Simulating Inextensible Cloth Using Impulses

Juntao Ye
Institute of Automation, Chinese Academy of Sciences, Beijing, China

Abstract
Computer simulation of cloth is often plagued by springs being over-stretched. The evaluation of impulses to prevent over-stretching is explained step-by-step. Our impulse approach controls the length of springs by a nonlinear system and then a novel linearization of it into a symmetric positive definite system. The cloth/solid collision handling is integrated into the linearized system seamlessly. Some results based on this method are also presented.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Physically based modeling

1. Introduction
Due to the fact that woven fabric is a complex mechanical network of interleaving yarn [BHW94], cloth simulation is often based on a mass-spring model. The fabric is represented by a 2D array of nodes, each having a given mass, and each being related to its neighbors by mutual stretching, shearing and bending. An interesting aspect of cloth is the way it stretches. Real woven fabric offers weak resistance to stretching in the initial phase of extension, but strong resistance as the threads lengthen. Threads seldom stretch more than 15% of their rest length. However, the springs in many cloth solver often exceed that limit (Figure 1). Very high stretching coefficients are seldom adopted to overcome the excessive elongation, because the solvers suffer performance loss as the material stiffness increases. The condition number of the resulted matrix grows with the increment of material stiffness, forcing an iterative solver to perform more steps to converge.

In this paper we alleviate this issue by an impulse based approach as a remedy for the cloth dynamics solver. We are motivated by the work of Bridson et al. [BFA02], using strain limits to filter velocities of the nodes as a second pass. Our solution to this problem is to let each spring generate impulse, which is applied to both end-nodes. The impulses are the unknowns of a linear system that approximate a nonlinear system of constraints on the velocities. In order to solve the linear system efficiently, we first prove that the coefficient matrix is symmetric positive definite, thus solving the system is done by sparse Cholesky decomposition with permutations. Our method also has solid/cloth collision seamlessly integrated into one system, which has been largely ignored by existing methods.

2. Related Work
A number of researchers have worked on the stretch resistance problem. Provot [Pro95] was the first to try to
give a solution by adjusting node positions. Over-stretched springs were identified after each step, then their endpoints were moved inward to keep the elongation to no more than 10%. This approach was also adopted by Desbrun et al. [DSB99,MDDB01]. One problem with this method was that the iteration could not be proved to converge. As a remedy, sorting was performed before position adjustments. For example, Kang et al. [KCC00] sorted all springs according to their elongated lengths, and Dochev and Vassilev [DV03] sorted all unconstrained nodes according to their distances to constrained ones. These position-adjusting methods moves objects without reference to an underlying dynamics and thus do not conserve physical properties such as momentum. In addition to those position-adjusting methods, Vassilev et al. [VSC01] and Bridson et al. [BFA02] tried to solve the problem by velocity-adjusting. They applied impulses to the nodes to adjust their velocities iteratively. Volino et al.’s momentum transfer method [VCM95] was believed to be similar to Bridson’s work. Recently, Tskinis [TB06] extended Bridson’s work and proposed triangle-based strain limiting.

An alternative is to use constrained dynamics [Wit01], which directly calculates the constraint forces to meet the requirement of constraints (constraining the spring length). Lagrange multipliers are used to enforce the constraints. An application of this method to cloth simulation was done by [HD96]. Hong et al. [HCJ05] used a linearized implicit formulation in order to improve stability of constrained dynamics. Most recently, Goldenthal et al. [GHF07] adopted constrained Lagrangian Mechanics to prevent excessive extension and proposed a novel fast project for efficient computation. Their method outperforms all other methods in both speed and stability. However, all works so far were silent on how to deal with collision constraints when adjusting the velocities/positions.

We follow Bridson’s approach and restrict extension to 10% and compression to 2%. The advantage of enforcing stronger resistance to compression is to introduce more out-of-plane bending, creating more folds and wrinkles. Unlike Bridson, who used Jacobin iterative approach to update node velocities, we construct a linear system that approximates a nonlinear system of constraints on the velocities of the nodes. Our method deals with stretching springs only, because we believe that the over-extension of the stretching springs is the most visible of the inaccuracies of the mass-spring model. Constraints are seamlessly incorporated into our impulse method to handle cloth/solid collisions and user interventions, which is one of the major contributions of this paper.

3. Add Impulses to the Cloth Dynamics Model

Goldenthal et al. [GHF07] has pointed that imposing inextensibility on all edges of a triangle mesh is nearly impossible, as the system would quickly run into locking. Therefore, the cloth in our experiment is based on a rectangular mesh, and linear springs are employed for stretching, shearing and bending. Our bending model follows Choi and Ko’s setup [CK02] to enrich folds and wrinkles. The ordinary differential equations (ODEs) is solved numerically by backward Euler according to [BW98]. We adopt moderate stiffness value for the stretching springs, and rely on the impulse mechanism to cure the excessive elongation.

Following Bridson [BFA02], the strain limits are set ahead of time by the user to be a and b. A spring is neither allowed to stretch beyond b, nor allowed to shrink shorter than a. The spring tension is a linear function of the spring length according to $f_s = k \left( |\mathbf{x}_{ij}|-l_0^a \right) \hat{\mathbf{x}}_{ij}$ when $|\mathbf{x}_{ij}| \in [a,b]$, where $\hat{\mathbf{x}}_{ij} = \frac{x_{ij}}{|\mathbf{x}_{ij}|}$. When $|\mathbf{x}_{ij}| \notin [a,b]$, impulses are generated and applied to the two end-nodes, so that the spring length after one step of update will be within the limits. This method works as if there was a massless string of length b passing through the spring, as in Figure 2. The string becomes tight and generates impulses when the spring is about to stretch beyond b.

We favor impulse $\mathbf{I}$ over force $\mathbf{f}$ due to the fact $\mathbf{I} = \mathbf{f} t = m \dot{\mathbf{v}}$, allowing us to relate impulse directly to velocity change without having to compute forces explicitly. If force were used, we would need to know not only the magnitude of the force, but also the time during which the force is in effect. How to compute the impulses (or velocity changes) is our new task.

We suppose a spring $x_{ij}$, connecting node $x_i$ and $x_j$, generates impulse $I_{ij}$ to $x_i$ and impulse $-I_{ij}$ to $x_j$, their directions being co-linear with that of $\mathbf{x}_{ij}$. We define $I_{ij} = s_{ij} \dot{\mathbf{x}}_{ij}$, where $s_{ij}$ is the unknown scalar specifying the magnitude of the impulse. All nodes are assumed to have equal mass, i.e., $m_i = m_j$. Thus node $x_i$ receives velocity change of $\frac{s_{ij} \dot{\mathbf{x}}_{ij}}{m_i}$, and node $x_j$ receives $\frac{s_{ij} \dot{\mathbf{x}}_{ij}}{m_j}$. To be consistent, we make an assumption that the first index is always smaller than the second (i.e., $i < j$), so the positive sign is assigned to the smaller index. A node receives impulses from all its incident springs. For example, Figure 3 shows a graph of seven springs connecting eight nodes labeled with the indexes satisfying $f < g < h < i < j < k < l < n$. The velocity changes for nodes $x_i$ and $x_j$ are

$$
\delta \mathbf{v}_i = \frac{1}{m_i} (s_{ij} \dot{\mathbf{x}}_{ij} + s_{ji} \dot{\mathbf{x}}_{ji} - s_j \dot{\mathbf{x}}_{fi} - s_i \dot{\mathbf{x}}_{hi}),
$$

Figure 2: Spring with an imaginary non-stretchable string inside.

Figure 3: Seven neighboring springs.
\[ \delta v_j = \frac{1}{m_j}(-s_{ij}\dot{x}_{ij} - s_{xj}\dot{x}_{xj} + s_{jk}\dot{x}_{jk} + s_{mk}\dot{x}_{mk}). \quad (2) \]

Suppose at time \( t_0 \), the spring length is \( L(t_0) = |x_j - x_i| \). Once the ODE solver computes the new velocities \( v_j(t_0) \) and \( v_x(t_0) \), the nodes is supposed to move to new positions accordingly. We predict the spring length at time \( t_0 + h \) to be \( \hat{L}(t_0 + h) = |(x_j(t_0) + v_j(t_0)h) - (x_i(t_0) + v_i(t_0)h)| \). If \( \hat{L}(t_0 + h) \not\in [a,b] \), it means the spring will be over-stretched or over-compressed. If so, we introduce impulses to further change the node velocities so that the final spring length

\[ L(t_0 + h) = |L(t_0) + (v_j(t_0) + \delta v_j)h - (v_i(t_0) + \delta v_i)h| \quad (3) \]

satisfies \( L(t_0 + h) \in [a,b] \). To solve for the unknowns \( s_{ij} \) contained in \( \delta v_j \) and \( \delta v_i \), we could choose the value for \( L(t_0 + h) \) according to the predicted value of \( \hat{L}(t_0 + h) \), i.e., we set

\[ L(t_0 + h) = \begin{cases} \ b & \text{if } \hat{L}(t_0 + h) > b; \\ \ a & \text{if } \hat{L}(t_0 + h) < a; \\ \hat{L}(t_0 + h) & \text{otherwise}. \end{cases} \]

This way, we get a system of quadratic equations with each stretching spring corresponding to one such equation and \( s_{ij} \) being the unknowns.

However, such a nonlinear system is not easy to solve. Considering the term \( \delta v_j \) depends on \( x_{xj} \), which is a function of time \( t \), solving the system can be even more complicated. We believe that formulating and solving such nonlinear system directly is more costly than the Lagrange Multiplier approach. However, since visual satisfactory, not physical accuracy, is pursued in computer animation, we approximate the above nonlinear system with a linear system of equations. This approximation is not guaranteed to result in every spring being within the limits at every step. But in our experiments, it tends to produce springs that are within the limits. Moreover, it greatly enhances the stability of the underlying ODE solver because modest spring coefficients are used. With this approximation, integrating collision constraints into the system becomes possible.

4. Linearization of the Impulse Method

Now we describe the linearized approximation of our impulse approach. Intuitively, a spring deforms due to the relative velocity of its two end-nodes. The relative velocity can be split into two components, one is along the direction of the spring (which we will call the \textit{in-line component} hereafter) and the other is normal to the direction of the spring (called the \textit{normal component}). The in-line component is the main source of the stretching/compression. Although the normal component could also cause elongation, its contribution within one step is negligible if the time step is small.

The normal component makes a spring change its orientation within each time step, i.e., it undergoes a rotation. This makes the in-line component keeps changing its direction, and only its direction at the beginning of the time interval is known. We make an approximation here and assume that the direction of the in-line component is unchanged over the whole time step. This approximation works like the forward Euler’s method for solving an ODE in which the derivative at the beginning of one time step is used to advance the integration for the whole time step. After the application of the impulses, the in-line component of the velocity \( \dot{v}^{new} \) has magnitude \( (v_j(t_0) - v_i(t_0) + \delta v_j - \delta v_i) \cdot \dot{x}_{ij}(t_0) \). Now we set

\[ (v_j(t_0) - v_i(t_0) + \delta v_j - \delta v_i) \cdot \dot{x}_{ij}(t_0) = \frac{z}{h}, \quad (4) \]

where \( z \) is the desired length change of the spring in one time step. The value of \( z \) is determined according to the current state of each spring:

\[ z = \begin{cases} \ b - L(t_0) & \text{if } L(t_0) > b; \\ a - L(t_0) & \text{if } L(t_0) < a; \\ 0 & \text{if } L(t_0) \in [a,b]; \\ h((v_j(t_0) - v_i(t_0)) \cdot \dot{x}_{ij}(t_0)) & \text{otherwise}. \end{cases} \]

We want to find impulses that will satisfy these constraints. By rewriting Eq. 4 as

\[ (\delta v_j - \delta v_i) \cdot \dot{x}_{ij}(t_0) = (v_j(t_0) - v_i(t_0)) \cdot \dot{x}_{ij}(t_0) - \frac{z}{h}, \quad (5) \]

and using Eq. 1 and 2, we can construct a linear system of equations \( Ax = b \). The vector \( s \in |E| \) is the solution for the unknowns \( s_{ij} \), where \( E \) is the set of all stretching springs. Then the spring \( x_{ij} \) can also be indexed as the \( p \)-th element of the set: \( e_p \in E \). The square matrix \( A \in |E| \times |E| \) has one row (or column) for each stretching spring \( e_p \) (or the corresponding \( x_{ij} \)). Each stretching spring, if not on the border of the mesh, has six neighboring stretching springs. Border springs have fewer neighbors. Therefore, the maximum number of nonzero elements in each row is seven, which means \( A \) is a sparse matrix. The right-hand side of Eq. 5 is used to form vector \( b \), according to each of the four different cases to which the spring belongs. The \( (p,q) \)-th element of \( A \) is denoted by \( a_{pq} \). Suppose the \( q \)-th stretching spring \( e_q \) (or \( x_q \)) is defined by nodes \( x_j \) and \( x_k \), which corresponds the \( q \)-th row of \( A \). Since \( e_p \) and \( e_q \) share a common vertex, we have

\[ a_{pp} = \frac{1}{m_j} = \frac{1}{m_j}, \quad a_{pq} = a_{qp} = \frac{(\dot{x}_{ij} \cdot \dot{x}_{jk})}{m_j}. \]

making \( A \) a symmetric matrix.

In a quadrilateral mesh of \( n \) nodes, the number of stretching springs is \( 2n \), so the size of matrix \( A \) is \( \approx 2n \times 2n \), the same as the matrix constructed by Goldenthal et al. [GHH*07] from the fast projection. This system is considerably cheaper to evaluate, assemble and solve than the traditional Constrained Lagrange Mechanics system.
5. Maintaining Constraints within the Impulse Method

In a cloth simulation system, collisions or user intervention often restrict the motion of particles. A particle can be: (1) completely unconstrained; (2) prevented from accelerating along a specific direction; (3) prevented from accelerating along a specific plane; or (4) completely constrained, i.e., not allowed to accelerate along any direction. An example of (2) could be a particle in contact with the surface of a rigid object and only sliding being allowed, thus acceleration along the surface normal is prohibited. A situation for (3) could be curtain simulation: cloth nodes fixed to gliders are prohibited from moving along planes perpendicular to the curtain rod.

We represent constraints using Baraff and Witkin’s [BW98] constraint matrix. We use their notation ndof(\(x_i\)) to indicate the number of degrees of freedom particle \(x_i\) has. Our way of constructing the constraint matrix is almost the same as that of Baraff and Witkin, except for case 3 (ndof(\(x_i\)) = 1). Let particle \(x_i\)’s prohibited direction be \(\hat{p}_i\) for case 2 (ndof(\(x_i\)) = 2), or the normal vector of the prohibited plane be \(\hat{n}_i\) for case 3 (ndof(\(x_i\)) = 1). A constraint matrix \(C_i \in \mathbb{R}^{3 \times 3}\) is defined as a matrix that when applied to the unconstrained velocity of particle \(x_i\), it produces the constrained velocity \(\delta \dot{v}_i^c = C_i \delta \dot{v}_i\). The matrix \(C_i\) is

\[
C_i = \begin{cases} 
1 & \text{if ndof}(x_i) = 3 \\
1 - \hat{p}_i \hat{p}_i^T & \text{if ndof}(x_i) = 2 \\
\hat{n}_i \hat{n}_i^T & \text{if ndof}(x_i) = 1 \\
0 & \text{if ndof}(x_i) = 0
\end{cases}
\]

(6)

where \(I\) is the 3 \times 3 identity matrix. When ndof(\(x_i\)) = 1, Baraff and Witkin defined the plane using two orthogonal unit vectors \(\hat{p}_i\) and \(\hat{q}_i\) in that plane: \(C_i = I - \hat{p}_i \hat{p}_i^T - \hat{q}_i \hat{q}_i^T\).

Most of the time, \(\hat{p}_i\) and \(\hat{q}_i\) have to be determined from the plane normal vector \(\hat{n}_i\), which is more readily available and thus allowing us to compute the result in fewer operations. Actually, it is not difficult to verify that \(I - \hat{p}_i \hat{p}_i^T - \hat{q}_i \hat{q}_i^T = \hat{n}_i \hat{n}_i^T\).

If node \(x_i\) is constrained, its velocity change \(\delta \dot{v}_i^c\) should be along the allowed direction(s) only: \(\delta \dot{v}_i^c = C_i \delta \dot{v}_i\). Thus Equ. 1 and Equ. 2 change to

\[
\delta \dot{v}_i^c = \frac{C_i}{m_i} (s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) - s_{ij} \dot{x}_{ij}(t_0) - s_{ij} \dot{x}_{ij}(t_0))
\]

(7)

and

\[
\delta \dot{v}_i^c = \frac{C_i}{m_j} (-s_{ij} \dot{x}_{ij}(t_0) - s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0)).
\]

(8)

The component of \(\delta \dot{v}_i\), along the prohibited direction is \((I - C_i)\delta \dot{v}_i\), which is canceled by the constraint forces. Note that our impulse method conserves both linear and angular momentum when no constraints is exerted onto the system. With the constraints, the system is no longer conservative. It is not a surprise since the constraints introduce external forces into the system, changing the momentum.

6. Symmetry and Positive Definiteness of the Coefficient Matrix \(A^c\)

A nice thing about our way of linearization and applying constraints is that the resulted matrix is symmetric and positive definite. In this section we give a brief proof.

From Equ. 7 and Equ. 8, elements of constrained impulse matrix \(A^c\) are

\[
a_{ij}^c = \frac{C_i}{m_i} s_{ij} \dot{x}_{ij} + \frac{C_j}{m_j} \dot{x}_{ij},
\]

(9)

\[
a_{ij}^c = -\frac{C_j}{m_j} \dot{x}_{ij}.
\]

(10)

If \(a_{ij}^c = a_{ij}^p\) holds, matrix \(A\) is symmetric. Actually, it is not hard to prove that for any 3 \times 1 vectors \(u\) and \(w\), \(C_i u \cdot w = C_i w \cdot u\) holds. Therefore, matrix \(A\) is symmetric.

Next, we will prove that \(A^c\) is positive definite. Before doing that, we give a useful equation

\[
C_i u \cdot C_i w = C_i w \cdot C_i u, \text{ for any } 3 \times 1 \text{ vectors } u \text{ and } w.
\]

(11)

This equation can be proved as \(C_i u \cdot C_i w = u^T C_i^T C_i w = u^T C_i w = (C_i u)^T w = C_i u \cdot w\).

The positive definiteness of \(A^c\) can be validated by proving \(s^T A^c s = \sum_{c \in N} m_i |\delta \dot{v}_i|^2\), for any nonzero vector \(s\), where \(N\) is the set of all nodes. Due to \(m_i |\delta \dot{v}_i|^2 = m_i (\delta \dot{v}_i \cdot \delta \dot{v}_i)\), expanding it according to Equ. 7 and Equ. 8 and applying Equ. 11 yields

\[
m_i |\delta \dot{v}_i|^2 = \frac{1}{m_i} \sum_{c \in N} \left( \begin{array}{c} s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) \\
+ s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) \\
+ s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) \\
+ s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) \end{array} \right)
\]

(11)

Il Re-organizing \(\sum_{c \in N} m_i |\delta \dot{v}_i|^2\) by numbering all edges as oppose to nodes, we get

\[
= \sum_{c \in E} \left( \begin{array}{c} s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) \\
+ s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) \\
+ s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) \\
+ s_{ij} \dot{x}_{ij}(t_0) + s_{ij} \dot{x}_{ij}(t_0) \end{array} \right)
\]

(11)

where \(E\) is the set of all stretching springs. Unless all \(\delta \dot{v}_i\) being zero, \(A^c\) is strictly positive. In the case of all \(\delta \dot{v}_i\) being zero, no velocity adjustment is needed, so the process of assembling and solving such a system is not activated at all. We can therefore consider all linear systems constructed to be symmetric positive definite, which enable us to choose a special algorithm for solving it.
Please note if both nodes of an edge are completely constrained, the corresponding row and column are null vectors, making the matrix rank deficient. In this case, that row and column need to be deleted, resulting in a matrix of size $|E| - 1$, which is still symmetric positive definite.

7. Solving the Linear System

Although sparse linear systems is typically solved by iterative methods, particularly the Conjugate Gradient method for symmetric positive definite systems, direct methods should not be neglected. A direct method is either the Gaussian elimination or its variant tailored to exploit the special structure of the matrix. Cholesky factorization is such a variant for symmetric positive definite matrices: $A = LDL^T$, where $L$ is the lower triangular matrix with all diagonal elements being one, and $D$ is a diagonal matrix. Thus solving the system $Ax = b$ turns into solving three sub-systems $Ly = b$, $Dz = y$ and $L^Tx = z$.

One issue with direct methods is that when a sparse matrix $A$ is factored, it typically suffers some fill-in and becomes less sparse. The fill-in means the triangular factor $L$ has non-zeros in positions which are zero in $A$. It is possible to reduce the amount of fill-in by reordering the matrix prior to the factorization, thus saving computer execution time and storage. A reordering means a symmetric permutation of the rows and columns of $A$, and the same permutation applied to $b$. Permutation of the rows is done by left-multiplying $A$ with a permutation matrix $P$, and permutation of the columns is done by right-multiplying $A$ with $P^T$. Once the reordering is done, factorizing $PAP^T$, instead of $A$, into $LDL^T$ requires less calculation and leads to much less fill-in in $L$. Thus instead of solving $Ax = b$, we solve $(PAP^T)(Px) = Pb$.

For the direct factorization, finding an optimal permutation that results in the minimum amount of fill-in is NP-hard [HEKP01]. Heuristics are usually used to find a decent permutation. Research on how to find a good permutation matrix $P$ has made much progress in the past few years. Particularly, Davis’s group provides some exciting results [ADD04] [CDHR06] [DH99] [DH01] [DH05]. The Approximated Minimum Degree (AMD) method, given by this group, is the most efficient in the family of Minimum Degree Reordering. To solve $As = b$, we can first use AMD to find a close-to-optimal permutation for matrix $A$. We then permute $A$ and $b$ followed by $LDL^T$ factorization and backward and forward substitution. Doing an inverse permutation to the solution vector gives the solution to the original system.

Which is better, the iterative method or the direct factorization, is really application-dependent. If a matrix is numerically diagonal-dominant, the CG method converges quickly when using its diagonal as the preconditioner. A proper initial guess for the solution is also very helpful to the convergence. However, the matrix in our system is far from diagonal-dominant, and a better-than-the-diagonal preconditioner can not be easily found. Moreover, it is not easy to find a decent initial guess but to use a zero vector. On the other hand, for the direct method, the performance of AMD usually increases nonlinearly as $A$ gets sparser. For our system, the number of nonzeros of each row is seven, with which the AMD performs fairly well. Our test results verified that AMD plus the factorization is at least 30% more efficient than CG algorithm in terms of the execution time.

There is another reason that makes us favor the direct factorization with AMD reordering over the CG method for our linear system. Table 1 shows the five stages of the direct method and their CPU time percentage for each stage while simulating a square cloth of 1,681 nodes for 1,000 steps (Figure 4). The first three stages depend only on the nonzero pattern of a matrix, not its numerical values. It is often the case that for a number of consecutive steps the matrix does not change its pattern, therefore these three steps only need to be done once. The matrix pattern changes only when a spring is removed from or added to the impulse system due to both end-nodes being completely constrained or released. This situation does not happen frequently, thus skipping the three stages will cut the CPU time for the impulse solver by 36%.

![Figure 4: Cloth with 1,681 nodes suspended from two corners, and swinging down after one corner released.](image-url)
Table 1: Task of the direct method breakdown

<table>
<thead>
<tr>
<th>subroutines</th>
<th>time (sec)</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD to find P</td>
<td>2.40</td>
<td>18.6</td>
</tr>
<tr>
<td>permutation PAP$^2$</td>
<td>1.86</td>
<td>14.5</td>
</tr>
<tr>
<td>symbolic factorization</td>
<td>0.39</td>
<td>3.0</td>
</tr>
<tr>
<td>numeric factorization</td>
<td>6.37</td>
<td>49.4</td>
</tr>
<tr>
<td>$Ly = b, Dz = y$</td>
<td>1.87</td>
<td>14.5</td>
</tr>
<tr>
<td>total</td>
<td>12.89</td>
<td>100.0</td>
</tr>
</tbody>
</table>

8. Experiment and Conclusion

We have described an impulse based method which intends to be used as an add-on to any existing simulation system that suffers the over-stretching problem. We set up several simulation scenarios to test the performance of our method. All experiments were tested on a Intel Core 2 Duo 2.13G system, but no multi-threading programming is involved.

The first example is the simulation of a sheet hanging by two/one corners and subject to gravity (Figure 4). The length of springs at the fixed corner(s) is within a desired limit. We also used the impulse method on a larger cloth mesh of 6,561 nodes, hit by a flying ball (Figure 5). The proper behavior is for the cloth to get out of the way of the ball rather than letting the ball stretch the cloth. Our approach also works well on more complicated meshes, such as T-shirt (Figure 7). For a mesh containing limited number of nodes, we are able to achieve real-time performance (Figure 6). The impulse method not only greatly alleviates the problem of extensibility, but also creates enough folds to greatly enhance the realism of the animation.

In order to integrate collision handling into the impulse based method, we adopted a special approximation to the nonlinear constrained system. Compared to the fast projection in [GHF∗07], our approximation scheme does not perform equally well in terms of convergence. If the predicted spring lengths are going to exceed the limit by too much, either because of a too-large time step or because of the spring coefficient being too low, the impulse application will result in a "rugged" cloth, creating unneeded out-of-plane displacement. However, it does not blow out the computation. If this happens, decreasing the time step or increasing the spring stiffness improves the result. For a mesh of modest size, e.g. the one having 1,681 nodes, the impulse method can be combined with the explicit Euler integration to take advantage of the small time step, and still achieve efficient overall execution time. In the future, we will certainly refine our linearization scheme to allow larger time step.

Acknowledgements

The author is grateful to Robert E. Webber and Irene Gargantini for their discussions. This work was partially funded by the National High Technology Research and Development Program of China (863 Program) #2007AA01Z341.

References


Figure 5: Ball hitting a piece of cloth modeled by 6,561 nodes, $\Delta t = 2.22$ms and the average simulation time was 5.5 seconds/frame taking no consideration of self-collision handling.

Figure 6: Real-time simulation of a skirt modeled by 108 nodes at 140 frames per second.

Figure 7: A T-shirt modeled by 2,792 nodes dropped onto a dummy model.


