Speeding Up Cloth Simulation

by

Eddy Boxerman

B.Eng., University of McGill, 1994

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

November 2003

Abstract

Simulating the motion of cloth is an important component in virtual character animation. Believable animations are now expected in feature films. Games and virtual reality are next, but the computational costs are still high. In this thesis we present a number of methods which reduce these costs, without losing accuracy or generality.

To this end, we introduce a novel *adaptive implicit-explicit* time integration scheme, which takes advantage of simulation parameters — locally in both space and time — to improve the efficiency of the computation. Building upon this technique, we present a *decomposition* method which attempts to decouple the cloth mesh into multiple components that can be solved separately and in parallel. These techniques are introduced in the context of particle-system models, and include discussions on a variety of modelling and simulation issues.

We also significantly improve the efficiency of the modified preconditioned conjugate gradient technique often used in cloth simulation for implicit integration schemes. We present improvements in the form of a preconditioner for the constrained problem and a better initial guess.

Contents

List of Tables

List of Figures

Acknowledgements

This thesis is the result of much reading, thinking, discussing, experimenting, writing and re-writing. But it is not a journey I have had to make alone. I am grateful to the following people for their help and companionship throughout:

- My supervisors, Uri Ascher and Dinesh Pai, for their invaluable guidance, insight and support.
- My family and friends for their patience and support in my research, and the many late nights it entailed. An honourable mention goes out to the illustrious Paula Obedkoff for her late-night help in editing this thesis.
- My friends and colleagues in the SCV and Imager labs, who I was fortunate enough to work with. In particular I would like to thank Robert Bridson, Chen Greif, Michiel van de Panne and David Pritchard for many helpful suggestions and stimulating conversations.

Eddy Boxerman

The University of British Columbia
November 2003

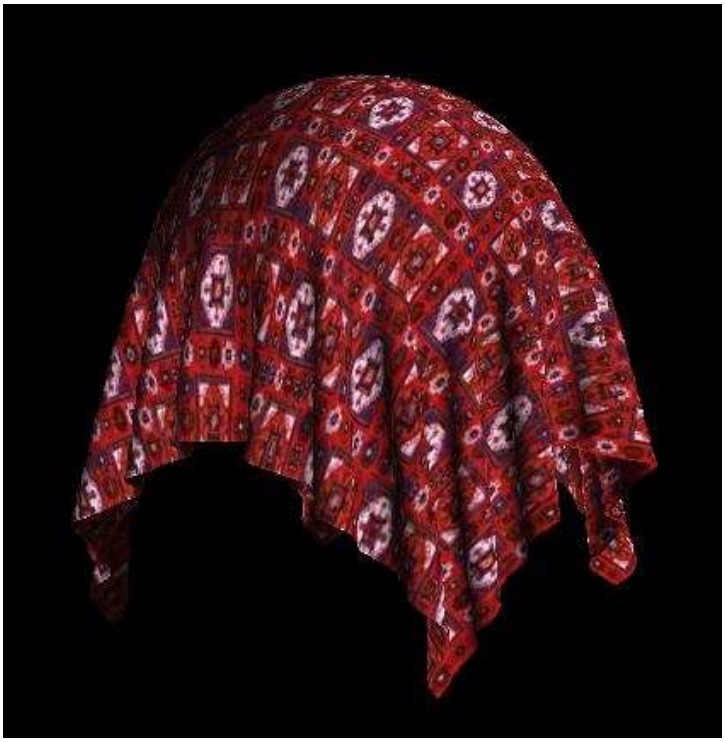

Figure 1: Cloth draping over a sphere. (Snapshot taken from our simulator.)

Chapter 1

Introduction

"The mathematician may be compared to a designer of garments, who is utterly oblivious of the creatures whom his garments may fit. To be sure, his art originated in the necessity for clothing such creatures, but this was long ago; to this day a shape will occasionally appear which will fit into the garment as if the garment had been made for it. Then there is no end of surprise and delight."

— David van Dantzig

Simulating the motion of cloth is an integral component of virtual character animation. Believable animations are now expected in feature films, and simulation relieves animators from the burden of animating this motion by hand. Garment designers and textile engineers are also interested in predicting the motion and drape of specific fabrics, thereby reducing the need to manufacture garment prototypes. Games and virtual reality applications are the next frontier.

Whereas engineers and garment designers are interested in the realistic modelling of fabrics, computer graphics researchers tend to place more emphasis on computational speed - so long as it looks realistic or attractive. Fast, accurate

and general techniques are the ideal. To this end, significant advances have been achieved, and this remains an active area of research.

In this thesis we present a number of methods which reduce the computational costs of simulating cloth, without losing accuracy or generality. We also discuss aspects of the modelling process, and its interaction with simulation methods.

The remainder of this chapter identifies the specific contributions of this thesis, as well as the experimental environment. Chapter 2 presents an overview of previous work in the area. Chapter 3 describes cloth modelling in general and the model we use in this thesis, including an improved damping model. Chapter 4 describes time-integration techniques as applied in the cloth simulation community, and presents a new "adaptive implicit-explicit" technique to improve performance. Chapter 5 illustrates a conjugate gradient technique commonly used in cloth simulation, presenting a proof and novel performance improvements to the method. Chapter 6 proposes a *decomposition* method which can further improve the performance of cloth simulators. Finally, Chapter 7 offers conclusions and future directions.

1.1 Contributions

Until [6], explicit time-stepping techniques were the norm in cloth simulation. Since then, implicit techniques have dominated the field. Recently, implicit-explicit (IMEX) techniques have also seen use [12, 19, 30]. We introduce a new type of IMEX scheme, called "adaptive IMEX," which takes advantage of the simulation parameters - locally in both space and time - to improve the efficiency of the computation. This also improves the sparsity of the system that must be solved at each time step. We build upon this and present a method that opportunistically *decomposes* the cloth

mesh into multiple components that can be solved separately and in parallel.

The modified preconditioned conjugate gradient technique introduced in [6] is widely used in the cloth simulation community. We present methods to improve the performance of this technique in the form of a preconditioner for the constrained problem, and a better initial guess.

Finally, excessive damping has been a topic of concern in cloth simulation, particularly in the context of implicit methods. This was partially alleviated by the model introduced by Choi and Ko [14], however their formulation damps rigid body rotations. We correct this by using a *projected* material damping model.

1.2 Implementation and Experiments

We have developed a cloth simulator using Java 1.4.1 which we used for all experiments described in this thesis. These were run on a 2.53GHz Pentium 4 with 2GB RAM and a GeForce4 graphics card, running Red Hat Linux 9 (Shrike).

Our results are presented within the flow of the thesis where relevant. For the reader's convenience, all experiments are listed here:

- Section 4.1.3: Stability of a forward-backward Euler Scheme.
- Section 4.2.4: Effect of Projected Damping (with Implicit Integration).
- Section 4.4.3: Stability, Results and Performance of an AIMEX Scheme.
- Section 5.3.1: Constrained Preconditioner: Unconstrained and Constrained Cases.
- Section 6.4: Decomposing Cloth: Costs and Performance.

Chapter 2

Previous Work

Over the past twenty years, the computer graphics community has applied a variety of techniques to the cloth simulation problem. In this section we provide a brief overview of the relevant research in the field. More in-depth expositions and histories can be found, where relevant, throughout this thesis.

It should be noted that the engineering community has also approached this problem, generally from a more quantitative point of view, typically employing finite element methods (FEM). Although we briefly discuss these methods in Chapter 3, they are not investigated in this thesis. Some principal papers in this area are [5, 16, 52, 64] and Chapter 4 of [32].

A thorough survey of early work in the field can be found in Ng and Grimsdale [42]. A year later, Gibson and Mirtich [27] presented a less focused but more general survey of deformable modelling techniques in computer graphics. The second chapter of [32] presents a more recent overview of cloth simulation techniques. Finally, the most recent comprehensive summary of the field was presented in Bridson's thesis [10].

The first cloth simulation was produced in 1986 by Weil [62], who used a geo-

metric technique involving catenaries to mimic the static drape of fabrics suspended at constraint points. In the same year, Feynmann [25] employed a model based on continuum mechanics — along with a multigrid solver to minimize the energy of the system — to predict the shape of fabrics draped over simple solids.

The work of Terzopoulos et al. [53, 54] followed, providing a more general solution — also based on continuum mechanics — to predict the dynamics of one, two and three-dimensional elastic models. They employed Lagrange's equations of motion, finite-difference discretizations, and a semi-implicit time integration scheme in their solution. Their model also supported contact with solids and simple aerodynamic forces. Their work set the bar for further research.

Starting in the early 1990s, the group at MIRAlab in Geneva (Carignan, Volino, the Thalmanns, et al.) began tackling the problem of simulating clothing on virtual actors [13, 38, 56, 57, 60]. They began with the model of Terzopoulos et al. [53], tailoring it especially for cloth by improving on the damping formulation and collision response with solids. They also introduced self-collision detection and response for cloth. They have since continued to work on virtual clothing, adopting newer models as they emerged in the literature, and making contributions along the way. Some of these contributions include the engineering of virtual clothing design software, self-collision consistency tracking, the geometric addition of wrinkles during simulation [28], and studies on the efficiency and behaviour of various numerical methods [58, 59] (including an implicit midpoint technique).

In 1994, Breen, House and Wozny [9] presented a new cloth model, based on particle systems and springs. They argued that cloth is a mechanism of warp and weft fibres — not a continuum — and that their model is thus more appropriate. Using data from the *Kawabata Evaluation System* [37] and minimizing the

cloth's energy via a gradient descent algorithm, they predicted the static drape of real materials quite accurately. In 1996, House, DeVaul and Breen [33] applied a Newtonian formulation to particle systems to simulate cloth dynamics. However, due to the stiffness of the springs required to maintain the cloth's structure, they chose to model the structure using fixed-length constraints. They solved this using a hierarchical Lagrange multiplier technique, specifically devised for the problem. A year later, DeVaul [18] proposed an interesting iterative technique to solve the constraint model. Although particle systems have become a mainstay in cloth simulation, constraint modelling techniques have seen little use since that time.

In 1995, Provot [45] presented a simple particle system model to simulate cloth dynamics efficiently. He also tackled the issue of stiffness — which he referred to as the "super-elastic" effect — by post-processing each time step, iteratively enforcing constraints: springs that were stretched by more than 10% were relaxed (shortened), thereby stretching neighbouring springs which were then relaxed, and so on until convergence was obtained. In practice, this method usually converged with attractive results. In subsequent papers, he approached the problem of parameter estimation (with Louchet and Crochemore [40]) and collision detection and response [46].

In 1996, Eberhardt et al. [22] expanded on the work of Breen et al., expanding the model to incorporate hysteresis and creases, and improving computational efficiency.

In 1998, Baraff and Witkin [6] published their seminal cloth paper, introducing a semi-implicit time integration scheme which allowed for large simulation time steps while maintaining stability. This proved to be a robust and efficient solution to the stiffness problem. In the same paper they introduced a modified conjugate gra-

dient solver which allowed for constraint enforcement within the implicit technique, as well as a semi-continuum cloth model which handled general mesh topologies. More recently, Baraff et al. [7] introduced a post-processing technique to handle degenerate cloth contact situations.

Since [6], others [17, 35] have attempted to improve the efficiency of this approach at the cost of further accuracy. Further details are given on these, as well as the original method, in Chapters 4 and 5.

Researchers at the University of Tübingen have been actively involved in cloth research since the paper by Eberhardt et al. in 1996 [22]. Their research spans issues in modelling [20, 21, 24], collision handling [23, 41], and numerical methods [19, 30]. In particular, the numerical methods and implicit-explicit schemes presented in these papers are discussed in Chapter 4. Also, Etzsmuss [24] presents a particle system derived from a continuum mechanics formulation which is of interest in Chapter 3.

In 2002, Choi and Ko [14] — building on Baraff and Witkin's numerical methods — presented a particle system model that overcomes the "post-buckling instability" problems of previous models. Their model also uses a more realistic, non-linear bending energy formulation. We employ a similar model in this thesis, described in Chapter 3. More recently [15], they extended their model to support more general triangular meshes.

Bridson et al. [10, 11, 12] focus on maximizing the realism of cloth simulation. Their contributions include the robust handling of collisions, friction and contact, a unique implicit-explicit scheme (further discussed in Chapter 4), and other innovations.

Chapter 3

Cloth Modelling

"Truth is much too complicated to allow anything but approximations."

— John von Neumann

"The best material model of a cat is another, or preferably the same, cat."

— A. Rosenbluth, Philosophy of Science, 1945

Cloth is composed of woven threads. The weave pattern and thread types that compose a piece of cloth determine the way it looks, the way it moves (*dynamics*), the way it feels (its *hand*), etc. For an excellent introduction to woven fabrics from a modellers' perspective, see Chapter 1 of [32].

A piece of fabric can be described by its geometry and physical properties. It can be idealized as a two dimensional surface moving in three dimensional space. It can *stretch* (i.e., experience in-plane deformations: tangential to the surface), and *bend* (i.e., out-of-plane deformations: perpendicular, or normal, to the surface). Cloth's resistance to stretching is typically *much* greater than its resistance to bending. In addition, most fabrics do not resist stretch orthotropically: they stretch diagonally, or *shear*, more easily than along the fibre directions. A standard-

ized system for measuring a fabric's resistance to these deformations is the Kawabata Evaluation System [37].

Unfortunately, as with many real phenomena, it is impossible to exactly model cloth, or to simulate its motion. This would require modelling at the quantum level. Even if we fully understood the physics involved, the problem would be computationally intractable. Our goal must be more modest: we seek only to approximate the motion of cloth. The properties we seek in our physical model are:

- Fidelity, with respect to the dynamics (and statics) of real cloth.
- Efficiency, from a computational standpoint.
- Elegance, so it can be explained, understood, and implemented with a minimum of difficulty.

In this chapter, we present a brief overview of the two dominant cloth models in the literature: continuum and particle-system models. We then show a relationship between these formulations which will prove useful in the next chapter. A description of particle systems follows — in particular the model we use throughout this work, based on Choi and Ko's model [14] with an improved damping formulation. We also include a description of external forces, such as aerodynamic effects, collisions and friction.

3.1 Continuum vs. Particle-System Models

3.1.1 Continuum Model

Continuum formulations have their origins in elasticity theory, which is the study of the deformation of elastic continua [48]; it has a long and rich history which

includes such names as Euler, Bernoulli, Poisson, Green, Laplace and many others. Continuum formulations consider the body to have a homogeneous structure, which allows the underlying physics to be modelled as a system of partial differential equations.

Continuum formulations are often employed by the engineering community in modelling cloth, which they solve numerically using the finite element method. The finite element method divides the body into a set of elements and seeks to find approximations to functions which satisfy deformation equilibrium equations between the elements; continuity of the function is enforced. A variety of element-types have been used in this way, including plates, shells, and beams.

Continuum formulations, in various simplified forms, have also been employed by the graphics community. Terzopoulos et al. [53, 54] solved simplified elasticity equations using a finite difference technique. Baraff and Witkin [6] used a triangulated mesh to represent the cloth structure, using a continuum formulation on a per-triangle basis for in-plane deformation, and the angle between adjacent triangles to measure out-of-plane deformation.

3.1.2 Particle-System Model

The other common choice for modelling cloth is the so-called "particle system" (sometimes referred to as a "mass-spring" system). A simple example is shown in Figure 3.1. These models represent the body as a set of discrete point masses; the masses are interconnected by damped springs which resist deformation of the structure.

Breen et al. [9] first introduced particle systems for simulating cloth in 1994. They argued that cloth is not a homogeneous material, but a mechanism of threads

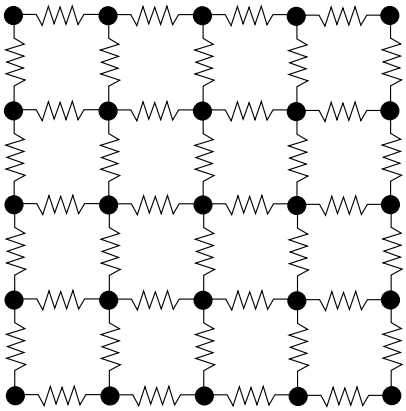

Figure 3.1: A section of a simple particle-system model for cloth

woven into an interlocking network; the fabric is not held together by molecular bonds, but by friction. They ran various simulations with their model, comparing their results with real data, and found reasonable correspondence.

Conceptually, modelling cloth as a particle system is quite intuitive, hence its appeal. Of course, particle systems found in the literature are usually more sophisticated than that depicted in Figure 3.1. A variety of different models have been used, with as many opinions as to which is best.

3.1.3 Relationship between Continuum and Particle-System Models

The whole of mathematics may be interpreted as a battle for supremacy between these two concepts [the continuous and the discrete]. This conflict may be but an echo of the older strife so prominent in early Greek philosophy, the struggle of the One to subdue the Many. But the image of a battle is not wholly appropriate, in mathematics at least, as the continuous and the discrete have frequently helped one another to progress. — E.T. Bell

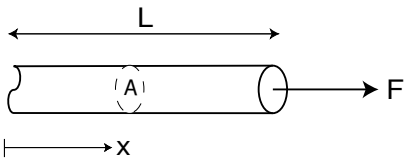

Figure 3.2: A member undergoing axial deformation

One might ask how these two popular models are related; do they give similar results? A number of researchers have investigated these questions. In [36], Kass presents a simple equivalence between these models in one dimension. In [24], Etzmuss et al. show their particle system represents a spatial semi-discretization of the continuum equation. They also provide experimental results to back this claim. Eischen and Bigliani also perform a comparison between the two models in Chapter 4 of [32]; again, results are fairly congruent. As a general rule, it seems that the differences *within* the varying particle-system models and continuum formulations are as great as the differences *between* the two categories of models. So, given the close-knit relationship between particle-system and continuum formulations, arguments about which is the better model may be immaterial.

We present here a relationship between particle-system and continuum formulations in 1D. In this case, they are identical. Material damping (energy dissipation due to internal friction) is included in this analysis. This relationship will be used in the next chapter to derive stability conditions for various time-integration schemes.

The physical scenario is depicted in Figure 3.2. We have a truss (or cloth fibre) which is constrained to deform in the axial direction. It has the following (constant) material properties:

- length L

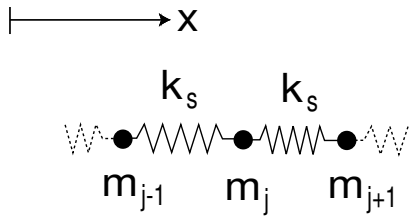

Figure 3.3: Zoomed in on the corresponding 1D particle system

- constant cross-sectional area A
- mass per unit length ρ
- Young's modulus E
- Damping coefficient β

The PDE from continuum mechanics that models this system is

$$\rho\ddot{u} = EA\frac{\partial^2 u}{\partial x^2} + \beta\frac{\partial^2 \dot{u}}{\partial x^2} \tag{3.1}$$

where u is the material strain and $\dot{u}$ is the strain rate [43].

A particle model of this system would look like Figure 3.3. The truss is divided into a set of n point-masses $m_1 \ldots m_n$, each separated by a damped spring of length h (where $h = \frac{L}{n-1}$), spring constant k_s, and damping constant k_d. Each point-mass has a mass ρh ($\frac{\rho h}{2}$ at the boundaries). The relationship between E and k_s, or between β and k_d, is not immediately evident — but will soon become apparent.

The force f on a given particle j is given by

$$\begin{aligned} f_j &= k_s(x_{j+1} - x_j - h) - k_s(x_j - x_{j-1} - h) + k_d(\dot{x}_{j+1} - \dot{x}_j) - kd(\dot{x}_j - \dot{x}_{j-1}) \\ &= k_s(x_{j+1} - 2x_j + x_{j-1}) + k_d(\dot{x}_{j+1} - 2\dot{x}_j + \dot{x}_{j-1}) \end{aligned}$$

where x_j and $\dot{x}_j$ are the respective position and velocity of the particle j.

The dynamics of the system are governed by Newton's second law $f = m\ddot{x}$. We obtain

$$\rho h\ddot{x}_j = k_s(x_{j+1} - 2x_j + x_{j-1}) + k_d(\dot{x}_{j+1} - 2\dot{x}_j + \dot{x}_{j-1}). \tag{3.2}$$

We now apply a change of variables to (3.2). For this simple 1D case, the strain u is simply the difference between a mesh point's current and original positions, given in body coordinates. Taking the truss member's left boundary as the body's origin, the relationship between the position x of a mesh point and its displacement u, along with their first and second time-derivatives, are given by

$$\begin{aligned} x_j &= \sum_{i=1}^{j} h + u_j = jh + u_j \\ \dot{x}_j &= \dot{u}_j \\ \ddot{x}_j &= \ddot{u}_j. \end{aligned}$$

Applying this to (3.2) (and cancelling jh terms), we obtain

$$\rho h\ddot{u}_j = k_s(u_{j+1} - 2u_j + u_{j-1}) + k_d(\dot{u}_{j+1} - 2\dot{u}_j + \dot{u}_{j-1}). \tag{3.3}$$

Recognizing the standard difference operator for the second order spatial derivatives $\frac{\partial^2 u_j}{\partial x^2} = \frac{1}{h^2}(u_{j+1} - 2u_j + u_{j-1})$, (3.3) becomes

$$\rho\ddot{u}_j = k_s h\frac{\partial^2 u_j}{\partial x^2} + k_d h\frac{\partial^2 \dot{u}_j}{\partial x^2} \tag{3.4}$$

which is of the same form as (3.1) for a particular point in the system.

We now see that the two formulations are identical for this simple 1D case. Moreover, we see the relation between the parameters of the continuous and the particle-system formulations:

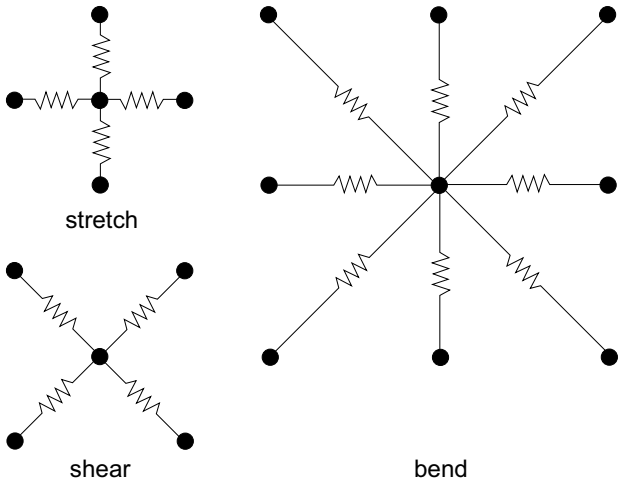

Figure 3.4: Choi and Ko model, showing the connectivity structure for *stretch*, *shear* and *bend* springs.

$$EA = k_s h \tag{3.5a}$$

$$\beta = k_d h \tag{3.5b}$$

This result will be experimentally verified and made use of, for various time-integration schemes, in the next chapter.

3.2 The Cloth Model

In this work, we employ a model similar (but not identical) to that used by Choi & Ko [14]. See Figure 3.4. 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 (less stiff) *shear* springs. Finally, each particle is connected to its eight next-nearest neighbours by (weak) non-linear *bend* springs.

Of course, other options are available. In [9], Breen et al. handled shear using angular (as opposed to axial) spring energies, and used a curvature-based energy function in the warp and weft thread directions to handle bending (similar in principle to Choi & Ko, but different in implementation). In [22], Eberhardt et al. expanded on the Breen model to include non-linear effects such as hysteresis. In [12], Bridson et al properly isolate the bending mode in a particle system by using the angle between adjacent *triangles*. In [24], Etmuss et al. handled shear and bending using finite-difference approximations for a continuum model; they also introduced a way to handle transverse contraction (non-zero Poisson ratio) in particle systems.

Although some of these other models have certain advantages over the Choi & Ko model, it is conceptually simple and has proven to give attractive results.[1]

Our formulation differs from Choi & Ko's in two respects:

1. We use different stiffnesses for the stretch and shear springs. This is a more general model; most fabrics have a lower resistance to shear; and varying the shear stiffness affects the visual behaviour of a fabric dramatically. Others have done this as well [45, 11].

2. We use a different damping model. This is further described in Section 3.3.

[1]However, despite its bending model being based on experimental data, the resulting simulations have only been evaluated — at least, in the literature — using the "eye"-norm (i.e., visual results). No comparison has yet been made against real data, nor has it been numerically compared to other models that have been more rigorously evaluated.

Spring Forces and Jacobians

As in [14], the stretch and shear springs are linear. The force acting on particle i due to the deformation between it and particle j is

$$\mathbf{f}_i = \begin{cases} k_s(|\mathbf{x}_{ij} - L|)\frac{\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|} & : \quad |\mathbf{x}_{ij}| \geq L \\ 0 & : \quad |\mathbf{x}_{ij}| < L \end{cases} \tag{3.6}$$

where $\mathbf{x}_{ij}$ is the difference between the two particles' position vectors $(\mathbf{x}_j - \mathbf{x}_i)$, and L is the spring's rest length. The Jacobian matrix of this force vector is

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} = \begin{cases} k_s\frac{\mathbf{x}_{ij}\mathbf{x}_{ij}^T}{\mathbf{x}_{ij}^T\mathbf{x}_{ij}} + k_s(1 - \frac{L}{|\mathbf{x}_{ij}|})(\mathbf{I} - \frac{\mathbf{x}_{ij}\mathbf{x}_{ij}^T}{\mathbf{x}_{ij}^T\mathbf{x}_{ij}}) & : \quad |\mathbf{x}_{ij}| \geq L \\ 0 & : \quad |\mathbf{x}_i j| < L \end{cases} \tag{3.7}$$

Note that this formulation guarantees the positive definiteness of the matrix A in (4.11).

One feature of this model is its non-linear handling of bending resistance. The equilibrium shape of buckled cloth is approximated to be a circular arc. The curvature is thus determined as a function of the axial spring strain $\frac{|x_{ij}|}{L}$, and the corresponding restorative force is (corrected here and) expressed as[2]

$$\mathbf{f}_i = \begin{cases} 0 & : \quad |\mathbf{x}_{ij}| \geq L \\ f_{bend}(\frac{|x_{ij}|}{L})k_s\frac{x_{ij}}{|x_{ij}|} & : \quad |\mathbf{x}_{ij}| < L \end{cases} \tag{3.8}$$

Choi & Ko approximated f as a fifth-order polynomial function of the axial strain. Following their methodology, we have computed this polynomial to be:

$$f_{bend}(\frac{|x_{ij}|}{L}) = f(s) = -11.541s^4 + 34.193s^3 - 39.083s^2 + 23.116s - 9.713 \tag{3.9}$$

The Jacobian matrix of this force vector is

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} = \begin{cases} 0 & : \quad |\mathbf{x}_{ij}| \geq L \\ \frac{df_{bend}}{d|\mathbf{x}_{ij}|}\frac{\mathbf{x}_{ij}\mathbf{x}_{ij}^T}{\mathbf{x}_{ij}^T\mathbf{x}_{ij}} & : \quad |\mathbf{x}_{ij}| < L \end{cases} \tag{3.10}$$

[2]Choi and Ko replaced this equation with a simple linear model for small deformations. See [14] for details. We have also done this.

where a term has been dropped to ensure its positive definiteness.

A unique feature of this model is its unification of bending and compressive resistances. Cloth is resistant to stretching, but has little resistance to compression; it responds by buckling (folding, wrinkling) out of the plane. In an attempt to model this, Choi & Ko disable any stretch (or shear) spring that is in compression; the compressive bending springs thus take over and simultaneously resist both bending and compression. While this method delivers convincing silhouettes, it does not guarantee preservation of area. In practice however (non-degenerate cases), area is generally preserved.

Damping Forces and Jacobians

We do not use Choi and Ko's damping model, but instead use a projected damping model that is presented in Section 3.3, along with the corresponding forces and Jacobians.

Note that we also take advantage of two other features specific to the Choi & Ko model:

1. All internal (material) forces are modelled using axial springs; this simplifies the stability analysis carried out in the next chapter.

2. The stiff (stretch and shear) springs are inactive in regions of the cloth that are in compression; this makes the mesh easier to *decompose*. Details on this can be found in Chapter 6.

3.3 Damping in Cloth Particle Models

Technically speaking, mass-spring systems should be called mass-spring-damper systems. Physical bodies — fabrics included — are not perfectly elastic; they dissipate energy during deformation. Thus for each ideal spring in our model, there is a corresponding ideal damper. Alternatively, we can think of the springs as being visco-elastic, thereby taking on the role of both ideal spring and damper.

An ideal spring stores the energy that deforms it, and attempts to release that energy (in an equal amount) by exerting a restorative force. An ideal damper, on the other hand, dissipates energy by opposing relative motion. For a damper connecting two particles i and j in 1D, the forces on the particles are

$$f_i = -f_j = k_d(v_j - v_i) \tag{3.11}$$

where $v = \dot{x}$ is a particle's velocity.

When extending this to a 3D cloth model, many authors [14, 17, 35, 19, 30] have simply used

$$\mathbf{f_i} = -\mathbf{f_j} = k_d(\mathbf{v_j} - \mathbf{v_i}). \tag{3.12}$$

Or worse still, some authors [45, 56] have used

$$\mathbf{f_i} = -k_d\mathbf{v_i}. \tag{3.13}$$

This is often unsatisfactory, as the model (3.12) damps rigid body rotations. In the case of cloth, this causes out-of-plane damping.[3]

Consider how cloth moves: if held under tension and then released, we do not observe it oscillating back and forth like a spring; instead it returns to rest in

[3]The model (3.13) is worse, damping *all* motion.

an unstretched state. A system that behaves in this way is categorized as critically-damped.

In [45], Provot states:

> Another lack of realism can be seen during the animation of the sheet: this "super elongation" does not come to stabilization easily, and leads to a high amplitude oscillation around the equilibrium position of the sheet. To avoid oscillation, it is therefore necessary to increase the damping coefficient C_{dis}. Though this operation can indeed suppress any oscillation, one of its shortcomings is that the sheet then looks like it has been immersed in some oily fluid and its movement loses its realism.

This has been a common complaint throughout the cloth simulation literature, especially in the context of implicit integration schemes (more on this in the next chapter); dynamic wrinkling and waving of the cloth is lost. And although this effect is partially mitigated by using (3.12) instead of (3.13), it still poses problems.

The problem can be summarized as follows: cloth resists stretching much more stiffly than bending, and as such it requires much greater spring and damping constants for its structural connections. However, the damping formulation does not behave as required: its effect "bleeds" out-of-plane, and the large magnitude of the in-plane damping coefficient impedes bending of the model.

This effect is minimized in [14] by using an extremely small damping constant. However, this can cause odd-looking, in-plane oscillations to occur, especially in "hard-constraint" situations.

All this can be easily remedied by restricting the damping to act only along

the direction of the connection, which is by definition in the plane. Thus, we use

$$\mathbf{f}_i = -\mathbf{f}_j = k_d(\frac{\mathbf{v}_{ij}^T \mathbf{x}_{ij}}{\mathbf{x}_{ij}^T \mathbf{x}_{ij}})\mathbf{x}_{ij} \tag{3.14}$$

where $\mathbf{v}_{ij} = \mathbf{v}_j - \mathbf{v}_i$. This *projects* the velocity difference onto the vector separating the particles, and only allows a force along that direction.

Damping Forces and Jacobians

In Choi and Ko's model, they simply have

$$\mathbf{f}_i = k_d(\mathbf{v}_j - \mathbf{v}_i)$$

and the Jacobian

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{v}_j} = k_d I$$

which as already stated damps rigid body rotations.

Instead, we use 3.14. The Jacobians for this formulation have the terms

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{v}_j} = k_d \frac{\mathbf{x}_{ij}\mathbf{x}_{ij}^T}{\mathbf{x}_{ij}^T \mathbf{x}_{ij}} \tag{3.15}$$

and

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} = \begin{cases} \frac{k_d}{\mathbf{x}_{ij}^T \mathbf{x}_{ij}}[\mathbf{x}_{ij}\mathbf{v}_{ij}^T + (\mathbf{x}_{ij}^T \mathbf{v}_{ij})(\mathbf{I} - 2\frac{\mathbf{x}_{ij}\mathbf{x}_{ij}^T}{\mathbf{x}_{ij}^T \mathbf{x}_{ij}})] & : \quad \mathbf{x}_{ij} \cdot \mathbf{v}_{ij} \geq 0 \\ 0 & : \quad \mathbf{x}_{ij} \cdot \mathbf{v}_{ij} < 0 \end{cases} \tag{3.16}$$

In order to maintain positive definiteness — analogously to the spring force — the damping force only acts during elongation. However, despite this filtering, we have found the inclusion of the $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ term to *detract* from the stability of our semi-implicit solver, and have therefore dropped it.

Since damping issues have been most problematic for implicit time-integration schemes, we present experimental results in that context in Section 4.2.

3.4 External Forces

The modelling of external forces such as aerodynamics, collisions and friction are necessary for producing realistic cloth simulations. This section briefly discusses our implementation of these phenomena.

3.4.1 Aerodynamic Forces

The model we have used for air resistance is a simple one, similar to [22], where the force on each particle is:

$$\mathbf{f_{air}} = \frac{1}{2}\rho c_w A(\hat{\mathbf{n}} \cdot \mathbf{v_{rel}})\mathbf{v_{rel}} \tag{3.17}$$

where ρ is the specific weight of air, c_w is the resistance coefficient, A is the surface area represented by the particle, $\hat{\mathbf{n}}$ is the unit surface normal at that point, and $\mathbf{v_{rel}}$ is the velocity of the particle with respect to an ambient wind vector.

For a more realistic treatment of aerodynamic effects in cloth simulation, see Ling's exposition in Chapter 7 of [32].

3.4.2 Collisions and Friction

A great deal of effort has been spent on collision handling in the cloth simulation community [11, 57, 41, 6, 46]. And although the subject is both challenging and interesting, we do not contribute to this area of research. We have, however, implemented collision detection, response and friction in our simulator; this section briefly describes our implementation.

Cloth-Cloth Contact

We have used a voxel-based technique for cloth-cloth collision detection, similar to that proposed by [63] (and also used by [14]). At each time step, the space enclosing the cloth is voxelised and each particle is registered in the appropriate voxel. Each particle is then tested for proximity with each other particle in its own voxel and its neighbouring voxels. If two particles lie within a given distance d_{min} from each other, a stiff, damped spring force[4] is used to separate them. These forces are handled implicitly where necessary (see Section 4.4). In practice we have found a value of $d_{min} = 0.6h$, where h is the mesh spacing, to work well. This has proven to be an efficient and surprisingly robust (if somewhat crude) method to handle most cloth-cloth contact situations. The main drawback of using a particle-particle method — rather than one that considers point-triangle and edge-edge collisions — is the "floating" effect: cloth does not appear to come into full contact with itself. However, for fine meshes this is barely noticeable.

Cloth-Solid Contact

As for solids, our implementation is restricted to collections of simple implicit surfaces (boxes, spheres, cylinders, etc.). As such, a simple set of inside-outside functions exist for each solid, against which each particle is tested. Detection is thus easily performed.

For cloth-solid collision response (including friction), we have used the method presented in [6]. When a cloth particle has penetrated a solid surface, its motion is constrained using the MPCG method (see Chapter 5) to push it to the surface.

[4]The damping used here is non-projected (i.e., Equation 3.12 is used); this roughly simulates kinetic friction. We have not implemented a solution for cloth-cloth static friction.

The constraint force is then calculated as the (unprojected) residual of the MPCG algorithm. If this force becomes attractive (i.e., causing the cloth to stick to the solid), the constraint is released. As has been noted in [30, 12], if the particle is completely ejected from the surface, a bouncing phenomenon occurs. Instead, the particle is moved some fraction of the distance to the surface. We have found a value of 0.9 to work well. This maintains cloth-solid contact so that friction can be applied.

If a particle's velocity is low relative to the colliding surface, static friction is applied: the particle becomes fully constrained ($ndof(i) = 0$). Alternatively, if the constrained tangential force f_T exceeds some fraction of the normal force f_N, such that $f_T > \mu_{static} f_N$, the particle is allowed to slide along the surface ($ndof(i) = 2$), and a kinetic friction force is applied $f_{fric} = \mu_{kinetic} f_N$ opposite the direction of relative motion.

Chapter 4

Time Integration

Given some initial configuration of the cloth, along with external forces, we wish to predict how it will move over time.

More formally, in the case of a particle system, we are working directly with a semi-discretization in space, solving an initial value problem (IVP) by integrating a set of ordinary differential equations (ODEs) in time using the *method of lines*. See Ascher and Petzold [2] for a general reference on the numerical solution of ODEs.

It is convenient to write the coupled set of ODEs as a single large system, expressed as

$$M\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}). \tag{4.1}$$

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

Many time integration techniques have been employed in the literature. Work in the late 1980s by Terzopoulos et al. [53, 54] used a semi-implicit solver. Subsequently, explicit methods — mainly explicit Euler and the classical, fourth order Runge-Kutta (RK4) — dominated the field until Baraff and Witkin [6] proposed a

semi-implicit backward Euler scheme in 1998. This scheme has favourable stability properties[1], and although it is only first order accurate and may occasionally diverge, it has provided significant improvement over previous techniques in situations where large time steps are desirable. As such, implicit methods have since become the new paradigm in cloth simulation. In [14], Choi and Ko used a second order *backward differentiation formula* (BDF2). Recently, researchers at the university of Tübingen [19, 30] have employed an implicit-explicit (IMEX) solution technique [4].

An excellent analysis of time integration techniques in the context of cloth simulation can be found in Hauth et al. [30]. Despite significant differences, their work is probably closest in spirit to our own.

There are several considerations when choosing a time integration technique, the most common ones being accuracy and stability. But there is more involved; one must also examine the nature of the true solution, and seek a solver which behaves similarly in some specific sense. For instance, one may ask: are there conserved quantities (such as energy), or is the solution damped?

In this chapter we present an overview of explicit, implicit, and IMEX schemes tailored to the context of cloth simulation. Stability and damping analyses are presented for several schemes. We then present a new IMEX technique, called "adaptive IMEX", which adaptively applies explicit and implicit schemes locally in both space and time to improve the efficiency of the computation. Experimental results are included.

[1] and perhaps not so favourable damping properties

4.1 Explicit Integration

Almost all explicit schemes used in the cloth simulation literature are of the one-step, Runge-Kutta type; these methods are based on quadrature schemes.

Given an IVP in canonical form

$$\mathbf{y}' = \phi(t, \mathbf{y}), \qquad \mathbf{y}(t_0) = \mathbf{y_0}, \tag{4.2}$$

a general, explicit, s-stage Runge-Kutta [2] scheme can be written in the form

$$\begin{aligned} \mathbf{Y}_i &= \mathbf{y}_n + k \sum_{j=1}^{i-1} a_{ij} \phi(t_n + c_j k, \mathbf{Y}_j), \qquad 1 \leq i \leq s \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + k \sum_{i=1}^{s} b_i \phi(t_n + c_i k, \mathbf{Y}_i). \end{aligned}$$

Where $\mathbf{y}_n$ is the approximate solution at time $t_n = nk$, k is the time-step size, and the $\mathbf{Y}_i$'s are intermediate approximations to the solution. The coefficients are chosen so as to maintain consistent quadrature approximations, and cancel error terms to maximize the accuracy of $\mathbf{y}_n$.

4.1.1 Forward Euler

The simplest scheme of this type is the familiar forward Euler:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + k\phi_n.$$

where $\phi_n \equiv \phi(t_n, \mathbf{y}_n)$. It is a first order accurate method. Although this scheme is rarely used in robust implementations, it serves as a convenient starting point for explanation and analysis.

The system (4.1) is a second order differential equation; in order to solve it numerically, we first put it into canonical form (4.2). Defining $\mathbf{v} \equiv \dot{\mathbf{x}}$, we re-write

(4.1) in the form (4.2)

$$\frac{d}{dt}\begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x},\mathbf{v}) \end{bmatrix}. \tag{4.3}$$

Applying forward Euler we have the following update formula:

$$\begin{bmatrix} \Delta\mathbf{x}_n \\ \Delta\mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{n+1} - \mathbf{x}_n \\ \mathbf{v}_{n+1} - \mathbf{v}_n \end{bmatrix} = k \begin{bmatrix} \mathbf{v}_n \\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}_n,\mathbf{v}_n) \end{bmatrix}. \tag{4.4}$$

Unfortunately, forward Euler has poor numerical stability properties. It relies on damping — either in the model, or artificially introduced in the scheme — to maintain stability; otherwise, the solution "explodes."

4.1.2 Forward-Backward Euler

For second order systems of ODEs such as (4.3), a better choice than forward Euler is the forward-backward (FB) Euler scheme [3]:

$$\begin{bmatrix} \Delta\mathbf{x}_n \\ \Delta\mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{n+1} - \mathbf{x}_n \\ \mathbf{v}_{n+1} - \mathbf{v}_n \end{bmatrix} = k \begin{bmatrix} \mathbf{v}_{n+1} \\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}_n,\mathbf{v}_n) \end{bmatrix}. \tag{4.5}$$

The update to $\mathbf{v}$ uses a forward Euler scheme, while the update to $\mathbf{x}$ uses a backward Euler scheme. Note that the method is still explicit ($\mathbf{v}_{n+1}$ is simply evaluated first).

In the absence of damping (i.e., the dependence of $\mathbf{f}$ on $\mathbf{v}$), the ODE (4.3) is Hamiltonian [29] and the method (4.5) is both *symplectic* and *symmetric*. In the presence of damping, these beautiful properties are lost, but the scheme is still more appropriate. Unlike forward Euler, the FB version does *not* require the addition of damping to maintain stability. And as will be seen in Section 4.3, it can be incorporated more naturally within an IMEX scheme.

It is easy to show that FB Euler is also the more "natural" choice. Assuming for the moment that $\mathbf{f}$ is a function of $\mathbf{x}$ only, upon eliminating $\mathbf{v}$ from (4.5) we

obtain

$$\mathbf{x_{n+1}} - 2\mathbf{x_n} + \mathbf{x_{n-1}} = k^2 M^{-1}\mathbf{f}(\mathbf{x}_n).$$

Doing the same for (4.4), we obtain

$$\mathbf{x_{n+1}} - 2\mathbf{x_n} + \mathbf{x_{n-1}} = k^2 M^{-1}\mathbf{f}(\mathbf{x}_{n-1}).$$

The former equation is centered as one would expect, whereas the latter is not.

4.1.3 Stability Analysis of FB Euler

A common method used in ODE analysis to determine the stability of a numerical scheme is to analyze its performance on the test equation

$$y' = \lambda y.$$

Such an analysis for various explicit and implicit schemes can be found in [2]. A more specific analysis in the context of cloth, including the calculation of eigenvalues, can be found in [30]. Here we analyze the stability of the FB Euler scheme applied to our cloth model by looking at the corresponding PDE and applying a von Neumann Fourier analysis [51].

Linearizing Equation (4.5) about the cloth's rest state (accounting only for stretch springs), and eliminating $\mathbf{v}$, we obtain

$$\mathbf{x_{n+1}} - 2\mathbf{x_n} + \mathbf{x_{n-1}} = \frac{k_s k^2}{\rho h^2}(D_+ D_-)\mathbf{x_n} + \frac{k_d k}{\rho h^2}(D_+ D_-)(\mathbf{x_n} - \mathbf{x_{n-1}}), \tag{4.6}$$

where $D_+ D_-$ is the second order finite difference approximation in two dimensions. Following the same methodology as seen in Section 3.1.3, the corresponding PDE to this discretization is

$$\rho\ddot{\mathbf{x}} = k_s \nabla^2 \mathbf{x} + k_d \nabla^2 \dot{\mathbf{x}}, \tag{4.7}$$

where ∇^2 is the Laplacian operator. Proceeding, we collect $\mathbf{x}$ terms in (4.6) as

$$\mathbf{x_{n+1}} = (2 + (\frac{k_s k^2}{\rho h^2} + \frac{k_d k}{\rho h^2})D_+ D_-)\mathbf{x_n} - (1 + \frac{k_d k}{\rho h^2} D_+ D_-)\mathbf{x_{n-1}}.$$

Applying a Fourier transform in space to this equation, it becomes

$$\hat{x}(t+k,\xi) = (2 - (a+b)\beta^2)\hat{x}(t,\xi) - (1 - b\beta^2)\hat{x}(t-k,\xi),$$

where ξ is a wave number — or mode — in the data, $a = \frac{k_s k^2}{\rho h^2}$, $b = \frac{k_d k}{\rho h^2}$, $\beta^2 = 4(\sin^2 \frac{\zeta_x}{2} + \sin^2 \frac{\zeta_y}{2})$, and $\zeta_x = \zeta_y = \xi h$ (assuming a homogeneous spatial discretization). Now, following Strikwerda's notation [51], the amplification polynomial for this scheme is

$$\Phi(g, \zeta_x, \zeta_y) = g^2 + ((a+b)\beta^2 - 2)g + (1 - b\beta^2)$$

and we demand that the magnitude of all roots of this polynomial satisfy $|g_{root_i}| \leq 1$. Applying the quadratic equation, the roots are

$$g_\pm = \frac{1}{2}(2 - (a+b)\beta^2 \pm \sqrt{(a+b)^2\beta^4 - 4a\beta^2}).$$

In order to satisfy our criterion (for all cases of ξ), we must have $(a + 2b) \leq \frac{1}{2}$. Finally, we can state our stability criterion for the discrete system as

$$\kappa = (a + 2b) = \frac{k}{m}(k_s k + 2k_d) \leq \frac{1}{2}. \tag{4.8}$$

This result[2] — although of a different form — is similar to the eigenvalue analysis in Hauth et al. [30] and the "numerical difficulty" in [59]. In this work, we refer to κ as the *numerical stiffness* of the discretization. It is an important (dimensionless, but grid dependent) parameter that will be used later in this chapter as well as in Chapter 5.

[2]note that the mesh spacing h is buried within the parameters m, k_s and k_d

Experiment: Stability of FB Euler

In this experiment we show the validity of the theoretical stability criterion (4.8). To this end, many simulations were run using a wide variety of parameters; for each set of parameters, the critically stable time step size k was found using a bisection search (accurate to within two significant digits). A time step size is deemed stable if the energy of the system does not grow unboundedly (i.e., beyond some small margin of error).

In one dimension, the test scenario resembles Figure 3.2. In this case, the discretizations resulting from the particle system and continuous models are identical. As such, the stability criterion[3] holds exactly. Results for this simple case are not included.

The results for the two and three dimensional cases are very similar, so we only include the more general, three-dimensional case here.

In three dimensions, the test scenario resembles Figure 4.1; a square sheet of cloth — held at two corners — is released from the horizontal position and swings until coming to rest. Simulation parameters are chosen randomly as a permutation of the following values:

- size of system (number of particles): 10×10, 20×20, 30×30
- h, mesh spacing (in m): 0.001, 0.01, 0.05, 0.1, 1
- ρ ($\frac{m}{h^2}$), cloth density (in $\frac{kg}{m^2}$): 0.01, 0.1, 1, 10, 100
- k_s, stretch stiffness (in N/m): 0, 10, 100, 1000, 10000
- k_s, shear stiffness (in N/m): 0, 10, 100, 1000, 10000

[3] in one dimension this is $\frac{k}{m}(k_s k + 2k_d) \leq 1$

- k_d, stretch damping (in $N \cdot s/\text{m}$): 0, 0.01, 0.1, 1, 10
- k_d, shear damping (in $N \cdot s/\text{m}$): 0, 0.01, 0.1, 1, 10

Note that the units used here are employed consistently throughout this work. These values are a superset of any realistic cloth simulation parameters.[4] We tested 500 parameter sets in this way, each simulation being run for 1000 time steps.

Evaluating κ for these parameter sets yielded the range 0.35 - 2.69. So, although the stability criterion held for most cases, for a few it did not (i.e., instability occurred within the region $\kappa \leq \frac{1}{2}$). We found that these violating cases represented very high-density, low-stiffness, low-damping materials that stretched wildly, even when using smaller time steps; these materials behave nothing like cloth. At the other extreme, values larger than 1.0 tended to represent cases where extremely small time steps were required so that little motion had an opportunity to occur (i.e., perhaps instability hadn't set in yet). We conclude that, for all intents and purposes, the stability criterion (4.8) is valid in practice.

4.1.4 Damping Analysis

The role of numerical damping has become a topic of concern in the cloth simulation community, particularly with respect to implicit methods (more on this in Section 4.2). It is enlightening to perform an analysis on the sources of damping in the numerical solution, even for a simple system, which we present here.[5]

The scenario is as depicted in Figure 4.2: a single point-mass is connected to the origin by a visco-elastic spring of zero rest-length, with spring and damping

[4] Except for bending stiffness, which is not included. However, in practice this term is too small to affect stability at current mesh resolutions.

[5] An analysis of this type is carried out in [44] for the implicit Euler method, without material damping.

Figure 4.1: Cloth suspended at two corners. (Snapshot taken from our simulator.)

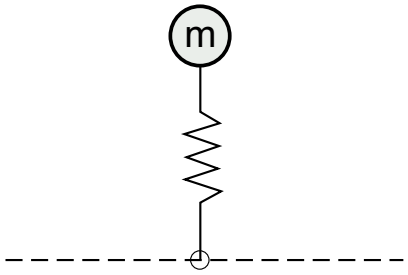

Figure 4.2: A single mass-spring system

coefficients k_s and k_d respectively. The FB Euler update formula for this system is

$$x_{n+1} = 2x_n - x_{n-1} - k^2 k_s x_n - k k_d (x_n - x_{n-1}).$$

Defining $w_{n+1} \equiv x_n$, we can rewrite this as a second order system

$$\begin{bmatrix} x \\ w \end{bmatrix}_{n+1} = \begin{bmatrix} 2 - k k_d - k^2 k_s & -1 + k k_d \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix}_n$$

The matrix appearing in this update formula is known as the *amplification matrix* A_{amp}. In order for a scheme to be stable, the eigenvalues λ_i of A_{amp} must satisfy $|\lambda_i| \leq 1$. Moreover, eigenvalues $|\lambda_i| < 1$ signify damping of the solution.

The magnitude of the eigenvalues of this system are $|\lambda_i| = 1 - k k_d$. For the case of no material damping, we see that the method does not damp either, with $|\lambda_i| = 1$. On the other hand, a similar analysis for forward Euler yields $|\lambda_i| = 1 - k(k_d - k k_s)$; we must have $k_d \geq k k_s$, otherwise the solution will grow unboundedly. As will be seen in Section 4.2, (implicit) BDF schemes do the reverse; they introduce an additional source of damping.

4.2 Implicit Integration

In recent years, implicit methods of various types have dominated the cloth simulation literature. In this section we give a brief overview of implicit methods and how they have been applied in cloth simulation. We also provide an analysis of the damping effects these methods cause, along with experimental support for using the projected damping formulation presented in Section 3.3.

4.2.1 Overview

Almost all implicit schemes used in the cloth simulation literature are of the multi-step, BDF type. They require the evaluation of $\mathbf{f}(t_{n+1}, \mathbf{y_{n+1}})$ at each step n, thus requiring the solution of a nonlinear system (for nonlinear $\mathbf{f}$) at each time step. For higher order methods, they use previous values of the solution and polynomial interpolation to improve the accuracy. Again, see [2] for a general reference on these schemes, and [30] for a presentation in the context of cloth simulation. A general k-step BDF — which has order k — can be written in the form

$$\sum_{i=0}^{k} \alpha_i \mathbf{y}_{n-i} = k\beta_0 \phi(t_{n+1}, \mathbf{y}_{n+1}),$$

Where $\alpha_0 = 1$ and $\beta_0 \neq 0$. The simplest schemes of this type — commonly used in cloth simulation — are backward Euler (order 1)

$$\mathbf{y}_{n+1} - \mathbf{y}_n = k\phi_{n+1},$$

and BDF2 (order 2)

$$\frac{3}{2}\mathbf{y}_{n+1} - 2\mathbf{y}_n + \frac{1}{2}\mathbf{y}_{n-1} = k\phi_{n+1}.$$

BDF are popular methods for solving *stiff* problems such as cloth. Although there is no formal measure for the stiffness of a problem, we can characterize it by looking at the time scales of the solution. In order to capture the details of the highest frequency mode appearing in the solution, a numerical scheme must take time steps smaller than the period of that mode. For some integration schemes, non-compliance with this restriction leads to numerical instability or "blowup". Other schemes such as BDF, as they possess stiff decay properties, simply "smooth over" the details of the solution that they cannot capture.

Stiffness typically manifests itself in the eigenvalues of the discrete system; the greater the ratio between the smallest and the largest eigenvalues (which generally correspond to low and high frequency solution modes), the stiffer the system. For the case of positive definite matrix operators, it is proportional to the condition number (see, for example, Saad [49]) of the matrix.

In the case of cloth, there are widely varying frequencies in the solution: high-frequency responses in the plane of the fabric, and low-frequency responses out of the plane. For the purposes of animation, we are not interested in visualizing the high-frequency, in-plane oscillations, but rather the low-frequency ones (e.g., waving, folding, wrinkling). In practice, BDF have proven to provide attractive results while avoiding overly prohibitive time step restrictions.[6] For the FB Euler method applied to our cloth model, the parameter κ is a reasonable quantitative measure of the system stiffness.

4.2.2 Implicit Methods in Cloth Simulation

Much effort has been spent in recent years on how to best apply implicit methods to cloth simulation. Applying a backward Euler scheme to (4.3) results in

$$\begin{bmatrix} \Delta\mathbf{x}_n \\ \Delta\mathbf{v}_n \end{bmatrix} = k \begin{bmatrix} \mathbf{v}_n + \Delta\mathbf{v}_n \\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}_n + \Delta\mathbf{x}_n, \mathbf{v}_n + \Delta\mathbf{v}_n) \end{bmatrix} \tag{4.9}$$

which is a nonlinear equation in $\Delta\mathbf{x}_n$ and $\Delta\mathbf{v}_n$. A semi-implicit version of (4.9) is obtained by using a first order Taylor series expansion of $\mathbf{f}$

$$\mathbf{f}(\mathbf{x}_n + \Delta\mathbf{x}_n, \mathbf{v}_n + \Delta\mathbf{v}_n) = \mathbf{f}_n + \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\Delta\mathbf{x}_n + \frac{\partial\mathbf{f}}{\partial\mathbf{v}}\Delta\mathbf{v}_n. \tag{4.10}$$

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. This is equivalent to applying one Newton itera-

[6] Although they do dampen frequencies in *all* directions.

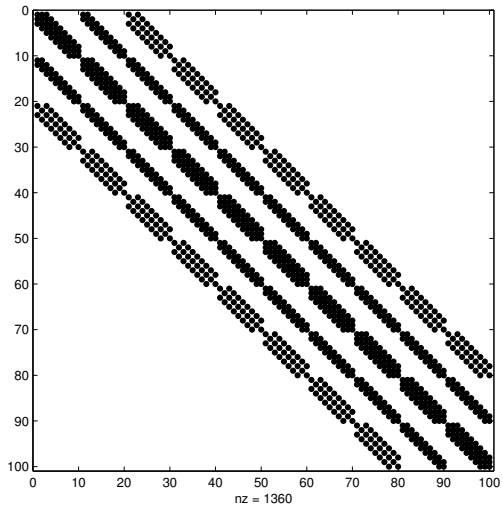

Figure 4.3: Sparsity structure of LHS of (4.11)

tion for (4.9). In [6], Baraff and Witkin adopt this idea and develop expressions for the Jacobians of the various internal forces. Due to the local connectivity structure of the mesh, these are sparse matrices, and they are further made to be symmetric positive definite by dropping some terms. Substituting this in (4.9) and rearranging, they obtained

$$A\Delta\mathbf{v} = (I - kM^{-1}\frac{\partial\mathbf{f}}{\partial\mathbf{v}} - k^2M^{-1}\frac{\partial\mathbf{f}}{\partial\mathbf{x}})\Delta\mathbf{v} = kM^{-1}(\mathbf{f}_n + k\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{v}_n). \tag{4.11}$$

The sparsity structure of this matrix is depicted in Figure 4.3 for a 10 by 10 regular mesh; each point represents a 3x3 matrix. They then proceed to solve this equation at each time step using a conjugate gradient algorithm with a reported cost of $O(n^{1.5})$; more will be said about this in Chapter 5. All this results in a practical semi-implicit method which often gives stable, visually appealing results.

The Baraff and Witkin methodology has several drawbacks, and others have attempted to improve upon it. Desbrun et al. [17] make further approximations to achieve an $O(n)$, unconditionally stable scheme. They pre-invert the matrix (for the

cloth's rest configuration) and use this solution at every time step, applying a post-correction factor for excessive deformation and global rotational momentum. Their technique, however, is inaccurate and does not generalize well to large systems. Kang et al. [35] improve upon this approximation, but ultimately, they are simply using a single, Jacobi-like solution iteration in place of a conjugate gradient one. Volino and Magnenat-Thalmann [58] used 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. Parks and Forsyth [44] used a generalized-α method in an attempt to mitigate some of the damping effects of implicit schemes, with some success. Choi and Ko [14] used the more accurate BDF2, solving for $\Delta\mathbf{x}$ instead of $\Delta\mathbf{v}$. Hauth et al [30] also use BDF2 within an IMEX solver (more on this — along with the method used by Bridson et al. [12] — in Section 4.3), and embed their version of (4.11) within a Newton solver (whereas Baraff and Witkin silently perform a single Newton iteration), making theirs more of a "fully implicit" technique.

4.2.3 Stability Analysis

In a manner similar to that presented in Section 4.1.3, it is fairly straightforward to prove that backward Euler and BDF2 are unconditionally stable when applied to (4.7). In practice, BDF have proven to be stable when applied to the full nonlinear problem.

4.2.4 Damping Analysis

As aforementioned, numerical damping caused by BDF schemes has been a concern in the cloth simulation community. One drawback to the model in [6] is that it

requires the artificial introduction of damping in order to maintain stability.[7] Choi and Ko's model eliminates this requirement, but their damping formulation is not ideal.[8].

In this section we quantitatively demonstrate the damping caused by implicit methods. We comment on this effect (both good and bad), and give experimental results on the improvement gained by using the projected damping model presented in Section 3.3.

Again considering the simple model presented in 4.1.4, for backward Euler we write the system as

$$\begin{bmatrix} 1+kk_d+k^2k_s & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix}_{n+1} = \begin{bmatrix} 2+kk_d & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix}_n$$

The magnitude of the eigenvalues of this system are $|\lambda_i| = \frac{1}{1+kk_d+k^2k_s} \leq 1$.[9] This shows us the nature of the numerical damping; even if we eliminate material damping ($k_d = 0$), the scheme will still damp the solution proportionally to k_s and k^2; for "large" values of k_s and k, the dynamics are lost. Of course, this effect is reduced if smaller time steps are used, but that partially defeats the purpose of using an implicit technique.

The problem here is subtle. Generally, we are not concerned with the fabric's in-plane oscillations.[10] Moreover, the bending stiffness of cloth is very small. So why do implicit methods damp this mode? There are three mechanisms:

[7] In fact, Bhat et al. [8] found that — when attempting to optimize simulation parameters to fit captured cloth motion — the method proved unworkable; they ended up resorting to an explicit RK4 method.

[8] And of course there is still the numerical damping associated with BDF

[9] The given value for $|\lambda_i|$ holds only if $k_d^2 < 4k_s$, otherwise the system possesses purely real eigenvalues (i.e., it is non-oscillatory). Such a system is categorized as *overdamped*; we are not really interested in this case.

[10] In fact, implicit schemes do a better job of reducing the "springiness" that earlier cloth simulations suffered from. They, in effect, change the model qualitatively.

1. Excessively large time steps. We are certainly interested in visualizing the out-of-plane behaviour of cloth; taking time steps comparable in size (within an order of magnitude) to this mode's period will damp it. (It will also produce drastically inaccurate results.)

2. In-plane stiffness and damping both "bleeding" out-of-plane, caused by inaccuracies in the numerical solution. This is more significant when using an approximate implicit solution technique; a true implicit solver — such as that published in [30] — should experience this to a much lesser degree. (This is one of the reasons for their improved results.)

3. A poor damping model. In mass-spring systems this can be remedied by using projected damping — as we will now demonstrate.

Experiment: Effect of Projected Damping (with Implicit Integration)

In this experiment we show the benefits of using the projected damping model presented in Section 3.3. To this end, we present three simulation examples: one with no model damping, one using non-projected damping, and one using projected damping.

All of these simulations are solved using the semi-implicit scheme (4.11). The configuration is "two corners pinned" as in Figure 4.1. The simulations share the following parameters in common: mesh size = 40×40 particles, $h = 0.025$, $\rho = 0.5$, k_s (stretch and shear) = 1000, k_s (bend) = 0.01, $k = 0.01$; collision response and aerodynamic forces are disabled.

The energy plots for these three cases can be seen in Figures 4.4 - 4.6. In each figure, the left plot shows the kinetic, internal, gravitational and total system energy

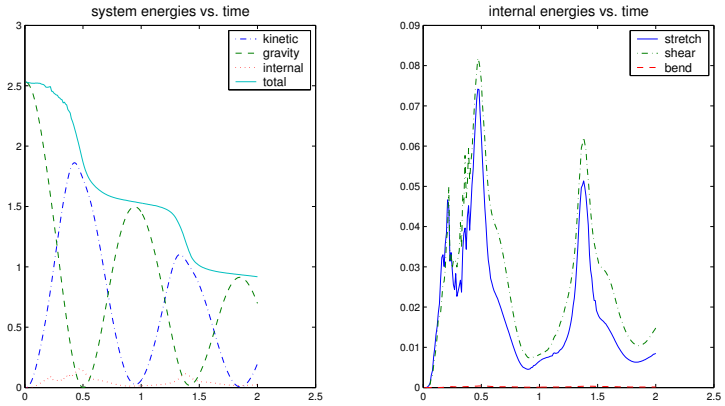

Figure 4.4: System energy plots — no damping

over time (two seconds). The right plot zooms in on the internal energies: stretch, shear and bend. Figure 4.4 shows the case for no damping; energy dissipates slowly, but spurious oscillations are evident in the internal energies. This is noticable in the corresponding animation as a (subtly) overly "bouncy" behaviour. Figure 4.5 shows the case for non-projected damping, using $k_d = 0.1$: energy dissipates far too quickly[11]. This can be mitigated by using smaller values of k_d, but then damping has little effect. Finally, Figure 4.6 shows the case for projected damping, using $k_d = 10$. Despite the large damping constant, overall system energy dissipates at approximately the same rate as for the case of no damping. Moreover, the oscillations evident in the no-damping case are eliminated — the cloth looks less "bouncy."

Visually, the differences between the no-damping and the projected-damping cases are subtle. Nevertheless, this demonstrates that if in-plane damping is desired, it is the projected formulation that should be used.

[11] And for larger values of k_d, the cloth doesn't even fall at the correct speed.

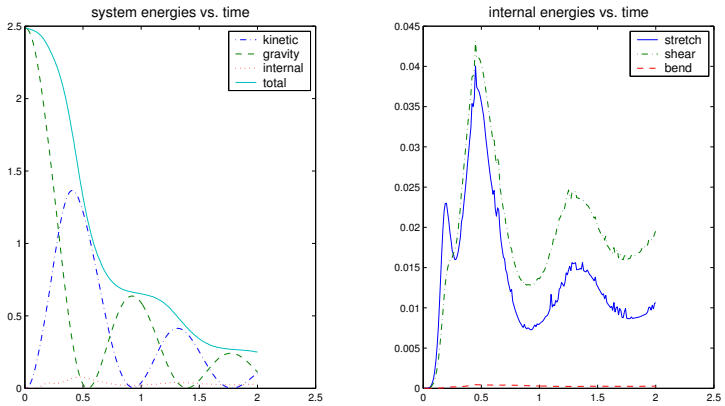

Figure 4.5: System energy plots — non-projected damping

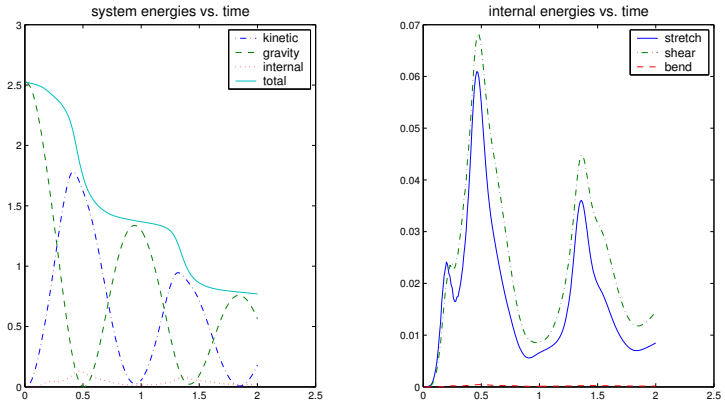

Figure 4.6: System energy plots — projected damping

4.3 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. In this section we give a brief overview of IMEX methods and how they have been applied in cloth simulation.

4.3.1 Overview

See Ascher et al. [4, 3] for general references on IMEX schemes for time-dependent PDEs. The essential idea is to separately treat the stiff and non-stiff parts of the PDE (or ODE) — to handle the stiff parts with an implicit method, and the non-stiff parts with an explicit method. Conceptually, we separate our canonical form (4.2) as

$$\mathbf{y}' = \psi(t, \mathbf{y}) + \phi(t, \mathbf{y}), \qquad \mathbf{y}(t_0) = \mathbf{y}_0, \tag{4.12}$$

where ψ is the collection of stiff terms, and ϕ is the collection of non-stiff terms. This is a common approach for solving advection-diffusion PDEs. It combines the stability of an implicit scheme where needed, and the simplicity of computation of an explicit scheme where possible.

A general, linear s-step IMEX scheme can be written as

$$\frac{1}{k}\mathbf{y}_{n+1} + \frac{1}{k}\sum_{j=0}^{s-1} a_j \mathbf{y}_{n-j} = \sum_{j=-1}^{s-1} c_j \psi(\mathbf{y}_{n-j}) + \sum_{j=0}^{s-1} b_j \phi(\mathbf{y}_{n-j}),$$

where $c_{-1} \neq 0$. Other constants are chosen so as to maintain consistency and obtain optimal order s. The simplest, first order scheme of this type is a combination of explicit and implicit Euler

$$\mathbf{y}_{n+1} = \mathbf{y}_n + k(\psi_{n+1} + \phi_n).$$

4.3.2 IMEX Methods in Cloth Simulation

Strictly speaking, all published cloth simulation techniques have been of the IMEX type; external forces such as friction and aerodynamic effects are evaluated at the current state and assumed constant throughout the time step. Typical cloth techniques have applied implicit methods only to the internal cloth energies/forces. In recent years, however, a few researchers have consciously applied IMEX schemes to cloth simulation.

Bridson et al. [11, 12] applied a similar IMEX approach to cloth as that taken for advection-diffusion equations [4].[12] They applied an implicit method to the damping term and an explicit method to the stretching term. Looking at the stability criterion (4.8), this makes sense for large k_d ($k_d \gg kk_s$), since the damping term then contributes much more than the stretching term. It is unclear if this is the best approach for cloth simulation, however, since most examples in the literature have $k_d \ll kk_s$[13]. In any case, they take time steps commensurate with the stretching stiffness term, which requires smaller time steps than what is typically used in conventional implicit solvers (but which also allows for much finer collision resolution). Their methods are thus slower than most, but produce undeniably convincing results. Many applications, however, have more stringent performance and laxer accuracy requirements.

Hauth, Eberhardt et al. [19, 30] based their IMEX splitting on connection type: stretch springs are handled implicitly, shear and bend "springs" are handled explicitly. This categorization applies to both the stretching and the damping terms. The IMEX splitting we use more closely resembles this approach.

[12] In fact, Equation (4.7) is very similar to the 2D advection-diffusion equation; although it is second order in time, it contains both a hyperbolic term and a diffusive term.

[13] Even in the high-damping experiment in Section 4.2.4, $k_d \approx kk_s$

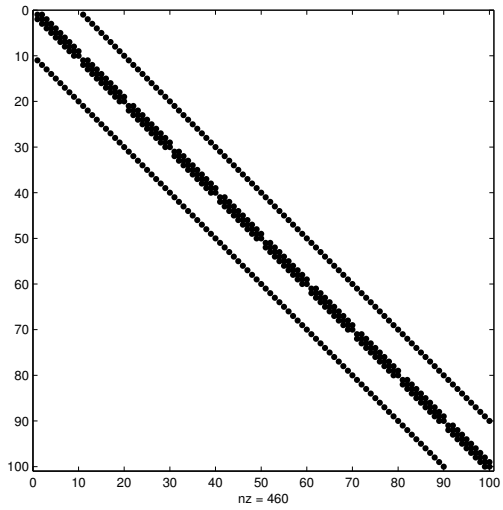

Figure 4.7: Sparsity structure of LHS of (4.11) for IMEX with implicit stretch only

A one-step IMEX scheme applied to Equation (4.3) gives

$$\begin{bmatrix} \Delta\mathbf{x}_n \\ \Delta\mathbf{v}_n \end{bmatrix} = k \begin{bmatrix} \mathbf{v}_n + \Delta\mathbf{v}_n \\ \mathbf{M}^{-1}[\mathbf{g}(\mathbf{x}_n + \Delta\mathbf{x}_n, \mathbf{v}_n + \Delta\mathbf{v}_n) + \mathbf{f}(\mathbf{x}_n, \mathbf{v}_n)] \end{bmatrix}. \tag{4.13}$$

This results in backward Euler for the stiff terms collected in $\mathbf{g}$ and FB Euler for the non-stiff terms in $\mathbf{f}$.

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. The sparsity pattern of the matrix A — when only the stretch springs are handled implicitly — is as depicted in Figure 4.7 for a 10 by 10 regular mesh (compare this to Figure 4.3). 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 conjugate gradient solver) are less expensive to compute.

4.3.3 Higher Order IMEX Methods

A second order accurate, semi-explicit BDF method for (4.12), taken from Ascher et al. [4], is

$$\mathbf{y}_{n+1} = \frac{1}{3}(4\mathbf{y}_n - \mathbf{y}_{n-1}) + \frac{2k}{3}(2\phi_n - \phi_{n-1} + \psi_{n+1}), \tag{4.14}$$

Adapting this to our second order system of ODEs, we obtain

$$\begin{bmatrix} \frac{3}{2}\mathbf{x}_{n+1} - 2\mathbf{x}_n + \frac{1}{2}\mathbf{x}_{n-1} \\ \frac{3}{2}\mathbf{v}_{n+1} - 2\mathbf{v}_n + \frac{1}{2}\mathbf{v}_{n-1} \end{bmatrix} = k \begin{bmatrix} \mathbf{v}_{n+1} \\ M^{-1}[2\mathbf{f}_n - \mathbf{f}_{n-1} + \mathbf{g}_{n+1}] \end{bmatrix} \tag{4.15}$$

Ideally, we would like a stability criterion analogous to (4.8) for this system. A stability analysis for this scheme — applied to the advection-diffusion equation — is carried out in [4]. Adapting this result to our purposes (i.e., defining and testing an adaptive IMEX scheme of order 2) is left to future work.

4.4 Adaptive IMEX Integration

Given the exposition thus far, the idea of using an adaptive IMEX (AIMEX) technique is fairly natural. Instead of deciding a-priori what IMEX splitting to apply to the governing PDE/ODE, we decide this on the fly based on the current simulation parameters and our stability criterion (4.8). Moreover — in cases where parameters vary locally in space — we do this on a per-spring-connection basis. In this section we provide details on the method, the motivations behind it, justifications for its use, and experimental results.

A note here before continuing: AIMEX schemes should be applicable to more than just cloth simulation. We posit (but do not investigate in this thesis), that they may be useful in any adaptive PDE solver, or for solving highly variable coefficient PDEs.

4.4.1 Implementation Details

Given a semi-implicit particle-system cloth simulator such as that found in Choi and Ko [14], implementing the AIMEX method is simple. When evaluating the forces applied on a pair of particles by a given spring-connection, we simply evaluate the expression (4.8)

$$\frac{k}{m}(k_s k + 2k_d) \leq 0.5.$$

If the relation is true, we skip the associated Jacobian calculation; if it is false, we evaluate the Jacobian as normal. This allows us to optimize the computation required. Following are some practical details.

- In practice, we do not want to use an explicit scheme in its marginally stable regime. So instead of using 0.5 on the right-hand-side of the equation, we typically use 0.2.

- This stability criterion is only applicable for the first order scheme (4.13). A different scheme would require the derivation and use of a different, though similar, criterion.

- The stability criterion as formulated is only applicable to axial springs, which makes the Choi and Ko model an ideal candidate to prototype this method. In the case of angular or deflection (bend) springs, a separate criterion would need to be derived and used. Moreover, although we do not investigate the Baraff and Witkin semi-continuous formulation here, we believe the criterion for this model to be very similar (only evaluated on a per-triangle basis).

- In practice, we always handle the bend springs explicitly. (Otherwise, a different stability criterion would be needed for these non-linear springs.)

- The evaluation of the criterion is a cheap computation. However, in the case where parameters do not vary locally in space (i.e., the Choi and Ko model), we can minimize the computation by evaluating the criterion once per time step and per connection type.

4.4.2 Motivation

When first experimenting with IMEX splitting, we were motivated by a simple yet encouraging result: by treating the bend springs explicitly, the performance of our simulator increased significantly. The next candidate was the shear springs. We imagine the researchers at the University of Tübingen had a similar experience. In their case, they chose to treat the shear springs explicitly as well.[14] This is fine when simulating fabric with a much smaller resistance to shear than stretch. But this is not the case for all materials; if these resistances are similar in magnitude (in the Choi and Ko model [14] they are equal), it makes sense to handle shear implicitly. Deciding this during simulation is a better option.

Clearly the motivation to use an AIMEX scheme is to minimize computation in the face of adaptive solution techniques.[15] In various examples from the literature, parameters such as the time step k, mesh-spacing h, particle mass m, and spring and damping stiffnesses k_s and k_d change during the course of the simulation:

- Adaptive Time Stepping (varying k) — Many researchers have used adaptive time stepping in the context of cloth simulation. Baraff and Witkin [6] based theirs on the proposed strain for a given time step: the state is rejected and

[14] They may have had other motivations for doing this since their shear formulation involves a more complex, four-particle relation.

[15] Of course, for non-adaptive techniques, the splitting can simply be chosen at the beginning of the simulation.

the time step halved if the cloth is stretched more than 10% its original length. Hauth et al. [30] based theirs on the convergence rate of their Newton solver, decreasing the time step upon slow convergence. Others, such as Bhat et al. [8] based theirs on the solution accuracy.

- Non-linear Springs (varying k_s) — Some researchers have used non-linear springs to improve the realism of their model. Eberhardt et al. [22] used non-linear springs (based on measured cloth data) to model hysteresis effects. Choi and Ko. [14] approximated cloth's bending response by a fifth order polynomial (3.9).

- Adaptive Mesh Spacing (varying h, thereby altering m, k_s and k_d) — Several researchers [34, 55, 61] have used adaptive local mesh refinement based on a curvature-based criterion for cloth. Etzmuss et al. [23] based their refinement on collisions.

4.4.3 AIMEX Experiments

The AIMEX method presented above is both simple and useful, but questions regarding stability, quality of results, and performance come to mind. In this section we answer these questions and provide supporting experimental evidence. Note that in these experiments, where comparisons are made against an implicit scheme, we use the semi-implicit backward Euler scheme (4.11).

Experiment: Stability of an AIMEX Scheme

First of all, can we really expect global stability based on the local stability criteria? Formally, though the local stability criterion is based in sound theory, we have no

proof for this. Experimentally, the method has worked without problem.[16] Finally, ours is not the first scheme to adjust its update formula locally based on stability criteria — upwind schemes for hyperbolic PDEs (see LeVeque [39] and references therein) do this as well, and have proven to be a very useful class of techniques.

In this experiment, we show that stability is maintained even when individual spring connections are handled variably — using either an explicit or implicit scheme — during the course of the simulation. To this end, we have run a series of simulations using adaptive time stepping. The time step size is varied (rather arbitrarily) between two extrema such that for the largest steps the stretch and shear springs are handled implicitly, whereas for the smallest time steps these springs are handled explicitly.

We have tested two different adaptive time step schemes for various simulation parameters. In the first scheme, the time step size simply alternates between the minimum and the maximum values; thus the handling of the stretch and shear springs are alternately handled explicitly and implicitly. In the second scheme, the time step size smoothly varies back and forth between the two extrema; at one point the handling of the shear springs changes, while at another the handling of the stretch springs changes. Stability was maintained for all cases.

Figures 4.8 - 4.10 are animation snapshots from one of these experiments. Wireframe images of the underlying mesh are displayed: black connections represent those that are being handled implicitly, grey connections are explicit, and missing connections are springs that are inactive due to compression[17]. Bend springs are not visualized.

[16] We suspect that for stiffly non-linear PDEs (where coefficients can change dramatically due to a small change in state), the method may fail; however, for problems of this type, we suspect that the semi-implicit technique of [6] would also fail.

[17] this is a feature of the Choi and Ko model as explained in Section 3.2

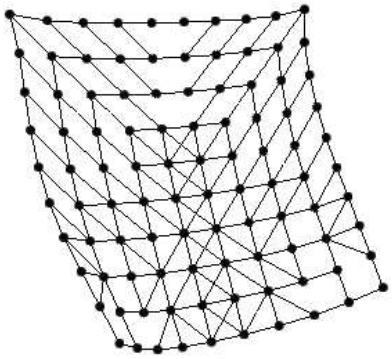

Figure 4.8: Wireframe snapshot when time step is large. All active connections are implicit (black)

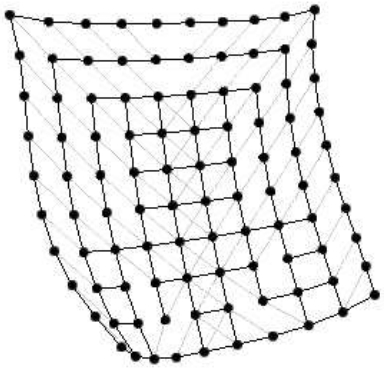

Figure 4.9: Wireframe snapshot when time step is modest. Stretch connections are implicit (black), shear connections are explicit (grey).

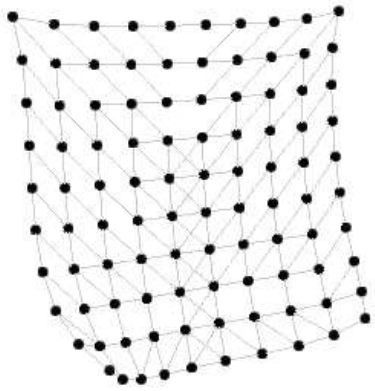

Figure 4.10: Wireframe snapshot when time step is small. All active connections are explicit (grey)

An example of the sparsity structure of A when using an AIMEX scheme (for implicit stretch and shear) is visualized in Figure 4.11.

Experiment: Visual Results of an AIMEX Scheme

Next we investigate how using an AIMEX solver affects the solution. First, there are accuracy considerations, but so long as both the explicit and implicit parts are consistent and of the same order of accuracy, this is not an issue. But there is also the *quality* of the results, such as the damping behaviour. If one area of the cloth is being solved implicitly, will it appear much more damped than an area that is being solved explicitly? Fortunately, the differences turn out to be relatively insignificant.

Analytically, we can determine this difference by looking at the magnitudes of the system eigenvalues along with the stability criterion; we seek the maximum

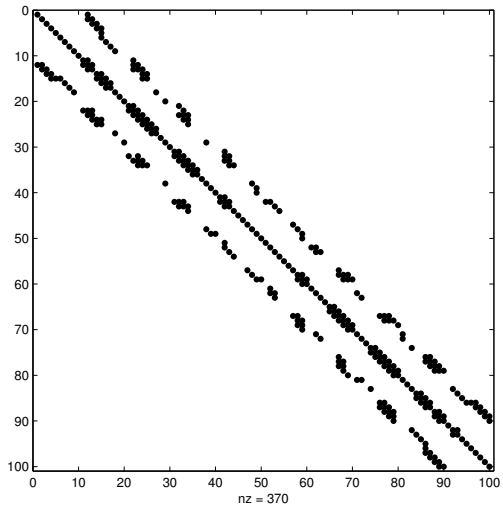

Figure 4.11: Sample sparsity structure when using an AIMEX scheme

magnitude of the difference

$$|1 - kk_d - \frac{1}{1 + kk_d + k^2 k_s}| \tag{4.16}$$

within the domain defined by $\frac{k}{m}(k_s k + 2k_d) \leq 0.5$. Clearly $(kk_d + k^2 k_s)$ is bounded by $\frac{m}{2}$, as is kk_d. Thus, for $m \ll 1$, (4.16) is also small. Therefore, when it is possible to use *either* scheme (within the stability region of both), they behave similarly.

In this experiment, we support this claim with visual results. To this end, many simulations were run using a wide variety of parameters. For each set of parameters, the variation between the AIMEX and implicit schemes is measured (using the largest infinity-norm on the position vectors over one second of simulated time). We then observe the worst cases by eye to judge the animation fidelity.

Simulation parameters are chosen randomly as a permutation of the following values:

- configuration: suspended from two corners, draping over a square table, draping over a sphere

- size of system (number of particles): 10×10, 20×20, 40×40
- h, mesh spacing (in m): 0.001, 0.01, 0.05, 0.1
- ρ ($\frac{m}{h^2}$), cloth density (in $\frac{kg}{m^2}$): 0.03, 0.1, 0.3, 1
- k_{s1}, stretch stiffness (in N/m): 10, 100, 1000, 3000
- k_{s2}, shear stiffness (in N/m): 10, 100, 1000, 3000
- k_{s3}, bend stiffness (in N/m): 0.001, 0.01, 0.1
- k_{d1}, stretch damping (in $N \cdot s/\text{m}$): 0, 0.01, 0.1, 1, 10
- k_{d2}, shear damping (in $N \cdot s/\text{m}$): 0, 0.01, 0.1, 1, 10
- k_d, air damping (in $N \cdot s/\text{m}$): 0, 0.001, 0.003, 0.01
- k, time step (in seconds): 0.0001, 0.001, 0.005, 0.01, 0.02

For most cases, the largest difference was under 10% of the mesh spacing h, which is not generally noticable to the eye. For some extreme cases — which resemble stretchy latex more than cloth — differences approached h. Not surprisingly (given the analysis above), these cases have a high mass density. Two animation snapshots are included here to show the difference in results; the grey surface is the result of the implicit solver, and the purple surface is the result of the AIMEX solver. Figure 4.12 shows the worst case scenario and Figure 4.13 shows an average case.

Experiment: Performance of an AIMEX Scheme

Finally, we investigate the performance benefits of using an AIMEX scheme. To this end, we compare it to the implicit scheme across 500 random simulations as

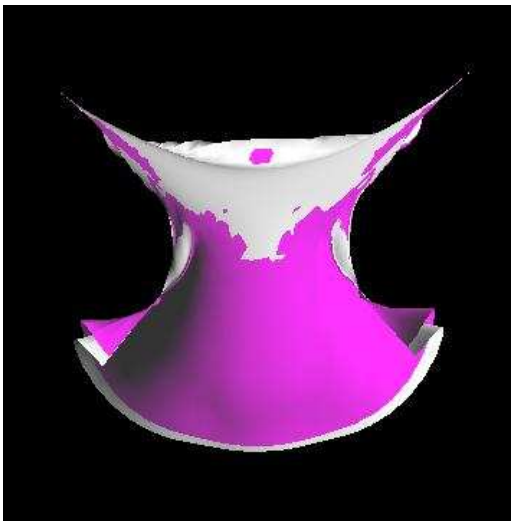

Figure 4.12: Snapshot of worst case scenario difference between AIMEX and implicit schemes. Cloth is pinned at its two upper corners (it's highly stretched), with the following parameters: size = 40×40, $\rho = 10$, $h = 0.03$, $k_{s1} = 100$, $k_{s2} = 1000$, $k_{s3} = 0.001$, $k_{d1} = 1$, $k_{d2} = 0$, $k_d = 0.001$, $k = 0.001$. Maximum error is $1.68h$.

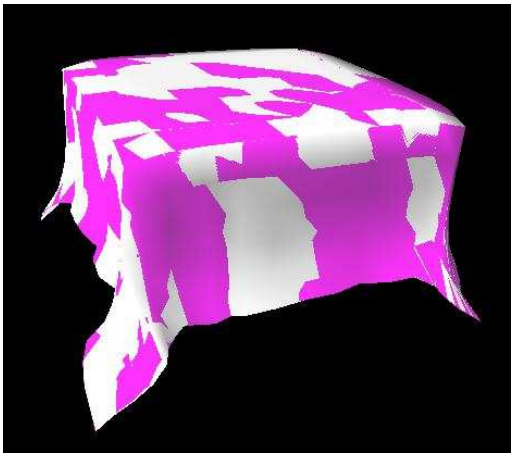

Figure 4.13: Snapshot of average case scenario difference between AIMEX and implicit schemes. Cloth is draping over a table with the following parameters: size = 20×20, $\rho = 0.1$, $h = 0.1$, $k_{s1} = 1000$, $k_{s2} = 1000$, $k_{s3} = 0.001$, $k_{d1} = 1$, $k_{d2} = 1$, $k_d = 0.003$, $k = 0.01$. Maximum error is $0.63h$ (the visual differences are due to aliasing).

in the previous experiment. For each set of parameters, we compare the number of conjugate gradient iterations and the total running time of the two schemes. The results are in table 4.1, where the "Speed Ratio" is defined as $\frac{Computation-time_{AIMEX}}{Computation-time_{implicit}}$ and the "CG Iteration Ratio" is defined as $\frac{CG-iteration-count_{AIMEX}}{CG-iteration-count_{implicit}}$. The results are divided into four categories, depending on how the AIMEX scheme handled the stretch and shear spring connections. Cloth-cloth collision handling was disabled for this series of experiments, as was rendering. Thus the computation time is dominated by the internal dynamics — with a small amount going towards cloth-solid collisions ($\approx 5\%$).

Connection Types (Stretch/Shear)	**# Runs**	**Average Speed Ratio**	**Average CG Iteration Ratio**
Implicit/Implicit	412	0.83	1.00
Implicit/Explicit	18	0.71	1.02
Explicit/Implicit	37	0.71	0.96
Explicit/Explicit	33	0.47	0.47

Table 4.1: Performance statistics: AIMEX vs. Implicit schemes

Note that for the random parameter distribution used in this experiment, the "implicit/implicit" splitting occurred much more frequently than the others. In practice, however, the "implicit/explicit" splitting is also quite common. Thus the AIMEX scheme generally requires 17-29% less computation time than the fully implicit scheme. The number of conjugate gradient iterations is essentially unaffected.[18]

[18]Except in the "explicit/explicit" case, where A becomes block diagonal for the AIMEX solver and converges in one iteration using the block-diagonal preconditioner. Note that for this fully explicit case, a CG solver is not required, but is used for simplicity/uniformity of treatment.

Chapter 5

The Modified Conjugate Gradient Method in Cloth Simulation

"I can't change the direction of the wind, but I can adjust my sails to always reach my destination."

— Jimmy Dean

The partly implicit time integration techniques discussed in the previous chapter require the solution of a sparse linear system at each time step. In their seminal paper [6], Baraff and Witkin present a modified preconditioned conjugate gradient (MPCG) algorithm for solving such systems in the presence of certain types of constraints.

In this chapter, we present a brief overview of the CG method, including preconditioning; the MPCG method and the type of constraints it supports; an overview of the proof of convergence of the MPCG method, along with some improvements

that follow, as given in Ascher and Boxerman [1]; and a new improvement to the algorithm in the form of a better preconditioner for the constrained problem — providing significantly faster convergence.

5.1 The Conjugate Gradient Method

The CG method was introduced by Hestenes and Stiefel in 1952 [31]. For a thorough exposition, see [49]. For a more "gentle" introduction, see [50].

CG is a popular iterative method for solving large systems of linear equations of the form

$$A\mathbf{x} = \mathbf{b},$$

where A is a sparse, positive-definite matrix. For systems of this type, the solution $\mathbf{x}$ is also the vector that minimizes the quadratic form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - b^T\mathbf{x} + c$$

for any scalar c. Thus we can recast this into an optimization problem; it is a method of optimized line-searches, where each direction is *A-orthogonal* — or conjugate — to all previous ones (it is thus a Krylov-space method). It provides the exact solution in n iterations for a system of size n. However, its popularity is due to its ability to provide a "reasonably" accurate solution in $O(\sqrt{n})$ iterations for systems like the ones we faced in the previous chapter. Each iteration involves one multiplication of a vector by A. Thus, for a sparse $n \times n$ system containing $O(n)$ entries, the method typically requires $O(n^{1.5})$ operations.

In addition, we can often obtain better convergence via preconditioning, a technique used to cluster the eigenvalues of A more tightly and/or reduce its condition number. Ideally, we choose a matrix P that approximates A well, but is easy

$$
\begin{aligned}
&\mathbf{x} = \mathbf{x}_0 \\
&\mathbf{r} = \mathbf{b} - A\mathbf{x} \\
&\mathbf{p} = P^{-1}\mathbf{r} \\
&b_\delta = \mathbf{r}^T\mathbf{p} \\
&\delta = b_\delta \\
&\textit{while } \delta > tol^2 b_\delta \\
&\qquad \mathbf{s} = A\mathbf{p} \\
&\qquad \alpha = \frac{\delta}{\mathbf{p}^T\mathbf{s}} \\
&\qquad \mathbf{x} = \mathbf{x} + \alpha\mathbf{p} \\
&\qquad \mathbf{r} = \mathbf{r} - \alpha\mathbf{s} \\
&\qquad \mathbf{h} = P^{-1}\mathbf{r} \\
&\qquad \delta_{old} = \delta \\
&\qquad \delta = \mathbf{r}^T\mathbf{h} \\
&\qquad \mathbf{p} = \mathbf{h} + \frac{\delta}{\delta_{old}}\mathbf{p}
\end{aligned}
$$

Figure 5.1: Preconditioned conjugate gradient algorithm

to invert (P must be positive definite as well). We can then solve our problem by applying CG iterations to

$$P^{-1}A\mathbf{x} = P^{-1}\mathbf{b}.$$

The number of iterations required now depends on the reduced condition number of $P^{-1}A$, but each iteration requires solving a linear system with P.

The algorithm is presented in Figure 5.1. Given $\mathbf{b}$, A, an initial guess $\mathbf{x_0}$ and an error tolerance tol^2, the algorithm produces an approximate solution $\mathbf{x}$ — where $\mathbf{r}$ are the residuals, $\mathbf{p}$ are the search directions, and δ are the error-norms of the current iteration.

5.2 MPCG (with corrections)

In the context of cloth simulation, the vector $\mathbf{x}$ represents the change in velocities (or positions, as in [14]) at the current time step. At times, we wish to constrain the particle from moving in certain directions; this typically occurs for cloth-solid contact, where we wish to constrain a cloth particle from penetrating the solid. Thus we wish to specify, in advance, the velocity change of certain particles in certain directions.

For each particle i let $ndof(i)$ denote its number of degrees of freedom: if $ndof(i) = 3$ then there are no constraints on this particle; if $ndof(i) = 2$ then there is one direction ξ_i ($|\xi_i| = 1$) of prescribed motion; if $ndof(i) = 1$ then there are two mutually orthogonal directions ξ_i and η_i ($|\xi_i| = 1,\ |\eta_i| = 1,\ \xi_i^T \eta_i = 0$) of prescribed motion; if $ndof(i) = 0$ then all motion is prescribed for this particle.

Define,

$$\begin{aligned} S_i &= \begin{cases} I & ndof(i) = 3 \\ I - \xi_i \xi_i^T & ndof(i) = 2 \\ I - \xi_i \xi_i^T - \eta_i \eta_i^T & ndof(i) = 1 \\ 0 & ndof(i) = 0 \end{cases} \\ S &= diag\{S_1, \ldots, S_N\}. \end{aligned} \tag{5.1}$$

Baraff and Witkin [6] used S to define a *filter* operation, embedded in the PCG algorithm, that operates on vectors to eliminate components in the constrained directions. In [1], Ascher and Boxerman made the crucial observation that S is an *orthogonal projection*. It can be used to decompose vectors as a direct sum. Thus,

if

$$\mathbf{x} = \mathbf{u} + \mathbf{v}, \quad \mathbf{u} = S\mathbf{x}\ ,\ \mathbf{v} = (I - S)\mathbf{x},$$

then $\mathbf{u}$ and $\mathbf{v}$ are orthogonal, $\mathbf{u}^T\mathbf{v} = 0$.

Here, the *constrained problem* can be written in terms of the projection matrix S defined in (5.1) and a given vector $\mathbf{z}$ of dimension n (like $\mathbf{x}'s$) such that the problem is

$$SA\mathbf{x} = S\mathbf{b}, \tag{5.2a}$$

$$(I - S)\mathbf{x} = (I - S)\mathbf{z}. \tag{5.2b}$$

In words, for each particle the equations of motion hold only in the subspace projected by S, $range(S)$, whereas in the subspace $range(I - S)$ the given values of $\mathbf{z}$ determine velocity (or position) changes[1]. As the two subspaces are orthogonal, the two types of motion are separated. This is all written in the form (5.2). A full proof of convergence is given in [1], where we show that the MPCG algorithm (with corrections) reduces to applying PCG to a projected version of the problem (which maintains positive-definiteness). The corrected MPCG algorithm is given in Figure 5.2.

There are two differences between this algorithm and that given in [6]. The first is the stopping criterion $\hat{\mathbf{b}}$, which — in light of the proof — is the natural choice. The other difference has a more profound effect on the performance of the algorithm; it is the choice of initial iterate $\mathbf{x}$. Whereas the original MPCG algorithm uses the initial iterate $\mathbf{x} = \mathbf{z}$, we use $\mathbf{x} = S\mathbf{x}_0 + (I - S)\mathbf{z}$, where $\mathbf{x}_0$ is the solution from the previous time step. Being able to incorporate this information into the initial guess — while still maintaining the constraints $\mathbf{z}$ — improves the performance of

[1] these are the constraints

$$
\begin{aligned}
&\mathbf{x} = S\mathbf{x}_0 + (I - S)\mathbf{z} \\
&\hat{\mathbf{b}} = S(\mathbf{b} - A(I - S)\mathbf{z}) \\
&\mathbf{r} = S(\mathbf{b} - A\mathbf{x}) \\
&\mathbf{p} = SP^{-1}\mathbf{r} \\
&b_\delta = \hat{\mathbf{b}}^T P^{-1}\hat{\mathbf{b}} \\
&\delta = \mathbf{r}^T\mathbf{p} \\
&while\ \delta > tol^2 b_\delta \\
&\qquad \mathbf{s} = SA\mathbf{p} \\
&\qquad \alpha = \frac{\delta}{\mathbf{p}^T\mathbf{s}} \\
&\qquad \mathbf{x} = \mathbf{x} + \alpha\mathbf{p} \\
&\qquad \mathbf{r} = \mathbf{r} - \alpha\mathbf{s} \\
&\qquad \mathbf{h} = P^{-1}\mathbf{r} \\
&\qquad \delta_{old} = \delta \\
&\qquad \delta = \mathbf{r}^T\mathbf{h} \\
&\qquad \mathbf{p} = S(\mathbf{h} + \frac{\delta}{\delta_{old}}\mathbf{p})
\end{aligned}
$$

Figure 5.2: Corrected modified preconditioned conjugate gradient algorithm

the algorithm. Experimental evidence for some problem instances in [1] showed that the number of iterations required by this version of the MPCG ranged from 45% to 75% of that required by the original algorithms.

5.3 Improved Preconditioner for MPCG

A number of researchers have attempted to improve the convergence of the MPCG algorithm by choosing a better preconditioner. Baraff and Witkin [6] used a diagonal preconditioner, $P = diag\{A\}$. Choi and Ko [14] used a 3×3 block diagonal preconditioner, reporting a 20% performance improvement; they also experimented with incomplete Cholesky (IC) and incomplete LU (ILU) factorizations [49], but reported no significant performance gain. In [30], Hauth et al. 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 literature, and it is unclear which cases their results apply to.

A significant improvement can be made by looking at the projected problem and choosing a preconditioner accordingly. Equations (5.2) can be combined so that the problem can be written as

$$(SA + (I - S))\mathbf{x} = S\mathbf{b} + (I - S)\mathbf{z}. \tag{5.3}$$

A good preconditioner should be an approximation to the matrix on the left hand side of Equation (5.3), rather than to A. If we let C be the matrix consisting of the 3x3 diagonal blocks of A then a *constrained preconditioner* P can be defined as

$$P = SC + (I - S). \tag{5.4}$$

This is also a block diagonal matrix which is easily inverted. Of course, in the unconstrained case, this reduces to simply using the 3x3 diagonal blocks of A. Note that although we employ a block diagonal C, any preconditioner for A which would be effective in the unconstrained case could be used.

5.3.1 Constrained Preconditioner Experiments

In this series of experiments, we investigate the performance improvements (and costs) that may be had by using the constrained preconditioner (5.4). In the process, we make the important distinction between constrained and unconstrained problems. As we will demonstrate, the preconditioner (5.4) does a much better job in the constrained case.

All simulations in this set of experiments are solved using the semi-implicit scheme (4.11). Cloth/cloth collision response is disabled to better see the performance improvements/costs. *A-Block* refers to the preconditioner comprised of the 3x3 block diagonals of A, and *CP* refers to the constrained preconditioner (5.4) based on *A-Block*.

Also, a note on CG tolerances: we use a value of $tol^2 = ck$ (where k is the time step); this maintains the order of accuracy of the scheme. We have found a value of $c = 0.01$ to provide stable results. Smaller values of c led to catastrophic errors when calculating the constraint and friction forces from the residuals of the MPCG algorithm (see Section 3.4.2).

Experiment: The Unconstrained (or Trivially Constrained) Case

In this experiment, we evaluate the performance of preconditioning for the case where all particles have ndof = 0 (unconstrained) or 3 (trivially constrained). To

this end, we test three preconditioners: the identity I (i.e., no preconditioner), *A-Block*, and the constrained preconditioner *CP*. The configuration is "two corners pinned" as in Figure 4.1, for a 1 meter square sheet of cloth.

Simulation parameters are chosen randomly as a permutation of the following values:

- size of system (number of particles): 5×5, 10×10, $\ldots 80 \times 80$ (in increments of 5)
- h, mesh spacing (in m): calculated as 1/size
- ρ ($\frac{m}{h^2}$), cloth density (in $\frac{kg}{m^2}$): 0.05, 0.1, 0.2, 0.5
- k_{s1}, stretch stiffness (in N/m): 300, 600, 1000, 2000, 3000
- k_{s2}, shear stiffness (in N/m): 300, 600, 1000, 2000, 3000
- k_{s3}, bend stiffness (in N/m): 0.001, 0.01, 0.1
- k_{d1}, stretch damping (in $N \cdot s/\mathrm{m}$): 0, 0.01, 0.1, 1, 10
- k_{d2}, shear damping (in $N \cdot s/\mathrm{m}$): 0, 0.01, 0.1, 1, 10
- k_d, air damping (in $N \cdot s/\mathrm{m}$): 0, 0.001, 0.003
- k, time step (in seconds): 0.005, 0.01, 0.015, 0.02

We tested 500 parameter sets in this way, each being run for one second of simulated time.

As expected, the CG iteration counts when using *A-Block* or *CP* are identical. This allows us to evaluate the additional cost of computing *CP* — which on average took 0.8% longer than *A-Block*. Given this minor difference, all further

comparisons between the different preconditioners will be based on CG iteration counts (as this is a more consistent indicator).

We plot in Figure 5.3 the number of CG iterations performed as a function of n (number of particles) for the three preconditioners (here, *A-Block* and *CP* are equal). In addition, the ratio between these counts is plotted in Figure 5.4. As can be seen, the *A-Block* and *CP* preconditioners decrease the number of CG iterations required by about 30%. Note that this improvement is greater than that stated in [14]; a possible explanation for this will be seen in the next experiment.

Finally, note that the number of CG iterations required is not solely dependant on the size of the problem n; it also depends on the stiffness of the problem (e.g., larger time steps, higher spring constants, etc.). To demonstrate this, we also plot the number of CG iterations vs. κ (4.8) in Figure 5.5[2]. Looking at the ratio between the counts in Figure 5.6, we see that the preconditioners do not simply provide a fixed improvement; it is apparent that the "stiffer" the problem is, the more we gain by preconditioning. In extreme cases, the count is reduced by more than 50%. That being said, most sets of simulation parameters in the literature have a value of κ in the range $10^2 - 10^3$; we can thus expect reductions of approximately 30-35%.

Experiment: Constrained Preconditioner Performance Improvements

In this experiment we evaluate the performance of the same three preconditioners (*I*, *A-Block* and *CP*) for the case of non-trivial constraints (i.e., particles having ndof = 1 or 2). This occurs often when cloth is in sliding contact with solids (e.g., clothing).

[2]this holds for fixed n.

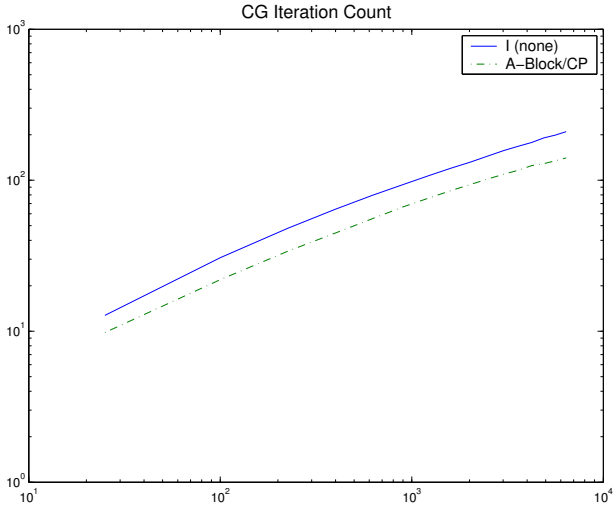

Figure 5.3: Plot: CG iteration count vs. number of particles (log/log plot). Unconstrained case.

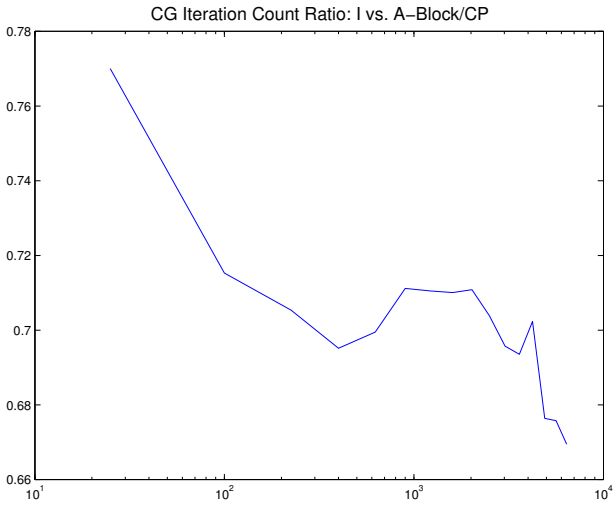

Figure 5.4: Plot: CG iteration count ratio vs. number of particles (semilog in x plot). Unconstrained case.

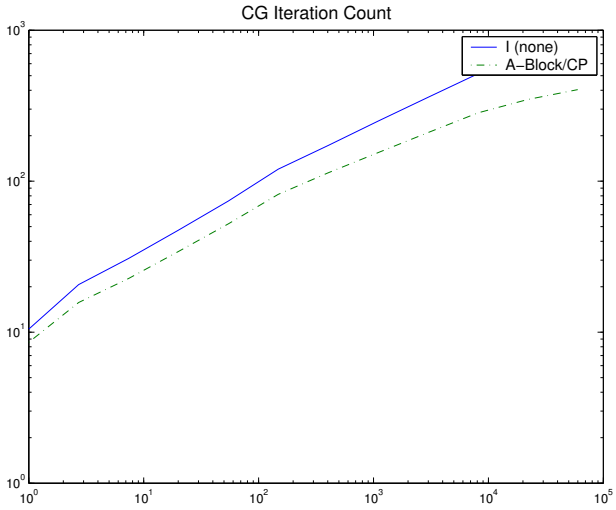

Figure 5.5: Plot: CG iteration count vs. κ (log/log plot). Unconstrained case.

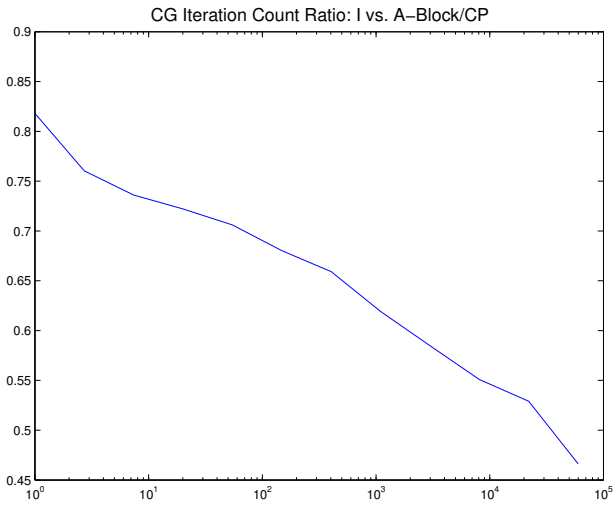

Figure 5.6: Plot: CG iteration count ratio vs. κ (semilog in x plot). Unconstrained case.

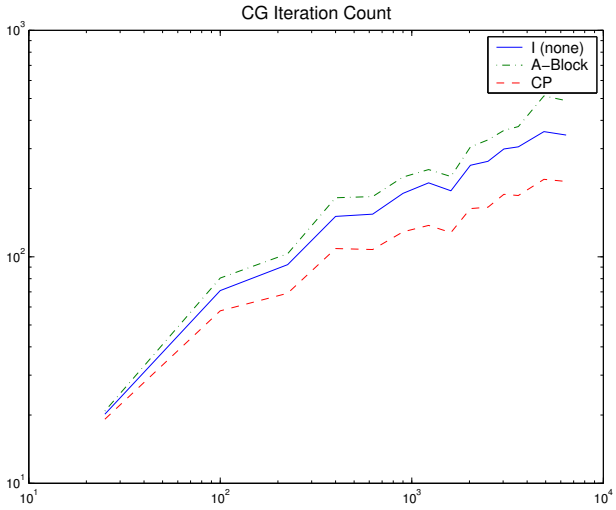

Figure 5.7: Plot: CG iteration count vs. number of particles (log/log plot). Constrained case.

To this end we test using the "cloth draping over a sphere" configuration as depicted in Figure 1, for a 1 meter square sheet of cloth. Simulation parameters are chosen from the same values as in the previous experiment. We tested 500 parameter sets in this way, each being run for one second of simulated time.

We plot in Figure 5.7 the number of CG iterations performed as a function of n for the three preconditioners. Note that here, the counts for *A-Block* and *CP* are not equal. In fact — *A-Block* actually requires *more* iterations than I![3] However, looking at the count ratios in Figure 5.8 we see that — for problems of any significant size — we can still expect a 30% decrease in the number of CG iterations when using *CP*.

As in the previous experiment, we plot the number of CG iterations vs. κ in Figure 5.9, along with the count ratios in Figure 5.10. Again we see that *CP* provides an improvement proportional to the stiffness of the problem, with a

[3]so I is actually a better approximation to the left hand side of (5.3) than *A-Block*.

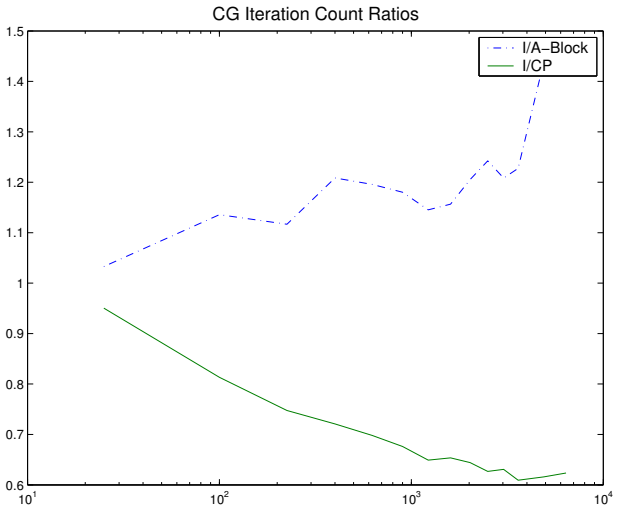

Figure 5.8: Plot: CG iteration count ratio vs. number of particles (semilog in x plot). Constrained case.

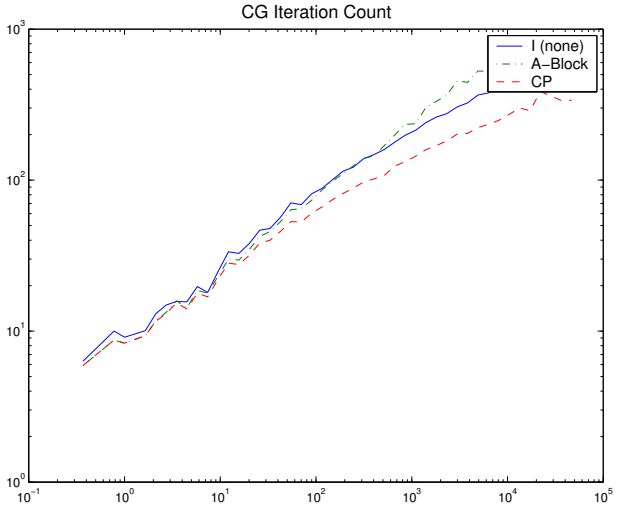

Figure 5.9: Plot: CG iteration count vs. κ (log/log plot). Constrained case.

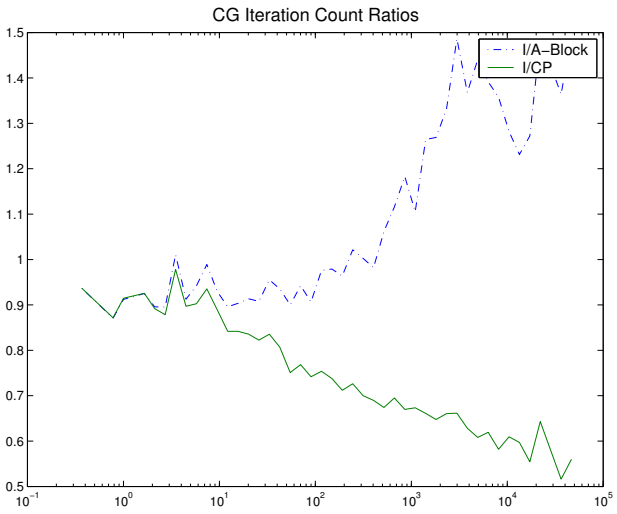

Figure 5.10: Plot: CG iteration count ratio vs. κ (semilog in x plot). Constrained case.

reduction of approximately 70% for typical cloth parameters. Conversely, the *A-Block* preconditioner performs more and more poorly as the stiffness increases for constrained problems. However, as seen in Figure 5.10, this may not have been noticed in practice since *A-Block* performs similarly to I for typical problems.

Note that preconditioning can also decrease the complexity of solving such systems. In [6], Baraff and Witkin stated the cost to be $O(n^{1.5})$; others have echoed this statement. Looking at the overall slopes in Figure 5.7, the iteration count for the problem without preconditioning is $O(n^{0.512})$, which is in agreement with the literature. However, for the *CP* preconditioned problem we found the count to be $O(n^{0.436})$. Thus, although the effect is subtle here, other *constrained* preconditioners may further decrease the asymptotic complexity of the problem.

Chapter 6

Decomposing Cloth

"... To grasp this sorry Scheme of Things entire,

Would we not shatter it to bits - and then

Re-mould it nearer to the Hearts Desire!

– Omar Khayyam

Imagine a tablecloth draped over a square table[1]. If we were to manipulate one corner of the cloth (assuming it does not slip with respect to the table) we would not affect the opposite corner. The same applies for the case of a virtual character tapping its foot, or moving its hand — local motion doesn't affect distant regions of the cloth. It is unfortunate then that, using an implicit solver, the entire cloth must be solved as a single system. It would be better if we could decompose it into subsections which could be solved independently. But — where to "cut"?

As seen in Chapter 4, implicit time integration schemes require the solution of linear systems of the form $A\mathbf{x} = \mathbf{b}$. Solving this using the MPCG method described in Chapter 5 has a computational cost $O(n^{1.5})$, where n is the dimension of the system. As seen in Section 4.3, the matrix A can become even more sparse

[1] or see Figure 6.1

Figure 6.1: Tablecloth draped over a square table. (Snapshot taken from our simulator.)

when using the methods described in this work. In fact, it can sometimes become sufficiently sparse so that it can be decomposed into a set of smaller systems. These smaller systems can be solved individually — and thus more quickly.

In this chapter, we present the mechanisms that allow cloth to be decomposed. We then show how potential decompositions can be easily and quickly detected. We also include some implementation details as to how the decoupled systems can be solved separately, in parallel, and with little data structure overhead. Finally, we present our experimental results.

6.1 Decomposition Mechanisms

Our technique can be seen as a (simple) special application of domain decomposition methods[2] (see Quarteroni and Valli [47] and references therein). In our case, we opportunistically seek independent subdomains such that their influence upon one another can be reduced to constant boundary conditions for a given time step. As such, we investigate two mechanisms by which the systems described in this thesis

[2]specifically, a *zonal*, *non-overlapping* method

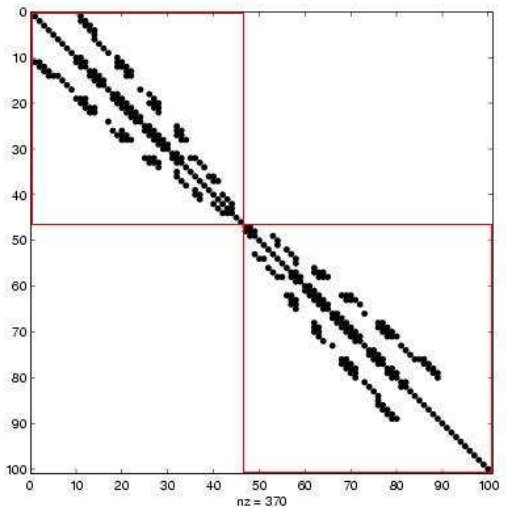

Figure 6.2: Reordered, block-diagonal matrix. (Red squares highlight the two main blocks.)

may be independently decomposed: sparsity and constraints.[3]

6.1.1 Mechanism 1: Sparsity

We begin with a simple example to demonstrate the concept of sparsity decomposition. Looking closely at Figure 4.11, we may note that a reordering of the rows and columns — corresponding to a different ordering of the particles — gives us the structure seen in Figure 6.2; the two large, separate blocks of this matrix can be solved independently.

For a solution technique such as that found in [6], 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 changing sparsity pattern for two reasons. First, a property of Choi and Ko's physical model is that the structural springs (stretch and

[3]That said, more general domain decomposition techniques — either in the form of a preconditioner to the linearized problem, or as a multi-domain/interface reformulation — may prove useful for very large cloth meshes.

shear) do not act in compression[4]. Thus the associated Jacobian entries disappear for any compressed spring. Second, the IMEX technique described in Section 4.3 handles spring connections implicitly at times and explicitly at other times. When treating a connection explicitly, the associated Jacobian entries also disappear.

6.1.2 Mechanism 2: Constraints

In some scenarios, the motion of certain cloth particles is fully prescribed, as in the case of static friction described in Section 3.4.2. This is handled by imposing such constraints directly as described in Chapter 5, where $ndof(i) = 0$ (and $S_i = 0$) for a fully constrained particle i. In this case, the i'th row of A is zeroed, save for the ones appearing along the diagonal; this particle's motion is unaffected by the motion of its neighbours (or by anything else for that matter, it is a known quantity). We can take advantage of this since the influence of this particle on the rest of the system is reduced to a constant (for the current time step), and the row/column pair can be removed. In fact, when looking at the projected problem as described in Chapter 5, the rows and columns corresponding to fully constrained particles are "filtered" or projected out. Thus constrained particles decrease the coupling of the system.

6.2 How to Decompose Cloth

Given the mechanisms just described, how can we detect in practice when independent decompositions are possible? The answer lies in simple graph theory and the relationship between matrices and graphs.

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

[4]Also, the bend springs do not act in extension, but this has little effect here.

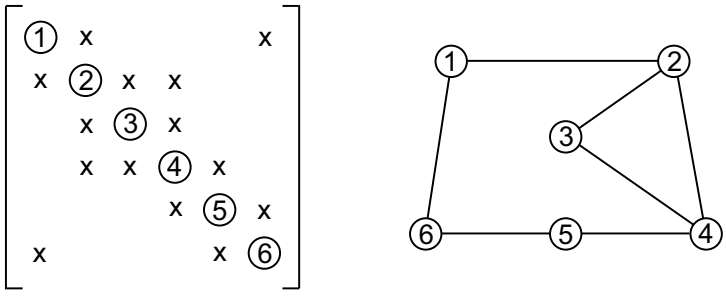

Figure 6.3: A symmetric matrix A and its labeled graph, with x denoting a nonzero entry of A.

pairs of vertices [26]. 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 6.3 illustrates the structure of a matrix and its labeled graph.

A graph is *connected* if every pair of vertices is joined by at least one path through the graph. Otherwise G is disconnected and consists of two or more *connected components*. In this case, there is a row ordering that will make the corresponding matrix block diagonal. Thus we can determine if a reordering exists which will make A block diagonal via simple graph searches.

Moreover, the graph has a clear association with the original physical problem: each vertex represents a particle, and each edge represents an *active* spring (remember, some springs can become disabled) handled implicitly by the solver. Intuitively, this makes sense; if a closed region of the cloth is connected to other regions only by explicit connections (which are considered constant throughout the time step), then it should be possible to solve for that region independently.

While the above explains how to handle graph connectivity, it doesn't deal with constrained particles. Consider a connected graph G that — by removing a single, constrained particle — would become separated into two connected components. Physically, these two components do not affect each other during the current

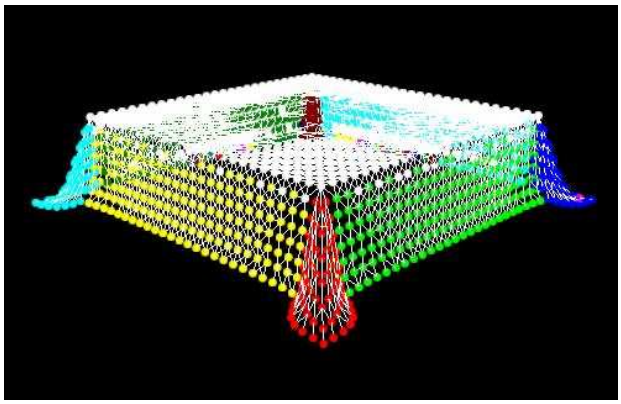

Figure 6.4: Decomposed Cloth Snapshot, Example 1. Cloth draping over a square table. (Implicit stretch and explicit shear.)

time step. However, the constrained particle does affect each component as a fixed boundary value. Thus, a constrained particle acts as a *dead end* during path traversals; it is included in the currently searched component, but cannot be used as a bridge to another component. This property is handled in the algorithm described in the next section.

Before continuing, it is illustrative to see a few snapshots of decomposed cloth; examples of this are seen in Figures 6.4-6.6. Particles of the same colour belong to the same connected component; white particles are fully constrained.

6.2.1 Decomposition Algorithm

Having understood the connection between the mesh and the corresponding matrix and graph, it is straightforward to implement our decomposing solver for cloth. We describe here the additional data structures and algorithmic elements required.

The data structure overhead is quite low. Assuming the structure of the mesh is represented by a connectivity graph (or something similar), we simply add a second (initially empty) set of edges, representing the "implicit" connectivity graph. We

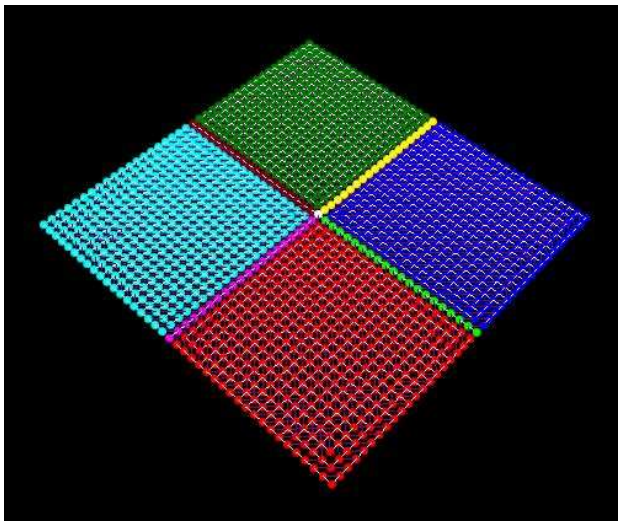

Figure 6.5: Decomposed Cloth Snapshot, Example 2. The cloth is constrained at its center point and has just begun falling. Initial decomposition is clean and regular. (Implicit stretch and explicit shear.)

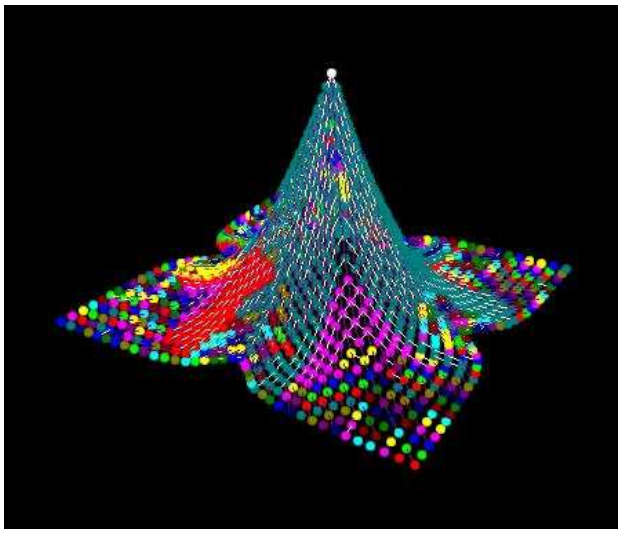

Figure 6.6: Decomposed Cloth Snapshot, Example 3. Moments after Figure 6.5, the decomposition has become quite thorough (colours are repeated).

also associate two integers with each particle: one, $ndof$, which specifies how many degrees of freedom it has at the current time step, and another, *group*, specifying which "group" it is a member of. Finally, we need a data structure to store our group lists.[5]

The algorithm additions are also straightforward. At the beginning of each time step, the edges of the implicit-connection graph are deleted. As spring connections are calculated, an edge is added to the graph *if* the connection is handled implicitly. When solving the system, graph searches are performed. Each connected component that is discovered is handed off to an MPCG solver. Pseudo-code details are presented in Figure 6.2.1.

6.3 MPCG Solution of Decomposed Components

We have not yet discussed how the MPCG solver must be changed to accommodate these decomposed components. The change is a simple one. The MPCG solver is modified to accept an additional argument: a list of particle numbers (corresponding to row numbers) contained within the component to be solved. We can think of each particle as "owning" the associated row in the matrix A and the vectors $\mathbf{r}$, $\mathbf{b}$, $\mathbf{x}$, etc. All operations (matrix/vector multiplies, inner products) are simply performed on this row subset.

[5]The combination of all groups taken together is a list length of n, regardless of the decomposition that occurs. Thus, a fixed length array can be used for this purpose to amortize the overhead.

```
1: loop {main time-stepping loop}
2:    reset forces, Jacobians, etc.
3:    reset implicit connection graph (delete edges)
4:    for all spring connections do
5:       perform usual force and (possibly) Jacobian calculations
6:       if connection is active and handled implicitly then
7:          add edge to implicit connection graph
8:       end if
9:    end for
10:   for all particles i do
11:      calculate external forces, collisions, etc.
12:      set z, and ndof(i) = {0, 1, 2, 3} (based on collisions with solids or otherwise)
13:   end for
14:   construct A**x** = **b** as usual (e.g., 4.11)
15:   // solve all decoupled systems
16:   group(1 ... n) = −1 // reset particles' group membership
17:   currentGroup ← 1
18:   for all particles i do
19:      if ndof(i) == 0 (i.e., particle is fully constrained) then
20:         group(i) ← 0 // 0th group signifies full constraint
21:         x(i) = z(i) // set prescribed solution
22:      end if
23:   end for
24:   for all particles i do
25:      if group(i) == −1 then
26:         begin new list LIST(++currentGroup) and add i to it
27:         group(i) ← currentGroup
28:         add all neighbours of particle i to search list
29:         for all particles j in search list do
30:            if group(j) == −1 then
31:               add j to LIST(currentGroup)
32:               group(j) ← currentGroup
33:               add neighbours of particle j to search list
34:            end if
35:         end for
36:         pass LIST(currentGroup), along with A, x, z and b to MPCG solver
37:      end if
38:   end for
39:   update system state (positions, velocities) given x
40: end loop
```

Figure 6.7: Shattering algorithm pseudo-code

6.4 Decomposing Cloth Experiments

Our decomposition technique offers attractive performance improvements, but there is also a computational overhead.

The improvements come in two forms: smaller, decomposed components converge more quickly (at times dramatically) than the system taken as a whole, and the separate components can be solved easily in parallel. Also note that in our algorithm, constrained particles are not a member of any group; row/vector multiplications are simply skipped for these rows. This can represent a substantial savings.[6]

The costs come in three forms. As described in Section 6.3, we perform operations on row subsets; as such, we can no longer use BLAS routines to perform vector inner products. We also need to perform graph searches as described in Section 6.2.1; these can be done in $O(n)$ time. Finally, we found that sorting the row subset list that is passed to our MPCG solver is necessary. If this isn't done, matrix/vector multiplications are performed in an (almost) random row order, causing severe caching overhead for large groups. This sorting can be done in $O(n)$ time as well. For reasonably large meshes, these costs are dominated by the cost of the MPCG algorithm.

In the following series of experiments, we investigate these performance improvements and costs. All simulations are solved using the AIMEX scheme described in Section 4.4.1 and using the constrained preconditioner (5.4).

[6] In fact, skipping rows which correspond to constrained particles can be implemented independently from our decomposition technique; but it fits most naturally into this context.

6.4.1 Experiment: Cost Overhead of our Decomposing Solver

As stated above, there are overheads associated with our decomposition method. In this experiment, we quantify this overhead by comparing the efficiency of our decomposing solver against a non-decomposing (or *full*) solver in the worst case scenario (i.e., where no decomposition is possible).[7]

The configuration is "two corners pinned" (depicted in Figure 4.1) for a 1.5 meter square sheet of cloth. Rendering and cloth/cloth collision handling was disabled for this series of experiments. Thus the computation time is dominated by the internal dynamics solver. Simulation parameters are chosen from the same values as in experiment 5.3.1. We tested 500 parameter sets in this way, each being run for one second of simulated time.

On average, our decomposing solver required 2.3% more computation time, with insignificant variance across mesh size n.

The number of CG iterations, however, cannot be used as a metric for these experiments. For most configurations, our decomposing solver performs many more iterations than the full solver, but on smaller systems. As such, we use a row/vector multiplication count (RV count), where the multiplication of a vector by one row of the matrix A counts as one RV operation. This is a sensible metric and we have found it to correspond well to CG computation times. In this experiment, the average RV count of our decomposing solver was 0.8% less than the full solver, showing that little decomposition occurred.

[7]Note that even in the worst cases we tried, some very minor decomposition occurs — in locally compressed regions for instance. However the effect here is insignificant.

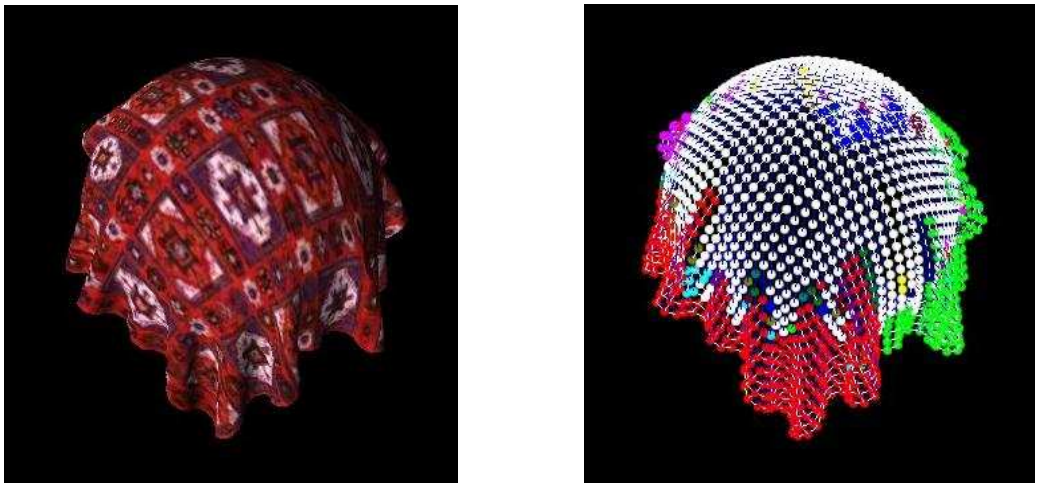

Figure 6.8: Animation Snapshots

6.4.2 Experiment: Performance Improvements of our Decomposing Solver

In this experiment, we examine the performance improvements that can be had via our technique in the case where decomposition *is* possible.

To this end, we compare the computation times and RV counts for the same solvers as above. The configuration is "draping over a sphere" for a 1.5 meter square sheet of cloth.[8] Rendering is disabled; however — to be realistic — cloth/cloth collision handling is enabled, as this decreases the possibility of decomposition. Simulation parameters are chosen from the same values as above. We tested 500 parameter sets in this way, each being run for one second of simulated time. Two animations of a sample experiment can be seen at the following web addresses: *http://www.cs.ubc.ca/~eddybox/decomp_sphere_tex.mov* and *http://www.cs.ubc.ca/~eddybox/decomp_sphere_wire.mov*, depicting textured and wire-

[8]The cloth sheet is not centered over the sphere. As such, in some experiments, the cloth "sticks" and in some it "slips," dependant on friction constants.

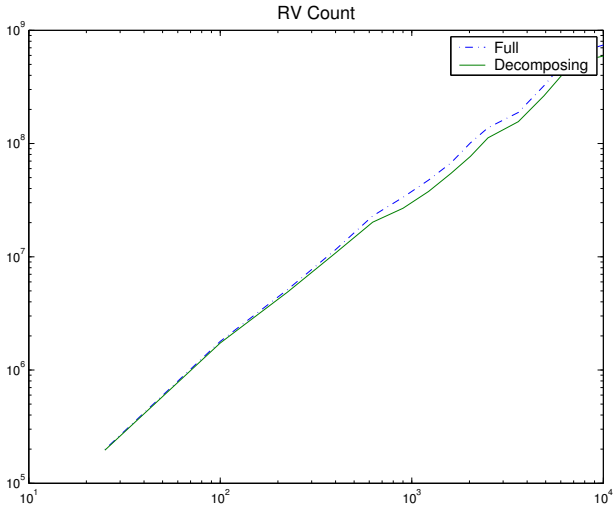

Figure 6.9: Plot: RV Count vs. n (loglog plot) for sphere test.

frame renderings respectively.[9] Figure 6.8 presents snapshots from these animations.

We plot in Figure 6.9 the RV count as a function of n for the decomposing and full solvers. In addition, the ratio between these counts is plotted in Figure 6.10. As can be seen, for small meshes our decomposition technique offers little improvement. However, for larger meshes (around 700+ particles), when the asymptotic complexity of the CG algorithm comes into play, decomposition becomes useful, offering a reduction in the RV count of approximately 20%. This translates directly into a performance improvement of our CG solver (minus roughly 2% as seen in the previous experiment).

During the course of our experiments, we often noticed small groups of particles being solved in a few CG iterations[10] as opposed to, for example, 100 or more. Thus we see an agreement with theory, and a clear view of the origin of our RV-count

[9] We recommend stepping frame by frame through the wireframe animation to observe the decomposition process.

[10] single particles are always solved in one iteration, thanks to our 3×3 block diagonal preconditioner

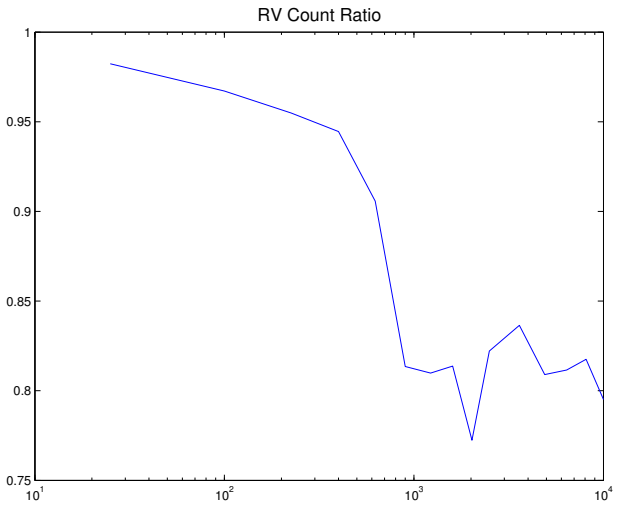

Figure 6.10: Plot: RV Count ratio vs. n (semilox plot in x) for sphere test.

savings.

Parallel Solution of Decomposed Blocks

Of course 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 nicely-decomposing test cases from above — on a dual processor machine. We do this for three solvers: the 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.[11]

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 inves-

[11] In practice we only create one thread per CPU. Additional threads provide no additional benefit; on the contrary, they introduce overhead in the form of context switching.

tigation is warranted (e.g., using larger numbers of processors, memory architecture impact, computing the spring forces/Jacobians in parallel, etc.).

6.4.3 Discussion of Results

As seen in the above experiments, our cloth decomposition technique becomes attractive for large meshes — which is exactly where such performance improvements are most welcome. But in practice, how often can we expect this decomposition to occur? The answer depends on the physical scenario. For example, a tablecloth will decompose more readily than a flag.

That said, we observed decompositions occurring with surprising regularity for large meshes in many scenarios. In fact, the "sphere" scenario above does not represent our best case performance; it simply seemed the most applicable. For instance, in the case of a loose piece of fabric falling to rest on the ground, we observed typical RV count reductions of 30-40%, and at times over 80%, implying a five times speedup in our CG solver!

Of course, the real test is how well it works on virtual clothing. For wildly flying skirts it may not be effective, but for pants, shirts, sweaters, socks, etc., we believe the potential savings to be significant (better than our results in the "sphere" experiment above). This may be a promising avenue for further research.

Chapter 7

Conclusion

In this thesis we have investigated a number of techniques that improve the efficiency of cloth simulation, specifically targeting the semi-implicit methods that are popular in the graphics community. Contrary to most other attempts to do this in the literature, our methods do not sacrifice accuracy.

In Chapter 4, we developed a stability criterion for the FB Euler scheme — applied to cloth — allowing us to devise an adaptive IMEX (AIMEX) scheme. Our AIMEX scheme, which is simple to implement, optimizes the implicit/explicit splitting, thereby decreasing the computational cost. Savings of roughly 30% are typical.

In Chapter 5, we introduced a new, *constrained* preconditioner for the MPCG algorithm popular in cloth simulation. In the presence of non-trivial constraints, this preconditioner clearly outperforms other choices in the literature, providing roughly a 30% reduction in the number of CG iterations required. Moreover, we showed that proper preconditioning can reduce the *asymptotic complexity* of the computation. Thus, as problem sizes grow, the benefits of preconditioning will only improve. The same is true for problem stiffness. We also presented an overview of the proof

of convergence of the MPCG algorithm, along with a superior initial guess that improves performance.

In Chapter 6, we presented a decomposition technique which opportunistically breaks the cloth mesh into separate components that can be solved more quickly and in parallel. Our method becomes attractive for large meshes, providing roughly a 20% reduction in the computational work required. Additional performance improvements are easily realized via parallel implementations of the technique.

Taken together, the above three methods roughly double to triple the speed of existing cloth simulation methods.

In addition, we discussed modelling issues in Chapters 3 and 4, along with the effect of numerical integration methods on the simulation. This included an analysis of the projected damping formulation presented in Section 3.3.

7.1 Future Work

We have not exhausted the ideas presented in this thesis; a number of related research avenues remain to be explored:

- The idea behind our AIMEX scheme, coupled with our decomposition technique, is surely applicable to other problem domains. In fact, it may prove *more* useful in solving highly variable coefficient PDEs, or in application areas where adaptive techniques are more prevalent.

- The AIMEX scheme employed in this thesis is only first-order accurate. As a prototype, this is acceptable[1]. However, higher order methods must be

[1] in addition, first order techniques are still quite common in cloth simulation

considered; this requires the development of new stability criteria and their subsequent testing in applications.

- The idea behind our constrained preconditioner can be extended beyond the 3×3 block diagonal structure. Incomplete Cholesky and SSOR versions may provide additional benefits.

- The parallel implementation of our decomposing solver is rudimentary. Experimentation with additional processors and different architectures is warranted. The force/Jacobian computations, along with collision detection and response, can also be done easily in parallel; this should be done to truly realize the benefits of the parallelism of our decomposition method.

- Our decomposition method relies on mechanisms which *completely* decouple the mesh into independent components. More general versions of domain decomposition methods (either in the form of a preconditioner to the linearized problem, or as a multi-domain/interface reformulation) may provide more reliable performance improvements than our method. Also, *heuristic* decomposition mechanisms — such as relative velocity measures between particles — may also prove to be viable "cutting" lines; such methods would affect the stability and accuracy of the solution, but perhaps not excessively.

Bibliography

[1] U. Ascher and E. Boxerman. On the modified conjugate gradient method in cloth simulation. *The Visual Computer*, 2003. Accepted for publication.

[2] U. Ascher and L. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial & Applied Mathematics, 1998.

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

[4] U. Ascher, S. Ruuth, and B. Wetton. Implicit-explicit methods for time-dependent pde's. *SIAM J. Numer. Anal.*, (32):797–823, 1995.

[5] J. Ascough, J. Bez, H., and A. Bricis. A simple beam element large displacement model for the finite element simulation of cloth drape. *Textile Institute*, 87(1):152–165, 1996.

[6] D. Baraff and A. Witkin. Large steps in cloth simulation. In *SIGGraph*, pages 43–54. ACM, 1998.

[7] D. Baraff, A. Witkin, and M. Kass. Untangling cloth. In *ACM Trans. Graphics*, pages 862–870. ACM Press, 2003.

[8] K. Bhat, C. Twigg, J. Hodgins, P. Khosla, Z. Popovic, and S. Seitz. Estimating cloth simulation parameters from video. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, July 2003.

[9] D. Breen, D. House, and M. Wozny. Predicting the drape of woven cloth using interacting particles. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 365–372. ACM Press, 1994.

[10] R. Bridson. *Computational Aspects of Dynamic Surfaces*. PhD thesis, Stanford University, 2003.

[11] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 594–603. ACM Press, 2002.

[12] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *ACM SIGGRAPH/Eurographics Symposium Computer Animation*, pages 28–36. ACM Press, 2003.

[13] M. Carignan, Y. Yang, N. Magnenat-Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex deformable clothes. *Computer Graphics*, 26(2):99–104, 1992.

[14] K. Choi and H. Ko. Stable but responsive cloth. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 604–611. ACM Press, 2002.

[15] K. Choi and H. Ko. Extending the immediate buckling model to triangular meshes for simulating complex clothes. In *Eurographics short presentations*. Eurographics Assoc, 2003.

[16] J. Collier, B. Collier, G. O'Toole, and S. Sargand. Drape prediction by means of finite-element analysis. *Journal for the Textile Institute*, 82(1):96–107, 1991.

[17] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In *Graphics Interface*, pages 1–8, 1999.

[18] R. DeVaul. Cloth dynamics simulation. Master's thesis, Texas A&M University, 1997.

[19] B. Eberhardt, O. Etzmuss, and M. Hauth. Implicit-explicit schemes for fast animation with particle systems. In *Eurographics Computer Animation and Simulation Workshop 2000*, 2000.

[20] B. Eberhardt and A. Weber. Modelling the draping behaviour of woven cloth. *Maple Tech. Journal*, 4(2):25–31, 1997.

[21] B. Eberhardt and A. Weber. A particle system approach to knitted textiles. *Computers & Graphics*, 23(4):599–606, 1999.

[22] B. Eberhardt, A. Weber, and W. Strasser. A fast, flexible particle-system model for cloth draping. *IEEE Computer Graphics and Applications*, 16(5):52–59, September 1996.

[23] O. Etzmuss, B. Eberhardt, M. Hauth, and W. Straßer. Collision adaptive particle systems. *Proceedings Pacific Graphics 2000*, 2000.

[24] O. Etzmuß, J. Groß, and W. Straßer. Deriving a Particle System from Continuum Mechanics for the Animation of Deformable Objects. *IEEE Transactions on Visualization and Computer Graphics*, 2002.

[25] C. Feynmann. Modeling the appearance of cloth. Master's thesis, Massachusetts Institute of Technology, 1986.

[26] A. George and J. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall, 1981.

[27] S. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical report, 1997.

[28] S. Hadap, E. Bangerter, P. Volino, and M. Magnenat-Thalmann. Animating wrinkles on clothes. pages 175–182. IEEE Computer Society, 1999.

[29] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration, Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2002.

[30] M. Hauth, O. Etzmuss, and W. Straßer. Analysis of numerical methods for the simulation of deformable models. *The Visual Computer*, 2002. Accepted for publication.

[31] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, pages 409–436, 1952.

[32] D. House and D. Breen. *Cloth modeling and animation*. A. K. Peters, Ltd., 2000.

[33] D. House, R. DeVaul, and D. Breen. Towards simulating cloth dynamics using interacting particles. *International Journal of Clothing Science and Technology*, pages 75–94, 1996.

[34] D. Hutchinson, M. Preston, and T. Hewitt. Adaptive refinement for mass/spring simulations. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 31–45. Springer-Verlag New York, Inc., 1996.

[35] Y. Kang, J. Choi, H. Cho, D. Lee, and C. Park. Real-time animation technique for flexible and thin objects. In *WSCG 2000*, pages 322–329, 2000.

[36] M. Kass. An introduction to physically based modeling, chapter: Introduction to continuum dynamics for computer graphics. *In SIGGRAPH 95 Course Notes*, 1995.

[37] S. Kawabata. The standardization and analysis of hand evaluation. *The Textile Machinery Society of Japan*, 1980.

[38] B. Lafleur, N. Magnenat-Thalmann, and D. Thalmann. Cloth animation with self-collision detection. In *Proceedings of Conference on Modeling in Computer Graphics*. Springer, 1991.

[39] R. LeVeque. *Finite volume methods for hyperbolic problems*. Cambridge Texts in applied mathematics. Cambridge University Press, 2002.

[40] J. Louchet, X. Provot, and D. Crochemore. Evolutionary identification of cloth animation models. In *Computer Animation and Simulation*, pages 44–54. Eurographics, 1995.

[41] J. Mezger, S. Kimmerle, and O. Etzmuß. Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG*, 11(2):322–329, 2003.

[42] H. Ng and R. Grimsdale. Computer graphics techniques for modeling cloth. *IEEE Computer Graphics and Applications*, pages 28–41, 1996.

[43] J. O'Brien and J. Hodgins. Graphical modeling and animation of brittle fracture. In *Siggraph 1999, Computer Grpahics Proceedings*, pages 137–146, Los Angeles, 1999. Addison Wesley Longman.

[44] D. Parks and D. Forsyth. Improved integration for cloth simulation. In *Proc. Eurographics short papers*. Eurographics Assoc, 2002.

[45] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Proc. Graphics Interface*, pages 147–154, 1995.

[46] X. Provot. Collision and self-collision handling in cloth model dedicated to design garments. In *Graphics Interface*, pages 177–189, 1997.

[47] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford Science Publications, 1999.

[48] J. Renton. *Applied Elasticity*. Ellis Horwood, Ltd., 1987.

[49] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial & Applied Mathematics, 1996.

[50] J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1994.

[51] J. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Wadsworth & Brooks/Cole, 1989.

[52] S. Tan, T. Wong, Y. Zhao, and W. Chen. A constrained finite element method for modeling cloth deformation. *The Visual Computer*, (15):90–99, 1999.

[53] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 205–214. ACM Press, 1987.

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

[55] J. Villard and H. Borouchaki. Adaptive meshing for cloth animation. In *11th International Meshing Roundtable*, pages 243–252, Ithaca, New York, USA, 15–18 September 2002. Sandia National Laboratories.

[56] P. Volino, M. Courchesne, and N. Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Computer Graphics Proceedings*, 1995.

[57] P. Volino and N. Magnenat-Thalmann. Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. In Dimitri Terzopoulos and Daniel Thalmann, editors, *Computer Animation and Simulation '95*, pages 55–65. Springer-Verlag, 1995.

[58] P. Volino and N. Magnenat-Thalmann. Implementing fast cloth simulation with collision response. *IEEE Computer Society*, pages 257–268, 2000.

[59] P. Volino and N. Magnenat-Thalmann. Comparing efficiency of integration methods for cloth animation. *IEEE Computer Society*, pages 265–274, 2001.

[60] P. Volino, N. Magnenat-Thalmann, S. Jianhua, and D. Thalmann. The evolution of a 3d system for simulating deformable clothes on virtual actors. *IEEE Computer Graphics and Applications*, pages 42–50, 1996.

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

[62] J. Weil. The synthesis of cloth objects. In *Computer Graphics Proceedings, Annual Conference Series*, pages 49–53. ACM SIGGRAPH, 1986.

[63] D. Zhang and M. Yuen. Collision detection for clothed human animation. *Proceedings Pacific Graphics 2000*, 2000.

[64] Y. Zhao, S. Tan, and T. Wong. An effective method for modelling flexible surfaces of cloth objects. *ASME*, pages 351–348, 1994.