Large Steps in Cloth Simulation

David Baraff Andrew Witkin

Robotics Institute
Carnegie Mellon University

Abstract

The bottle-neck in most cloth simulation systems is that time steps must be small to avoid numerical instability. This paper describes a cloth simulation system that can stably take large time steps. The simulation system couples a new technique for enforcing constraints on individual cloth particles with an implicit integration method. The simulator models cloth as a triangular mesh, with internal cloth forces derived using a simple continuum formulation that supports modeling operations such as local anisotropic stretch or compression; a unified treatment of damping forces is included as well. The implicit integration method generates a large, unbanded sparse linear system at each time step which is solved using a modified conjugate gradient method that simultaneously enforces particles' constraints. The constraints are always maintained exactly, independent of the number of conjugate gradient iterations, which is typically small. The resulting simulation system is significantly faster than previous accounts of cloth simulation systems in the literature.

Keywords—Cloth, simulation, constraints, implicit integration, physically-based modeling.

1 Introduction

Physically-based cloth animation has been a problem of interest to the graphics community for more than a decade. Early work by Terzopoulos *et al*. [17] and Terzopoulos and Fleischer [15, 16] on deformable models correctly characterized cloth simulation as a problem in deformable surfaces, and applied techniques from the mechanical engineering and finite element communities to the problem. Since then, other research groups (notably Carignan *et al*. [4] and Volino *et al*. [20, 21]; Breen *et al*. [3]; and Eberhardt *et al*. [5]) have taken up the challenge of cloth.

Although specific details vary (underlying representations, numerical solution methods, collision detection and constraint methods, etc.), there is a deep commonality amongst all the approaches: physically-based cloth simulation is formulated as a time-varying partial differential equation which, after discretization, is numerically solved as an ordinary differential equation

$$\ddot{\mathbf{x}} = \mathbf{M}^{-1}\left(-\frac{\partial E}{\partial \mathbf{x}} + \mathbf{F}\right). \tag{1}$$

In this equation the vector $\mathbf{x}$ and diagonal matrix $\mathbf{M}$ represent the geometric state and mass distribution of the cloth, E—a scalar function of $\mathbf{x}$—yields the cloth's internal energy, and $\mathbf{F}$ (a function of $\mathbf{x}$ and $\dot{\mathbf{x}}$) describes other forces (air-drag, contact and constraint forces, internal damping, etc.) acting on the cloth.

In this paper, we describe a cloth simulation system that is much faster than previously reported simulation systems. Our system's faster performance begins with the choice of an *implicit* numerical integration method to solve equation (1). The reader should note that the use of implicit integration methods in cloth simulation is far from novel: initial work by Terzopoulos *et al*. [15, 16, 17] applied such methods to the problem.[1] Since this time though, research on cloth simulation has generally relied on *explicit* numerical integration (such as Euler's method or Runge-Kutta methods) to advance the simulation, or, in the case of of energy minimization, analogous methods such as steepest-descent [3, 10].

This is unfortunate. Cloth strongly resists stretching motions while being comparatively permissive in allowing bending or shearing motions. This results in a "stiff" underlying differential equation of motion [12]. Explicit methods are ill-suited to solving stiff equations because they require many small steps to stably advance the simulation forward in time.[2] In practice, the computational cost of an explicit method greatly limits the realizable resolution of the cloth. For some applications, the required spatial resolution—that is, the dimension n of the state vector $\mathbf{x}$—can be quite low: a resolution of only a few hundred particles (or nodal points, depending on your formulation/terminology) can be sufficient when it comes to modeling flags or tablecloths. To animate clothing, which is our main concern, requires much higher spatial resolution to adequately represent realistic (or even semi-realistic) wrinkling and folding configurations.

In this paper, we demonstrate that implicit methods for cloth overcome the performance limits inherent in explicit simulation methods. We describe a simulation system that uses a triangular mesh for cloth surfaces, eliminating topological restrictions of rectangular meshes, and a simple but versatile formulation of the internal cloth energy forces. (Unlike previous metric-tensor-based formulations [15, 16, 17, 4] which model some deformation energies as quartic functions of positions, we model deformation energies only as quadratic functions with suitably large scaling. Quadratic energy models mesh well with implicit integration's numerical properties.) We also introduce a simple, unified treatment of damping forces, a subject which has been largely ignored thus far. A key step in our simulation process is the solution of an $O(n) \times O(n)$ sparse linear system, which arises from the implicit integration method. In this respect, our implementation differs greatly from the implementation by Terzopoulos *et al*. [15, 17], which for large simulations

Author affiliation (September 1998): David Baraff, Andrew Witkin, Pixar Animation Studios, 1001 West Cutting Blvd., Richmond, CA 94804. Email: deb@pixar.com, aw@pixar.com.

[1] Additional use of implicit methods in animation and dynamics work includes Kass and Miller [8], Terzopoulos and Qin [18], and Tu [19].

[2] Even worse, the number of time steps per frame tends to increase along with the problem size, for an explicit method. Cloth simulations of size n—meaning $\mathbf{x} \in \mathbb{R}^{O(n)}$—generally require $O(n)$ explicit steps per unit simulated time. Because the cost of an explicit step is also $O(n)$ (setting aside complications such as collision detection for now) explicit methods for cloth require time $O(n^2)$—or worse.

used an "alternating-direction" implicit (ADI) method [12]. An ADI method generates a series of tightly banded (and thus quickly solved) linear systems rather than one large sparse system. (The price, however, is that some of the forces in the system—notably between diagonally-adjacent and non-adjacent nodes involved in self-collisions—are treated explicitly, not implicitly.) The speed (and ease) with which our sparse linear systems can be robustly solved—even for systems involving 25,000 variables or more—has convinced us that there is no benefit to be gained from using an ADI method instead (even if ADI methods could be applied to irregular triangular meshes). Thus, regardless of simulation size, we treat *all* forces as part of the implicit formulation. Even for extremely stiff systems, numerical stability has not been an issue for our simulator.

1.1 Specific Contributions

Much of the performance of our system stems from the development of an implicit integration formulation that handles contact and geometric constraints in a direct fashion. Specifically, our simulator enforces constraints without introducing additional penalty terms in the energy function E or adding Lagrange-multiplier forces into the force $\mathbf{F}$. (This sort of direct constraint treatment is trivial if equation (1) is integrated using explicit techniques, but is problematic for implicit methods.) Our formulation for directly imposing and maintaining constraints is harmonious with the use of an extremely fast iterative solution algorithm—a modified version of the conjugate gradient (CG) method—to solve the $O(n) \times O(n)$ linear system generated by the implicit integrator. Iterative methods do not in general solve linear systems exactly—they are run until the solution error drops below some tolerance threshold. A property of our approach, however, is that the constraints are maintained exactly, regardless of the number of iterations taken by the linear solver. Additionally, we introduce a simple method, tailored to cloth simulation, for dynamically adapting the size of time steps over the course of a simulation.

The combination of implicit integration and direct constraint satisfaction is very powerful, because this approach almost always allows us to take large steps forward. In general, most of our simulations require on average from two to three time steps per frame of 30 Hz animation, even for (relatively) fast moving cloth. The large step sizes complement the fact that the CG solver requires relatively few iterations to converge. For example, in simulating a $6,000$ node system, the solver takes only 50–100 iterations to solve the $18,000 \times 18,000$ linear system formed at each step. Additionally, the running time of our simulator is remarkably insensitive to the cloth's material properties (quite the opposite behavior of explicit methods). All of the above advantages translate directly into a fast running time. For example, we demonstrate results similar to those in Breen *et al*. [3] and Eberhardt *et al*. [5] (draping of a 2,600 node cloth) with a running time just over 2 seconds per frame on an SGI Octane R10000 195 Mhz processor. Similarly, we show garments (shirts, pants, skirts) exhibiting complex wrinkling and folding behavior on both key-framed and motion-captured characters. Representative running times include a long skirt with 4,530 nodes (8,844 triangles) on a dancing character at a cost of 10 seconds per frame, and a shirt with 6,450 nodes (12,654 triangles) with a cost varying between 8 to 14 seconds per frame, depending on the underlying character's motion.

1.2 Previous Work

Terzopoulos *et al*. [15, 17] discretized cloth as a rectangular mesh. Energy functions were derived using a continuum formulation. This work recognized the need for damping forces; however, only a simple viscous drag force $-k\dot{\mathbf{x}}$ was used. The linear systems resulting from the use of implicit integration techniques were solved, for small systems, by direct methods such as Choleski factorization, or using iterative techniques such as Gauss-Seidel relaxation or conjugate gradients. (For a square system of n nodes, the resulting linear system has bandwidth $\sqrt{n}$. In this case, banded Choleski factorization [6] requires time $O(n^2)$.) As previously discussed, Terzopoulos *et al*. made use of an ADI method for larger cloth simulations.

Following Terzopoulos *et al*.'s treatment of deformable surfaces, work by Carignan *et al*. [4] described a cloth simulation system using rectangular discretization and the same formulation as Terzopoulos *et al*. Explicit integration was used. Carignan *et al*. recognized the need for damping functions which do not penalize rigid-body motions of the cloth (as simple viscous damping does) and they added a force which damps cloth stretch and shear (but not bend). Later work by the same group includes Volino *et al*. [20], which focuses mainly on collision detection/response and uses a triangular mesh; no mention is made of damping forces. The system uses the midpoint method (an explicit method) to advance the simulation. Thus far, the accumulated work by this group (see Volino *et al*. [21] for an overview) gives the only published results we know of for simulated garments on moving characters. Reported resolutions of the garments are approximately two thousand triangles per garment (roughly 1,000 nodal points) [21] with running times of several minutes per frame for each garment on an SGI R4400 150 Mhz processor.

Breen *et al*. [3] depart completely from continuum formulations of the energy function, and describe what they call a "particle-based" approach to the problem. By making use of real-world cloth material properties (the Kawabata measuring system) they produced highly realistic static images of draped rectangular cloth meshes with reported resolutions of up to 51×51 nodes. The focus of this work is on static poses for cloth, as opposed to animation: thus, their simulation process is best described as energy minimization, although methods analogous to explicit methods are used. Speed was of secondary concern in this work. Refinements by Eberhardt *et al*. [5]—notably, the use of higher-order explicit integration methods and Maple-optimized code, as well as a dynamic, not static treatment of the problem—obtain similarly realistic results, while dropping the computational cost to approximately 20–30 minutes per frame on an SGI R8000 processor. No mention is made of damping terms. Provot [13] focuses on improving the performance of explicit methods by a post-step modification of nodal positions. He iteratively adjusts nodal positions to eliminate unwanted stretch; the convergence properties of this method are unclear. A more comprehensive discussion on cloth research can be found in the survey paper by Ng and Grimsdale [9].

2 Simulation Overview

In this section, we give a brief overview of our simulator's architecture and introduce some notation. The next section derives the linear system used to step the simulator forward implicitly while section 4 describes the specifics of the internal forces and their derivatives that form the linear system. Section 5 describes how constraints are maintained (once established), with a discussion in section 6 on collision detection and constraint initialization. Section 7 describes our adaptive step-size control, and we conclude in section 8 with some simulation results.

2.1 Notation and Geometry

Our simulator models cloth as a triangular mesh of particles. Given a mesh of n particles, the position in world-space of the ith particle is $\mathbf{x}_i \in \mathbb{R}^3$. The geometric state of all the particles is simply $\mathbf{x} \in \mathbb{R}^{3n}$.

The same component notation applies to forces: a force $\mathbf{f} \in \mathbb{R}^{3n}$ acting on the cloth exerts a force $\mathbf{f}_i$ on the ith particle. Real-world cloth is cut from flat sheets of material and tends to resist deformations away from this initial flat state (creases and pleats not withstanding). We capture the rest state of cloth by assigning each particle an unchanging coordinate (u_i, v_i) in the plane.[3] Section 4 makes use of these planar coordinates.

Collisions between cloth and solid objects are handled by preventing cloth particles from interpenetrating solid objects. Our current implementation models solid objects as triangularly faced polyhedra. Each face has an associated thickness and an orientation; particles found to be sufficiently near a face, and on the wrong side, are deemed to have collided with that face, and become subject to a contact constraint. (If relative velocities are extremely high, this simple test may miss some collisions. In this case, analytically checking for intersection between previous and current positions can guarantee that no collisions are missed.) For cloth/cloth collisions, we detect both face-vertex collisions between cloth particles and triangles, as well as edge/edge collisions between portions of the cloth. As in the case of solids, close proximity or actual intersection of cloth with itself initiates contact handling.

2.2 Energy and Forces

The most critical forces in the system are the internal cloth forces which impart much of the cloth's characteristic behavior. Breen *et al*. [3] describes the use of the Kawabata system of measurement for realistic determination of the in-plane shearing and out-of-plane bending forces in cloth. We call these two forces the shear and bend forces. We formulate the shear force on a per triangle basis, while the bend force is formulated on a per edge basis—between pairs of adjacent triangles.

The strongest internal force—which we call the stretch force—resists in-plane stretching or compression, and is also formulated per triangle. Under normal conditions, cloth does not stretch appreciably under its own weight. This requires the stretch force to have a high coefficient of stiffness, and in fact, it is the stretch force that is most responsible for the stiffness of equation (1). A common practice in explicitly integrated cloth systems is to improve running time by decreasing the strength of the stretch force; however, this leads to "rubbery" or "bouncy" cloth. Our system uses a very stiff stretch force to combat this problem, without any detrimental effects on the run-time performance. While the shear and bend force stiffness coefficients depend on the material being simulated, the stretch coefficient is essentially the same (large) value for all simulations. (Of course, if stretchy cloth is specifically called for, the stretch coefficient can be made smaller.)

Complementing the above three internal forces are three damping forces. In section 5, we formulate damping forces that subdue any oscillations having to do with, respectively, stretching, shearing, and bending motions of the cloth. The damping forces do not dissipate energy due to other modes of motion. Additional forces include air-drag, gravity, and user-generated generated mouse-forces (for interactive simulations). Cloth/cloth contacts generate strong repulsive linear-spring forces between cloth particles.

Combining all forces into a net force vector $\mathbf{f}$, the acceleration $\ddot{\mathbf{x}}_i$ of the ith particle is simply $\ddot{\mathbf{x}}_i = \mathbf{f}_i / m_i$, where m_i is the ith particle's mass. The mass m_i is determined by summing one third the mass of all triangles containing the ith particle. (A triangle's mass is the product of the cloth's density and the triangle's fixed area in the uv coordinate system.) Defining the diagonal mass matrix $\mathbf{M} \in \mathbb{R}^{3n \times 3n}$ by $\mathrm{diag}(\mathbf{M}) = (m_1, m_1, m_1, m_2, m_2, m_2, \ldots, m_n, m_n, m_n)$, we can write simply that

$$\ddot{\mathbf{x}} = \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}). \tag{2}$$

2.3 Sparse Matrices

The use of an implicit integration method, described in the next section, generates large unbanded sparse linear systems. We solve these systems through a modified conjugate gradient (CG) iterative method, described in section 5. CG methods exploit sparsity quite easily, since they are based solely on matrix-vector multiplies, and require only rudimentary sparse storage techniques. The sparsity of the matrix generated by the implicit integrator is best represented in block-fashion: for a system with n particles, we deal with an $n \times n$ matrix, whose non-zero entries are represented as dense 3×3 matrices of scalars. The matrix is represented as an array of n rows; each row is a linked list of the non-zero elements of that row, to accommodate possible run-time changes in the sparsity pattern, due to cloth/cloth contact. The (dense) vectors that are multiplied against this matrix are stored simply as n element arrays of three-component vectors. The overall implementation of sparsity is completely straightforward.

2.4 Constraints

An individual particle's position and velocity can be completely controlled in either one, two, or three dimensions. Particles can thus be attached to a fixed or moving point in space, or constrained to a fixed or moving surface or curve. Constraints are either user-defined (the time period that a constraint is active is user-controlled) or automatically generated, in the case of contact constraints between cloth and solids. During cloth/solid contacts, the particle may be attached to the surface, depending on the magnitudes of the frictional forces required; otherwise, the particle is constrained to remain on the surface, with sliding allowed. The mechanism for releasing a contact constraint, or switching between sliding or not sliding, is described in section 5.

The constraint techniques we use on individual particles work just as well for collections of particles; thus, we could handle cloth/cloth intersections using the technique described in section 5, but the cost is potentially large. For that reason, we have chosen to deal with cloth/cloth contacts using penalty forces: whenever a particle is near a cloth triangle or is detected to have passed through a cloth triangle, we add a stiff spring with damping to pull the particle back to the correct side of the triangle. The implicit solver easily tolerates these stiff forces.

3 Implicit Integration

Given the known position $\mathbf{x}(t_0)$ and velocity $\dot{\mathbf{x}}(t_0)$ of the system at time t_0, our goal is to determine a new position $\mathbf{x}(t_0 + h)$ and velocity $\dot{\mathbf{x}}(t_0 + h)$ at time $t_0 + h$. To compute the new state and velocity using an implicit technique, we must first transform equation (2) into a first-order differential equation. This is accomplished simply by defining the system's velocity $\mathbf{v}$ as $\mathbf{v} = \dot{\mathbf{x}}$ and then writing

$$\frac{d}{dt}\begin{pmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{pmatrix} = \frac{d}{dt}\begin{pmatrix} \mathbf{x} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}, \mathbf{v}) \end{pmatrix}. \tag{3}$$

To simplify notation, we will define $\mathbf{x_0} = \mathbf{x}(t_0)$ and $\mathbf{v_0} = \mathbf{v}(t_0)$. We also define $\Delta\mathbf{x} = \mathbf{x}(t_0 + h) - \mathbf{x}(t_0)$ and $\Delta\mathbf{v} = \mathbf{v}(t_0 + h) - \mathbf{v}(t_0)$.

[3] In general, each particle has a unique (u, v) coordinate; however, to accommodate pieces of cloth that have been topologically seamed together (such as a sleeve), particles lying on the seam must have multiple (u, v) coordinates. For these particles, we let the (u, v) coordinate depend on which triangle we are currently examining. The (u, v) coordinates are useful for texturing.

The explicit *forward* Euler method applied to equation (3) approximates $\Delta\mathbf{x}$ and $\Delta\mathbf{v}$ as

$$\begin{pmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{v} \end{pmatrix} = h \begin{pmatrix} \mathbf{v_0} \\ \mathbf{M}^{-1}\mathbf{f_0} \end{pmatrix}$$

where the force $\mathbf{f_0}$ is defined by $\mathbf{f_0} = \mathbf{f}(\mathbf{x_0}, \mathbf{v_0})$. As previously discussed, the step size h must be quite small to ensure stability when using this method. The implicit *backward* Euler method appears similar at first: $\Delta\mathbf{x}$ and $\Delta\mathbf{v}$ are approximated by

$$\begin{pmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{v} \end{pmatrix} = h \begin{pmatrix} \mathbf{v_0} + \Delta\mathbf{v} \\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x_0} + \Delta\mathbf{x}, \mathbf{v_0} + \Delta\mathbf{v}) \end{pmatrix}. \quad (4)$$

The difference in the two methods is that the forward method's step is based solely on conditions at time t_0 while the backward method's step is written in terms of conditions at the terminus of the step itself.[4]

The forward method requires only an evaluation of the function $\mathbf{f}$ but the backward method requires that we *solve* for values of $\Delta\mathbf{x}$ and $\Delta\mathbf{v}$ that satisfy equation (4). Equation (4) is a nonlinear equation: rather than solve this equation exactly (which would require iteration) we apply a Taylor series expansion to $\mathbf{f}$ and make the first-order approximation

$$\mathbf{f}(\mathbf{x_0} + \Delta\mathbf{x}, \mathbf{v_0} + \Delta\mathbf{v}) = \mathbf{f_0} + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Delta\mathbf{x} + \frac{\partial \mathbf{f}}{\partial \mathbf{v}}\Delta\mathbf{v}.$$

In this equation, the derivative $\partial\mathbf{f}/\partial\mathbf{x}$ is evaluated for the state $(\mathbf{x_0}, \mathbf{v_0})$ and similarly for $\partial\mathbf{f}/\partial\mathbf{v}$. Substituting this approximation into equation (4) yields the linear system

$$\begin{pmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{v} \end{pmatrix} = h \begin{pmatrix} \mathbf{v_0} + \Delta\mathbf{v} \\ \mathbf{M}^{-1}(\mathbf{f_0} + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Delta\mathbf{x} + \frac{\partial \mathbf{f}}{\partial \mathbf{v}}\Delta\mathbf{v}) \end{pmatrix}. \quad (5)$$

Taking the bottom row of equation (5) and substituting $\Delta\mathbf{x} = h(\mathbf{v_0} + \Delta\mathbf{v})$ yields

$$\Delta\mathbf{v} = h\mathbf{M}^{-1}\left(\mathbf{f_0} + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}h(\mathbf{v_0} + \Delta\mathbf{v}) + \frac{\partial \mathbf{f}}{\partial \mathbf{v}}\Delta\mathbf{v}\right).$$

Letting $\mathbf{I}$ denote the identity matrix, and regrouping, we obtain

$$\left(\mathbf{I} - h\mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2\mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)\Delta\mathbf{v} = h\mathbf{M}^{-1}\left(\mathbf{f_0} + h\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{v_0}\right) \quad (6)$$

which we then solve for $\Delta\mathbf{v}$. Given $\Delta\mathbf{v}$, we trivially compute $\Delta\mathbf{x} = h(\mathbf{v_0} + \Delta\mathbf{v})$.

Thus, the backward Euler step consists of evaluating $\mathbf{f_0}$, $\partial\mathbf{f}/\partial\mathbf{x}$ and $\partial\mathbf{f}/\partial\mathbf{v}$; forming the system in equation (6); solving the system for $\Delta\mathbf{v}$; and then updating $\mathbf{x}$ and $\mathbf{v}$. We use the sparse data structures described in section 2.3 to store the linear system. The sparsity pattern of equation (6) is described in the next section, while solution techniques are deferred to section 5.

[4] The method is called "backward" Euler because starting from the output state $(\mathbf{x_0} + \Delta\mathbf{x}, \mathbf{v_0} + \Delta\mathbf{v})$ and using a forward Euler step to run the system *backward* in time (i.e. taking the step $-h(\mathbf{v}(t_0+h), \mathbf{f}(\mathbf{x}(t_0+h), \mathbf{v}(t_0+h)))$ brings you back to $(\mathbf{x_0}, \mathbf{v_0})$. What is the value in this? Forward Euler takes no notice of wildly changing derivatives, and proceeds forward quite blindly. Backward Euler, however, forces one to find an output state whose derivative at least points back to where you came from, imparting, essentially, an additional layer of consistency (or sanity-checking, if you will).

4 Forces

Cloth's material behavior is customarily described in terms of a scalar potential energy function $E(\mathbf{x})$; the force $\mathbf{f}$ arising from this energy is $\mathbf{f} = -\partial E/\partial\mathbf{x}$. Equation (6) requires both the vector $\mathbf{f}$ and the matrix $\partial\mathbf{f}/\partial\mathbf{x}$. Expressing the energy E as a single monolithic function—encompassing all aspects of the cloth's internal behavior—and then taking derivatives is impractical, from a bookkeeping point of view. A better approach is decompose E into a sum of sparse energy functions; that is, to write $E(\mathbf{x}) = \sum_\alpha E_\alpha(\mathbf{x})$ where each E_α depends on as few elements of $\mathbf{x}$—as few particles—as possible.

However, even decomposing E into sparse energy functions is not enough. Energy functions are an undesirable starting point because sensible damping functions cannot be derived from energy functions. Instead, we define internal behavior by formulating a vector condition $\mathbf{C}(\mathbf{x})$ which we want to be zero, and then defining the associated energy as $\frac{k}{2}\mathbf{C}(\mathbf{x})^T\mathbf{C}(\mathbf{x})$ where k is a stiffness constant. In section 4.5, we show how sensible damping functions can be constructed based on this formulation. An added bonus is that starting from this vector-based energy description tends to result in a simpler, more compact, and more easily coded formulation for $\partial\mathbf{f}/\partial\mathbf{x}$ than proceeding from an energy function in which the structure of $\mathbf{C}$ has been lost.

4.1 Forces and Force Derivatives

Given a condition $\mathbf{C}(\mathbf{x})$ which we want to be zero, we associate an energy function $E_\mathbf{C}$ with $\mathbf{C}$ by writing $E_\mathbf{C}(\mathbf{x}) = \frac{k}{2}\mathbf{C}(\mathbf{x})^T\mathbf{C}(\mathbf{x})$ where k is a stiffness constant of our choice. Assuming that $\mathbf{C}$ depends on only a few particle, $\mathbf{C}$ gives rise to a sparse force vector $\mathbf{f}$. Recall from section 2.1 that we view the vector $\mathbf{f}$ in block form; each element $\mathbf{f}_i$ is a vector in $\mathbb{R}^3$. For each particle i that $\mathbf{C}$ depends on,

$$\mathbf{f}_i = -\frac{\partial E_\mathbf{C}}{\partial \mathbf{x}_i} = -k\frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}_i}\mathbf{C}(\mathbf{x}); \quad (7)$$

all the other elements of $\mathbf{f}$ are zero.

Similarly, the derivative of $\mathbf{f}$ is also sparse. Defining the derivative matrix $\mathbf{K} = \partial\mathbf{f}/\partial\mathbf{x}$, the nonzero entries of $\mathbf{K}$ are $\mathbf{K}_{ij}$ for all *pairs* of particles i and j that $\mathbf{C}$ depends on. Again, we treat $\mathbf{K}$ in block fashion: $\mathbf{K} \in \mathbb{R}^{3n\times 3n}$, so an element $\mathbf{K}_{ij}$ is a 3×3 matrix. From equation (7), we have

$$\mathbf{K}_{ij} = \frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} = -k\left(\frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}_i}\frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}_j}^T + \frac{\partial^2 \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_j}\mathbf{C}(\mathbf{x})\right). \quad (8)$$

Additionally, since $\mathbf{K}_{ij}$ is a second derivative—that is, $\mathbf{K}_{ij} = \partial\mathbf{f}_i/\partial\mathbf{x}_j = \partial^2 E/\partial\mathbf{x}_i\partial\mathbf{x}_j$—we have $\mathbf{K}_{ij} = \mathbf{K}_{ji}^T$ so $\mathbf{K}$ is symmetric. Note that since $\mathbf{C}$ does not depend on $\mathbf{v}$, the matrix $\partial\mathbf{f}/\partial\mathbf{v}$ is zero.

We can now easily describe the internal forces acting on the cloth, by just writing condition functions. Forces and their derivatives are easily derived using equations (7) and (8).

4.2 Stretch Forces

Recall that every cloth particle has a changing position $\mathbf{x}_i$ in world space, and a fixed plane coordinate (u_i, v_i). Even though our cloth is modeled as a discrete set of points, grouped into triangles, it will be convenient to pretend momentarily that we have a single continuous function $\mathbf{w}(u, v)$ that maps from plane coordinates to world space. Stretch can be measured at any point in the cloth surface by examining the derivatives $\mathbf{w}_u = \partial\mathbf{w}/\partial u$ and $\mathbf{w}_v = \partial\mathbf{w}/\partial v$ at that point. The magnitude of $\mathbf{w}_u$ describes the stretch or compression in the u direction; the material is unstretched wherever $\|\mathbf{w}_u\| = 1$. Stretch in the

v direction is measured by $\|\mathbf{w}_v\|$. (Some previous continuum formulations have modeled stretch energy along an axis as essentially $(\mathbf{w}_u^T\mathbf{w}_u - 1)^2$, which is a quartic function of position [15, 16, 17, 4]. We find this to be needlessly stiff; worse, near the rest state, the force gradient—a quadratic function of position—is quite small, which partially negates the advantage implicit integration has in exploiting knowledge of the force gradient. A quadratic model for energy is, numerically, a better choice.)

We apply this stretch/compression measure to a triangle as follows. Let us consider a triangle whose vertices are particles i, j and k. Define $\Delta\mathbf{x}_1 = \mathbf{x}_j - \mathbf{x}_i$ and $\Delta\mathbf{x}_2 = \mathbf{x}_k - \mathbf{x}_i$. Also, let $\Delta u_1 = u_j - u_i$, while $\Delta u_2 = u_k - u_i$ and similarly for Δv_1 and Δv_2. We approximate $\mathbf{w}(u, v)$ as a linear function over each triangle; this is equivalent to saying that $\mathbf{w}_u$ and $\mathbf{w}_v$ are constant over each triangle. This lets us write $\Delta\mathbf{x}_1 = \mathbf{w}_u\Delta u_1 + \mathbf{w}_v\Delta v_1$ and $\Delta\mathbf{x}_2 = \mathbf{w}_u\Delta u_2 + \mathbf{w}_v\Delta v_2$. Solving for $\mathbf{w}_u$ and $\mathbf{w}_v$ yields

$$(\mathbf{w}_u \quad \mathbf{w}_v) = (\Delta\mathbf{x}_1 \quad \Delta\mathbf{x}_2)\begin{pmatrix}\Delta u_1 & \Delta u_2\\ \Delta v_1 & \Delta v_2\end{pmatrix}^{-1}. \tag{9}$$

Note that $\mathbf{x}_1$ and $\mathbf{x}_2$ vary during the simulation but the matrix in the above equation does not.

We can treat $\mathbf{w}_u$ and $\mathbf{w}_v$ as functions of $\mathbf{x}$, realizing that they depend only on $\mathbf{x}_i$, $\mathbf{x}_j$ and $\mathbf{x}_k$ and using equation (9) to obtain derivatives. The condition we use for the stretch energy is

$$\mathbf{C}(\mathbf{x}) = a\begin{pmatrix}\|\mathbf{w}_u(\mathbf{x})\| - b_u\\ \|\mathbf{w}_v(\mathbf{x})\| - b_v\end{pmatrix} \tag{10}$$

where a is the triangle's area in uv coordinates. Usually, we set $b_u = b_v = 1$, though we need not always do so. In particular, if we want to slightly lengthen a garment (for example, a sleeve) in the u direction, we can increase b_u, which causes $\mathbf{w}_u$ to seek a larger value, and tends to induce wrinkles across the u direction. Likewise, we might decrease b_v near the end of a sleeve, inducing a tight cuff, as on a sweatshirt. We have found the ability to control shrink/stretch anisotropically to be an indispensable modeling tool.

4.3 Shear and Bend Forces

Cloth likewise resists shearing in the plane. We can measure the extent to which cloth has sheared in a triangle by considering the inner product $\mathbf{w}_u^T\mathbf{w}_v$. In its rest state, this product is zero. Since the stretch term prevents the magnitudes of $\mathbf{w}_u$ and $\mathbf{w}_v$ from changing overly much, we need not normalize. By the small angle approximation, the product $\mathbf{w}_u^T\mathbf{w}_v$ is a reasonable approximation to the shear angle. The condition for shearing is simply

$$C(\mathbf{x}) = a\mathbf{w}_u(\mathbf{x})^T\mathbf{w}_v(\mathbf{x})$$

with a the triangle's area in the uv plane.

We measure bend between pairs of adjacent triangles. The condition we write for the bend energy depends upon the four particles defining the two adjoining triangles. If we let $\mathbf{n}_1$ and $\mathbf{n}_2$ denote the unit normals of the two triangles and let $\mathbf{e}$ be a unit vector parallel to the common edge, the angle θ between the two faces is defined by the relations $\sin\theta = (\mathbf{n}_1 \times \mathbf{n}_2) \cdot \mathbf{e}$ and $\cos\theta = \mathbf{n}_1 \cdot \mathbf{n}_2$. We define a condition for bending by writing simply $C(\mathbf{x}) = \theta$ which results in a force that counters bending.[5] The assumption that the stretch energy will keep the cloth from stretching much allows us to treat $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{e}$ as having a constant length at each step of the simulation. This makes differentiating θ with respect to $\mathbf{x}$ a manageable task.

Rectangular meshes make it simple to treat bending anisotropically. The uv coordinates associated with particles make this possible for triangular meshes as well. Given material for which bending in the u and v directions are weighted by stiffnesses k_u and k_v, we can emulate this anisotropy as follows. Let the edge between the triangles be between particles i and j, and define $\Delta u = u_i - u_j$ and $\Delta v = v_i - v_j$. The stiffness weighting for this edge should simply be

$$\frac{k_u(\Delta u)^2 + k_v(\Delta v)^2}{(\Delta u)^2 + (\Delta v)^2}.$$

4.4 Additional Forces

To the above forces we also add easily implemented forces such as gravity and air-drag (which is formulated on a per-triangle basis, and opposes velocities along the triangle's normal direction). When the simulation is fast enough to interact with, we add user-controlled "mouse" forces. These forces and their gradients are easily derived.

4.5 Damping

The energies we have just described are functions of position only. Robust dynamic cloth simulation, however, is critically dependent on well-chosen damping forces that are a function of both position *and* velocity. For example, the strong stretch force must be accompanied by a suitably strong damping force if we are to prevent anomalous in-plane oscillations from arising between connected particles. However, this strong damping force must confine itself solely to damping in-plane stretching/compressing motions: stretch damping should not arise due to motions that are not causing stretch or compression. Terzopoulos *et al.*'s [16, 17] treatment of cloth used a simple viscous damping function which dissipated kinetic energy, independent of the type of motion. Carignan *et al.* [4] improved upon this somewhat, borrowing a formulation due to Platt and Barr [11]; however, their damping function—a linear function of velocity—does not match the quartic energy functions of their continuum formulation. In this section we describe a general treatment for damping that is independent of the specific energy function being damped.

It is tempting to formulate a damping function for an energy function $E(\mathbf{x})$ by measuring the velocity of the energy, $\dot{E} = \frac{d}{dt}E(\mathbf{x})$. This is an easy trap to fall into, but it gives nonsensical results. At an equilibrium point of E, the gradient $\partial E/\partial\mathbf{x}$ vanishes. Since $\dot{E} = (\partial E/\partial\mathbf{x})^T\dot{\mathbf{x}}$, we find that $\dot{E}$ is zero when E is at its minimum, *regardless* of the system's velocity $\dot{\mathbf{x}} = \mathbf{v}$. In general, $\dot{E}$ is always too small near the system's rest state. Clearly, basing the damping force on $\dot{E}$ is not what we want to do.

We believe that the damping function should be defined not in terms of the energy E, but in terms of the condition $\mathbf{C}(\mathbf{x})$ we have been using to define energies. The force $\mathbf{f}$ arising from the energy acts only in the direction $\partial\mathbf{C}(\mathbf{x})/\partial\mathbf{x}$, and so should the damping force. Additionally, the damping force should depend on the component of the system's velocity in the $\partial\mathbf{C}(\mathbf{x})/\partial\mathbf{x}$ direction; in other words, the damping strength should depend on $(\partial\mathbf{C}(\mathbf{x})/\partial\mathbf{x})^T\dot{\mathbf{x}} = \dot{\mathbf{C}}(\mathbf{x})$. Putting this together, we propose that the damping force $\mathbf{d}$ associated with a condition $\mathbf{C}$ have the form

$$\mathbf{d} = -k_d\frac{\partial\mathbf{C}(\mathbf{x})}{\partial\mathbf{x}}\dot{\mathbf{C}}(\mathbf{x}). \tag{11}$$

This neatly parallels the fact that $\mathbf{f} = -k_s\dfrac{\partial\mathbf{C}(\mathbf{x})}{\partial\mathbf{x}}\mathbf{C}(\mathbf{x})$.

[5] For reasonably equilateral triangles, as edge lengths decrease, the curvature represented by a particular angle θ between triangles increases. Since the square of the curvature—a good measure of the bend energy in cloth—increases at the same rate that the triangle's area decreases, the condition C should *not* be scaled by the triangles' areas. See Breen *et al.* [3] for a further discussion of relating curvature to bend angle.

Given the condition functions $\mathbf{C}$ we have defined in this section for stretch, bend and shear forces, we can now add accompanying damping forces by applying equation (11). As before, $\mathbf{d}_i$ is nonzero only for those particles that $\mathbf{C}$ depends on, and $\partial\mathbf{d}/\partial\mathbf{x}$ has the same sparsity pattern as $\partial\mathbf{f}/\partial\mathbf{x}$. Differentiating equation (11), we obtain

$$\frac{\partial \mathbf{d}_i}{\partial \mathbf{x}_j} = -k_d \left(\frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}_i} \frac{\partial \dot{\mathbf{C}}(\mathbf{x})}{\partial \mathbf{x}_j}^T + \frac{\partial^2 \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_j} \dot{\mathbf{C}}(\mathbf{x}) \right). \qquad (12)$$

Note that $\partial\mathbf{d}/\partial\mathbf{x}$ is *not* a second derivative of some function as was the case in equation (8) so we cannot expect $\partial\mathbf{d}/\partial\mathbf{x}$ to be symmetrical. In equation (12), it is the term $(\partial\mathbf{C}(\mathbf{x})/\partial\mathbf{x}_i)(\partial\dot{\mathbf{C}}(\mathbf{x})/\partial\mathbf{x}_j)^T$ which breaks the symmetry. Anticipating section 5.2, we find it expedient simply to leave this term out, thereby restoring symmetry. This simplification is clearly not physically justifiable, but we have not observed any ill effects from this omission. (Omitting *all* of equation (12), however, causes serious problems.)

Finally, equation (6) requires the derivative $\partial\mathbf{d}/\partial\mathbf{v}$. Since $\dot{\mathbf{C}}(\mathbf{x}) = (\partial\mathbf{C}(\mathbf{x})/\partial\mathbf{x})^T\mathbf{v}$, we have

$$\frac{\partial \dot{\mathbf{C}}(\mathbf{x})}{\partial \mathbf{v}} = \frac{\partial}{\partial \mathbf{v}} \left(\frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}}^T \mathbf{v} \right) = \frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}}.$$

Using this fact, we can write

$$\frac{\partial \mathbf{d}_i}{\partial \mathbf{v}_j} = -k_d \frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}_i} \frac{\partial \dot{\mathbf{C}}(\mathbf{x})}{\partial \mathbf{v}_j}^T = -k_d \frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}_i} \frac{\partial \mathbf{C}(\mathbf{x})}{\partial \mathbf{x}_j}^T.$$

In this case, the result is symmetrical without dropping any terms.

5 Constraints

In this section, we describe how constraints are imposed on individual cloth particles. The constraints we discuss in this section are either automatically determined by the user (such as geometric attachment constraints on a particle) or are contact constraints (generated by the system) between a solid object and a particle. The techniques we describe in this section could be used for multi-particle constraints; however, constraints that share particle would need to be merged. Thus, a set of four-particle constraints (such as vertex/triangle or edge/edge contacts in the cloth) might merge to form a single constraint on arbitrarily many particles, which would be expensive to maintain. Because of this, we handle cloth/cloth contacts with strong springs (easily dealt with, given the simulator's underlying implicit integration base) and "position alteration," a technique described in section 6.

At any given step of the simulation, a cloth particle is either completely unconstrained (though subject to forces), or the particle may be constrained in either one, two or three dimensions. Given the differential nature of our formulation, it is the particle's acceleration, or equivalently, the change in the particle's velocity, that is constrained. If the particle is constrained in all three dimensions, then we are explicitly setting the particle's velocity (at the next step). If the constraint is in two or one dimensions, we are constraining the particle's velocity along either two or one mutually orthogonal axes. Before describing our constraint method, we discuss several other possible enforcement mechanisms and explain why we chose not to use them.

Reduced Coordinates

An obvious and quite exact method for constraining a particle is to reduce the number of coordinates describing the particle's position and velocity. A completely constrained particle would have no coordinates, while a particle with one dimension of constraint would have two coordinates. This is possible—but it complicates the system immensely. If we change the number of coordinates per particle, we alter the size of the derivative matrices in equation (6), as well as the sparsity pattern (this happens when a particle changes from having no coordinates to some coordinates, or vice versa). Given the transient nature of contact constraints between cloth and solids, this is most unappealing. The computation of the derivative matrices' entries is also greatly complicated, because we must now introduce extra Jacobian matrices that relate a particle's reduced coordinates to its motion in world-space. Finally, correct constraint-release behavior between cloth and solid objects is difficult to achieve using a reduced coordinate formulation. Considering all of this, we immediately rejected this method of constraints.

Penalty Methods

We could constrain particles through the use of strong energy functions—essentially, stiff springs that attempt to prevent illegal particle motions. Since our entire formulation is geared to handle stiffness, the usual objections to enforcing constraints with springs—very stiff equations—do not carry as much weight. We tried this for a time, and found it to be a not unreasonable constraint enforcement mechanism. However, penalty methods do not enforce constraints exactly, and they do add some additional stiffness to the system. Since the mechanism we describe enforces constraints exactly, and adds no extra stiffness, we turned away from penalty methods except in the case of cloth/cloth interactions.

Lagrange Multipliers

We could introduce additional constraint forces—that is, Lagrange multipliers—into our system to satisfy the constraints. This involves augmenting the linear system of equation (6) with extra variables (the multipliers) and extra equations (the constraint conditions). Unfortunately, this turns a positive definite system into an indefinite system, which means that iterative methods such as CG will need to square the system first, thereby doubling the running time and degrading the numerical conditionining of the linear system. Additionally, an iterative method will generally not enforce the constraints exactly without a large number of iterations. (A direct method for solving the augmented system would, however, avoid this problem.) Again, the constraint method we describe steps past these difficulties, so we turned away from using Lagrange multipliers.

5.1 Mass Modification

The idea behind our constraint enforcement mechanism is described quite simply, although the actual implementation is somewhat more complicated, to maximize performance. A dynamic simulation usually requires knowledge of the *inverse* mass of objects; for example, note the appearance of $\mathbf{M}^{-1}$, and not $\mathbf{M}$ in equation (6). In the case of a single particle, we write $\ddot{\mathbf{x}}_i = \frac{1}{m_i}\mathbf{f}_i$ to describe a particle's acceleration. When inverse mass is used, it becomes trivial to enforce constraints by altering the mass.

Suppose for example that we want to keep particle i's velocity from changing. If we take $1/m_i$ to be zero, we give the particle an infinite mass, making it ignore all forces exerted on it. Complete control over a particle's acceleration is thus taken care of by storing a value of zero for the particle's inverse mass. What if we wish to constrain the particle's acceleration in only one or two dimensions? Although we normally think of a particle's mass as a scalar, we need not always do so. Suppose we write $\ddot{\mathbf{x}}_i = \begin{pmatrix} 1/m_i & 0 & 0 \\ 0 & 1/m_i & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{f}_i$. Now $\ddot{\mathbf{x}}_i$

must lie in the *xy* plane; no acceleration in the *z* direction is possible. Note than an unconstrained particle can be considered to have the 3×3 inverse mass matrix $\frac{1}{m_i}\mathbf{I}$, with $\mathbf{I}$ the identity matrix.

Of course, we are not restricted to coordinate-aligned constraints. More generally, given a unit vector $\mathbf{p} \in \mathbb{R}^3$, a particle is prevented from accelerating along $\mathbf{p}$ by using an inverse mass matrix $\frac{1}{m_i}(\mathbf{I} - \mathbf{p}\mathbf{p}^T)$; this follows from the fact that $(\mathbf{I} - \mathbf{p}\mathbf{p}^T)\mathbf{p} = \mathbf{0}$. Similarly, given two mutually orthogonal unit vectors $\mathbf{p}$ and $\mathbf{q}$, we prevent a particle from accelerating in either the $\mathbf{p}$ or $\mathbf{q}$ direction by using the inverse mass matrix $\frac{1}{m_i}(\mathbf{I} - \mathbf{p}\mathbf{p}^T - \mathbf{q}\mathbf{q}^T)$.

By allowing constrained particles to have these sorts of inverse masses, we can build constraints directly into equation (6). We will create a modified version $\mathbf{W}$ of $\mathbf{M}^{-1}$; $\mathbf{W}$ will be a block-diagonal matrix, with off-diagonal blocks being zero, and diagonal blocks defined as follows: let $\mathrm{ndof}(i)$ indicate the number of degrees of freedom particle i has, and let particle i's prohibited directions be $\mathbf{p}_i$ (if $\mathrm{ndof}(i) = 2$) or $\mathbf{p}_i$ and $\mathbf{q}_i$ (if $\mathrm{ndof}(i) = 1$) with $\mathbf{p}_i$ and $\mathbf{q}_i$ mutually orthogonal unit vectors. $\mathbf{W}$'s diagonal blocks are $\mathbf{W}_{ii} = \frac{1}{m_i}\mathbf{S}_i$ where

$$\mathbf{S}_i = \begin{cases} \mathbf{I} & \text{if } \mathrm{ndof}(i) = 3 \\ (\mathbf{I} - \mathbf{p}_i\mathbf{p}_i^T) & \text{if } \mathrm{ndof}(i) = 2 \\ (\mathbf{I} - \mathbf{p}_i\mathbf{p}_i^T - \mathbf{q}_i\mathbf{q}_i^T) & \text{if } \mathrm{ndof}(i) = 1 \\ \mathbf{0} & \text{if } \mathrm{ndof}(i) = 0. \end{cases} \tag{13}$$

We are not limited to constraining particles to have zero accelerations in certain directions; rather, we control exactly *what* the change in velocity is along the constrained directions. For every particle i, let $\mathbf{z}_i$ be the change in velocity we wish to enforce in the particle's constrained direction(s). (This implies we can choose any value of $\mathbf{z}_i$ for a completely constrained particle, since all directions are constrained; an unconstrained particle must have $\mathbf{z}_i = \mathbf{0}$ since it has *no* constrained directions.) Using $\mathbf{W}$ and $\mathbf{z}$, we rewrite equation (6) to directly enforce constraints. If we solve

$$\left(\mathbf{I} - h\mathbf{W}\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2\mathbf{W}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)\Delta\mathbf{v} = h\mathbf{W}\left(\mathbf{f}_0 + h\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{v}_0\right) + \mathbf{z} \tag{14}$$

for $\Delta\mathbf{v}$, we will obtain a $\Delta\mathbf{v}$ which is consistent with our constraints. Completely constrained particles will have $\Delta\mathbf{v}_i = \mathbf{z}_i$, while partially constrained particles will have a $\Delta\mathbf{v}_i$ whose component in the constrained direction(s) is equal to $\mathbf{z}_i$.

5.2 Implementation

We initially implemented constraints using equation (14) and found that it worked exactly as advertised. For very small test systems, we solved equation (14) using a direct method (Gaussian elimination) without any problems. For larger systems, we planned to use the iterative, sparsity-exploiting CG method, which immediately presents us with a problem: equation (14) is not a symmetric linear system. (For that matter, neither is equation (6) unless all particles have the same mass.) CG methods, however, require symmetric matrices.[6] We *could* apply a CG method to the unsymmetric matrix of equation (14) by use of the "normal equations"; but this involves multiplying the matrix of equation (14) with its transpose which doubles the cost of each iteration while squaring the condition number of the system [14]—a less than desirable plan. We decided that using a CG method to solve the unsymmetric problem was not acceptable.

Note that without constraints, applying a CG method to equation (6) is not difficult, because we can transform this equation to a symmetric (and positive definite) system by left-multiplying the entire equation by $\mathbf{M}$: the system

$$\left(\mathbf{M} - h\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)\Delta\mathbf{v} = h\left(\mathbf{f}_0 + h\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{v}_0\right) \tag{15}$$

is symmetric and has the same solution $\Delta\mathbf{v}$ as equation (6). Unfortunately, we cannot apply the same transformation to equation (14), because $\mathbf{W}$ is singular—the filtering blocks in equation (13) are rank deficient—so we cannot multiply through by $\mathbf{W}^{-1}$.

The solution to the problem of asymmetry is to modify the CG method so that it can operate on equation (15), while procedurally applying the constraints inherent in the matrix $\mathbf{W}$ at each iteration. The modified method will need to know about the particles' constraints and the vector $\mathbf{z}$. Let us define the symmetric positive definite matrix $\mathbf{A}$ by

$$\mathbf{A} = \left(\mathbf{M} - h\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right) \tag{16}$$

and the vector $\mathbf{b}$ and residual vector $\mathbf{r}$ as

$$\mathbf{b} = h\left(\mathbf{f}_0 + h\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{v}_0\right) \quad \text{and} \quad \mathbf{r} = \mathbf{A}\Delta\mathbf{v} - \mathbf{b}.$$

Given $\mathbf{A}$, $\mathbf{b}$, constraints on the particles, and $\mathbf{z}$, our modified CG method will try to find $\Delta\mathbf{v}$ that satisfies two conditions:

- For each particle i, the component of $\mathbf{r}_i$ in the particle's *unconstrained* direction(s) will be made equal to zero (assuming the method is run for sufficiently many iterations).
- For each particle i, the component of $\Delta\mathbf{v}_i$ in the particle's *constrained* direction(s) will be exactly $\mathbf{z}_i$ (no matter how many iterations are taken).

Note that these two conditions imply that unconstrained particles have $\mathbf{r}_i$ close to zero, while completely constrained particles have $\Delta\mathbf{v}_i = \mathbf{z}_i$. Thus in the case when no particles are constrained, our modified CG method should produce the same result as the regular CG method.

5.3 The Modified Conjugate Gradient Method

The CG method (technically, the preconditioned CG method) takes a symmetric positive semi-definite matrix $\mathbf{A}$, a symmetric positive definite preconditioning matrix $\mathbf{P}$ of the same dimension as $\mathbf{A}$, a vector $\mathbf{b}$ and iteratively solves $\mathbf{A}\Delta\mathbf{v} = \mathbf{b}$. The iteration stops when $\|\mathbf{b} - \mathbf{A}\Delta\mathbf{v}\|$ is less than $\epsilon\|\mathbf{b}\|$ where ϵ is a user-defined tolerance value. The preconditioning matrix $\mathbf{P}$, which must be easily invertible, speeds convergence to the extent that $\mathbf{P}^{-1}$ approximates $\mathbf{A}$. We wholeheartedly refer the reader to Shewchuk [14] for information on the CG method.

We derive our modified conjugate gradient method by observing that the effect of the matrix $\mathbf{W}$ in equation (14) is to filter out velocity changes in the constrained directions. Our idea then is to define an invariant— for all i, the component of $\Delta\mathbf{v}_i$ in the constrained direction(s) of particle i is equal to $\mathbf{z}_i$—and then establish and maintain the invariant at each iteration, by defining a filtering procedure **filter**. The role of **filter** is to take a vector $\mathbf{a}$ and perform the same filtering operation (see equation (13)) as multiplying by $\mathbf{W}$, but leaving out the scaling by $1/m_i$:

procedure filter(a)
for i = 1 **to** n
    $\hat{\mathbf{a}}_i = \mathbf{S}_i\mathbf{a}_i$
return $\hat{\mathbf{a}}$

[6] In fact, they work best on positive definite symmetric matrices. The matrices we ultimately hand to our CG method are positive definite.

Using **filter**, we define the modified CG method **modified-pcg** as follows:

```
1    procedure modified-pcg
2    Δv = z
3    δ0 = filter(b)^T P filter(b)
4    r = filter(b − AΔv)
5    c = filter(P^−1 r)
6    δnew = r^T c
7    while δnew > ε^2 δ0
8        q = filter(Ac)
9        α = δnew/(c^T q)
10       Δv = Δv + αc
11       r = r − αq
12       s = P^−1 r
13       δold = δnew
14       δnew = r^T s
15       c = filter(s + (δnew/δold) c)
```

Line 2 of the procedure establishes our invariant. Lines 5 and 15 maintain the invariant by filtering **c** before adding it to $\Delta\mathbf{v}$. The unmodified conjugate gradient method establishes a stopping criterion based on $\mathbf{b}^T\mathbf{P}\mathbf{b}$. Since our constrained formulation ignores certain components of **b**, our stopping criterion should as well, so we add filtering to line 3. The vector **r** measures the solution error $\mathbf{b}-\mathbf{A}\Delta\mathbf{v}$, and should not include error due to the constraints; hence we add filtering at lines 4 and 8. (Note that removing the calls to **filter** and changing line 2 to $\Delta\mathbf{v}=\mathbf{0}$ yields the standard preconditioned conjugate gradient method.)

We use a simple preconditioner **P** by making **P** be a diagonal matrix with $\mathbf{P}_{ii}=1/\mathbf{A}_{ii}$ so products involving $\mathbf{P}^{-1}$ are trivially computed. More elaborate preconditioners could be used, though we doubt there is a large speedup to be gained. Matrix-vector products with **A** are of course implemented in sparse matrix-vector fashion, using the data structures defined in section 2.3.

Given **modified-pcg**, obvious questions are "does it work?" followed by "how does it compare with the unmodified CG method?" Proofs about CG methods are difficult in general; in practice, our method always converges, which answers the first question. Prior to implementing **modified-pcg**, we used a penalty method and applied the standard CG method to equation (15). When we began using procedure **modified-pcg**, we did not notice any substantial change in the number of iterations required by the method. Empirically, we conclude that the two methods have similar convergence behavior. Result in section 8 indicate that the running time is close to $O(n^{1.5})$, which is what unmodified CG would be expected to deliver on this sort of problem [14].

5.4 Determining the Constraint Forces

For contact constraints (between cloth and solid objects) we need to know what the actual force of constraint is, in order to determine when to terminate a constraint. Additionally, we need to know the constraint force actually exerted in order to model frictional forces properly. Fortunately, it is easy to add one more step to **modified-pcg** to determine the constraint force. When **modified-pcg** terminates, the residual error $\mathbf{e}=\mathbf{A}\Delta\mathbf{v}-\mathbf{b}$ has the property that $\mathbf{e}_i$ need not be close to zero if particle i is constrained. In fact, $\mathbf{e}_i$ is exactly the extra constraint force that must have been supplied to enforce the constraint. Thus, we can compute constraint forces at the end of **modified-pcg** by performing one last matrix-vector product to compute $\mathbf{A}\Delta\mathbf{v}-\mathbf{b}$. (The vector **r** in **modified-pcg** is equal to $\mathbf{filter}(\mathbf{A}\Delta\mathbf{v}-\mathbf{b})$, so the extra matrix-vector product to compute **e** really is necessary.)

The particles' accelerations are inherently dependent on one another through the matrix **A** of equation (16). This means that the correct approach to determing constraint release is combinatoric, as in Baraff [2]. We reject this approach as impractical given the dimension of **A**. Instead, we allow contacts to release when the constraint force between a particle and a solid switches from a repulsive force to an attractive one. In practice, this has proven to work well.

Friction presents a similar problem. When cloth contacts a solid, we lock the particle onto the surface, if the relative tangential velocity is low. We monitor the constraint force, and if the tangential force exceeds some fraction of the normal force, we allow the particle to slide on the surface. For high sliding velocities, we apply a dissipative tangential force, opposite the relative sliding direction, proportional to the normal force.

6 Collisions

Much has been written about collision detection for cloth; we have nothing substantial to add to the subject of collision detection *per se*. Cloth/cloth collisions are detected by checking pairs (p,t) and (e_1,e_2) for intersections, where p and t are a cloth particle and a cloth triangle respectively, and e_1 and e_2 are edges of cloth triangles. Given a previous known legal state of the cloth, we postulate a linear motion for the cloth particles to the current (possibly illegal) state and check for either particle/triangle or edge/edge crossings. To avoid $O(n^2)$ comparisons, we use a coherency-based bounding-box approach [1] to cull out the majority of pairs.

When collisions between a cloth vertex and triangle, or two cloth edges are detected, we insert a strong damped spring force to push the cloth apart. A dissipative force tangent to the contact is also applied, countering any sliding motion. The force is not, strictly speaking, a frictional force: rather it is proportional to the slip velocity, so it is in actuality a damping force, although it reasonably emulates dynamic friction. Applying static friction forces to cloth contacts is far more difficult, and is a problem we have not solved yet. The forces, and their derivatives with respect to position and velocity, are of course included in equation (15).

Our system detects collisions between cloth particles and solid objects by testing each individual cloth particle against the faces of each solid object. A solid object's faces are grouped in a hierarchical bounding box tree, with the leaves of the tree being individual faces of the solid. The tree is created by a simple recursive splitting along coordinate axes. The maintenance of contacts and the application of friction forces was described in the previous section.

6.1 Constraint Initiation

Both cloth/cloth and cloth/solid collisions give rise to the same problem whenever two contacts form. For both types of collisions, our detection algorithm reports an intersection, and then takes action to remedy the situation: either by enforcing a constraint (cloth/solid collisions) or by adding a penalty force (cloth/cloth) collisions. However, since our simulator proceeds in discrete steps, collisions resulting in a reasonably substantial interpenetration depth can occur between one step and the next. Clearly, this situation needs to be remedied.

For cloth/cloth collisions, this would not appear to be a problem: the spring forces that are added work to counter the colliding velocities and then push the cloth apart. For cloth/solid collisions, however, the situation is more complicated. If we simply enforce a constraint which causes the colliding cloth particle to have a velocity consistent with the solid object's velocity, and continue to enforce that constraint, the cloth particle will continue to remain embedded somewhere below the solid object's surface. This is unacceptable.

One solution is to use Baumgarte stabilization [18], which schedules the particle's acceleration so that the position and velocity error of the particle with respect to the surface decay asymptotically to zero. We experimented with this technique, but found it lacking. In particular, a fast rise to the surface was prone to noise and "jumpiness"; this could be eliminated, but at the cost of decreasing the step size. A slower rise to the surface caused visual artifacts.

We tried a simpler solution: when intersections occurred, rather than wait for a scheduled constraint or a penalty force to eliminate the intersection, we simply altered the positions of the cloth particles, effecting an instantaneous (and discontinuous) change in position. While this would be problematic when using a multi-step differential equation solver which expects continuity (such as a Runge-Kutta method), it should not interfere with a one-step solver such as the backward Euler method. Unfortunately, simply changing particle positions produced disastrous results. The stretch energy term in a cloth system is extremely strong, and altering particle positions arbitrarily introduced excessively large deformation energies in an altered particle's neighborhood. This resulted in visibly "jumpy" behavior of the cloth in localized regions.

6.2 Position Alteration

Despite its initial failure, the ability to make arbitrary small changes in a particle's position continued to attract our attention. The entire process of implicit integration can be considered to be a filtering process [7], and we postulated that a mechanism for filtering energy changes caused by displacing particles might make position alteration a viable technique. We considered that perhaps some sort of extra implicit step could be used as a filter, but forming and solving an additional linear system at each step seemed too expensive. Happily, we can make use of the filtering effect of implicit integration without any extra work.

Consider a particle that has collided with a solid object. The particle's change in velocity at each step is under our control, using the constraint techniques described in section 5. Meanwhile, the particle's *position* at the next step follows from equation (4):

$$\Delta\mathbf{x}_i = h(\mathbf{v}_{0i} + \Delta\mathbf{v}_i)$$

(recall that $\mathbf{v}_{0i}$ is the particle's current velocity). The reason that changing positions *after* a step has been taken doesn't work is because the particle's neighbors receive no advance notification of the change in position: they are confronted with the alteration at the beginning of the next step. This presents an obvious solution: we simply modify the top row of equation (4) to

$$\Delta\mathbf{x}_i = h(\mathbf{v}_{0i} + \Delta\mathbf{v}_i) + \mathbf{y}_i \tag{17}$$

where $\mathbf{y}_i$ is an arbitrary correction term of our choice, introduced solely to move a particle to a desired location during the backward Euler step. Having modified the top row of equation (4), we must follow this change through: using equation (17) and repeating the derivation of section 3 and the symmetric transform from section 5 yields the modified symmetric system

$$\left(\mathbf{M} - h\frac{\partial\mathbf{f}}{\partial\mathbf{v}} - h^2\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\right)\Delta\mathbf{v} = h\left(\mathbf{f}_0 + h\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{v}_0 + \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{y}\right). \tag{18}$$

This modification gives us complete control over both the position and velocity of a constrained particle in just one step, without any extra computational cost. We use this technique to bring particles quickly and stably to the surface of solid objects without creating visual artifacts or limiting the allowable step size. We can also add correction terms to particles involved in cloth/cloth collisions. Without a constraint on those particles' velocities there is no guarantee that they will go exactly where we want in one step, but the ability to induce sizeable jumps in position without excessively stiff spring forces adds greatly to the stability of the simulation.

7 Adaptive Time Stepping

The methods introduced in all of the previous sections usually allow us to take sizeable steps forward, without loss of stability. Even so, there are still times when the step size must be reduced to avoid divergence. There are a large number of methods for altering the size of a time step, for both explicit and implicit integrators, but these methods tend to concentrate on the accuracy of the simulation, and not the stability. Our goal is animation, not engineering; thus visually pleasing results, meaning a numerically stable solution, rather than overall accuracy, is the deciding voice. The trick is to recognize instability before you see it on your screen—by then it's too late.

Stiffness, and thus any potential instability, arises almost completely from the strong stretch forces in the cloth. After each implicit step, we treat the resulting $\Delta\mathbf{x}$ as a proposed change in the cloth's state, and examine the stretch terms (section 4.2) for each triangle in the newly proposed state. If any triangle undergoes a drastic change in its stretch (in either the u or v direction) we discard the proposed state, reduce the step size, and try again. Subtlety is not required: we find that an unstable step invariably results in stretch changes that are quite large, and are thus easily detected.

Our simulation is run with a parameter that indicates the maximum allowable step size: this parameter is set by the user, and is always less than or equal to one frame. (Most of our simulations involving human motions use a step size of 0.02 seconds.) Whenever the simulator reduces the step size, after two successes with the reduced step size the simulator tries to increase the step size. If the simulator fails at the larger step size, it reduces the size again, and waits for a longer period of time before retrying to increase the step size. At its limit, the simulator will try increasing the step size every 40 steps; thus, if the user chooses too large a step, the simulator settles down to wasting only one out of every 40 steps in attempting too large a step. This method, though simple, has served us well.

8 Results

Table 1 gives a performance summary of assorted animations, shown in figures 1–6. Unaccounted overhead of the simulation (typically about 5%) includes tasks such as geometry transformations, memory allocation, etc. The clothes in figures 3–6 were modeled as discrete planar panels, and then topologically seamed. The simulator was used to relax the clothing from an initial deformed state, that got the clothes around the characters, to a well-fitting state on the characters. The b_u and b_v parameters (see equation (10)) were then made smaller in certain regions to produce cuffs and waistbands, or strategically increased to induce wrinkling behavior in other regions.

We also ran the simulation in figure 1 with a range of stiffnesses for the bend term. Using the stiffness parameters in figure 1 as a reference, we ran the simulation with those bend stiffnesses multiplied by 0.1, 1.0, 10, 100 and 1,000 (for a total range of 10,000 in the stiffness). The variance in the running times was under 5%. We doubt that simulators based on explicit integration methods could make a similar claim.

Finally, we tried to estimate our simulator's performance as a function of n, the number of cloth particles. We ran the simulation in figure 1 with cloth resolutions of 500, 899, 2,602 (shown in figure 1) and 7,359 particles. The running times were, respectively, 0.23 seconds/frame, 0.46 seconds/frame, 2.23 seconds/frame, and 10.3 seconds/frame. This is slightly better than $O(n^{1.5})$ performance, which is in line with the convergence rates of the conjugate gradient method [14] for systems such as equation (18).

figure	no. vertices/no. triangles cloth	solid	time/frame (CPU sec.)	step size min/max (ms)	total frames/ total steps	task breakdown percentage EVAL	CG	C/C	C/S
1	2,602/4,9442	322/640	2.23	16.5/33	75/80	25.7	50.4	18.3	1.4
2	2,602/4,9442	322/640	3.06	16.5/33	75/80	17.9	63.6	15.3	0.2
3	6,450/12,654	9,941/18,110	7.32	16.5/33	50/52	18.9	37.9	30.9	2.6
4 (shirt)	6,450/12,654	9,941/18,110	14.5	2.5/20	430/748	16.7	29.9	46.1	2.2
(pants)	8,757/17,352	9,941/18,110	38.5	0.625/20	430/1214	16.4	35.7	42.5	1.7
5 (skirt)	2,153/4,020	7,630/14,008	3.68	5/20	393/715	18.1	30.0	44.5	1.5
(blouse)	5,108/10,016	7,630/14,008	16.7	5/20	393/701	11.2	26.0	57.7	1.3
6 (skirt)	4,530/8,844	7,630/14,008	10.2	10/20	393/670	20.1	36.8	29.7	2.6
(blouse)	5,188/10,194	7,630/14,008	16.6	1.25/20	393/753	13.2	30.9	50.2	1.4

Table 1: System performance for simulations in figures 1–6. Minimum and maximum time steps are in milliseconds of simulation time. Time/frame indicates actual CPU time for each frame, averaged over the simulation. Percentages of total running time are given for four tasks: EVAL— forming the linear system of equation (18); CG—solving equation (18); C/C—cloth/cloth collision detection; and C/S—cloth/solid collision detection.

9 Acknowledgments

This research was supported in part by an ONR Young Investigator award, an NSF CAREER award, and grants from the Intel Corporation. We thank Alias|Wavefront for supplying the models and motion capture data used in figures 5 and 6.

References

[1] D. Baraff. *Dynamic Simulation of Non-penetrating Rigid Bodies*. PhD thesis, Cornell University, May 1992.

[2] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. *Computer Graphics* (*Proc. SIGGRAPH*), 28:23–34, 1994.

[3] D.E. Breen, D.H. House, and M.J. Wozny. Predicting the drape of woven cloth using interacting particles. *Computer Graphics* (*Proc. SIGGRAPH*), pages 365–372, 1994.

[4] M. Carignan, Y. Yang, N. Magenenat-Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex deformable clothes. *Computer Graphics* (*Proc. SIGGRAPH*), pages 99–104, 1992.

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

[6] G. Golub and C. Van Loan. *Matrix Computations*. John Hopkins University Press, 1983.

[7] M. Kass. *An Introduction To Physically Based Modeling*, chapter Introduction to Continuum Dynamics for Computer Graphics. SIGGRAPH Course Notes, ACM SIGGRAPH, 1995.

[8] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. *Computer Graphics* (*Proc. SIGGRAPH*), pages 49–58, 1990.

[9] H.N. Ng and R.L. Grimsdale. Computer graphics techniques for modeling cloth. *IEEE Computer Graphics and Applications*, 16:28–41, 1996.

[10] H. Okabe, H. Imaoka, T. Tomiha, and H. Niwaya. Three dimensional apparel cad system. *Computer Graphics* (*Proc. SIGGRAPH*), pages 105–110, 1992.

[11] J.C. Platt and A.H. Barr. Constraint methods for flexible models. In *Computer Graphics* (*Proc. SIGGRAPH*), volume 22, pages 279–288. ACM, July 1988.

[12] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes*. Cambridge University Press, 1986.

[13] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface*, pages 147–155, 1995.

[14] J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-TR-94-125, Carnegie Mellon University, 1994. (See also `http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.ps`.).

[15] D. Terzopoulos and K. Fleischer. Deformable models. *Visual Computer*, 4:306–331, 1988.

[16] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Computer Graphics* (*Proc. SIGGRAPH*), volume 22, pages 269–278. ACM, August 1988.

[17] D. Terzopoulos, J.C. Platt, and A.H. Barr. Elastically deformable models. *Computer Graphics* (*Proc. SIGGRAPH*), 21:205–214, 1987.

[18] D. Terzopoulos and H. Qin. Dynamics nurbs with geometric constraints for interactive sculpting. *ACM Transactions on Graphics*, 13:103–136, 1994.

[19] X. Tu. *Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception and Behavior*. PhD thesis, University of Toronto, May 1996.

[20] P. Volino, M. Courchesne, and N. Magnenat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. *Computer Graphics* (*Proc. SIGGRAPH*), pages 137–144, 1995.

[21] P. Volino, N. Magnenat Thalmann, S. Jianhua, and D. Thalmann. An evolving system for simulating clothes on virtual actors. *IEEE Computer Graphics and Applications*, 16:42–51, 1996.

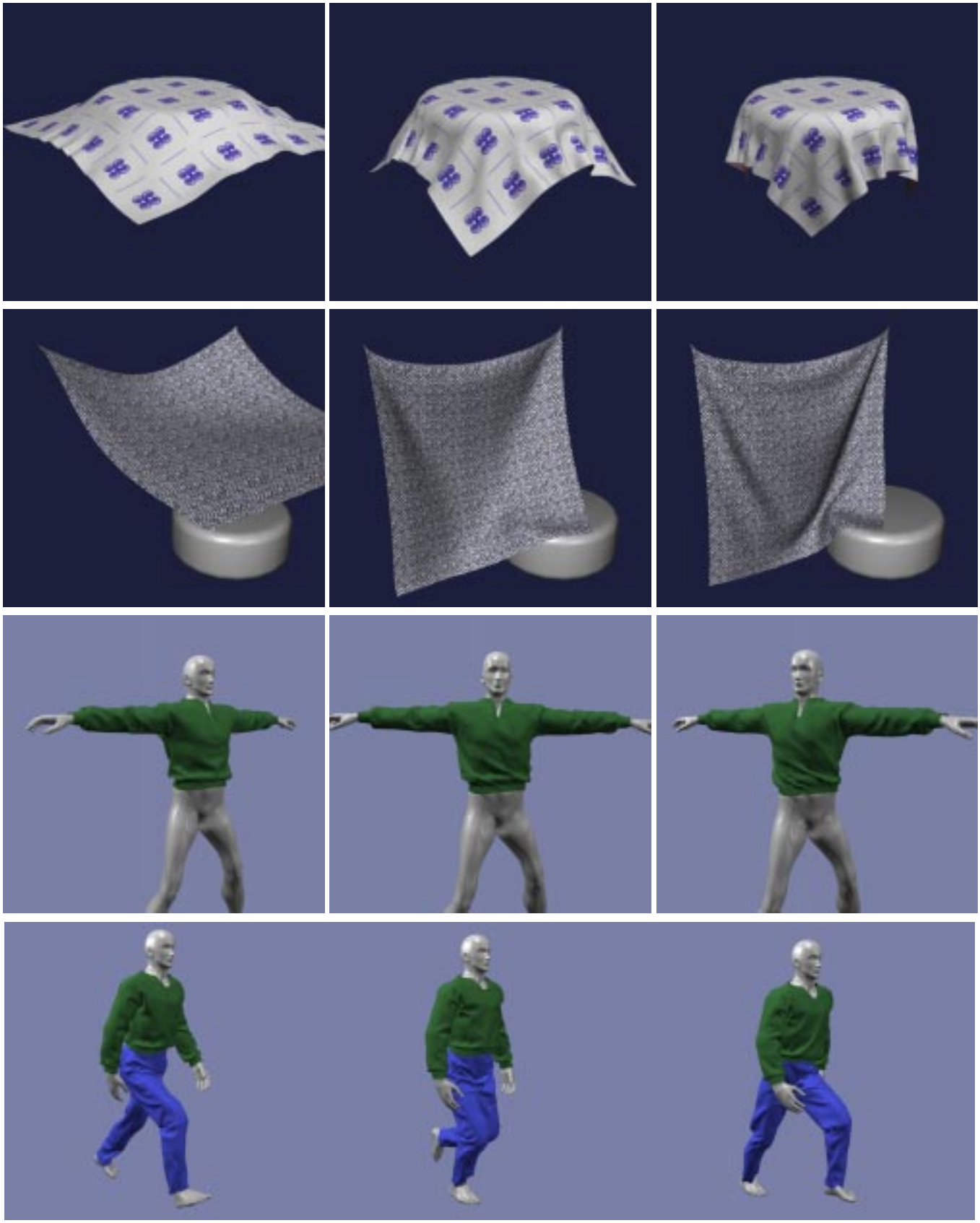

Figure 1 (top row): Cloth draping on cylinder; frames 8, 13 and 35. Figure 2 (second row): Sheet with two fixed particles; frames 10, 29 and 67. Figure 3 (third row): Shirt on twisting figure; frames 1, 24 and 46. Figure 4 (bottom row): Walking man; frames 30, 45 and 58.

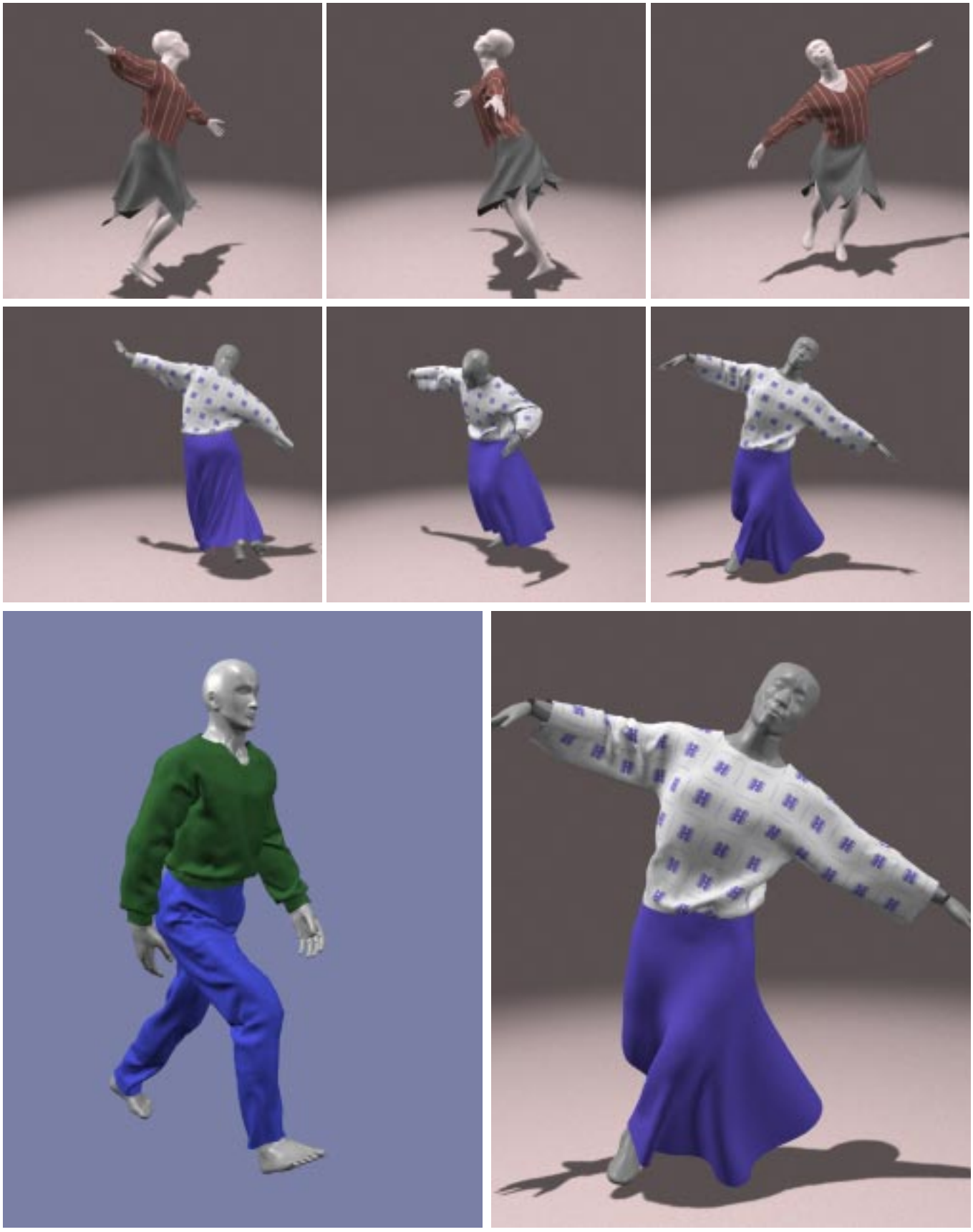

Figure 5 (top row): Dancer with short skirt; frames 110, 136 and 155. Figure 6 (middle row): Dancer with long skirt; frames 185, 215 and 236. Figure 7 (bottom row): Closeups from figures 4 and 6.