is further made to be symmetric positive definite by dropping some terms from the Jacobians. The system (5) is solved in [BW98] at each time step using a MPCG algorithm.

This methodology has several drawbacks, and others have attempted to improve upon it. Desbrun et al. [DSB99] make further approximations to achieve an $O(n)$, unconditionally stable scheme. Their technique, however, is inaccurate and does not generalize well to large systems. Kang et al. [KCC*00] improve upon this approximation, but ultimately, they are using a single, Jacobi-like solution iteration in place of a CG one. Volino and Magnenat-Thalmann [VMT00] use a weighted implicit-midpoint method that appeared to give attractive dynamic results but which is less stable and may be difficult to tune in practice. Choi and Ko [CK02] use the more accurate BDF2, solving for $\Delta\mathbf{x}$ instead of $\Delta\mathbf{v}$. Hauth et al [HES02] also use BDF2 within an IMEX solver (more on this below), and embed their version of (5) within a Newton solver, making theirs a "fully implicit" technique.

3.4. IMEX Integration

Of course, our options are not restricted to explicit *or* implicit. An entire spectrum of implicit-explicit (IMEX) schemes, combining the two, are possible. See Ascher et al.

[ARW95, ARS97] for general references on IMEX schemes for time-dependent PDEs, and [HES02] for a presentation in the context of cloth simulation. The essential idea is to separately treat the stiff and non-stiff parts of the ODE, handling the stiff parts with an implicit method and the non-stiff parts with an explicit method. This combines the stability of an implicit scheme where needed with the simplicity of computation of an explicit scheme where possible. Hauth, Eberhardt et al. [EEH00, HES02] based their IMEX splitting on connection type: stretch springs are handled implicitly, whereas shear and bend "springs" are handled explicitly (this categorization applies to both the stretching and the damping terms).

Writing $\mathbf{f} = \mathbf{g}^I + \mathbf{g}^{II}$, where $\mathbf{g}^I$ accounts for the stiff terms and $\mathbf{g}^{II}$ are the non-stiff ones, the simplest IMEX scheme applied to Equation (2) gives

$$\left[\begin{array}{c} \Delta\mathbf{x}_n \\ \Delta\mathbf{v}_n \end{array} \right] = h \left[\begin{array}{c} \mathbf{v}_n + \Delta\mathbf{v}_n \\ M^{-1}[\mathbf{g}^I(\mathbf{x}_n + \Delta\mathbf{x}_n, \mathbf{v}_n + \Delta\mathbf{v}_n) + \mathbf{g}^{II}(\mathbf{x}_n, \mathbf{v}_n)] \end{array} \right].$$
(6)

This results in backward Euler for the stiff terms and FB Euler for the non-stiff terms.

In the case of a semi-implicit solver that uses a single Newton iteration at each time step, handling a spring connection explicitly is as simple as dropping (or zeroing) its contribution to the Jacobian matrices. Thus, the computation at each time step is reduced for such an IMEX scheme. We need not calculate the Jacobians for the explicitly handled connections. More importantly, the matrix A is sparser, so matrix-vector products (the dominant cost of the CG solver) are less expensive to compute.

3.5. Adaptive IMEX Integration

Generally, we always treat the bend springs explicitly. But shear springs vary more in relative stiffness. Treating these explicitly as well is fine when simulating fabric with a relatively small resistance to shear. But this is not the case for all materials; if these resistances are "too large", it makes sense to handle shear implicitly.

For a given stretch or shear spring, denote its stiffness by k_s, its damping by k_d and its rest length by L. Ideally we desire a stability criterion that would allow us to decide our IMEX splitting during the simulation. Moreover, in the face of adaptive simulation techniques — where h varies from step to step (as in [BW98, HES02]), or where the local mesh parameters m, L, k_s and k_d vary (as in [EWS96, EEHS00, HPH96, VB02, VL02]) — we require a criterion that can be applied locally in space and time.

A stability criterion for the FB Euler scheme as applied to our model is given by

$$\kappa = \frac{h}{m}(k_s h + 2k_d) \leq \frac{1}{2}.$$
(7)

(Note that the mesh spacing is buried within the parameters m, k_s and k_d.) This result is obtained by applying a von Neumann Fourier analysis to the FB Euler scheme and a centered spatial discretization for the corresponding, simplified PDE [HES02]

$$\rho\ddot{\mathbf{x}} = k_s \nabla^2 \mathbf{x} + k_d \nabla^2 \dot{\mathbf{x}}$$
(8)

(where ∇^2 is the Laplacian operator). See [Box03] for details.

We evaluate (7) for each spring connection at each time step, using the current time-step size h and (worst case) local parameters m, k_s and k_d. If the criterion is true, we handle that spring explicitly; if not, we handle it implicitly. Thus we are able to optimize our splitting instead of having to decide it (conservatively) a priori. To our knowledge, this is the first time an IMEX scheme has been split adaptively (ie. based on local stability criteria) in either space or time.

Examples of the sparsity structure of A for implicit and adaptive IMEX schemes are given in Figures 3 and 4 respectively. The matrices are both 300×300; each point represents a 3×3 block, and nz denotes the number of nonzero blocks.
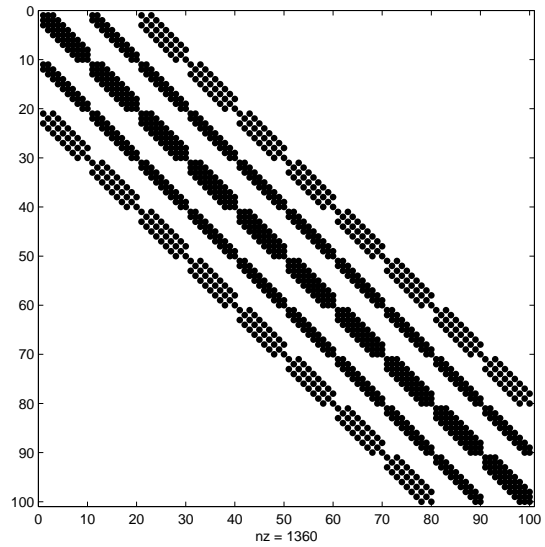

Figure 3: *Static sparsity structure for implicit scheme*

In practice we replace 0.5 by 0.2 in the stability criterion (7). This works well. In addition, this criterion is only valid for the linear stretch and shear springs; we always handle the weak bend springs explicitly, and have not experienced stability problems.

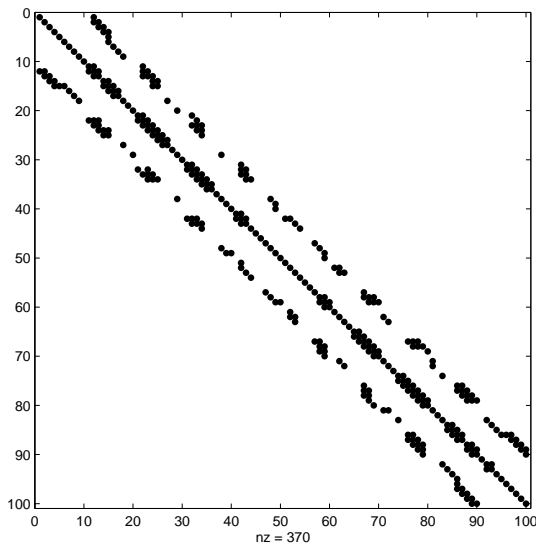Performance gains for this technique are demonstrated in §6.

Figure 4: *Sample sparsity structure for an adaptive IMEX scheme*

4. A *Modified* Preconditioner for the MPCG Technique

The partly implicit time integration techniques discussed in the previous section require the solution of a sparse linear system at each time step. The seminal paper [BW98] presents a MPCG algorithm for solving such systems in the presence of constraints. These constraints typically occur in cloth-solid contact, removing degrees of freedom from the particles' motion so as to prevent them from penetrating the solid.

In [AB03], Ascher and Boxerman give a proof of convergence for the algorithm and increase its performance through the use of an improved initial guess. The key observation in their paper is that the "constraint-filters" are *orthogonal projections*. As such, the constrained version of the problem (5) becomes

$$SA\mathbf{x} = S\mathbf{b}, \tag{9a}$$

$$(I-S)\mathbf{x} = (I-S)\mathbf{z}, \tag{9b}$$

where $\mathbf{x}$ are the velocity or position changes, S is a block diagonal projection matrix, and $\mathbf{z}$ are prescribed constraint values. Thus, for each particle the equations of motion hold only in the range subspace of S, $range(S)$, whereas in the orthogonal subspace $range(I-S)$ the given values of $\mathbf{z}$ determine those of $\mathbf{x}$.

Several researchers have attempted to improve the convergence of the MPCG algorithm by choosing a good preconditioner for A. Baraff and Witkin [BW98] used a diagonal preconditioner, $C = diag\{A\}$. Choi and Ko [CK02] used a 3×3 block diagonal preconditioner, reporting a 20% performance improvement; they also experimented with incom-

plete Cholesky (IC) and incomplete LU (ILU) factorizations [Saa96], but saw no significant performance gain. Hauth et al. [HES02] experimented with IC and symmetric successive overrelaxation (SSOR) preconditioners, both of which gave reported performance improvements of approximately 20%.

An important distinction, however, must be made between the constrained and unconstrained cases. This has not been done in the references cited above, and it is unclear which cases their results apply to.

A significant improvement in the constrained case can be realized by looking at the projected problem and choosing a preconditioner accordingly. Because S is an orthogonal projection, equations (9) can be written equivalently as

$$(SA + (I-S))\mathbf{x} = S\mathbf{b} + (I-S)\mathbf{z}. \tag{10}$$

A good preconditioner should therefore be an approximation to the matrix $SA + I - S$ rather than to A. If we let C be a preconditioner for A in the unconstrained case (5) then a *modified* preconditioner P can be defined as

$$P = SC + (I-S). \tag{11}$$

We have used for C the block diagonal matrix consisting of the 3×3 diagonal blocks of A. Then P is also an easily invertible block diagonal matrix. Experimental performance improvements for the preconditioner P are presented in §6.

5. Decomposing Cloth

Building upon the previous sections, we can now present our decomposition technique. At a given time step we seek independent subdomains such that their influence upon one another can be reduced to constant boundary conditions.

5.1. Decomposition Mechanisms

We investigate two mechanisms by which the systems described earlier may be independently decomposed: matrix reordering and separation by constraints.

5.1.1. Mechanism 1: Matrix reordering

Observe that a reordering of the rows and columns — corresponding to a different ordering of the particles — of the matrix depicted in Figure 4 yields the structure depicted in Figure 5; the two large, separate blocks of this matrix can be solved independently.

The principle thus demonstrated is general. For a solution technique such as that found in [BW98], the sparsity pattern of the matrix is fixed and this kind of separation does not occur. The methods used in this work, on the other hand, exhibit a dynamic sparsity pattern for two reasons. First, the structural springs (stretch and shear) do not act in compression [CK02]. Thus the associated Jacobian entries disappear for any compressed spring. Second, when the adaptive
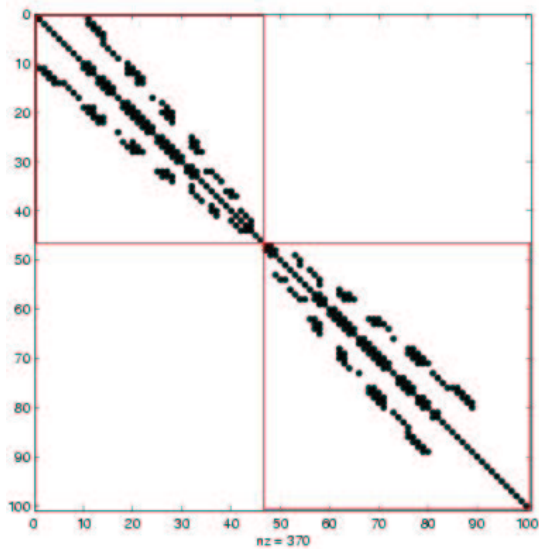
Figure 5: *Reordered, block-diagonal matrix. (Red squares highlight the two main blocks.)*

IMEX technique described in §3.5 handles spring connections explicitly, the associated Jacobian entries also disappear.

5.1.2. Mechanism 2: Separation by constraints

In many scenarios, the motion of certain cloth particles is fully prescribed (i.e., it has zero degrees of freedom). This occurs in the case of static friction as described in [BW98], or when using the "flypapering" technique of [BWK03]. The influence of such particles on the rest of the system is thus reduced to a constant for the current time step. (This is handled by imposing such constraints directly using the MPCG algorithm.)

When looking at the projected problem as described in [AB03], the rows and columns corresponding to fully constrained particles are "filtered" or projected out. (For such a particle i, the corresponding block of S is zeroed.) Thus, constrained particles decrease the size and coupling of the system. We take advantage of this fact. When decomposing the mesh, such particles do not become a member of *any* component; they simply act as boundary conditions.

5.2. Decomposing via Graph Searches

To employ the mechanisms just described, we must detect when independent decompositions are possible. This is done by searching the graph corresponding to the sparsity pattern of a given matrix.

Recall that a symmetric, n x n matrix A can be represented by an undirected graph $G(V, E)$, where V is a set of n *vertices*

and E is a set of *edges*, which are unordered pairs of vertices [GL81]. The *ordered* (or *adjacency*) *graph* of A is one for which the vertices V are numbered from 1 to n, and $i, j \in E$ if and only if $a_{ij} = a_{ji} \neq 0, i \neq j$. Figure 6 illustrates the structure of a matrix and its labeled graph.
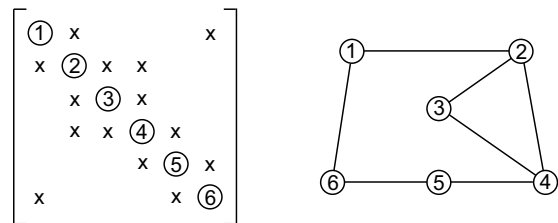

Figure 6: *A symmetric matrix A and its labeled graph, with x denoting a nonzero entry of A.*

The graph has a clear association with the original physical problem: each vertex represents a particle, and each edge represents an *active* spring handled implicitly by the solver. If a region of the cloth is connected to other regions only by explicit connections (which are considered constant throughout the time step), then it is possible to solve for that region independently.

Thus, a simple graph search is performed at each time step to determine its *connected components*. If there are two or more of those then there is a row ordering that will make the corresponding matrix block diagonal.

Next, consider two components that are connected solely via a constrained particle. Physically, these two components can not influence one another *through* the particle (they can't even influence the particle itself); they are independent during the current time step. Thus, a constrained particle acts as a *dead end* during path traversals.

Figures 7 and 8 illustrate decomposed cloth. Particles of the same colour belong to the same connected component; white particles are fully constrained.

5.3. Decomposition Algorithm

The implementation of the decomposing solver is straightforward and has $O(n)$ complexity. The main addition is the maintenance and searching of a graph which keeps track of implicit connections and constrained particles as described above. In our implementation, the associated computations add approximately a 2% overhead.

When the linear algebra system has been assembled, graph searches are performed using a colouring technique. For each connected component that is discovered, the list of particles in that component (along with pointers to the matrix A and the various vectors) is handed off to the MPCG solver. We can think of each particle as "owning" the associated row in the matrix A and the vectors **x**, **z**, etc. All operations in our
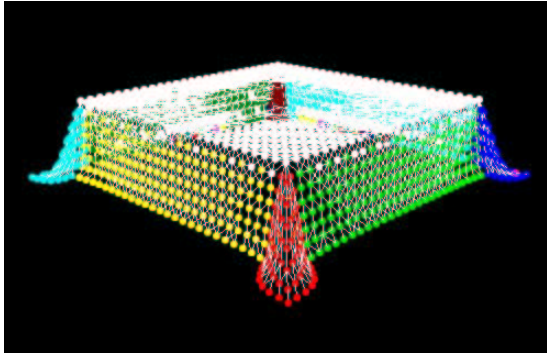
Figure 7: *Decomposed Cloth Snapshot, Example 1. Cloth draping over a square table. (Implicit stretch and explicit shear.)*
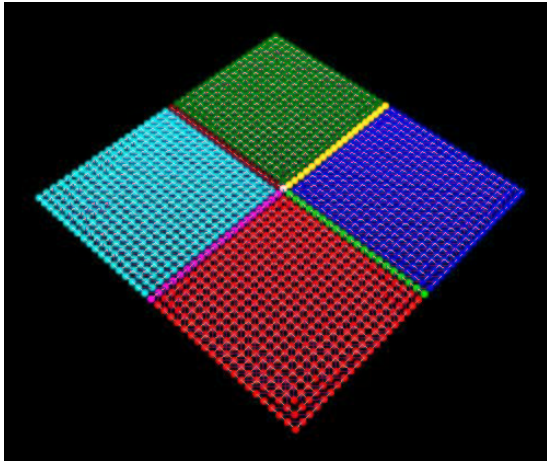

Figure 8: *Decomposed Cloth Snapshot, Example 2. The cloth is pinned at its center point and has just begun falling. (Implicit stretch and explicit shear.)*

MPCG solver (matrix/vector multiplies, inner products) are simply performed on this row subset. When all such systems have been solved, the solution vector **x** is used in updating the cloth's state.

Note that fully constrained particles are not members of any component and are *never* passed to our MPCG solver. Instead, we (first) simply compute their contribution to the solution **x** using **z**. This represents a computational saving *in itself*, as the corresponding rows of A are never included in CG iterations. Also, particle lists that are passed to the MPCG solver are first sorted (using a bin-sorting technique); this avoids excessive cache swapping issues. For additional details see [Box03].

6. Results

In this section we demonstrate the techniques presented in this paper. All experiments were run using our cloth simulator, developed in Java 1.4.1, on a 2.53GHz Pentium 4 with 2GB RAM and a GeForce4 graphics card, running Red Hat Linux 9 (Shrike).

6.1. Adaptive IMEX Results

In practice, we have found our Adaptive IMEX scheme to provide results that are stable and nearly indistinguishable from the "standard" semi-implicit scheme for cloth. Comparing the efficiency of these two schemes, adaptive IMEX typically requires 17-29% less computation time. Moreover, the sparsity of the system is improved, allowing greater possibility of decomposition.

6.2. Constrained Preconditioner Results

For the unconstrained case, P is identical to the 3×3 block diagonal preconditioner C, requiring negligible additional time to compute.

In constrained cases where particles have only 1 or 2 degrees of freedom, P performs much better. In Figure 9, we plot the number of CG iterations performed as a function of n for three preconditioners: I (i.e., no preconditioner), C and P. In this case, C actually requires *more* iterations than I. On the other hand, P still yields roughly a 30% decrease in the number of CG iterations for the larger problems (where this fact matters).
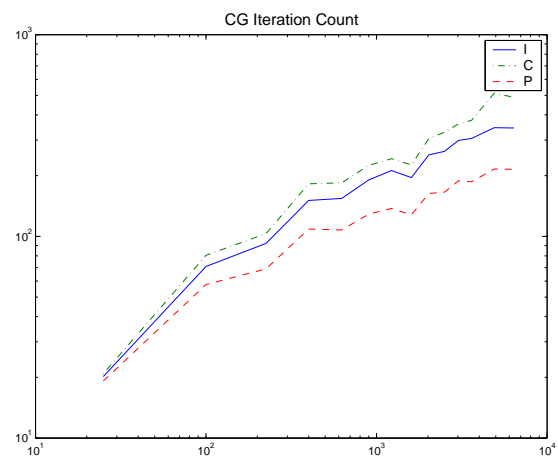

Figure 9: *Plot: CG iteration count vs. number of particles (log/log plot). Constrained case.*

6.3. Decomposition Results

For small meshes, our decomposition technique yields little or no performance improvements. For larger meshes (700+

particles), however, we have observed decompositions occurring with surprising regularity in many scenarios.

In order to measure the performance improvements provided by decomposition, the number of CG iterations cannot be used as a metric because the system sizes (and number of systems) differ. Instead, we count the number of row-vector multiplies (RVMs). This is a sensible metric and we have found it to correspond well to CG computation times.

During the course of our experiments, we often noticed small groups of particles being solved in only a few CG iterations. In addition, rows corresponding to fully constrained particles are never included in matrix-vector multiplies. These confirm expectations for gained efficiency.

Comparing our decomposing solver to a full solver (both using the adaptive IMEX scheme and the modified preconditioner), the decomposing solver performed anywhere from 0-80% fewer RVMs. This translates directly into a performance improvement of the CG solver (minus the roughly 2% overhead).

Beyond this, it is difficult to give a typical RVM reduction count, as it is highly dependent on simulation parameters and physical scenario. That said, in various "arm bend" experiments, we observed RVM reductions of 35-50%; and in "draping over a sphere" experiments, we observed reductions of 20%. Figure 1 presents textured and wireframe snapshots from these animations. We recommend stepping frame by frame through the wireframe animations to observe the decomposition process.

7. Conclusions and Further remarks

We have presented a number of techniques that improve the efficiency of cloth simulation, specifically targeting the popular semi-implicit methods. Contrary to most other attempts to do this in the literature, our methods do not sacrifice accuracy.

Our adaptive IMEX scheme — which is simple to implement — optimizes implicit-explicit splitting, thereby reducing computational costs. It also improves the sparsity of the system, making it easier to decompose. Building upon this and other mechanisms, our decomposition method offers further efficiency improvements. We have also introduced a new class of *modified* preconditioners for the MPCG algorithm, demonstrating its improved efficiency.

One of the main advantages of this technique is its adaptability to parallelism. To demonstrate this we ran a simple experiment — picking one of the test cases from above — on a dual processor machine. We have done this for three solvers: a full solver, our decomposing solver (DS1), and a small extension to our decomposing solver that embeds the MPCG algorithm within a java thread (DS2). In DS2 the main thread simply starts MPCG threads to solve the decomposed systems, waiting until they are all done before proceeding with the next time step. (In practice we only create one thread per CPU. Additional threads provide no additional benefit and introduce overhead in the form of context switching.) In our test case, DS1 required 18% less computation time than the full solver, whereas DS2 required 30% less. This is a promising initial result and further development and investigation is warranted (e.g., also computing the spring forces/Jacobians in parallel, using a larger number of processors, memory architecture impact, etc.).

Taken together, these methods can provide speedups anywhere from twenty to several hundred percent. Given this variability, an important question is how well it will do for virtual clothing — the prime application for such techniques. Based on our "arm bend" and other experiments, we believe the potential savings to be significant.

Acknowledgements

References

[AB03]   ASCHER U., BOXERMAN E.: On the modified conjugate gradient method in cloth simulation. *The Visual Computer*, 19 (2003), 526–531.

[AP98]   ASCHER U., PETZOLD L.: *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* Society for Industrial & Applied Mathematics, 1998.

[ARS97]   ASCHER U., RUUTH S., SPITERI R.: Implicit–explicit Runge–Kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics 25*, 2–3 (1997), 151–167.

[ARW95]   ASCHER U., RUUTH S., WETTON B.: Implicit-explicit methods for time-dependent pde's. *SIAM J. Numer. Anal.*, 32 (1995), 797–823.

[BFA02]   BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 594–603.

[BHW94]   BREEN D., HOUSE D., WOZNY M.: Predicting the drape of woven cloth using interacting particles. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (1994), ACM Press, pp. 365–372.

[BMF03]   BRIDSON R., MARINO S., FEDKIW R.: Simulation of clothing with folds and wrinkles. In *ACM SIGGRAPH/Eurographics Symposium Computer Animation* (2003), ACM Press, pp. 28–36.

[Box03]   BOXERMAN E.: *Speeding Up Cloth Simulation*. Master's thesis, University of British Columbia, 2003.

[BW98]   BARAFF D., WITKIN A.: Large steps in cloth simulation. In *SIGGraph* (1998), ACM, pp. 43–54.

[BWK03]   BARAFF D., WITKIN A., KASS M.: Untangling cloth. In *ACM Trans. Graphics* (2003), ACM Press, pp. 862–870.

[CK02]   CHOI K., KO H.: Stable but responsive cloth. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 604–611.

[CYMTT92]   CARIGNAN M., YANG Y., MAGNENAT-THALMANN N., THALMANN D.: Dressing animated synthetic actors with complex deformable clothes. *Computer Graphics 26*, 2 (1992), 99–104.

[DSB99]   DESBRUN M., SCHRÖDER P., BARR A.: Interactive animation of structured deformable objects. In *Graphics Interface* (1999), pp. 1–8.

[EEH00]   EBERHARDT B., ETZMUSS O., HAUTH M.: Implicit-explicit schemes for fast animation with particle systems. In *Eurographics Computer Animation and Simulation Workshop 2000* (2000).

[EEHS00]   ETZMUSS O., EBERHARDT B., HAUTH M., STRASSER W.: Collision adaptive particle systems. *Proceedings Pacific Graphics 2000* (2000).

[EWS96]   EBERHARDT B., WEBER A., STRASSER W.: A fast, flexible particle-system model for cloth draping. *IEEE Computer Graphics and Applications 16*, 5 (Sept. 1996), 52–59.

[GL81]   GEORGE A., LIU J.: *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall, 1981.

[HES02]   HAUTH M., ETZMUSS O., STRASSER W.: Analysis of numerical methods for the simulation of deformable models. *The Visual Computer* (2002). Accepted for publication.

[HPH96]   HUTCHINSON D., PRESTON M., HEWITT T.: Adaptive refinement for mass/spring simulations. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96* (1996), Springer-Verlag New York, Inc., pp. 31–45.

[KCC*00]   KANG Y., CHOI J., CHO H., LEE D., PARK C.: Real-time animation technique for flexible and thin objects. In *WSCG 2000* (2000), pp. 322–329.

[Pro95]   PROVOT X.: Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Proc. Graphics Interface* (1995), pp. 147–154.

[Saa96]   SAAD Y.: *Iterative Methods for Sparse Linear Systems*. Society for Industrial & Applied Mathematics, 1996.

[TPBF87]   TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM Press, pp. 205–214.

[TW88]   TERZOPOULOS D., WITKIN A.: Deformable models. *IEEE Computer Graphics and Applications 8*, 6 (November 1988), 41–51.

[VB02]   VILLARD J., BOROUCHAKI H.: Adaptive meshing for cloth animation. In *11th International Meshing Roundtable* (Ithaca, New York, USA, 15–18 September 2002), Sandia National Laboratories, pp. 243–252.

[VCT95]   VOLINO P., COURCHESNE M., THALMANN N.: Versatile and efficient techniques for simulating cloth and other deformable objects. In *Computer Graphics Proceedings* (1995).

[VL02]   VOLKOV V., LI L.: Adaptive local refinement and simplification of cloth meshes. In *First International Conference on Information Technology & Applications (ICITA 2002)* (2002).

[VMT00]   VOLINO P., MAGNENAT-THALMANN N.: Implementing fast cloth simulation with collision response. *IEEE Computer Society* (2000), 257–268.

[ZY00]   ZHANG D., YUEN M.: Collision detection for clothed human animation. *Proceedings Pacific Graphics 2000* (2000).