



저작자표시-비영리-동일조건변경허락 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.
- 이차적 저작물을 작성할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



동일조건변경허락. 귀하가 이 저작물을 개작, 변형 또는 가공했을 경우에는, 이 저작물과 동일한 이용허락조건하에서만 배포할 수 있습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

공학박사학위논문

Linear Models for Real-time Clothing Simulation

실시간 의복 시뮬레이션을 위한 선형 모델 연구

2013년 8월

서울대학교 대학원
전기·컴퓨터 공학부

최 봉 욱

Abstract

Linear Models for Real-time Clothing Simulation

Bong-Ouk Choi

Department of Electrical Engineering and Computer Science

The Graduate School

Seoul National University

Deformations occurring in cloth can be decomposed into two components: the in-plane and the out-of-plane deformations. Stretch and shear are in-plane deformation, and bending is out-of-plane deformation. Clothing simulation involves all the above types of deformations. This paper proposes a new physical model for deformations of clothes. The significance of the proposed models is that (1) their numerical simulation can be done in real-time, and (2) the models fix some flaws that existed in previous real-time models, leading to conspicuous reduction of artifacts. The essential idea in inventing the new models is to replace $(|\mathbf{x}| - C)^2$ in the energy function with $|\mathbf{x} - \mathbf{x}^*|^2$ for some constant vector \mathbf{x}^* . Then, the force jacobian becomes a constant, and so does the system matrix. As a result, its inverse matrix can be pre-computed only once in off-line, so that the on-line semi-implicit integration can be replaced with (the constant) matrix-vector multiplications. This paper develops such simplified physical models for both edge-based and triangle-based systems. In ad-

dition, to speed up the process of matrix-vector multiplications, this work reviews the current state-of the art in the sparse Cholesky factorization methods and introduces an effective method for the current purpose.

Keywords: physically based animation, cloth simulation, real-time simulation, linear model, sparse Cholesky factorization

Student Number: 2003-21717

Contents

Abstract	i
Contents	iii
List of Figures	v
List of Tables	vii
1 Introduction	1
1.1 Notations	3
1.2 Edge-Based Formulation of Stretch Energy and Force	4
1.3 Explicit Formulation	5
1.4 Implicit Formulation	6
2 Related Work	9
3 Edge-Based Linear Stretch Model	13
3.1 Conventional Stretch Model	14
3.2 Our Stretch Model	15

3.3	Representation of Shear Deformations	20
3.4	A Killer Application of This Model	21
4	Triangle-Based Linear Stretch/Shear Model	22
4.1	Material Space to 3D Space Mapping S	23
4.2	Conventional Stretch and Shear Model	24
4.3	Our Stretch and Shear Model	24
5	Linear Bending Model	28
5.1	Calculating Bending Vector	28
5.2	Applying Bending Force	30
5.3	Jacobian of the Bending Force	31
6	Sparse Cholesky Factorization	32
6.1	Cholesky Factorization	32
6.2	Reordering	40
7	Experimental Results	48
8	Conclusion	65
	Bibliography	67
	초 록	71

List of Figures

1.1	The mass-spring model	4
5.1	Creating bending force in a triangle mesh	29
6.1	System matrix diagram of linear systems	36
6.2	Examples of matrices and those adjacency graphs	38
6.3	An example of the graph model of an elimination	39
6.4	An extreme case of the graph model of an elimination	40
6.5	Effect of ordering	41
6.6	The original and the ordered arrow matrix A	42
6.7	Examples of the minimum degree ordering	43
6.8	Examples of the nested dissection ordering	45
6.9	Examples of Cuthill-McKee & reverse Cuthill-McKee ordering	46
7.1	An effect of reordering in low resolution	49
7.2	An effect of reordering in medium resolution	50
7.3	An effect of reordering in high resolution	51
7.4	An effect of reordering in different meshes	53

7.5	An effect of reordering in different meshes	54
7.6	An effect of reordering in different meshes	55
7.7	The swinging cloth test	57
7.8	The sphere test in edge-based systems	63
7.9	The sphere test in triangle-based systems	64

List of Tables

6.1	Minimum and maximum eigenvalues of the system matrix . . .	37
7.1	This table summarizes the statics in reordering at different resolutions of Figure 7.1, 7.2, and 7.3	52
7.2	This table summarizes the statics in reordering at different shapes of Figure 7.4, 7.5, and 7.6	56
7.3	This table summarizes the statistics in simulating swinging cloth at three different mesh resolutions: from top to bottom, the number of vertices, the number of triangles, the number of non-zero terms in system matrix, the number of non-zero terms in original lower triangular matrix, the number of non-zero terms in reordered lower triangular matrix, the average computation time per each time step at different models and speed gain with respect to conventional model.	58

- 7.4 This table present a computation time of one time step of swinging cloth simulation in detail: from top to bottom, the time for calculating forces, the time for building system matrix, the time for solving system, the time for collision handling, the total time for integrating one time step and speed gain with respect to previous non-linear model. From left to right, each time is measured in the conventional edge-based model, the linear edge-based IER model and the linear edge-based IVT model. The time unit is millisecond, and all simulations are executed with same rectangular mesh of 5472 vertices. 59
- 7.5 This table present a computation time of one time step of swinging cloth simulation in detail: from top to bottom, the time for calculating forces, the time for building system matrix, the time for solving system, the time for collision handling, the total time for integrating one time step and speed gain with respect to previous non-linear model. From left to right, each time is measured the in conventional triangle-based model, the linear triangle-based IAR model and the linear triangle-based IVT model. The time unit is millisecond, and all simulations are executed with same rectangular mesh of 5472 vertices. . . . 60
- 7.6 This table presents the computation time of forward/backward substitution of Cholesky factorization with or without reordering in an edge-based linear system at different resolutions. . . . 62

7.7 This table presents the computation time of forward/backward substitution of Cholesky factorization with or without reordering in a triangle-based linear system at different resolutions. . . 62

Chapter 1

Introduction

When we need to create virtual humans, whether it is in a movie, animation, game, or VR simulation, the problem of simulating dynamic movements of clothes arises. Unfortunately, clothing animation has been and still remains a nontrivial task to the animators. One of the reasons is that, because the speed of simulation is far from real-time, the animators cannot see the draping of the garments immediately.

This paper proposes a new technique for real-time simulation of clothes. This remarkable speed up is based on the use of a new energy formulation. This work presents that, under certain assumptions, with the proposed energy formulation, the system matrix that represents the effects of stretch, shear, and bending altogether reduces to a constant matrix. Then, its inverse matrix can be pre-computed only once in off-line, so that the on-line semi-implicit integration can be replaced with (the constant) matrix-vector multiplications. The

proposed technique cannot match off-line techniques in the animation quality. Nor the idea of using linear stretch and bending forces (so that their Jacobians become constant) is completely new. However, to our knowledge, linearization of all the deformations (i.e., stretch, shear, and bending forces) altogether has not been attempted yet, nor the result such linearization produces has been analyzed.

This work makes the following technical contributions: it proposes a new way of formulating the stretch and shear energy such that the derived force jacobian becomes constant, which fixes a flaw that existed in Desbrun et al.'s linear stretch model [9]. In the development of the linearized clothing simulator, as for the bending, we simply adopt the linear bending model proposed by Volino and Thalmann [21]. As for the stretch and shear, however, we develop new models from the scratch. This work is the first attempt to linearize the stretch, shear, and bending forces altogether in the same simulator. Unfortunately, having a constant system matrix does not imply fast simulation by itself, because a naïve inversion of the system matrix results in a non-sparse matrix, in which case the complexity of the matrix-vector multiplication can amount to $O(n^2)$, which is even more costly than the PCG. A few techniques have been studied to remedy such situation; Sparse Cholesky factorization methods can maintain the sparsity during the inversion through careful reordering of the linear equations. The use of sparse Cholesky factorization has not been explored yet in the context of linearized clothing simulation. This work reviews the current state-of-the-art in the Sparse Cholesky

factorization methods and introduces a most effective method for the current purpose. The linearized clothing simulation will exhibit artifacts compared to the conventional clothing simulation. This work makes an analysis of the linearized simulation, with the hope of characterizing the conditions when the proposed linearized clothing simulation does not experience severe artifacts.

Practically the proposed technique allows for several levels of quality vs. speed trade-offs, providing a wide gamut of scalability. It can have immediate uses in real-time applications such as game and VR simulation. It can be also employed into off-line systems for previewing the draping of the constructed clothes before performing more accurate simulations of them.

The subsequent sections are organized as follows. In the first two sections, we present linearizable physical models for stretch/shear deformations in the edge-based systems (Chapter 3) and in the triangle based systems (Chapter 4). Chapter 5 gives a brief introduction of the linear bending model proposed by Volino et al. Chapter 6 introduces how a sparse inverse of the system matrix can be obtained with a sparse Cholesky factorization. Chapter 7 reports a few experiments performed with the new simulator and gives analysis of those results. Finally Chapter 8 concludes the paper.

1.1 Notations

Imagine a piece of cloth. In the particle-based simulation, we internally represent it with N particles P_1, \dots, P_N . Let's denote the position and mass of each

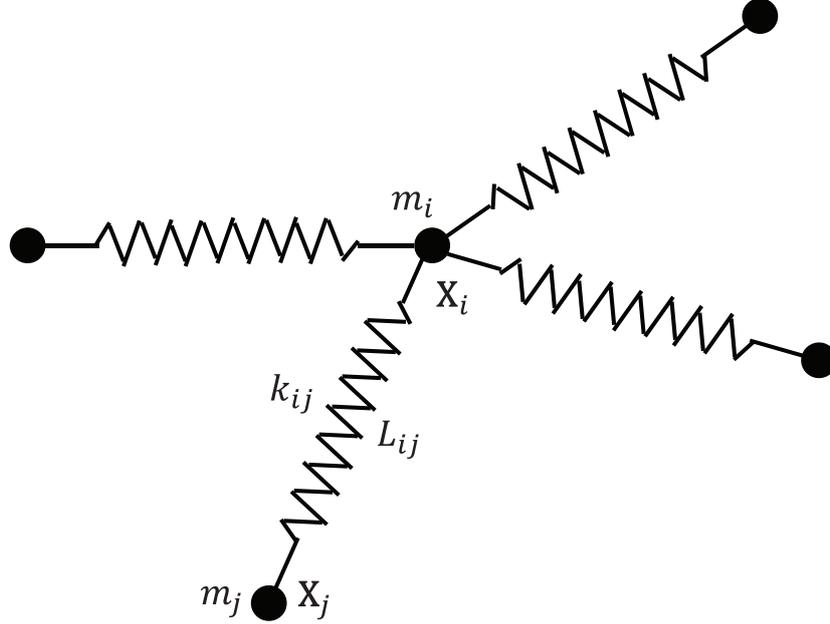


Figure 1.1 The mass-spring model

particle as \mathbf{x}_i and m_i , respectively, as shown in Figure 1.1. Let \mathbf{x} be the $3N$ -dimensional vector which is constructed by concatenating $\mathbf{x}_1, \dots, \mathbf{x}_N$. \mathbf{x}_i and \mathbf{x} vary over time t . To explicitly denote the fact, we may write them as $\mathbf{x}_i(t)$ and $\mathbf{x}(t)$. To denote the values of \mathbf{x}_i and \mathbf{x} at n -th discrete time step, we use the notations \mathbf{x}_i^n and \mathbf{x}^n . Also, we use the notational convention $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$.

1.2 Edge-Based Formulation of Stretch Energy and Force

Conventionally, the stretch energy E_{ij}^s held on the deformed edge \mathbf{x}_{ij} is modeled by the formula

$$E_{ij}^s = \frac{1}{2}k_{ij}(|\mathbf{x}_{ij}| - L_{ij})^2, \quad (1.1)$$

where L_{ij} and k_{ij} are the rest length and the stiffness, respectively, of the edge. Once the energy potential is defined, then the restorative force acting on P_i that is contributed from the edge \mathbf{x}_{ij} is given by the spatial derivative of the energy potential:

$$-\frac{\partial E_{ij}^s}{\partial \mathbf{x}_i} = k_{ij}(|\mathbf{x}_{ij}| - L_{ij}) \frac{\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|}. \quad (1.2)$$

The above stems from the material's restoration tendency and is called the *internal force*. The restorative force \mathbf{f}_i acting on P_i from all the adjacent edges is given by the summation

$$\mathbf{f}_i = \sum_{j \in N(i)} k_{ij}(|\mathbf{x}_{ij}| - L_{ij}) \frac{\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|}, \quad (1.3)$$

where $N(i)$ is the set of indices of the particles which P_i is connected to.

1.3 Explicit Formulation

To find out the mechanical movement of a particle, we need to know the total force acting on the particle. In addition to the internal force, P_i also experiences external forces such as gravity and air drag. The details about the external forces are out of the scope of this paper. Let's just use \mathbf{f}_i^{ext} to denote summation of all the external forces acting on P_i . Let's use \mathbf{F}_i to denote the summation of all the internal and external forces acting on P_i , i.e., $\mathbf{F}_i = \mathbf{f}_i + \mathbf{f}_i^{ext}$. Newton's second law states that the movement of P_i is governed by the equa-

tion

$$m_i \ddot{\mathbf{x}}_i = \mathbf{F}_i. \quad (1.4)$$

Equation 1.4 should be formulated for every (non-constrained) particles, which can be assembled into a system of differential equations

$$\mathbf{M} \ddot{\mathbf{x}} = \mathbf{F}, \quad (1.5)$$

where \mathbf{M} and \mathbf{F} are the $3N \times 3N$ mass matrix and $3N \times 1$ force vector, respectively. Solving the above equation gives time-varying trajectories of the particles.

1.4 Implicit Formulation

The above explicit formulation sounds intuitive. Moreover, it does not call for solving a system of linear equations thus is straightforward to implement. Unfortunately, it is prone to numerical instabilities; Unless we use very small time steps, the system often diverges. The implicit formulation we describe below has been known to be more stable. So the models we propose in this paper are all based on the implicit formulation.

In comparison with the explicit Euler method shown in Equation 1.5, the implicit Euler method samples the derivatives at t^{n+1} , i.e., the implicit Euler

method updates the system state according to

$$\begin{bmatrix} \Delta \mathbf{x} \\ \Delta \mathbf{v} \end{bmatrix} = h \begin{bmatrix} \mathbf{v}^n + \Delta \mathbf{v} \\ \mathbf{M}^{-1} \mathbf{F}(\mathbf{x}^n + \Delta \mathbf{x}, \mathbf{v}^n + \Delta \mathbf{v}) \end{bmatrix}. \quad (1.6)$$

We apply a first order Taylor series expansion on $\mathbf{F}(\mathbf{x}^n + \Delta \mathbf{x}, \mathbf{v}^n + \Delta \mathbf{v})$ to get

$$\mathbf{F}(\mathbf{x}^n + \Delta \mathbf{x}, \mathbf{v}^n + \Delta \mathbf{v}) = \mathbf{F}^n + \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \Delta \mathbf{x} + \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \Delta \mathbf{v}. \quad (1.7)$$

Substituting the above to Equation 1.6 and eliminating $\Delta \mathbf{x}$ produces

$$\left(\mathbf{I} - h\mathbf{M}^{-1} \frac{\partial \mathbf{F}}{\partial \mathbf{v}} - h^2 \mathbf{M}^{-1} \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \right) \Delta \mathbf{v} = h\mathbf{M}^{-1} \left(\mathbf{F}^n + h \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \mathbf{v}^n \right). \quad (1.8)$$

Now, the equation can be solved for $\Delta \mathbf{v}$, and the result can be used to compute $\Delta \mathbf{x} = \mathbf{v}^n + \Delta \mathbf{v}$. For later references, let's denote the system matrix and the right hand side vector of the above equation as

$$A = \left(\mathbf{I} - h\mathbf{M}^{-1} \frac{\partial \mathbf{F}}{\partial \mathbf{v}} - h^2 \mathbf{M}^{-1} \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \right) \quad (1.9)$$

$$b = h\mathbf{M}^{-1} \left(\mathbf{F}^n + h \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \mathbf{v}^n \right) \quad (1.10)$$

so that Equation 1.8 can be written as

$$A\Delta \mathbf{v} = b \quad (1.11)$$

A is a $3N \times 3N$ matrix, and b is a $3N \times 1$ vector. Compared to explicit methods, calculation of $\Delta \mathbf{v}$ out of Equation 1.11 calls for solving a large system of linear equations. A striking feature of the new technique is that the stretch/shear model we propose makes the system matrix A constant over time. Then, the matrix inversion can be pre-computed only once in off-line, so that the on-line simulation can be done fast. The above speed-up is the result of adopting a simplified (less accurate) physical model, details of which we will address later. A stretch model that causes A to become constant in the context of implicit formulation has already been proposed by Desbrun et al. [9]. Their simplification was based on overlooking the rotational movements, which obviously resulted in artifacts. The difference of our model from theirs is that our simplification accounts for the rotational movements. As a consequence, the proposed method can produce conspicuously improved results.

Chapter 2

Related Work

Thanks to the pioneering work of various groups over the past decade [5, 2, 20, 10, 1, 7, 3], cloth can now be simulated with remarkable realism. For example, natural wrinkles can now be produced using the particle model, and the robustness of the collision handling in cloth simulations has been considerably improved. Along with the improvements that have been made in animation quality, the overall simulation algorithm has been refined such that it runs at a reasonable speed. For example, producing a 30 seconds long animation of an outfit represented with about 10,000 particles takes a few days excluding the time for rendering.

However, there are several application areas in which simulation speed should be faster. For example, in games, animation of cloth should be generated in real-time. Desbrun, Schröder, and Barr [9] pioneered the problem of real-time simulation of cloth-like objects. To accelerate the semi-implicit method, they

omitted the non-linear force components and used only the linear components in calculating the force jacobian, which resulted in a constant system matrix. Therefore, once the inverted system matrix was pre-computed at the initial stage, the linear system did not need to be solved at every time step. Omission of the non-linear components could lead to artifacts. To compensate the possible errors from the ignored non-linear forces, the angular momentum correction steps were taken at the end of simulating each time step. But those steps could not completely fix the problem; Even though the correction steps could preserve global angular momentum, they could not preserve angular momenta of local regions. The physical model we propose also makes the system matrix constant. But we do not overlook the non-linear forces altogether in the jacobian calculation.

Kang, Choi, and Cho [16, 15] also proposed a simplification of the semi-implicit method that pre-computes the inverted hessian matrix or avoids solving the large linear system. To achieve $O(n)$ time complexity, the velocity change of each cloth particle is directly updated using explicitly estimated future velocities of the nearby particles that are connected to it.

Recently, Cordiner and Thalmann presented a real-time cloth animation system in [8]. They classified the cloth into three categories (tight, loose, floating) based on the cloth movement pattern and adopted different approach to animate them in each category. The movement of tight/loose regions is highly dependent on the body motion, and therefore such regions may not need full 3D simulation. Geometrical techniques were used for the tight and loose re-

gions. On the other hand, the particle system with semi-implicit method was used for the floating regions. Since the floating regions might take up only a small portion in normal garments, the above technique could achieve real-time animation.

About a bending force, several ideas have been proposed in the field, and they can be classified to two major approaches. The first is to use crossover springs that extend the surface, opposing transversal bending [17, 10]. The second is to evaluate accurately the angle between adjacent mesh elements and to create between them normal forces that oppose this angle through opposite bending momentum [14, 4, 20, 19]. This approach can reach similar accuracy as grid continuum-mechanics [18, 2] and grid particle system derivatives [1] which are fairly complex to evaluate.

The crossover spring approach is usually implemented in mass-spring system usually, so it is simple to implement and allowing a homogeneous simulation system. Unfortunately, this approach is also very inaccurate. On the other hand, the normal force approach can accurately apply bending forces according to precisely calculated bending angles, and those bending forces don't interfere with the stretch and shear forces, significantly. However, the computational cost of this model is much higher.

Volino and Thalmann propose an alternative approach in [21]. They combine fairly good accuracy for representing quantitative bending stiffness with a very simple and efficient computational procedure.

About sparse Cholesky factorization, there are several applications which

exploit matrix reordering problem [13, 11, 12]. With proposed techniques, Cholesky factorization methods can maintain the sparsity during the inversion.

Chapter 3

Edge-Based Linear Stretch Model

An edge-based system refers to mass-spring representation of cloth in which in-plane deformation is realized by the stretches along the edges. The elementary deformable unit is an edge. This section proposes the stretch energy function for the edge between two particles P_i and P_j , so that the restorative force and the force jacobians can be derived from it. Since it judges the deformation by looking at only limited part (i.e., only the edges) of the mesh, edge-based system makes more sense when the topological connectivity is kept the same at all edges. Therefore edge-based system is adopted mostly in the context of regular meshes. There are two kinds of regular meshes that can be considered for cloth simulation: regular-triangular meshes and regular-rectangular meshes. When a rectangular mesh is used, shear deformation can be simulated using the above edge-based stretch model by making diagonal connections.

3.1 Conventional Stretch Model

The stretch force and its jacobians are derived from the energy function. So, the task of developing a physical model that leads to constant system matrix is reduced to finding a new energy function that has such simplifying property. Differentiation of the conventional energy function

$$E_{ij}^s = \frac{1}{2}k_{ij}(|\mathbf{x}_{ij}| - L_{ij})^2 \quad (3.1)$$

produces

$$\mathbf{f}_i = \sum_{j \in N(i)} k_{ij}(|\mathbf{x}_{ij}| - L_{ij}) \frac{\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|}, \quad (3.2)$$

which is not linear with respect to \mathbf{x}_i or \mathbf{x}_j . Therefore, the force jacobian $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ is not constant over time. Then, the linear system given in Equation 1.11 should be solved at every time step.

Let's go back to Equation 3.1 and see what is the critical feature of the equation that hinders the linearization. We note that if there had been no norm operator $|\cdot|$ in the equation, then \mathbf{f}_i would have been linear. So the problem can be rephrased as: *Can we find an energy function that obviates the use of the norm operator but its value is close to that of Equation 3.1?*

3.2 Our Stretch Model

We use the energy function

$$E_{ij}^s = \frac{1}{2}k(\mathbf{x}_{ij} - \mathbf{x}_{ij}^*) \cdot (\mathbf{x}_{ij} - \mathbf{x}_{ij}^*) \quad (3.3)$$

where \mathbf{x}_{ij}^* is some vector that is newly calculated at each time step but regarded constant when differentiating. There is no norm operator in the energy. Therefore the force jacobian $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ is now constant.

An immediate question that can be raised would be whether there can exist the vector quantity \mathbf{x}_{ij}^* such that use of the above energy function produces acceptable stretch behavior of cloth? We find the answer is yes through our experiments.

Then, what value should we use for \mathbf{x}_{ij}^* ? To make Equation 3.3 match Equation 3.1 in the physical meaning, ideally, \mathbf{x}_{ij}^* should represent an unstretched version of \mathbf{x}_{ij} . We can express the situation as the conditions: $|\mathbf{x}_{ij}^*| = L$ and $\mathbf{x}_{ij}^* \parallel \mathbf{x}_{ij}$, where \parallel means “paralle”. Note that we are using Equation 3.3 for updating the value of \mathbf{x}_{ij} , from \mathbf{x}_{ij}^n to \mathbf{x}_{ij}^{n+1} . Therefore, the second condition can be more explicitly phrased as: $\mathbf{x}_{ij}^* \parallel \mathbf{x}_{ij}^{n+1}$. Here, the problem is that \mathbf{x}_{ij}^{n+1} is not known yet. One way to circumvent this problem would be to use an estimated direction of \mathbf{x}_{ij}^{n+1} .

Desbrun et al. [9] states that “... we simply decide to overlook the rotation, and suppose that this non-linear part will stay constant ...”. In our notation, their decision amounts to using $\mathbf{x}_{ij}^n / |\mathbf{x}_{ij}^n|$ for the estimated direction of \mathbf{x}_{ij}^{n+1} .

Therefore, their decision is equivalent to using

$$\mathbf{x}_{ij}^* = L \frac{\mathbf{x}_{ij}^n}{|\mathbf{x}_{ij}^n|}. \quad (3.4)$$

We will refer this scheme as the *previous direction* method. Using the direction of the previous step is a flaw. The scheme produces a side effect: the simulated cloth exhibits a tendency to maintain the old (global) orientation of \mathbf{x}_{ij} ; such effect can accumulate over time. This side effect is persistent even when small time steps are used.

We propose two new methods to estimate \mathbf{x}_{ij}^* . One is called the *inertial edge rotation*, and the other is called the *inertial vertex translation*. Both of these methods do not have the above kind of flaw. When time steps are taken small enough, those methods do not experience any particular side effect to our knowledge.

The Inertial Edge Rotation The principal idea of the *inertial edge rotation* (IER) method is to predict the direction of \mathbf{x}_{ij}^{n+1} based on the angular velocity ω of \mathbf{x}_{ij} at t^n ; we simply assume that \mathbf{x}_{ij} will continue to rotate with the angular velocity for the duration h . Let the rotation matrix \mathbf{R} represent the estimated incremental orientation change that occurs during $[t^n, t^{n+1}]$. Then, the method calculates \mathbf{x}_{ij}^* according to

$$\mathbf{x}_{ij}^* = L \frac{\mathbf{R}\mathbf{x}_{ij}^n}{|\mathbf{x}_{ij}^n|}. \quad (3.5)$$

The calculation of \mathbf{R} is done according to the Rodrigues' formula. We can write $h\boldsymbol{\omega} = \theta \hat{\boldsymbol{\omega}}$ for some scalar θ , where $\hat{\boldsymbol{\omega}}$ is the unit vector along $\boldsymbol{\omega}$. Then the rotation matrix \mathbf{R} corresponding to the incremental rotation $h\boldsymbol{\omega}$ is given by $I + \sin \theta [\hat{\boldsymbol{\omega}}] + (1 - \cos \theta) [\hat{\boldsymbol{\omega}}]^2$, where $[\hat{\boldsymbol{\omega}}]$ is the skew-symmetric matrix representing the cross-product operator $\hat{\boldsymbol{\omega}}_{\times}$.

Finally, the angular velocity $\boldsymbol{\omega}$ needs to be estimated. If we denote the angle between \mathbf{x}_{ij}^{n-1} and \mathbf{x}_{ij}^n as $\angle(\mathbf{x}_{ij}^{n-1}, \mathbf{x}_{ij}^n)$, then we can approximate the angular velocity $\boldsymbol{\omega}^n$ at t^n with

$$\boldsymbol{\omega}^n = \frac{\angle(\mathbf{x}_{ij}^{n-1}, \mathbf{x}_{ij}^n)}{h} \frac{\mathbf{x}_{ij}^{n-1} \times \mathbf{x}_{ij}^n}{|\mathbf{x}_{ij}^{n-1} \times \mathbf{x}_{ij}^n|}. \quad (3.6)$$

Similarly, we can approximate the angular velocity $\boldsymbol{\omega}^{n-1}$ at t^{n-1} with

$$\boldsymbol{\omega}^{n-1} = \frac{\angle(\mathbf{x}_{ij}^{n-2}, \mathbf{x}_{ij}^{n-1})}{h} \frac{\mathbf{x}_{ij}^{n-2} \times \mathbf{x}_{ij}^{n-1}}{|\mathbf{x}_{ij}^{n-2} \times \mathbf{x}_{ij}^{n-1}|}. \quad (3.7)$$

Now, we can estimate $\boldsymbol{\omega}$, the angular velocity at t^{n+1} . If we decide to use the first-order prediction, $\boldsymbol{\omega}$ is calculated by

$$\boldsymbol{\omega} = \boldsymbol{\omega}^n. \quad (3.8)$$

If we decide to use the second-order prediction, $\boldsymbol{\omega}$ is calculated by

$$\boldsymbol{\omega} = \boldsymbol{\omega}^n + (\boldsymbol{\omega}^n - \boldsymbol{\omega}^{n-1}) = 2\boldsymbol{\omega}^n - \boldsymbol{\omega}^{n-1}, \quad (3.9)$$

or we can use more general second-order prediction which calculates ω with

$$\omega = \alpha \omega^n + \beta \omega^{n-1}, \quad (3.10)$$

where α and β are constants we adjust. General higher-order explicit prediction of ω can be done with

$$\omega = \sum_{m=0}^M \alpha_m \omega^{n-m}, \quad (3.11)$$

where $\alpha_m (m = 0, \dots, M)$ are constants we adjust.

If we substitute Equation 3.5 into Equation 3.3, and if we regard \mathbf{x}_{ij}^* as constant in differentiation, then the stretch force is now linear with respect to \mathbf{x} . Therefore, $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ in Equation 1.8 is constant. The conventional model used for representing *damping* is already linear with respect to \mathbf{v} . Therefore, $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ is also constant. Therefore, the whole system matrix A is constant.

One problem that has been overlooked so far is that treating \mathbf{x}_{ij}^* as constant in the differentiation can lead to an artifact. Experimentation with the IER method reveals that the method occasionally produces vibratory results. It is predictable that the artifact will be more noticeable when there is a large fluctuation in the value of \mathbf{x}_{ij}^* . The inertial vertex translation method we propose in the next section is more robust in that aspect.

The Inertial Vertex Translation We propose another method to estimate \mathbf{x}_{ij}^* , the *inertial vertex translation* (IVT) method. The major difference from the

IER is that, instead of assuming the *edges* are making inertial movements, IVT regards each vertex moves independently. When there is no specific information about the vertices movements, we may assume that each vertex makes an inertial movement.

The IVT method first calculates the estimated position \mathbf{x}_i^* of \mathbf{x}_i^{n+1} by assuming that it makes an inertial movement. For example, a first-order estimation can be made by

$$\mathbf{x}_i^* = \mathbf{x}_i^n + (\mathbf{x}_i^n - \mathbf{x}_i^{n-1}) = 2\mathbf{x}_i^n - \mathbf{x}_i^{n-1}, \quad (3.12)$$

but more general first-order estimation can be made with

$$\mathbf{x}_i^* = \alpha\mathbf{x}_i^n + \beta\mathbf{x}_i^{n-1} \quad (3.13)$$

where α and β are constants we adjust. A second-order estimation can be made with

$$\mathbf{x}_i^* = \mathbf{x}_i^n + (\mathbf{x}_i^n - \mathbf{x}_i^{n-1}) + \frac{1}{2}(\mathbf{x}_i^n - 2\mathbf{x}_i^{n-1} + \mathbf{x}_i^{n-2}), \quad (3.14)$$

but more general second-order estimation can be made with

$$\mathbf{x}_i^* = \gamma\mathbf{x}_i^n + \delta\mathbf{x}_i^{n-1} + \lambda\mathbf{x}_i^{n-2}, \quad (3.15)$$

where γ , δ , and λ are constants we adjust. General higher-order estimation

can be made with

$$\mathbf{x}_i^* = \sum_{m=0}^M \alpha_m \mathbf{x}_i^{n-m}, \quad (3.16)$$

where $\alpha_m (m = 0, \dots, M)$ are constants we adjust.

We would like to emphasize that the principal idea of the IVT method (compared to the IER method) does not lie in the specific scheme for estimating the new vertex position, but it lies in treating the vertices *freely moving* particles.

Finally, we can estimate \mathbf{x}_i^* with

$$\mathbf{x}_{ij}^* = L \frac{\mathbf{x}_j^* - \mathbf{x}_i^*}{|\mathbf{x}_j^* - \mathbf{x}_i^*|}. \quad (3.17)$$

When the IVT method is used, the vibratory artifact is reduced compared to the IER method. The reduction is more significant when the second order estimation is used.

3.3 Representation of Shear Deformations

We have established a linear stretch model for the edge-based systems. In representing cloth, the other two kinds of deformation should also be considered: shear and bending.

In rectangular meshes, shear deformation can be represented with the model presented in the above by making diagonal connections with additional springs. For representing the shear deformations in triangular meshes, the proposed

model is not particularly suited for. However, the triangular mesh intrinsically has a tendency to come back from shear deformations due to the mesh structure itself. This is a side effect the model originally did not intend to produce. If you need to explicitly represent shear deformation in triangular meshes, we recommend to use the technique presented in Chapter 4.

3.4 A Killer Application of This Model

We propose an effective use of the technique we have presented above. When the computation that can be allocated to clothing simulation is highly scarce, then the edge-based linear stretch model presented in this section can be adopted on an irregular triangular mesh, *without* explicitly modeling shear deformation. The only springs in the model are the linear springs at the edges. We rely on the side-effects of the model described in Chapter 3.3. to represent shear deformation;; the triangular network by its nature will create tendency to restore from shear deformation. So, by applying the linear stretch model to an irregular triangular mesh, we obtain shear behavior of the edge-based model as a by-product. The result may not be physically accurate, but can be useful in implementing real-time systems like a game or a virtual reality, when speed is the utmost issue. Even in off-line systems, it adds another scalable option. We want to make a note that the very idea of using the edge-based linear stretch model in the above way for simple but very fast representation of cloth forms a contribution.

Chapter 4

Triangle-Based Linear Stretch/Shear Model

A triangle-based system refers to interacting particles representation of cloth in which in-plane deformation is realized by the displacements of three vertices. The elementary deformable unit is a triangle. This chapter proposes the stretch and shear energy functions for the triangle formed by P_i , P_j , and P_k so that the restorative force and the force jacobians can be derived from it. Since it judges the deformation by looking at the (triangular) area rather than just looking at the edges, triangle-based system is less sensitive to irregularity of the mesh. A striking difference from the edge-based linear stretch model is that, in the triangle-based model, the shear deformation can be accounted for.

4.1 Material Space to 3D Space Mapping \mathbf{S}

Imagine a triangle composed of three particles. Let their material space coordinates be $\mathbf{u}_i = [u_i \ v_i]^T$, $\mathbf{u}_j = [u_j \ v_j]^T$, $\mathbf{u}_k = [u_k \ v_k]^T$, and let the corresponding 3D Cartesian space locations be \mathbf{x}_i , \mathbf{x}_j , \mathbf{x}_k , respectively. Let \mathbf{S} be the mapping from the 3D material space to 3D Cartesian space, such that $\mathbf{S}(u, v)$ gives the 3D position of the material point (u, v) . A simplifying assumption we adopt in this work is that, even though triangles will have different stretch and shear strains, the strain is constant within each triangle. Under the above simplifying assumption, the partial derivatives $S_u = \frac{\partial \mathbf{S}}{\partial u}$ and $S_v = \frac{\partial \mathbf{S}}{\partial v}$ can be expressed in terms of \mathbf{u}_i , \mathbf{u}_j , \mathbf{u}_k , \mathbf{x}_i , \mathbf{x}_j , \mathbf{x}_k :

$$[S_u \ S_v] = [\mathbf{x}_j - \mathbf{x}_i \ \mathbf{x}_k - \mathbf{x}_i] \begin{bmatrix} u_j - u_i & u_k - u_i \\ v_j - v_i & v_k - v_i \end{bmatrix}^{-1}. \quad (4.1)$$

A notable consequence of the above equation is that both S_u and S_v can be expressed as linear combinations of \mathbf{x}_i , \mathbf{x}_j , and \mathbf{x}_k , i.e., we can write

$$S_u = a\mathbf{x}_i + b\mathbf{x}_j + c\mathbf{x}_k \quad (4.2)$$

$$S_v = p\mathbf{x}_i + q\mathbf{x}_j + r\mathbf{x}_k \quad (4.3)$$

where a , b , c , p , q , and r are determined from the (undeformed) triangulization in the material space.

4.2 Conventional Stretch and Shear Model

In previously proposed triangle-based models, the formula which was popularly used for representing the stretch energy was [1]

$$E^{st} = \frac{1}{2}A\{k_u(|S_u| - 1)^2 + k_v(|S_v| - 1)^2\}, \quad (4.4)$$

where A , k_u , k_v are the area of the triangle (in the undeformed state), u - and v -directional stiffnesses, respectively. As for the shear energy, some groups [1] used

$$E^{sh} = \frac{1}{2}Ak_{sh}(S_u \cdot S_v)^2, \quad (4.5)$$

and other groups [6] used

$$E^{sh} = \frac{1}{2}A\{k_{\tilde{u}}(|S_{\tilde{u}}| - 1)^2 + k_{\tilde{v}}(|S_{\tilde{v}}| - 1)^2\}, \quad (4.6)$$

where \tilde{u} and \tilde{v} represent diagonal axes obtained by rotating u and v axes by +90 degrees, $k_{\tilde{u}}$ and $k_{\tilde{v}}$ represent \tilde{u} - and \tilde{v} -directional stiffnesses.

Note that none of the above formulations make the system matrix constant.

4.3 Our Stretch and Shear Model

We propose to use the energy functions

$$E^{st} = \frac{1}{2}A\{k_u|S_u - S_u^*|^2 + k_v|S_v - S_v^*|^2\} \quad (4.7)$$

$$E^{sh} = \frac{1}{2}A \left\{ k_{\tilde{u}} \left| \frac{S_u + S_v}{\sqrt{2}} - S_{\tilde{u}}^* \right|^2 + k_{\tilde{v}} \left| \frac{S_u - S_v}{\sqrt{2}} - S_{\tilde{v}}^* \right|^2 \right\}, \quad (4.8)$$

where the linear combinations given in Equations 4.2 and 4.3 are used for S_u and S_v . We have not described yet how to calculate the vector quantities S_u^* , S_v^* , $S_{\tilde{u}}^*$, and $S_{\tilde{v}}^*$ that are supplied at each time step and are regarded constant in differentiating. But one thing is clear at this moment: the formulation makes the system matrix constant! Now, an immediate question would be whether there exist vector quantities S_u^* , S_v^* , $S_{\tilde{u}}^*$, and $S_{\tilde{v}}^*$ such that use of the above energy functions will produce acceptable stretch and shear behavior of cloth. As in the edge-based systems, the answer is yes and we propose two ways to make the estimations: the inertial axes rotation and the inertial vertex translation.

The Inertial Axes Rotation In principle, S_u^* and S_v^* should represent unstretched version of S_u^{n+1} and S_v^{n+1} , respectively. Since S_u^{n+1} and S_v^{n+1} are not available, we propose to use the estimations of them. Let \tilde{S}_u^{n+1} and \tilde{S}_v^{n+1} denote the estimation of S_u^{n+1} and S_v^{n+1} . Then, one possibility we can calculate the unstretched versions of them is:

$$S_u^* = \tilde{S}_u^{n+1} / |\tilde{S}_u^{n+1}| \quad (4.9)$$

$$S_v^* = \tilde{S}_v^{n+1} / |\tilde{S}_v^{n+1}| \quad (4.10)$$

Similarly, $S_{\tilde{u}}^*$ and $S_{\tilde{v}}^*$ should in principle represent unstretched version of $S_{\tilde{u}}^{n+1}$ and $S_{\tilde{v}}^{n+1}$, respectively. If we denote the estimations of $S_{\tilde{u}}^{n+1}$ and $S_{\tilde{v}}^{n+1}$ as $\tilde{S}_{\tilde{u}}^{n+1}$

and \tilde{S}_v^{n+1} , respectively, then one possibility we can calculate S_u^* and S_v^* is:

$$S_u^* = \tilde{S}_u^{n+1} / |\tilde{S}_u^{n+1}| \quad (4.11)$$

$$S_v^* = \tilde{S}_v^{n+1} / |\tilde{S}_v^{n+1}|. \quad (4.12)$$

A simpler way of calculating S_u^* and S_v^* is to assume that they are just perpendicular to S_u^* and S_v^* . Under this assumption, we can use

$$S_u^* = \frac{S_u^* + S_v^*}{|S_u^* + S_v^*|} \quad (4.13)$$

$$S_v^* = \frac{S_u^* - S_v^*}{|S_u^* - S_v^*|}. \quad (4.14)$$

Now, we describe how we make estimations \tilde{S}_u^{n+1} , \tilde{S}_v^{n+1} , \tilde{S}_u^{n+1} , and \tilde{S}_v^{n+1} . Estimations of \tilde{S}_u^{n+1} and \tilde{S}_v^{n+1} can be done in a similar manner as \tilde{S}_u^{n+1} , \tilde{S}_v^{n+1} . So we only describe the methods to estimate \tilde{S}_u^{n+1} , \tilde{S}_v^{n+1} .

Assuming that the triangle will make an inertial movement during the short duration of $[t^n, t^{n+1}]$, we can predict S_u^{n+1} and S_v^{n+1} by rotating S_u^n and S_v^n by $h\omega$, i.e., we can calculate \tilde{S}_u^{n+1} and \tilde{S}_v^{n+1} by

$$\tilde{S}_u^{n+1} = \frac{\mathbf{R}(h\omega)S_u^n}{|S_u^n|} \quad (4.15)$$

$$\tilde{S}_v^{n+1} = \frac{\mathbf{R}(h\omega)S_v^n}{|S_v^n|}. \quad (4.16)$$

The estimation of ω can be done by the procedures introduced in Chapter 3.2.

The Inertial Vertex Translation When there is no prior information about the cloth movements, another plausible postulation we can make is that the vertices will make inertial movements. We can apply the same procedures introduced in Chapter 3.2 to obtain the predicted position \mathbf{x}_i^* of \mathbf{x}_i^{n+1} . Then, we can take the linear combinations of them according to Equations 4.2 and 4.3 to make the estimations:

$$S_u^* = a\mathbf{x}_i^* + b\mathbf{x}_j^* + c\mathbf{x}_k^* \quad (4.17)$$

$$S_v^* = p\mathbf{x}_i^* + q\mathbf{x}_j^* + r\mathbf{x}_k^* \quad (4.18)$$

Chapter 5

Linear Bending Model

As for the bending force, we simply adopt the linear bending model proposed by Volino and Thalmann [21], and in this chapter, we briefly summarize that bending model.

Volino et al. propose a bending vector that can be calculated by a simple linear combination of particle positions. This bending vector represents the bending of the surface, and when apply bending forces, they redistribute bending vector to each particles according to the bending stiffness of the surface.

5.1 Calculating Bending Vector

The bending vector is computed by a simple linear combination of particle positions that make-up two adjacent triangles $(\mathbf{x}_i, \mathbf{x}_k, \mathbf{x}_l)$ and $(\mathbf{x}_j, \mathbf{x}_l, \mathbf{x}_k)$ as shown

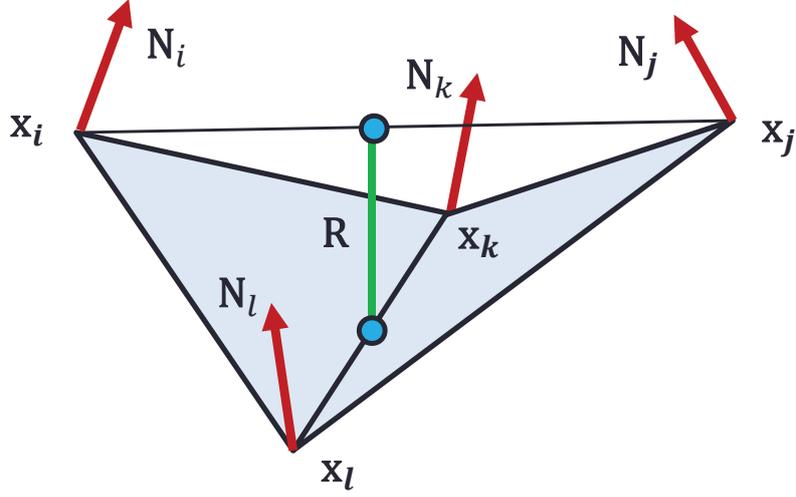


Figure 5.1 Creating bending force in a triangle mesh

in Figure 5.1. Therefore, the bending vector \mathbf{R} can be represented as follows:

$$\mathbf{R} = \alpha_i \mathbf{x}_i + \alpha_j \mathbf{x}_j + \alpha_k \mathbf{x}_k + \alpha_l \mathbf{x}_l, \quad (5.1)$$

and the coefficients α_i , α_j , α_k , and α_l are computed through the segment intersection rule:

$$\alpha_i = \frac{|\mathbf{N}_j|}{|\mathbf{N}_i| + |\mathbf{N}_j|} \quad (5.2)$$

$$\alpha_j = \frac{|\mathbf{N}_i|}{|\mathbf{N}_i| + |\mathbf{N}_j|} \quad (5.3)$$

$$\alpha_k = -\frac{|\mathbf{N}_l|}{|\mathbf{N}_k| + |\mathbf{N}_l|} \quad (5.4)$$

$$\alpha_l = -\frac{|\mathbf{N}_k|}{|\mathbf{N}_k| + |\mathbf{N}_l|}, \quad (5.5)$$

using the normal:

$$\mathbf{N}_i = (\mathbf{x}_i - \mathbf{x}_k) \wedge (\mathbf{x}_i - \mathbf{x}_l) \quad (5.6)$$

$$\mathbf{N}_j = (\mathbf{x}_j - \mathbf{x}_l) \wedge (\mathbf{x}_j - \mathbf{x}_k) \quad (5.7)$$

$$\mathbf{N}_k = (\mathbf{x}_k - \mathbf{x}_j) \wedge (\mathbf{x}_k - \mathbf{x}_i) \quad (5.8)$$

$$\mathbf{N}_l = (\mathbf{x}_l - \mathbf{x}_i) \wedge (\mathbf{x}_l - \mathbf{x}_j) \quad (5.9)$$

5.2 Applying Bending Force

The bending forces \mathbf{F}_i , \mathbf{F}_j , \mathbf{F}_k , \mathbf{F}_l are applied on the vertices, \mathbf{x}_i , \mathbf{x}_j , \mathbf{x}_k , \mathbf{x}_l respectively along the bending vector \mathbf{R} . Therefore, the bending forces can be formulated as follows:

$$\mathbf{F}_i = -\lambda \alpha_i \mathbf{R} \quad (5.10)$$

$$\mathbf{F}_j = -\lambda \alpha_j \mathbf{R} \quad (5.11)$$

$$\mathbf{F}_k = -\lambda \alpha_k \mathbf{R} \quad (5.12)$$

$$\mathbf{F}_l = -\lambda \alpha_l \mathbf{R}, \quad (5.13)$$

and a stiffness coefficient λ can be evaluated as follows:

$$\lambda = \frac{2}{3} \frac{|\mathbf{N}_i| + |\mathbf{N}_j|}{(|\mathbf{N}_i||\mathbf{N}_j|)^2} l \mu \quad (5.14)$$

where l is the length of the edge $(\mathbf{x}_k, \mathbf{x}_l)$, and μ is the bending stiffness modulus of the bent surface.

5.3 Jacobian of the Bending Force

The coefficients $\alpha_i, \alpha_j, \alpha_k, \alpha_l$ do not depend on the current position of the vertices $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k, \mathbf{x}_l$. Therefore, the Jacobian of the bending forces is constant over time steps and can be calculated as follows:

$$\frac{\partial \mathbf{F}_i}{\partial \mathbf{x}_j} = -\lambda \alpha_i \alpha_k I \quad (5.15)$$

where I denotes the identity matrix.

Chapter 6

Sparse Cholesky Factorization

The stretch/shear model we propose makes the system matrix A constant over time. Then, the matrix inversion can be pre-computed only once in off-line, so that the on-line simulation can be done fast. Unfortunately, having a constant system matrix does not imply fast simulation by itself, because a naive inversion of the system matrix results in a non-sparse matrix, in which case the complexity of the matrix-vector multiplication can amount to $O(n^2)$, which is even more costly than the PCG. A few techniques have been studied to remedy such situation; Sparse Cholesky factorization methods can maintain the sparsity during the inversion through careful reordering of the linear equations.

6.1 Cholesky Factorization

To apply Cholesky factorization, the system matrix A has to be symmetric positive definite (SPD) matrix. Prior to experiment sparse Cholesky factoriza-

tion, we verified that the system matrices of our edge-based linear system and triangle-based system is SPD.

As Equation 1.9, the system matrix A is $\left(\mathbf{I} - h\mathbf{M}^{-1}\frac{\partial\mathbf{F}}{\partial\mathbf{v}} - h^2\mathbf{M}^{-1}\frac{\partial\mathbf{F}}{\partial\mathbf{x}}\right)$. As shown in the formulation, if the force jacobian $\frac{\partial\mathbf{F}}{\partial\mathbf{x}}$ is symmetric, the system matrix is also symmetric.

Firstly, at the edge-based linear system, a stretch force of a vertex \mathbf{x}_i is

$$\mathbf{f}_i^s = k_{ij}(\mathbf{x}_{ij} - \mathbf{x}_{ij}^*). \quad (6.1)$$

Accordingly, a component of the stretch force jacobian of i -th row and j -th column is

$$\frac{\partial\mathbf{f}_i^s}{\partial\mathbf{x}_j} = k_{ij}I, \quad (6.2)$$

with I denoting an identity matrix. Oppositely, j -th row and i -th column component of the stretch force jacobian is

$$\frac{\partial\mathbf{f}_j^s}{\partial\mathbf{x}_i} = k_{ij}I. \quad (6.3)$$

As shown, (i, j) and (j, i) components of the force jacobian are equal. So, the edge-based linear stretch force jacobian and the system matrix are symmetric.

Similarly, at the triangle-based linear system, according to the linear stretch energy formulation (Equation 4.7), a stretch force of a vertex \mathbf{x}_i is

$$\mathbf{f}_i^{st} = -A\{k_u a(S_u - S_u^*) + k_v p(S_v - S_v^*)\}. \quad (6.4)$$

Accordingly, a component of the stretch force jacobian of i -th row and j -th column is

$$\frac{\partial \mathbf{f}_i^{st}}{\partial \mathbf{x}_j} = -A(k_u ab + k_v pq)I. \quad (6.5)$$

Oppositely, j -th row and i -th column component of the stretch force jacobian is

$$\frac{\partial \mathbf{f}_j^{st}}{\partial \mathbf{x}_i} = -A(k_u ba + k_v qp)I. \quad (6.6)$$

Alikely, (j,k) and (k,j) components of the stretch force jacobian are

$$\frac{\partial \mathbf{f}_j^{st}}{\partial \mathbf{x}_k} = -A(k_u bc + k_v qr)I \quad (6.7)$$

$$\frac{\partial \mathbf{f}_k^{st}}{\partial \mathbf{x}_j} = -A(k_u cb + k_v rq)I, \quad (6.8)$$

and (k,i) , (i,k) components of the force stretch jacobian are

$$\frac{\partial \mathbf{f}_k^{st}}{\partial \mathbf{x}_i} = -A(k_u ca + k_v rp)I \quad (6.9)$$

$$\frac{\partial \mathbf{f}_i^{st}}{\partial \mathbf{x}_k} = -A(k_u ac + k_v pr)I. \quad (6.10)$$

As shown, (i,j) and (j,i) , (j,k) and (k,j) , (k,i) and (i,k) components of the stretch force jacobian are equal mutually. Next to a stretch force, according to the linear shear energy formulation (Equation 4.8), a shear force of a vertex

\mathbf{x}_i is

$$\mathbf{f}_i^{sh} = -A \left\{ k_{\bar{u}} \frac{a+p}{\sqrt{2}} \left(\frac{S_u + S_v}{\sqrt{2}} - S_{\bar{u}}^* \right) + k_{\bar{v}} \frac{a-p}{\sqrt{2}} \left(\frac{S_u - S_v}{\sqrt{2}} - S_{\bar{v}}^* \right) \right\}. \quad (6.11)$$

Accordingly, a component of the shear force jacobian of i -th row and j -th column is

$$\frac{\partial \mathbf{f}_i^{sh}}{\partial \mathbf{x}_j} = -\frac{A}{2} \{ k_{\bar{u}}(a+p)(b+q) + k_{\bar{v}}(a-p)(b-q) \} I. \quad (6.12)$$

Oppositely, j -th row and i -th column component of the shear force jacobian is

$$\frac{\partial \mathbf{f}_j^{sh}}{\partial \mathbf{x}_i} = -\frac{A}{2} \{ k_{\bar{u}}(b+q)(a+p) + k_{\bar{v}}(b-q)(a-p) \} I. \quad (6.13)$$

Alikely, (j,k) and (k,j) components of the force jacobian are

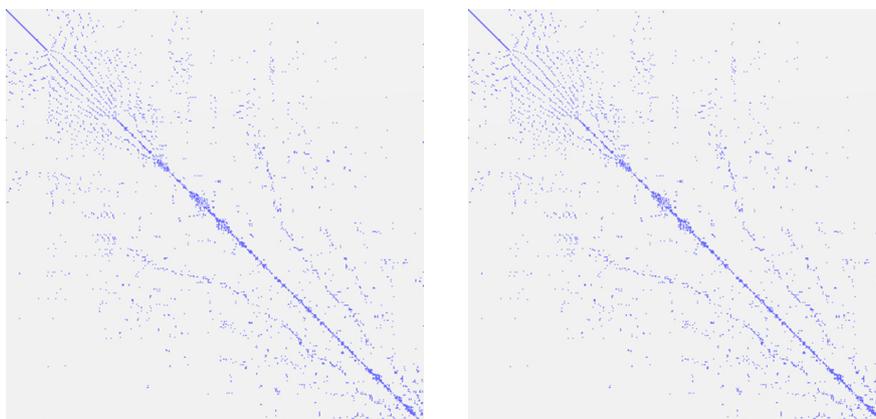
$$\frac{\partial \mathbf{f}_j^{sh}}{\partial \mathbf{x}_k} = -\frac{A}{2} \{ k_{\bar{u}}(b+q)(c+r) + k_{\bar{v}}(b-q)(c-r) \} I, \quad (6.14)$$

$$\frac{\partial \mathbf{f}_k^{sh}}{\partial \mathbf{x}_j} = -\frac{A}{2} \{ k_{\bar{u}}(c+r)(b+q) + k_{\bar{v}}(c-r)(b-q) \} I. \quad (6.15)$$

and (k,i) , (i,k) components of the force jacobian are

$$\frac{\partial \mathbf{f}_k^{sh}}{\partial \mathbf{x}_i} = -\frac{A}{2} \{ k_{\bar{u}}(c+r)(a+p) + k_{\bar{v}}(c-r)(a-p) \} I, \quad (6.16)$$

$$\frac{\partial \mathbf{f}_i^{sh}}{\partial \mathbf{x}_k} = -\frac{A}{2} \{ k_{\bar{u}}(a+p)(c+r) + k_{\bar{v}}(a-p)(c-r) \} I. \quad (6.17)$$



(a) An edge-based model

(b) A triangle-based model

Figure 6.1 System matrix diagram of linear systems

As shown, (i, j) and (j, i) , (j, k) and (k, j) , (k, i) and (i, k) components of the shear force jacobian are equal mutually, So, the triangle-based linear stretch and shear force jacobians are symmetric. Therefore, the system matrix of the triangle-based linear system is symmetric.

As a result, all our linear models produce symmetric system matrix. Figure 6.1 shows a system matrix diagram of an edge-based and a triangle-based system. In this figure we can visually verify that system matrices are symmetric in all linear systems.

From now, if the system matrix A is positive definite, we can adopt Cholesky factorization in system solving process. To check whether A is positive definite or not, we evaluated an eigenvalue of the system matrix in each case of linear models. Table 6.1 shows the result.

According to the result, in the edge-based and the triangle-based system, all eigenvalues of the system matrix have positive values. As a result, we can

	Edge-based system	Triangle-based system
#vertices	166	166
Min eigenvalue	0.9817	1.0
Max eigenvalue	2.2693	2.8761

Table 6.1 Minimum and maximum eigenvalues of the system matrix

say that system matrices of proposed linear models are symmetric and positive definite.

Assume A is symmetric positive definite (SPD) and A has Cholesky factorization $A = LL^T$ where L is lower triangular matrix with positive diagonal entries. Linear system $Ax = b$ can then be solved by forward-substitution in lower triangular system $Ly = b$, followed by backward-substitution in upper triangular system $L^T x = y$.

SPD matrix A can be factored as follows in the process of Cholesky factorization

$$A = \begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{1}{\sqrt{a_{11}}}A_{21} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & A_1 \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & \frac{1}{\sqrt{a_{11}}}A_{21}^T \\ 0 & I \end{bmatrix}, \quad (6.18)$$

where $A_1 = A_{22} - \frac{1}{a_{11}}A_{21}A_{21}^T$ is the *Schur complement*. If A_1 is further factorized as $A_1 = L_1L_1^T$ then

$$A = \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{1}{\sqrt{a_{11}}}A_{21} & L_1 \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & \frac{1}{\sqrt{a_{11}}}A_{21}^T \\ 0 & L_1^T \end{bmatrix} \quad (6.19)$$

The Schur complements may get more and more dense. We hope that the final

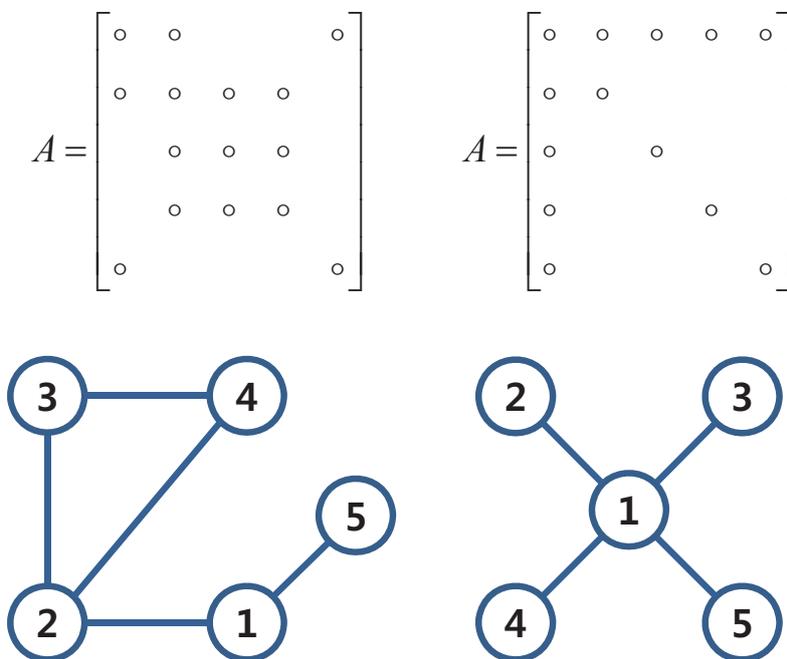


Figure 6.2 Examples of matrices and those adjacency graphs

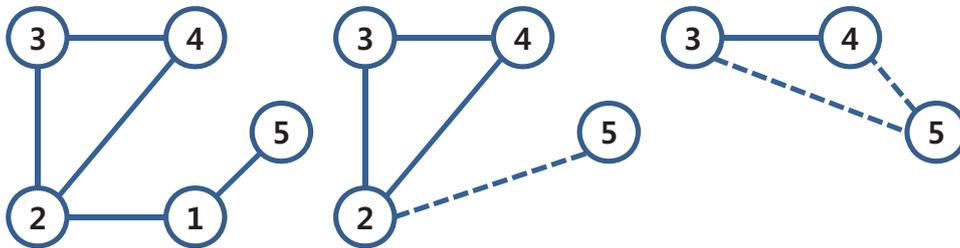
L remains the sparsity. But L is usually more dense.

Graph theory can be used in analysis of matrices. A graph $G = (V, E)$ is defined by a vector of vertices $V = \{v_1, \dots, v_n\}$ and a vector of edges $E = \{(v_i, v_j)\} = V \times V$. $G = (V, E)$ of an $n \times n$ sparse matrix A is an adjacency graph having n vertices, with edge between vertices i and j if $a_{ij} \neq 0$ (Figure 6.2).

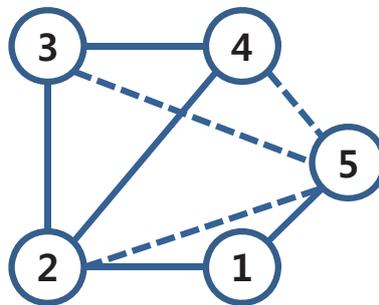
Vertices that are not directly connected can get connected via matrix operations. This causes fill-in. At each step of Cholesky factorization, the eliminations correspond to the removal of vertices from the adjacency graphs of A, A_1, A_2, \dots , and edges are added during the eliminations. They are fill edges and correspond to zero entries in A but non-zeroes in L . Vertices get connected if they were indirectly connected by a path. The removal of a vertex connects

$$A = \begin{bmatrix} \circ & \circ & & & \circ \\ \circ & \circ & \circ & \circ & \\ & \circ & \circ & \circ & \\ & \circ & \circ & \circ & \\ \circ & & & & \circ \end{bmatrix} \quad L = \begin{bmatrix} \circ & & & & \\ \circ & \circ & & & \\ & \circ & \circ & & \\ & \circ & \circ & \circ & \\ \circ & \bullet & \bullet & \bullet & \circ \end{bmatrix}$$

(a) Matrix A and its factored matrix L



(b) Adjacency graphs of A, A_1, A_2



(c) Filled graph of matrix A

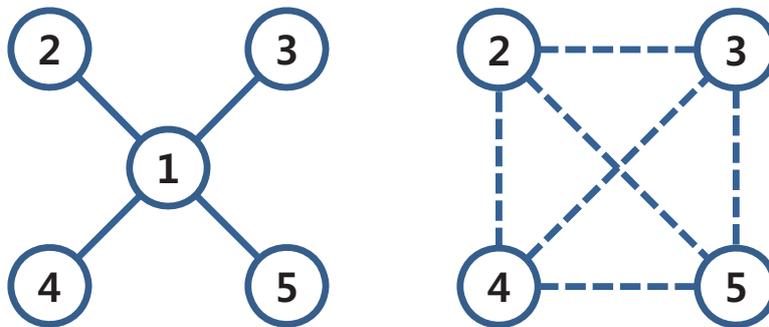
Figure 6.3 An example of the graph model of an elimination

all higher numbered vertices that were previously connected to it.

For matrix in left side of Figure 6.2, Figure 6.3 shows the graph model of an elimination. As shown in Figure 6.3, L is not as sparse as A . Black dots in matrix L (right side of Figure 6.3(a)) are due to the fill edges described as dotted lines in Figure 6.3(c). Figure 6.4 shows an adjacency graph for the Schur

$$A = \begin{bmatrix} \circ & \circ & \circ & \circ & \circ \\ \circ & \circ & & & \\ \circ & & \circ & & \\ \circ & & & \circ & \\ \circ & & & & \circ \end{bmatrix} \quad L = \begin{bmatrix} \circ & & & & \\ \circ & \circ & & & \\ \circ & \bullet & \circ & & \\ \circ & \bullet & \bullet & \circ & \\ \circ & \bullet & \bullet & \bullet & \circ \end{bmatrix}$$

(a) Matrix A and its factored matrix L



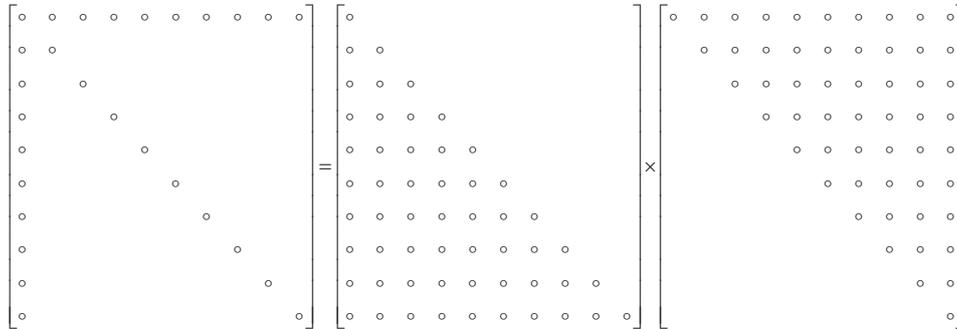
(b) Adjacency graphs for A and A_1

Figure 6.4 An extreme case of the graph model of an elimination

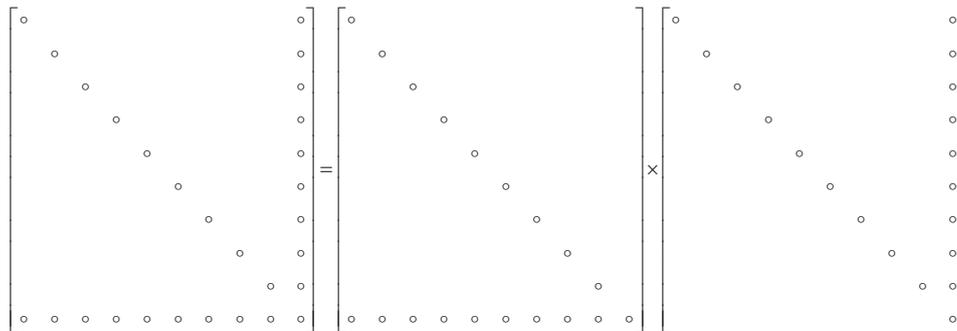
complement after 1 step of elimination of matrix in right side of Figure 6.2. In an extreme case like Figure 6.4 (an arrow matrix), the first elimination step will create a fully dense A_1 .

6.2 Reordering

Amount of fill depends on order in which vertices are eliminated. The Cholesky factorization of SPD matrices is numerically stable, and symmetrically permut-



(a) Factorization with 100% fill-in

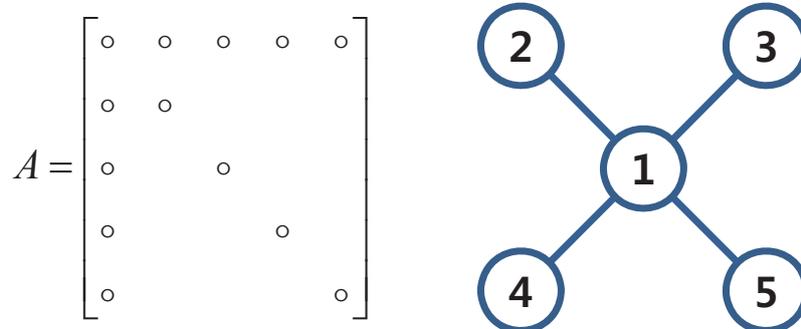


(b) Factorization with zero fill-in

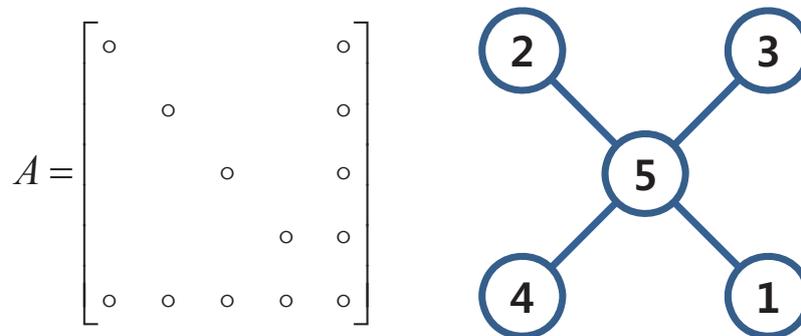
Figure 6.5 Effect of ordering

ing the rows and columns of an SPD matrix yields another SPD matrix. Therefore, we can try to symmetrically permute the rows and columns of a sparse matrix to reduce fill and work in the factorization. We do not need to worry that the permutation will numerically destabilize the factorization. In terms of the graph theory, a symmetric row and column permutation corresponds to re-labeling the vertices of the graphs. In other words, given a graph we seek an elimination ordering for its vertices.

As an example, the arrow matrix in Figure 6.4 can be reordered to a new arrow matrix. The factorization of this matrix has no fill-in (Figure 6.6).



(a) The original arrow matrix A and its adjacency graph



(b) A new arrow matrix A and its adjacency graph

Figure 6.6 The original and the ordered arrow matrix A

General problem of finding ordering that minimizes fill is NP-complete, but there are relatively cheap heuristics that limit fill effectively.

Minimum Degree If the elimination of a yet-uneliminated vertex creates a fill whose amount is the number of the uneliminated neighbors of the chosen vertex, it makes sense to eliminate the vertex with the fewer uneliminated neighbors. Choosing this vertex minimizes the amount of fill that is created in the next step and minimizes the arithmetic work in the next step. Ordering

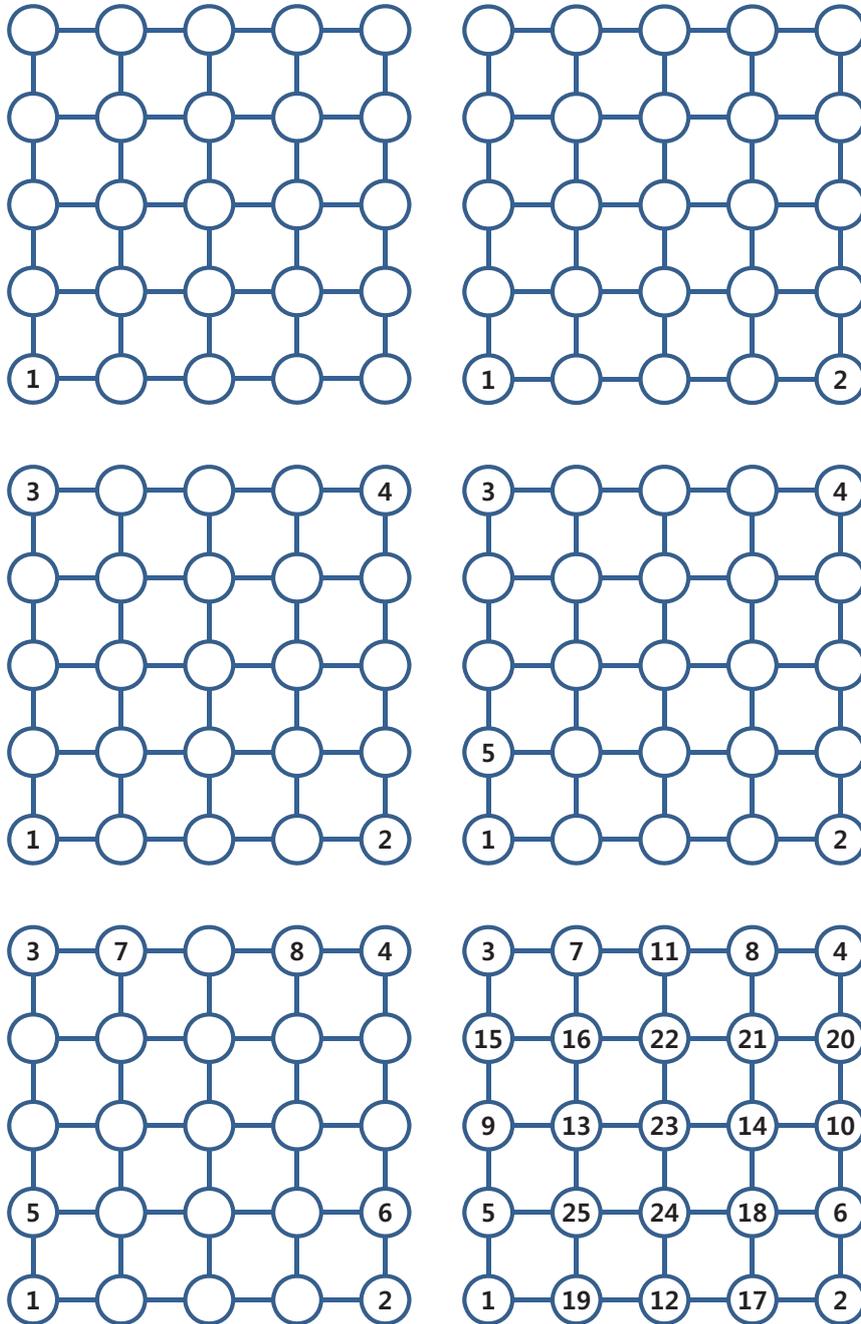


Figure 6.7 Examples of the minimum degree ordering

heuristics based on this idea are called minimum-degree heuristics. That is, for the matrix: at each step of the eliminations, rows and columns are permuted so that the pivot row (column) to be eliminated is the one with the least nonzero off-diagonal entries (Figure 6.7).

There are two problems in the minimum-degree idea. First, not all the edges in the new fills are new; some of them might be part of adjacency graph of A or might have already been created by a previous elimination step. It is possible to minimize not the degree but the actual new fill, but this turns out to be more expensive algorithmically. Minimizing fill in this way also turns out to produce orderings that are not significantly better than minimum-degree orderings. Second, an optimal choice for the next vertex to eliminate may be suboptimal globally. There are families of matrices on which minimum-degree orderings generate asymptotically more fill than the optimal orderings. Minimum-degree heuristics is greedy and myopic. It selects vertices for elimination one at a time, and the lack of global planning can be local minimization on the fill only.

Even though minimum-degree algorithms are theoretically known to be suboptimal, they are easy to implement, very fast and often produce effective ordering results. On huge matrices nested-dissection orderings, which we discuss next, are often more effective than minimum-degree orderings, but on smaller matrices, minimum-degree orderings are sometimes more effective in reducing fill and work.

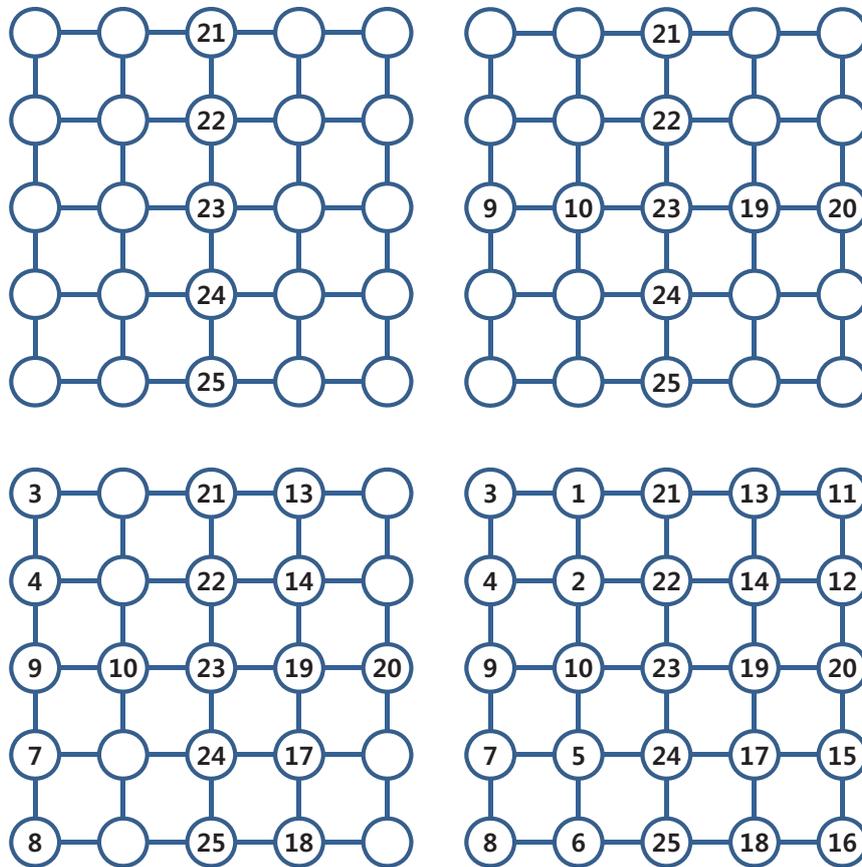


Figure 6.8 Examples of the nested dissection ordering

Nested Dissection Nested dissection orderings are known to be approximately optimal, and on huge matrices that can be significantly more effective than other ordering heuristics. Nested dissection orderings are defined recursively using vertex subsets called separators. Given a vertex separator, we order the rows and columns of matrix of the separator last, and for each subset, apply the same idea recursively (Figure 6.8). The size of the separator determines the circumference of the rectangular block of the subset. The size imbalance

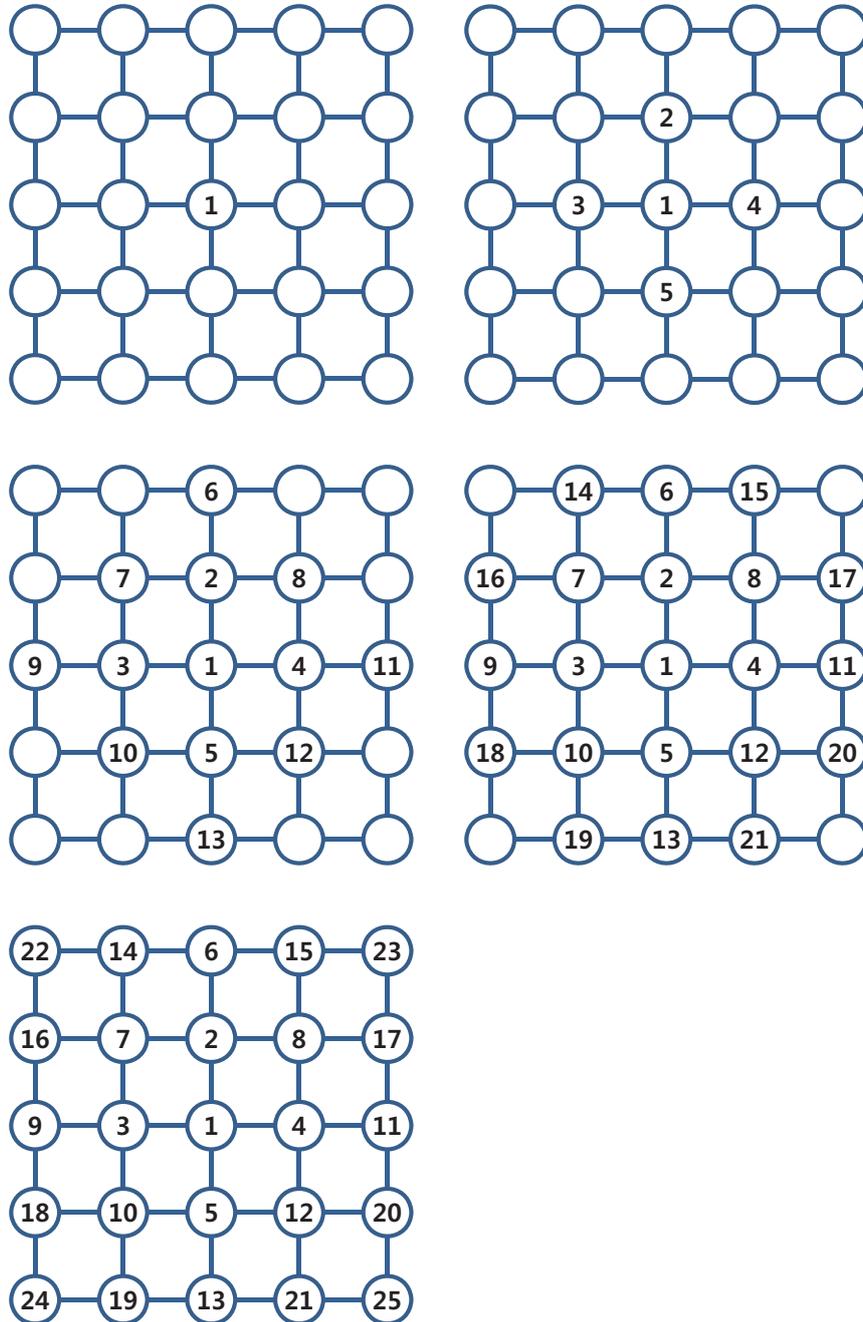


Figure 6.9 Examples of Cuthill-McKee & reverse Cuthill-McKee ordering

between subsets determines the aspect ratio of this rectangle. For a given circumference, the area is maximized the rectangle is as close to square as possible. Therefore, a small balanced separator reduces fill more effectively than a large or unbalanced separator.

Cuthill-McKee & Reverse Cuthill-McKee They are level-set orderings based on traversing the matrix by level sets. A level set is defined recursively as the set of all unlabeled neighbors of all the vertices of a previous level set. One traversal example: for each vertex u in the current level set, its neighbors are inspected. If a neighbor v is not numbered, add v to the next level set (Figure 6.9). This is called breadth first search, and this reduces distance of nonzero diagonals from main diagonal. In a current level set, the nodes can be traversed according to their degrees.

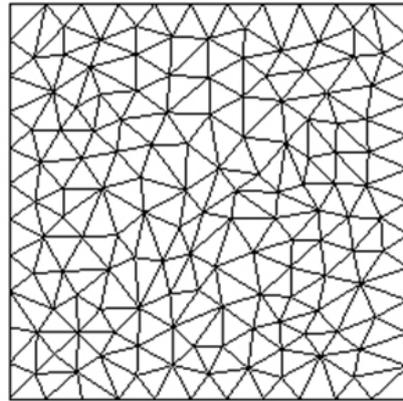
Chapter 7

Experimental Results

We experimented our method with various cloth and clothing examples. All simulations were performed on a single desktop computer with Intel Core i7(3770K) 3.50GHz CPU and 16GB RAM.

Sparse Cholesky Factorization To evaluate the performance of the sparse Cholesky factorization, we measured sparsity of the lower triangular matrix L before reordering and after reordering with various resolutions and shapes. We used Cuthill-McKee method when reordering.

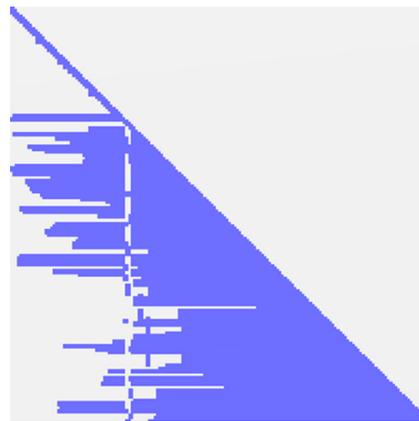
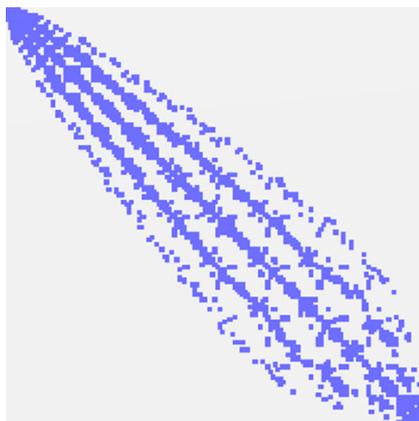
Firstly, Figures 7.1, 7.2, and 7.3 show the change of sparsity according to a variation of a mesh resolution. According to figures, at high resolution mesh, non-zero terms of matrix gather around diagonal terms. Therefore, the higher resolution of cloth mesh, the more gain of sparsity could be earned. Table 7.1 shows this result more precisely with numbers. At the table, bottom row shows the reduction rate of number of non-zeroes between the original



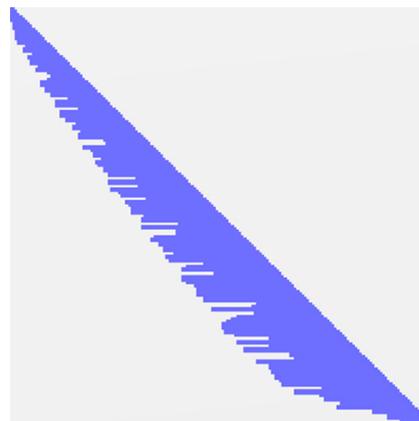
(a) Number of vertices: 166

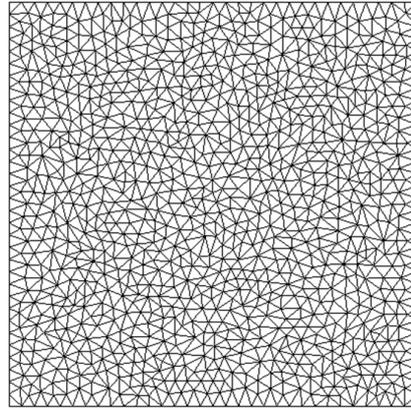


(b) System matrix before reordering

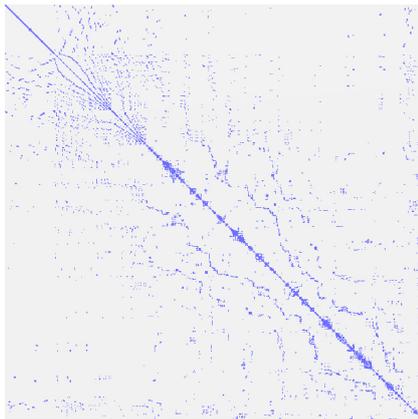
(c) Lower triangular matrix L 

(d) System matrix after reordering

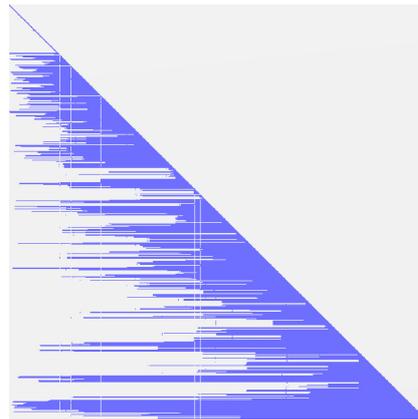
(e) Lower triangular matrix L **Figure 7.1** An effect of reordering in low resolution



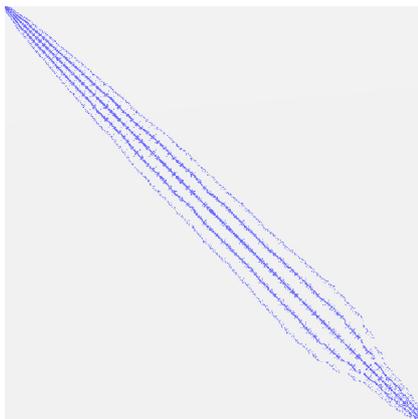
(a) Number of vertices: 1098



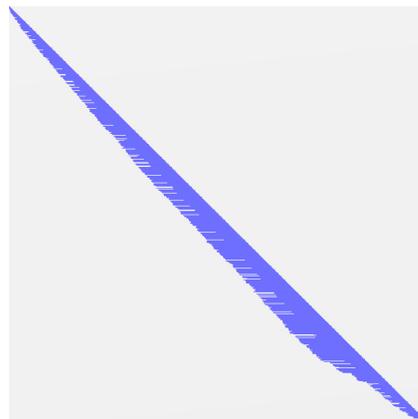
(b) System matrix before reordering



(c) Lower triangular matrix L

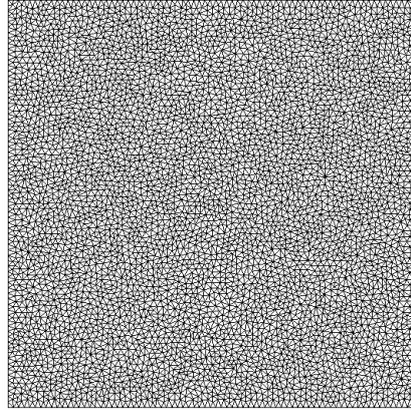


(d) System matrix after reordering

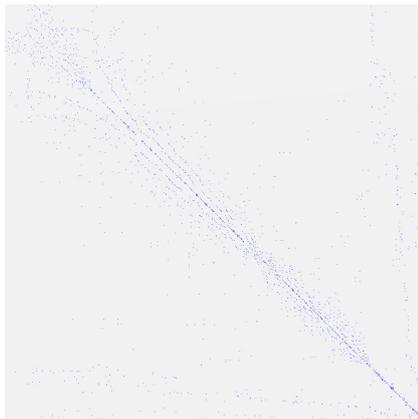


(e) Lower triangular matrix L

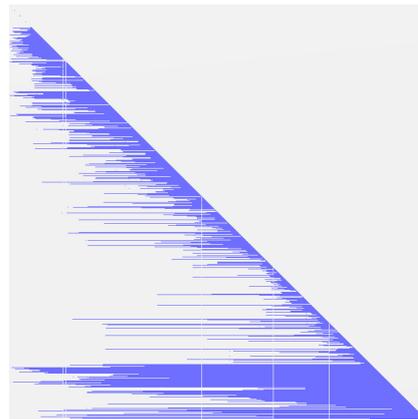
Figure 7.2 An effect of reordering in medium resolution



(a) Number of vertices: 5498



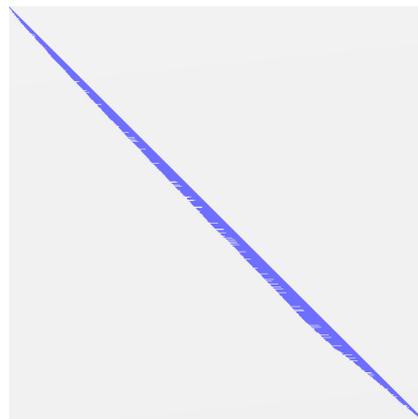
(b) System matrix before reordering



(c) Lower triangular matrix L



(d) System matrix after reordering



(e) Lower triangular matrix L

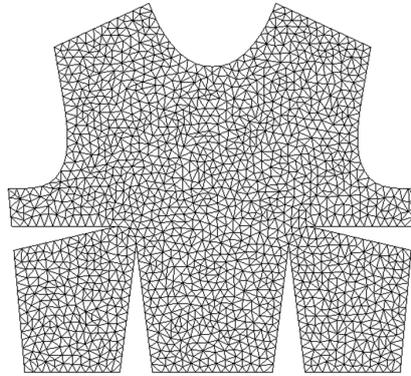
Figure 7.3 An effect of reordering in high resolution

#vertices	166	1098	5498
#non-zeroes of system matrix	1872	13426	69406
#non-zeroes of matrix L before reordering	7352	223734	3738863
#non-zeroes of matrix L after reordering	3952	68614	749633
Reduction rate of #non-zeroes	53.75%	30.67%	20.05%

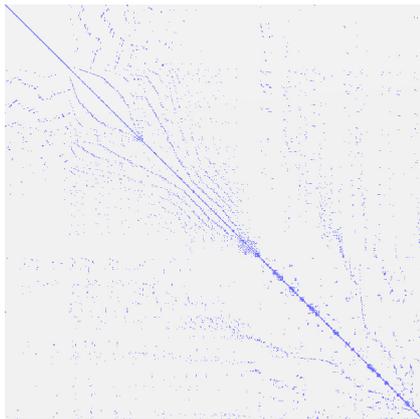
Table 7.1 This table summarizes the statics in reordering at different resolutions of Figure 7.1, 7.2, and 7.3

matrix and the reordered matrix. This rate is calculated by dividing the number of non-zeroes in the original lower triangular matrix L by the number of non-zeroes in the reordered lower triangular matrix L . Therefore, the smaller rate is good in simulation speed, and at high resolution the rate is smaller. As a result, we can expect more speed gain at large or high resolution meshes when implementing simulation.

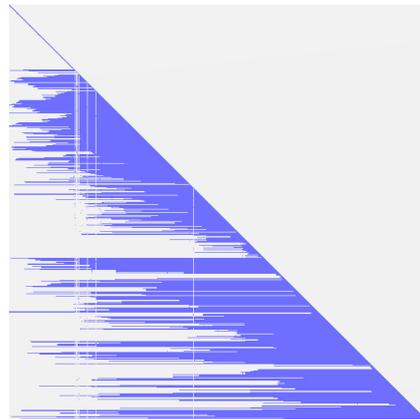
Secondly, Figures 7.4, 7.5, and 7.6 show a change of sparsity according to a variation of a mesh shape. Those shapes are actual cloth patterns that used in real clothes. Figures 7.4(a) and 7.5(a) are upper parts and Figure 7.6(a) is a lower part of a women's one-piece dress. According to these figures, we can see that the reordered lower triangular matrix L (Figure 7.4(e), 7.5(e), and 7.6(e)) maintained sparsity well at various shapes of meshes. Especially in Figure 7.5, although the unordered lower triangular matrix L (Figure 7.5(c)) is a lot dense, the ordered L (Figure 7.5(e)) shows almost same sparsity with the other shape of meshes. As a result, the mesh of Figure 7.5 shows better reduction rate of number of non-zeroes, and this result can be verified by Table 7.2.



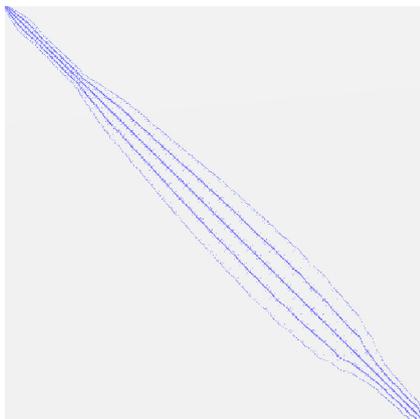
(a) Number of vertices: 1780



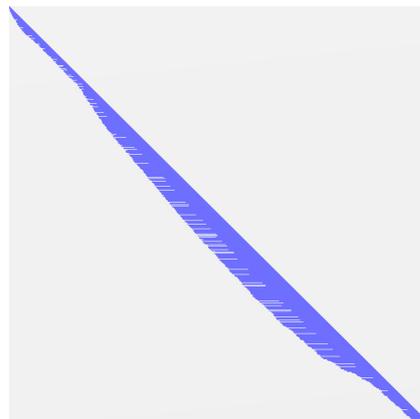
(b) System matrix before reordering



(c) Lower triangular matrix L

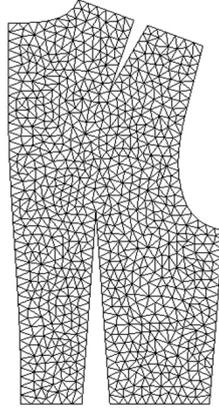


(d) System matrix after reordering

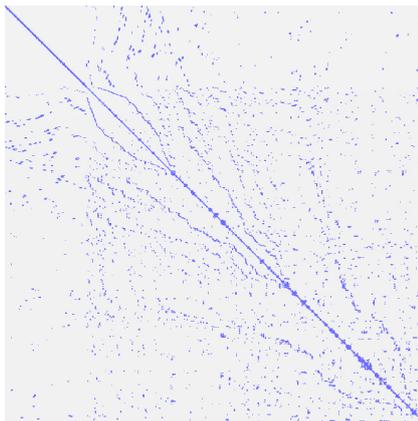


(e) Lower triangular matrix L

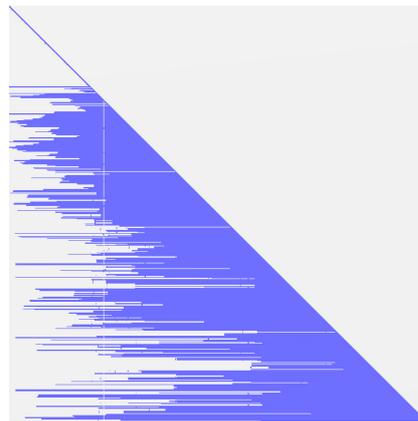
Figure 7.4 An effect of reordering in different meshes



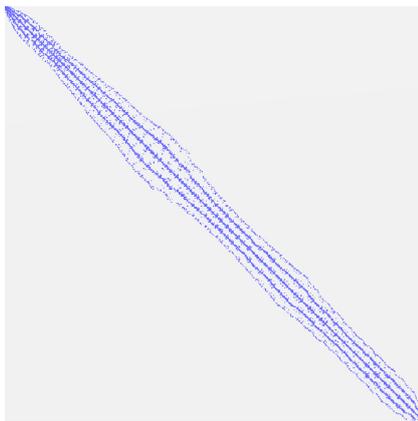
(a) Number of vertices: 918



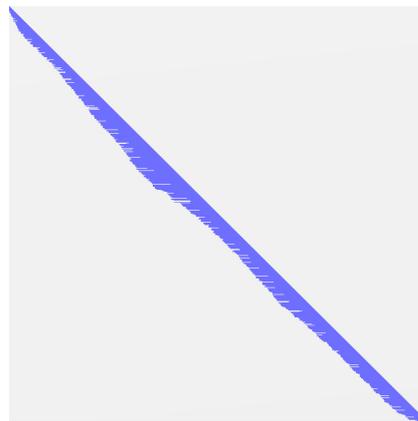
(b) System matrix before reordering



(c) Lower triangular matrix L

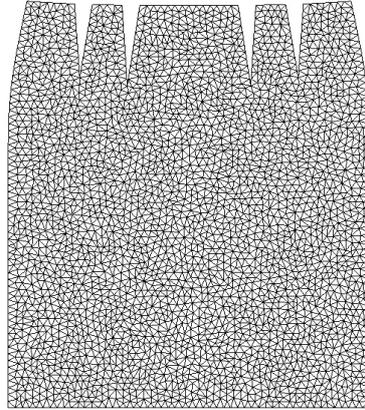


(d) System matrix after reordering

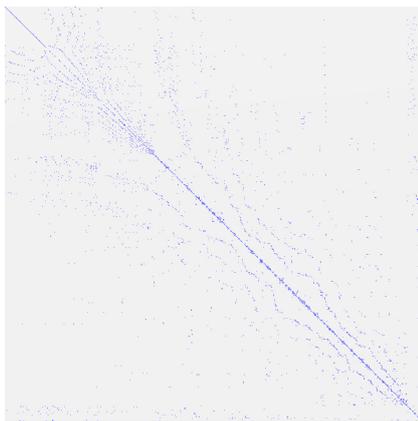


(e) Lower triangular matrix L

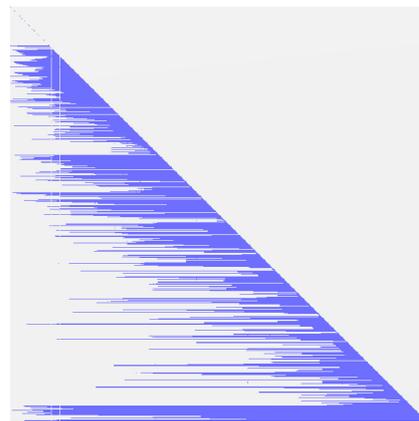
Figure 7.5 An effect of reordering in different meshes



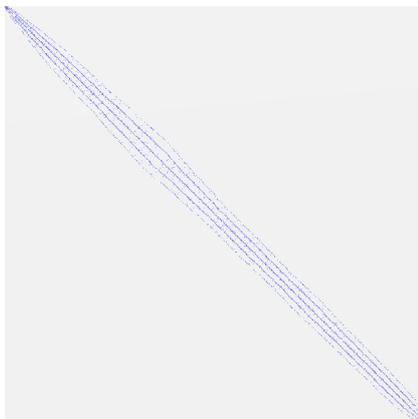
(a) Number of vertices: 3176



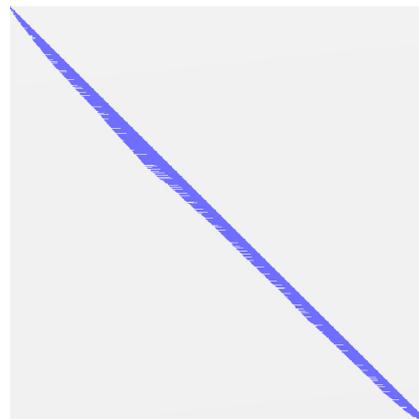
(b) System matrix before reordering



(c) Lower triangular matrix L



(d) System matrix after reordering



(e) Lower triangular matrix L

Figure 7.6 An effect of reordering in different meshes

Mesh	of Figure 7.4	of Figure 7.5	of Figure 7.6
#vertices	1780	918	3176
#non-zeroes of system matrix	21412	10816	39352
#non-zeroes of matrix L before reordering	626214	204179	1478721
#non-zeroes of matrix L after reordering	182132	38722	337654
Reduction rate of #non-zeroes	29.08%	18.96%	22.83%

Table 7.2 This table summarizes the statics in reordering at different shapes of Figure 7.4, 7.5, and 7.6

Performance To evaluate the speed-up, we measured the average computation time of calculating one time step for both the conventional and proposed linear methods for various experiments. We applied PCG for solving system in the conventional method.

We measured the computation time when the resolution of mesh is varying with the simple swinging cloth test (Figure 7.7). In this test, Vertices of top corners of the square cloth are constrained, and other part of cloth is freely draped. At each case, the size of cloth is same, and number of vertices and triangles are varying.

Table 7.3 summarizes the statics collected during swinging cloth test. It shows that the proposed linear models are about 2-5 times faster than the conventional model. As expected before, the higher mesh resolution, the more speed up is earned because a gain from preserving sparsity in sparse Cholesky factorization (SCF) is bigger when the resolution of mesh is higher. In the case

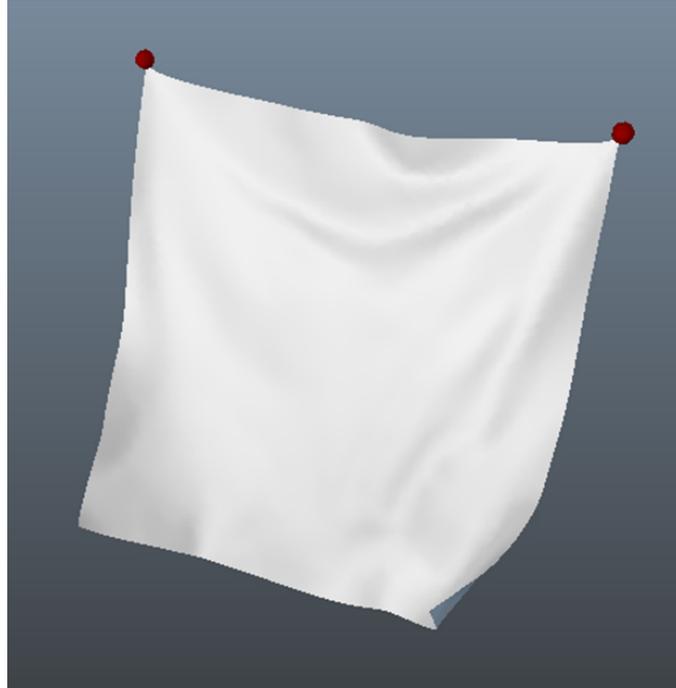


Figure 7.7 The swinging cloth test

of the number of third column of Table 7.3, the speed up in edge-based IER models is above 5 times. If more large and high resolution mesh is simulated, we can expect that the amount of the speed up would be higher. In addition, the speed up in an edge-based system is slightly larger than a triangle-based system, and the speed up in IVT method is larger than IER and IAR method. It is because a difference of formulation and the number of applied forces: an edge-based system applies stretch and bending forces, but a triangle-based system applies stretch, shear and bending forces.

To analysis the cause of the speed up more precisely, we partitioned the one time step into some blocks with respect to its function. First block is a process of calculating force, second block is about building a system matrix,

#vertices	1245		2485		4219	
#triangles	2348		4768		8176	
#non-zeroes of system matrix	15293		30969		53067	
#non-zeroes of matrix L before reordering	286430		829610		2200249	
#non-zeroes of matrix L after reordering	84145		236537		505102	
Average time (ms) prev. edge-based model	5.75		14.74		29.81	
Average time (ms)/speed up linear edge-based IER	2.17	×2.65	4.97	×2.97	9.08	×3.28
Average time (ms)/speed up linear edge-based IVT	1.11	×5.18	2.72	×5.42	5.39	×5.53
Average time (ms) prev. triangle-based model	7.18		16.79		33.68	
Average time (ms)/speed up linear triangle-based IAR	2.73	×2.63	6.25	×2.69	11.39	×2.96
Average time (ms)/speed up linear triangle-based IVT	1.40	×5.13	3.18	×5.28	6.26	×5.38

Table 7.3 This table summarizes the statistics in simulating swinging cloth at three different mesh resolutions: from top to bottom, the number of vertices, the number of triangles, the number of non-zero terms in system matrix, the number of non-zero terms in original lower triangular matrix, the number of non-zero terms in reordered lower triangular matrix, the average computation time per each time step at different models and speed gain with respect to conventional model.

third block is a process of solving system, and last part is a collision handling process. At each block, we measured the average computation time for both the conventional and the proposed linear methods with a same cloth mesh in swinging cloth test (Figure 7.7).

#vertices: 5472 (ms)	Prev. edge- based model	Linear edge- based IER	Linear edge- based IVT
Calculating force	0.983	5.722	0.887
Bulding system matrix	23.249	0	0
Solving system	25.346	6.863	6.905
Collision process	0.001	0.001	0.001
Total	49.579	12.586	7.793
Speed up	×1.0	×3.939	×6.362

Table 7.4 This table present a computation time of one time step of swinging cloth simulation in detail: from top to bottom, the time for calculating forces, the time for building system matrix, the time for solving system, the time for collision handling, the total time for integrating one time step and speed gain with respect to previous non-linear model. From left to right, each time is measured in the conventional edge-based model, the linear edge-based IER model and the linear edge-based IVT model. The time unit is millisecond, and all simulations are executed with same rectangular mesh of 5472 vertices.

Tables 7.4 and 7.5 show the results of measurement. According to the results, when we simulate the swinging cloth mesh with 5472 vertices, proposed linear models are about 3-6 times faster than the conventional model.

The main cause of speed up is that the proposed linear model doesn't need to build the system matrix at every time step, because the system matrix is constant over time. In the proposed linear model, the system matrix can be pre-computed only once in off-line. In addition, the conventional non-linear model also has to solve a large system at every time step, but the proposed linear model can calculate the inverse of the system matrix in pre-computation time, and at each time step, system solving can be replaced with simple matrix-

#vertices: 5472 (ms)	Prev. triangle- based model	Linear triangle- based IAR	Linear triangle- based IVT
Calculating force	1.71	8.928	1.944
Bulding system matrix	28.028	0	0
Solving system	22.157	6.169	6.225
Collision process	0.001	0.001	0.001
Total	51.896	15.098	8.17
Speed up	×1.0	×3.437	×6.352

Table 7.5 This table present a computation time of one time step of swinging cloth simulation in detail: from top to bottom, the time for calculating forces, the time for building system matrix, the time for solving system, the time for collision handling, the total time for integrating one time step and speed gain with respect to previous non-linear model. From left to right, each time is measured the in conventional triangle-based model, the linear triangle-based IAR model and the linear triangle-based IVT model. The time unit is millisecond, and all simulations are executed with same rectangular mesh of 5472 vertices.

vector operations. This is another main cause of speed up. In an edge-based system, system matrix building time decreases from 23.249ms to 0ms, and time of solving system decreases from 25.346ms to 6.863ms with the IER method, and to 6.905ms with the IVT method, respectively. As shown in Table 7.5, a triangle-based model shows a similar tendency. Time of building system matrix decreases from 28.028ms to 0ms, and system solving time decreases from 22.157ms to 6.169ms with the IAR method, and to 6.225ms with the IVT method, respectively.

On the other hand, when forces are calculated, linear models spent more

time, as compared with the conventional non-linear model. The reason is that linear models need to calculate the term, \mathbf{x}_{ij}^* in an edge-based system, and S_u^* , S_v^* in a triangle-based system, additionally. According to Tables 7.4 and 7.5, force calculating time increases from 0.983ms to 5.722ms in the linear edge-based IER model, and from 1.71ms to 8.928ms in the linear triangle-based IAR model, respectively. In the IVT model of both an edge-based and a triangle-based model, the force calculation takes similar time with conventional models. It is because that when the IVT method is used, the process of calculating \mathbf{x}_{ij}^* , S_u^* , and S_v^* is relatively simple.

Although, the force calculation time increases in a linear model, the decrements in the process of building system matrix and solving system are very larger than the increments, so, overall simulation speed of a linear models are higher than a non-linear model. Additionally, the collision process takes almost same time at every method.

The speed up earned through the system matrix reordering process is also noticeable. Table 7.6 and 7.7 show the effect of system matrix reordering at different resolutions of mesh in swinging cloth test. When the system matrix is reordered and the factored triangular matrix preserves sparsity of original system matrix, computation time of a forward and backward substitution, in other words, solving system is about 2-4 times faster than the unordered case. As expected before, the higher mesh resolution, the more speed up is earned because a gain from preserving sparsity in sparse Cholesky factorization is bigger when the resolution of mesh is higher.

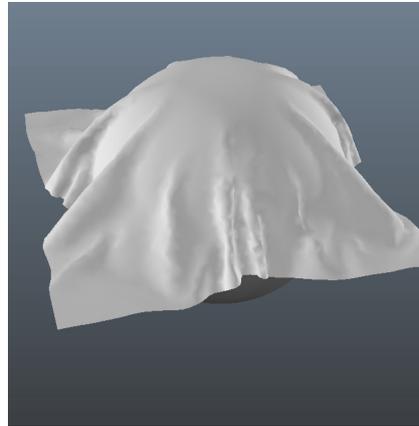
#vertices	Unordered system matrix	Reordered system matrix	Speed up
1245	2.36	0.92	$\times 2.57$
2485	6.51	2.21	$\times 2.95$
4219	16.84	4.65	$\times 3.62$

Table 7.6 This table presents the computation time of forward/backward substitution of Cholesky factorization with or without reordering in an edge-based linear system at different resolutions.

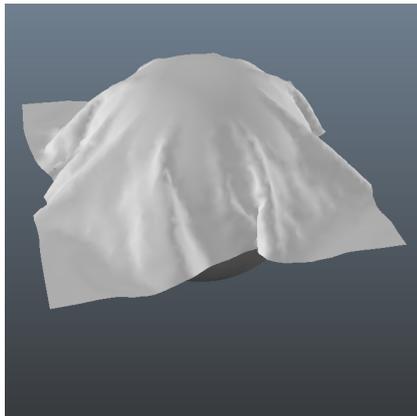
#vertices	Unordered system matrix	Reordered system matrix	Speed up
1245	2.85	1.26	$\times 2.26$
2485	6.63	2.25	$\times 2.94$
4219	16.83	4.68	$\times 3.60$

Table 7.7 This table presents the computation time of forward/backward substitution of Cholesky factorization with or without reordering in a triangle-based linear system at different resolutions.

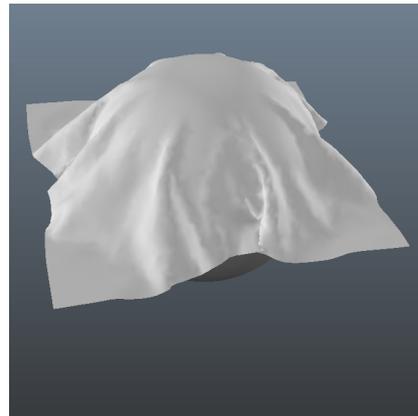
Simulation Quality Figure 7.8 shows side-by-side comparison between the conventional non-linear and our linear edge-based models. It is the sphere test, where a piece of cloth is draped on a sphere and slips down. The top of images (Figures 7.8(a)) is solved by PCG with a non-linear force model, and results in the bottom row of images (Figures 7.8(b) and 7.8(c)) are solved by SCF with a linear force model. Figure 7.8(b) applies the edge-based IER model, and Figure 7.8(c) applies the edge-based IVT model. Figure 7.9 also shows side-by-side comparison between the conventional non-linear and the our linear model, but in this case, simulations apply a triangle-based model. Similarly, the top of images is solved by PCG with a non-linear force model,



(a) The non-linear method



(b) The linear IER method

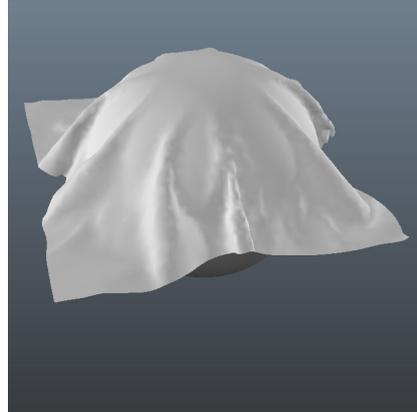


(c) The linear IVT method

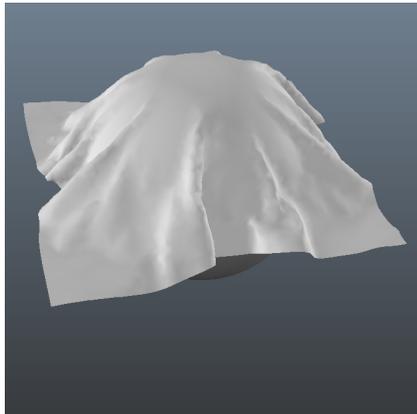
Figure 7.8 The sphere test in edge-based systems

and results in the bottom row of images is solved by SCF with a linear force model. In this case, Figure 7.9(b) applies the triangle-based IAR model, and Figure 7.9(c) applies the triangle-based IVT model.

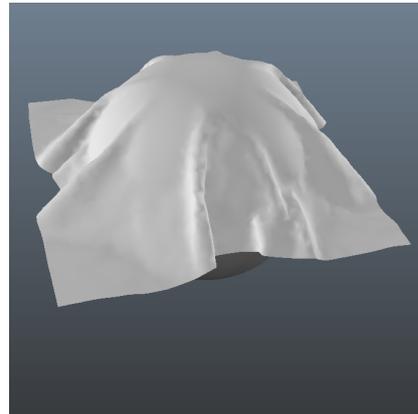
The simulated results of the non-linear and linear methods look almost similar, and we did not experience any particular side effect to our knowledge. Occasionally, the edge-based IER and the triangle based IAR method produce vibratory results. It is predictable that the artifact will be more noticeable when



(a) The non-linear method



(b) The linear IAR method



(c) The linear IVT method

Figure 7.9 The sphere test in triangle-based systems

there is a large fluctuation in the value of \mathbf{x}_{ij}^* . The IVT method is more robust in that aspect. When the IVT method is used, the vibratory artifact is reduced compared to the IER and IAR method. The reduction is more significant when the higher order estimation is used.

Chapter 8

Conclusion

In this paper, we presented a new way of formulating the linear stretch and shear model using $(\mathbf{x}_{ij} - \mathbf{x}_{ij}^*) \cdot (\mathbf{x}_{ij} - \mathbf{x}_{ij}^*)$ instead of $(|\mathbf{x}| - C)^2$, where \mathbf{x}_{ij}^* is some value supplied at each time step and regarded as constant when differentiating. In addition, we proposed the methods to calculate \mathbf{x}^* : the edge-based IER, IVT method, and the triangle-based IAR, IVT method. Edge-based methods can handle stretch force, and triangle-based methods can handle stretch and shear force.

Those linear models lead to a constant force jacobian and a constant system matrix, as a result. To solve a constant system matrix faster, we adopt sparse Cholesky factorization. Sparse Cholesky factorization utilizes matrix re-ordering, so the factorized lower triangular matrix maintains sparsity of system matrix.

In addition, we adopt Volino's linear bending model to complete the linear

force model of cloth simulation.

Practically the proposed techniques is easy to implement and allows for several levels of quality vs. speed trade-offs, providing a wide gamut of scalability. It can have immediate uses in real-time applications such as game and VR simulation. It can be also employed into off-line systems for previewing the draping of the constructed clothes before performing more accurate simulations of them.

Bibliography

- [1] BARAFF, D., AND WITKIN, A. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 43–54.
- [2] BREEN, D. E., HOUSE, D. H., AND WOZNY, M. J. Predicting the drape of woven cloth using interacting particles. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), SIGGRAPH '94, ACM, pp. 365–372.
- [3] BRIDSON, R., FEDKIW, R., AND ANDERSON, J. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 594–603.
- [4] BRIDSON, R., MARINO, S., AND FEDKIW, R. Simulation of clothing with folds and wrinkles. In *ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), SIGGRAPH '05, ACM.

- [5] CARIGNAN, M., YANG, Y., THALMANN, N. M., AND THALMANN, D. Dressing animated synthetic actors with complex deformable clothes. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1992), SIGGRAPH '92, ACM, pp. 99–104.
- [6] CHOI, K.-J., AND KO, H.-S. Extending the immediate buckling model to triangular meshes for simulating complex clothes. In *Eurographics 2003 Short Presentations* (2003), pp. 187–191.
- [7] CHOI, K.-J., AND KO, H.-S. Stable but responsive cloth. In *ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), SIGGRAPH '05, ACM.
- [8] CORDIER, F., AND MAGNENAT-THALMANN, N. Real-time animation of dressed virtual humans. *Comput. Graph. Forum* 21, 3 (2002), 327–335.
- [9] DESBRUN, M., SCHRÖDER, P., AND BARR, A. Interactive animation of structured deformable objects. In *Proceedings of the 1999 conference on Graphics interface '99* (San Francisco, CA, USA, 1999), Morgan Kaufmann Publishers Inc., pp. 1–8.
- [10] EBERHARDT, B., WEBER, A., AND STRASSER, W. A fast, flexible, particle-system model for cloth draping. *IEEE Computer Graphics and Applications* 16, 5 (Sept. 1996), 52–59.

- [11] GEORGE, A. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis* 10, 2 (1973), 345–363.
- [12] GEORGE, A., AND LIU, J. W. *Computer Solution of Large Sparse Positive Definite*. Prentice Hall Professional Technical Reference, 1981.
- [13] GEORGE, A., AND LIU, W. H. The evolution of the minimum degree ordering algorithm. *SIAM Rev.* 31, 1 (Mar. 1989), 1–19.
- [14] GRINSPUN, E., HIRANI, A. N., DESBRUN, M., AND SCHRÖDER, P. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 62–67.
- [15] KANG, Y.-M., AND CHO, H.-G. Bilayered approximate integration for rapid and plausible animation of virtual cloth with realistic wrinkles. In *Proceedings of the Computer Animation* (Washington, DC, USA, 2002), CA '02, IEEE Computer Society, pp. 203–.
- [16] KANG, Y.-M., CHOI, J.-H., CHO, H.-G., AND PARK, C.-J. Fast and stable animation of cloth with an approximated implicit method. In *Proceedings of the International Conference on Computer Graphics* (Washington, DC, USA, 2000), CGI '00, IEEE Computer Society, pp. 247–.
- [17] PROVOT, X. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *In Graphics Interface* (1996), pp. 147–154.

- [18] TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 205–214.
- [19] THOMASZEWSKI, B., AND WACKER, M. Bending models for thin flexible objects. In *WSCG (Short Papers)* (2006), pp. 171–178.
- [20] VOLINO, P., COURCHESNE, M., AND MAGNENAT THALMANN, N. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 137–144.
- [21] VOLINO, P., AND MAGNENAT-THALMANN, N. Simple linear bending stiffness in particle systems. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), SCA '06, Eurographics Association, pp. 101–105.

초 록

옷에서 일어나는 변형은 크게 평면 내 변형과 평면 외 변형으로 나눌 수 있다. 인장 변형과 전단 변형이 평면 내 변형, 굽힘 변형이 평면 외 변형에 속한다. 의류 시뮬레이션은 위 세 가지 변형을 모두 포함한다. 본 논문에서는 이러한 옷의 변형을 다루기 위한 새로운 물리 모델을 제시한다. 본 논문에서 제시하는 모델이 가지는 의의는 시뮬레이션을 수행하는데 필요한 계산량을 줄임으로써 수치적 시뮬레이션이 실시간에 이루어질 수 있도록 했다는 점과 기존의 실시간 모델에 존재했던 몇가지 결함을 해결함으로써 시뮬레이션 결과에서 보였던 문제점들을 해결했다는 점에 있다. 본 논문이 새로운 물리 모델을 개발함에 있어 주요한 아이디어는 인장, 전단, 굽힘 변형에 대한 에너지 함수에 존재하는 $(|\mathbf{x}| - C)^2$ 항을 \mathbf{x}^* 라는 벡터를 도입하여 $|\mathbf{x} - \mathbf{x}^*|$ 항으로 바꾼 데 있다. \mathbf{x}^* 벡터는 매 시간 구간마다 새로 구하는 값이나 힘 자코비안 행렬을 구하는 미분 과정에서는 상수로 간주한다. 이렇게 함으로써 힘 자코비안 행렬을 시간에 따라 변하지 않도록 만들고 그에 따라 시스템 행렬 역시 시뮬레이션을 수행하는 동안 고정된 값을 갖도록 만든다. 그 결과 시스템 행렬의 역행렬을 시뮬레이션 시작 전 사전 계산 시간에 미리 구할 수 있고, 내연적 시뮬레이션 진행 과정에서 시스템 행렬을 매번 새로 구성하고 해를 구해야 했던 것을 단순한 행렬과 벡터의 곱셈으로 대체할 수 있다. 본 논문은 이러한 선형 물리 모델을 선분 기반 시스템과 삼각형 기반 시스템에 대해 제시한다. 추가적으로 행렬과 벡터 곱셈 과정의 속도를 향상하기 위해 최신의

희소 출레스키 분해 방법을 살펴보고 의상 시뮬레이션에 효과적인 적용 방법을 소개한다.

주요어: 물리 기반 애니메이션, 의복 시뮬레이션, 실시간 시뮬레이션, 선형 모델, 희소 출레스키 분해

Keywords: physically based animation, cloth simulation, real-time simulation, linear model, sparse Cholesky factorization

학 번: 2003-21717