Real-time Simulation of Physically Realistic Global Deformations

by

Yan Zhuang

B.A. (University of Arizona) 1992
M.A. (University of Southern California) 1995
M.S. (University of Southern California) 1995

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Fall 2000

Real-time Simulation of Physically Realistic Global Deformations

Abstract

Real-time Simulation of Physically Realistic Global Deformations

by

Yan Zhuang

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor John Canny, Chair

In this thesis, we model and simulate large global deformations of linear viscous materials. Furthermore, we simulate the dynamic behaviors of such deformations using finite element methods (*FEM*).

Real-time simulation and animation of global deformation of 3D objects, using the finite element method, is difficult due to the following 3 fundamental problems: (1) The linear elastic model is inappropriate for simulating large motions and large deformations (unacceptable distortion will occur); (2) The time step for dynamic integration has to be drastically reduced to simulate collisions, if the traditional penalty methods are applied; (3) The size of the problem (the number of elements in the FEM mesh) is in n magnitude larger than that of a 2D problem.

In this thesis, we counter these 3 difficulties as following: (1) using quadratic strain instead of the popular linear strain to simulate arbitrarily large motions and global deformations of a 3D object; (2) applying an efficient collision constraint to a decoupled system, which makes an integration step for collision as cheap as a regular dynamic integration step; (3) using a graded mesh instead of a uniform mesh, which reduces the asymptotic complexity of a 3D problem to that of a 2D problem.

In order to preserve some of the subtle material properties such as viscous elasticity, we also present an alternative real-time solution without compromising the mass matrix in the FEM system. Instead of decoupling the system by diagonalizing the mass and damping matrix, we preprocess the system using modified nested dissection, which improves the sparsity of the system.

In this thesis, we also present how we can apply the same FEM model to simulate haptic feedback to a human operation in the virtual environment.

Professor John Canny
Dissertation Committee Chair

To Haixin,

my wife,

who is always there at the most difficult time,

and my parents,

Ming-de and Yuhua,

whose encouragement made my pursuit of a Ph.D. possible.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my Ph.D. advisor, Prof. John Canny, whose vision provides unlimited motivation for this project. I would like to thank my committee members, Prof. Papadopoulos and Prof. Shewchuk. Prof. Papadopoulos is an invaluable resource for FEM due to his expertise on the subject. Prof. Shewchuk introduced me into the field of meshing algorithms. I also thank Prof. Forsyth and Prof. Goldberg for their discussions and feedbacks about this project. Without those people's help, it would be impossible for me to conduct this research.

Chapter 1

Introduction

Physically realistic modeling and manipulation of deformable objects has been the bottleneck of many applications, such as human tissue modeling, character animation, surgical simulation, etc. Among the potential applications, a virtual surgical training system is the most demanding for real-time performance because of the requirement of real-time interaction with virtual human tissue.

So far *real-time* simulation and animation of deformation has only been achieved in two special cases: 2D problems such as cloth simulation [4], and small or local deformations for 3D objects [7, 24, 27, 18].

In this thesis, instead of focusing on one particular application, we address the bottleneck problem of real-time simulation of physically realistic large *global deformations* of 3D objects. By *global deformation*, we mean deformations, such as large twisting or bending of an object (figure (2.3) and (2.4)), which involve the entire body, in contrast to poking and squeezing, which involve a relatively small region of the deformable object. Large global deformations are essential for applications such as surgical simulation, in which tissue will often be folded.

In contrast to most graphics research, we demand accuracy in addition to visually satisfactory rendering. This makes finite element method (*FEM*) a better choice than the popular mass-spring model often used by the graphics community, because the finite element method is much more mathematically rigorous than a mass-spring model. In particular we apply the geometrically nonlinear finite element method (FEM) to model large global deformations.

Finite element methods usually lead to a large system of differential equations. It is difficult and challenging to solve such a system in real-time. In this thesis, different approaches are explored and discussed.

Collision is always a bottleneck in real-time dynamic simulation. The existing collision handling methods lead to dramatic slow down of the simulation when collision occurs. Simulating deformable object collisions using a penalty method [39] requires tiny time steps to generate visually satisfactory animations. A general impulse collision [3] is considered more efficient and accurate but still requires more computational power than collision-free dynamics. In this thesis, we present an extremely simple and efficient collision time integration scheme, which makes the time integration of collision dynamics as cheap as that of collision-free dynamics.

We observe that simulation of 3D deformation is at least one order of magnitude more difficult than a similar 2D problem because the size of the problem (the number of elements in its mesh) is one order of magnitude higher. By *one order of magnitude*, we mean that the size of a 3D finite element mesh is $O(n)$ larger than that of a 2D finite element mesh, where n is the number of elements in each principle direction. In order to counter this problem we propose a *graded* mesh that reduces the complexity of the 3D problem by one order of magnitude asymptotically.

In this thesis, we also present how to apply the FEM model to provide haptic feedback to the human operator.

1.1 Related Works

Our work of modeling and simulating a deformable object falls into the realm of physically based modeling. Witkin *et al* [44] summarizes the methods and principles of physically based modeling, which has emerged as an important new approach to computer animation and computer graphics modeling.

In general, there are several different approaches to modeling deformable objects: mass-spring models, finite element methods, finite difference methods, and boundary element methods. Gibson and Mirtich [14] gives a comprehensive review on mass-spring models and finite element methods.

The mass spring model has had good success in creating visually satisfactory animations. Waters [42] uses a spring model to create a realistic 3D facial expression. Provot *et al* [29] describes a 2D model for animating cloth, using double cross springs.

Promayon *et al* [28] presents a mass-spring model of 3D deformable objects and develops some control techniques.

Despite the success in some animation applications, the mass spring models do not model the underlying physics accurately, which makes them unsuitable for simulations that require more accuracy. The structure of the mass spring is often application dependent and hard to interpret. The animation results often vary dramatically with different spring structures. The distribution of the mass to nodes is somewhat (if not completely) arbitrary. Despite its inaccuracy, it does not have visual distortion and it is computationally cheap to integrate over time because the system is, by its very nature, a set of independent algebraic equations, which requires no matrix inversions to solve.

As an alternative, finite element methods (*FEM*) model the continuum much more accurately and their underlying mathematics are well studied and developed. Finite element methods approximate the complex geometry and deformations by piecewise simple functions, such as low order polynomials. Due to the accuracy and mathematical rigorousness, FEM is a better choice for applications such as surgical simulations.

Another similar method is the finite difference method, which is less accurate and simpler. Indeed a linear finite difference method over a uniform mesh is just a special case of FEM. Both finite element methods and finite difference methods provide approximate solutions to differential equations. The difference lies where the approximation occurs. Finite element methods approximate the solution using piecewise smooth functions such as low order polynomials, while using the exact differential operators. Finite difference methods approximate the differential operators themselves by difference operators. Usually finite difference methods require meshes over regular grids.

Terzopoulos *et al* [39, 38, 40] applies both finite difference and finite element methods in modeling elastically deformable objects. Celniker *et al* [24] applies FEM to generate primitives that build continuous deformable shapes designed to support a new free-form modeling paradigm. Pieper *et al* [27] applies FEM to computer-aided plastic surgery. Chen [5] animates human muscle using a 20 node hexahedral FEM mesh. Keeve *et al* [20] develops a static anatomy-based facial tissue model for surgical simulation using the FEM. Most recently, Cotin *et al* [7] presents real-time elastic deformation of soft tissues for surgery simulation, which only simulates static deformations.

Boundary element methods (*BEM*) have the advantage of solving a smaller system because they only deal with degrees of freedom on the surface of the model. However, it is usually more expensive to solve such a system than an FEM system, because the matrices derived in boundary element methods are dense, while the matrices derived by FEM are sparse. The asymptotic bounds for solving a system derived from boundary element method and that from FEM are both $O(n^3)$, where n is the number of nodes in each principle direction. However it is, in practice, computationally cheaper to solve an FEM system because of its sparsity. Besides the numerical disadvantages, it is difficult to apply boundary element methods to model non-homogeneous material and materials with internal state. This makes boundary element methods inappropriate for tissue modeling because tissues are not necessarily homogeneous materials. James and Pai [18] model real-time quasi-static local deformations using the boundary element method (BEM). The deformations are simulated with a locally linear model. The real-time performance achieved by [18] requires the assumption that the types of boundary condition changes at a very small region. The solution update takes $O(s^3 + n^2 s)$ time, where n is the number of nodes in each principle direction and s is the number of nodes on which the boundary condition changes types. In the worst case, its complexity is $O(n^6)$.

Our work described in this thesis differs from the previous work by either one or all of the following: (1) we simulate large global deformations instead of small local deformations; (2) we simulate the dynamic behavior of soft objects rather than the static deformation. Also instead of using a uniform mesh as all the related works have done, we use a graded mesh to reduce the complexity of the model.

1.2 Thesis Outline

In chapter 2, we present the background work on theory of elasticity and finite element methods. In this chapter, the nonlinear system of differential equations that model the dynamic behavior of global deformations of linear viscous material are derived.

In chapter 3, we present the *real-time* numerical solution that solves the system derived in chapter 2. In particular, in chapter 3.1.1, we discuss how to decouple the system such that we can apply an explicit integration algorithm. Chapter 3.2 is dedicated to real-time collision handling. In chapter 3.3, we introduce the concept of graded meshes and discuss its favorable properties in terms of both complexity and numerical accuracy. Part of the material in this chapter is published by Zhuang and Canny in [46].

In chapter 4, we present the modified nested dissection that solve the FEM system in real-time without compromising the mass property as in chapter 3.1.1. We also present comparisons of our algorithm versus several related algorithms, such as minimal degree algorithm and graph partition algorithm on the connectivity graph. The comparison shows that the original geometry of finite element mesh itself provides better heuristics than the corresponding connectivity graph. This chapter is based on published work by Zhuang and Canny [48].

Chapter 5 is relatively independent from the rest of the thesis. It applies the same FEM model to simulate the haptic feedback when a human operator interacts with a virtual deformable object in a virtual environment. The material in this chapter is published by Zhuang and Canny in [47].

Finally, we summarize the thesis and discuss the future directions of our research in chapter 6.

Chapter 2

Finite Element Formulation of Dynamic Global Deformations

In this chapter, we discuss how we model the global deformation using geometrically nonlinear finite element methods. First, we will briefly introduce the fundamental concepts in continuum mechanics. Then we will derive the finite element formulation of deformations using *virtual work* principle. This chapter also serves as a quick tutorial to finite element method. For those who are interested in more details on this subject, we would like to refer them to [10, 15, 30, 50].

2.1 Traction, Stress and Strain

In this section, we introduces the fundamental concepts for the description and measurement of the state of deformations.

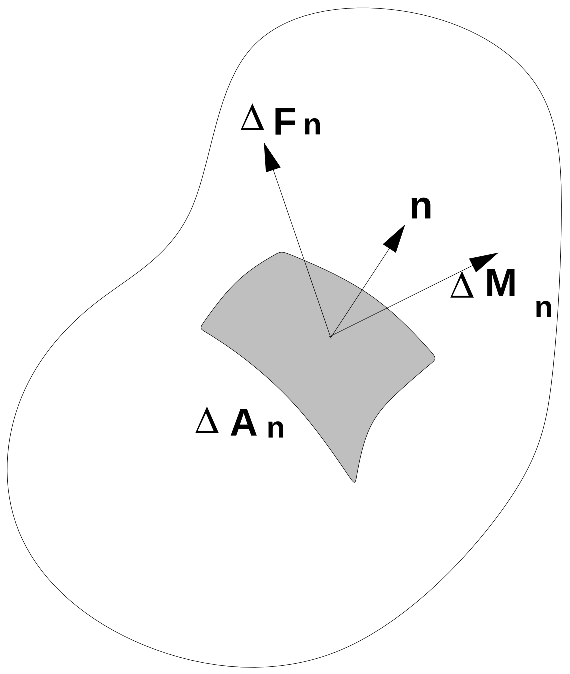


Figure 2.1: Resultant force and moment on a small oriented area.

Given a deformable body (Figure 2.1), consider a small area $\triangle A_n$ on the surface of or within the body, which has an orientation specified by unit normal vector $\hat{\mathbf{n}}$. Let $\triangle \mathbf{F_n}$ and $\triangle \mathbf{M_n}$ be the resultant force vector and moment vector, respectively, exerted on oriented area $\triangle A_n$. We seek the intensity of the resultants on the oriented area element $\triangle A_n$ by taking the limit as following:

$$\lim_{\triangle A_n \to 0} \frac{\triangle \mathbf{F_n}}{\triangle A_n} = \frac{d\mathbf{F_n}}{dA_n} = \mathbf{T_n} \tag{2.1}$$

$$\lim_{\triangle A_n \to 0} \frac{\triangle \mathbf{M_n}}{\triangle A_n} = \frac{d\mathbf{M_n}}{dA_n} = \mathbf{C_n} \tag{2.2}$$

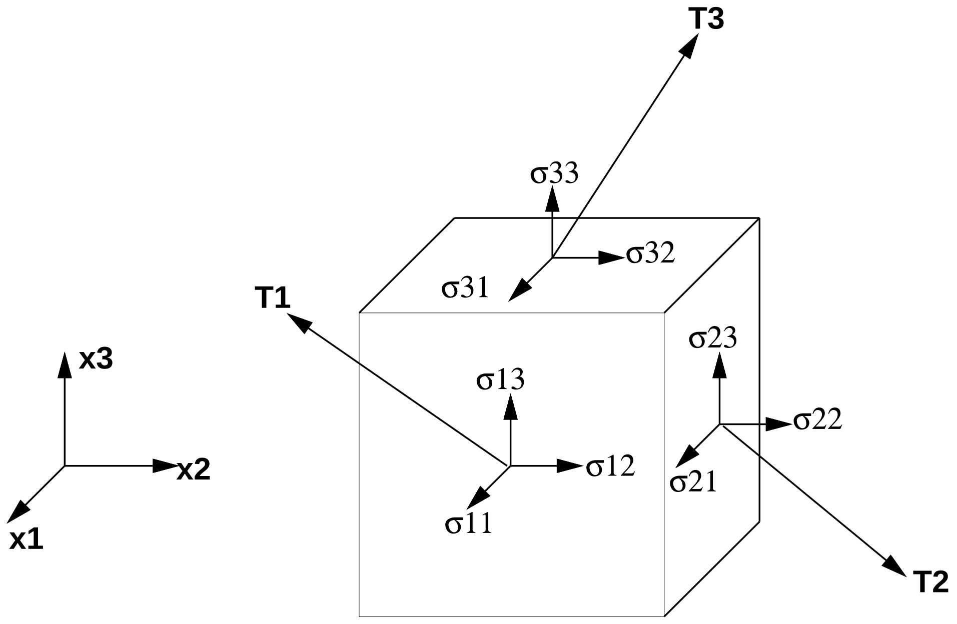


Figure 2.2: Components of stress.

$\mathbf{T_n}$ is the stress vector or *traction*, $\mathbf{C_n}$ is the *couple-stress vector*. The elementary theory of elasticity proceeds on the assumption that $\mathbf{C_n} = \mathbf{0}$, while the traction $\mathbf{T_n}$ represents the stress intensity at the point for the particular area element of orientation specified by unit normal vector $\hat{\mathbf{n}}$. A complete description of the stress at the point requires the traction for all directions. Fortunately this problem can be solved by traction in principle directions.

Given a point (x_1, x_2, x_3)[1] in the deformable body, consider an infinitesimal cube centered at this point. Figure 2.2 shows the 3 tractions T_i, $i = 1, 2, 3$, on the faces with surface normal vectors along the principle directions. Each such traction T_i has components along the principle axis. The 9 stress components (each a scale value)

[1]In this thesis, we will use (x, y, z) and (x_1, x_2, x_3) interchangably.

shown in figure 2.2 form a second order tensor P as follows:

$$P = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{pmatrix} \tag{2.3}$$

This tensor P uniquely defines the state of stress at point (x_1, x_2, x_3). Given an area element of surface normal $\hat{n}$, the corresponding traction is given by $P^T\hat{n}$.

Stress tensor P gives a complete description of the internal force intensity at a given point. Similarly, a strain tensor at a point gives a complete description of the state of deformation at that point. The strain tensor is also second order tensor as following:

$$\begin{pmatrix} \epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\ \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\ \epsilon_{31} & \epsilon_{32} & \epsilon_{33} \end{pmatrix} \tag{2.4}$$

Using the equilibrium condition of zero moment at point (x_1, x_2, x_3) and shrinking the cube to zero-volume, one can easily derive the following:

$$\sigma_{ij} = \sigma_{ji} \tag{2.5}$$

It requires more complicated mathematical treatment to establish the symmetry of strain tensor ([10]). Here we simply state this fact without the mathematical derivation:

$$\epsilon_{ij} = \epsilon_{ji} \tag{2.6}$$

Due to the symmetry, there are only 6 independent variables in the stress tensor and the strain tensor. In the rest of this thesis, we will simply use the vector representation for stress and strain, instead of their tensor representation. The vector representation of stress is:

$$\sigma = \left\{ \begin{array}{c} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{yz} \\ \tau_{zx} \\ \tau_{xy} \end{array} \right\} \tag{2.7}$$

where $\sigma_x = \sigma_{11}$, $\sigma_y = \sigma_{22}$, $\sigma_z = \sigma_{33}$, $\tau_{yz} = \sigma_{23} = \sigma_{32}$, $\tau_{zx} = \sigma_{31} = \sigma_{13}$, and $\tau_{xy} = \sigma_{12} = \sigma_{21}$. The first 3 components are called *normal stresses*, the other 3 are called *shear stresses.*

The vector representation of strain is:

$$\epsilon = \begin{Bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{yz} \\ \gamma_{zx} \\ \gamma_{xy} \end{Bmatrix} \tag{2.8}$$

Given the point-wise displacement (u, v, w) of the deformable body, the strain is given as following:

$$\epsilon_x = \frac{\partial u}{\partial x} + \frac{1}{2}\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial x}\right)^2\right] \tag{2.9}$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} + \left[\frac{\partial u}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\frac{\partial v}{\partial y} + \frac{\partial w}{\partial x}\frac{\partial w}{\partial y}\right] \tag{2.10}$$

The other 4 terms of the strain vector is defined similarly. If the motion of the deformable body is small, the second order term can be neglected. This leads to the linear strain approximation for small motion (deformation), which have the following forms:

$$\epsilon_x = \frac{\partial u}{\partial x} \tag{2.11}$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{2.12}$$

A more mathematically complete development of traction, stress and strain can be found in [10].

2.2 Linear Strain and Nonlinear Strain

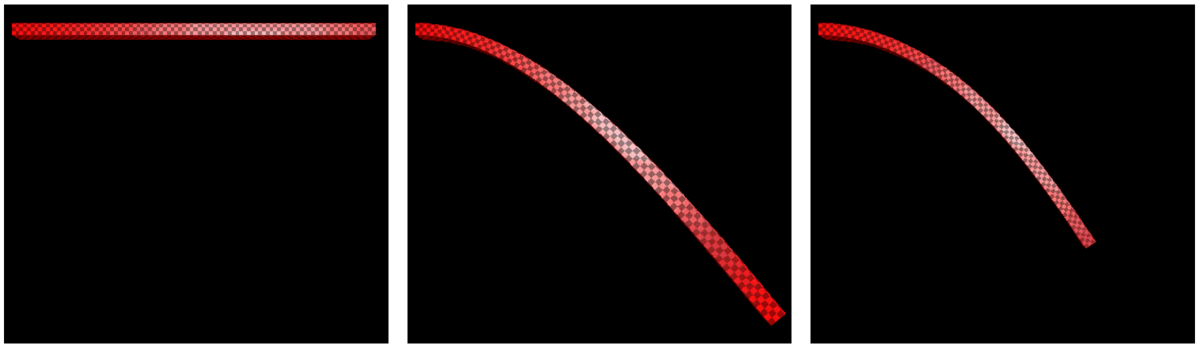


Figure 2.3: The left image shows a beam at its initial configuration with a fixed left end and a free right end. The middle image shows the *distorted* deformation under gravity, using linear strain. The right image shows the *undistorted* deformation, under the same gravitational force, using quadratic strain (equation (2.9) and (2.10)).

In this thesis, we focus on large global deformations. By *global deformations*, we mean deformations that are large and involve the entire body, such as high amplitude bending and twisting (figure 2.3 and 2.4). These types of deformations are essential to surgical simulations. For example, in a surgical simulation, it is essential to simulate how tissue folds. Although we are not building a surgical simulator, this thesis is meant to establish the fundamental framework for modeling tissues. Therefore we would like to choose a modeling paradigm that can handle large global deformations.

To our best knowledge the published research ([27, 5, 20, 7]) assume small deformations in their virtual environment. The most simulated deformations are those

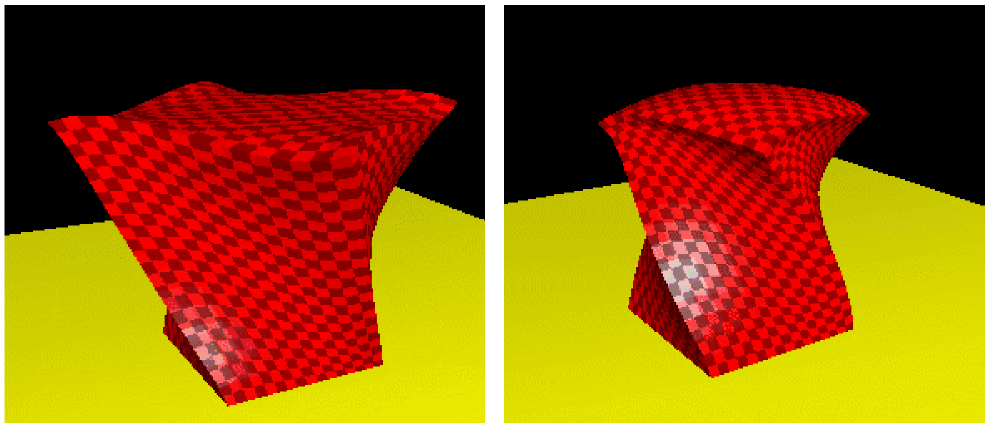


Figure 2.4: The bottom of the object is fixed and its top is twisted. The top in the left image is distorted (expanded bigger) because it is simulated using linear elasticity. The right image shows that the same distortion does not occur with nonlinear elasticity.

caused by squeezing and poking at a relatively small surface region. The small deformation assumption leads to the often used linear elasticity model, which is based on the linear strain approximation (2.11) and (2.12).

For large global deformations, the linear strain is not appropriate because it does not model large motions exactly. The linear strain is derived from the assumption of small motion. Recall that the exact strain defined in (2.9) and (2.10) are derived without the assumption of small motions. Therefore it mathematically applies to any large motion and deformation.

To further illustrate the difference between the exact strain and the linear strain, let us examine an artificial motion of a deformable object Ω in details. For the sake of illustration, assume that we subject the object Ω to an arbitrary rotation R. Furthermore, we assume that the object Ω is originally undeformed and remains undeformed after this rotation. Any reasonable model should be able to handle such a "rigid" body rotation exactly. In terms of strain, an appropriate model should be able to give zero strain in such a rotation, because there is no deformation after a pure rotation.

After the rotation of Ω, every point in the Ω has a displacement. Let (x, y, z) be an arbitrary point in the deformable body Ω. The pointwise displacement of Ω is:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = R \begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r_{11}-1 & r_{12} & r_{13} \\ r_{21} & r_{22}-1 & r_{23} \\ r_{31} & r_{32} & r_{33}-1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{2.13}$$

If we apply the linear strain (2.11) to this displacement field of Ω, we have

$$\begin{aligned} \epsilon_x &= r_{11} - 1 \neq 0 \\ \gamma_{xy} &= r_{12} + r_{12} \neq 0 \end{aligned} \tag{2.14}$$

However this is apparently problematic. The undeformed body is still undeformed after a rigid body rotation, therefore the strain should be zero regardless how large the rotation is. If we apply such a linear strain to model large deformations, unacceptable distortions will occur. (figure 2.3 and 2.4). This artificial non-zero strain introduces artificial non-zero stress. The deformable body has to deform distortionally to counter balance such artificial stress.

Now let us exam the quadratic *exact strain* given by (2.9) and (2.10). If we apply

(2.9) and (2.10), we have

$$\begin{aligned}
\epsilon_x &= (r_{11}-1)+\tfrac{1}{2}[(r_{11}-1)^2+r_{21}^2+r_{31}^2] \\
&= r_{11}-1+\tfrac{1}{2}[r_{11}^2-2r_{11}+1+r_{21}^2+r_{31}^2] \\
&= r_{11}-1+\tfrac{1}{2}[2-2r_{11}] \\
&= 0 \qquad (2.15) \\
\gamma_{xy} &= r_{12}+r_{21}+[(r_{11}-1)r_{12}+r_{21}(r_{22}-1)+r_{31}r_{32}] \\
&= r_{11}r_{12}+r_{21}r_{22}+r_{31}r_{32} \\
&= 0 \quad (Orthonormality)
\end{aligned}$$

Other 4 terms of the exact strain can be verified similarly. This shows that the exact strain can handle arbitrarily large deformations and motions of deformable body. Because large motions and deformations are essential in character animation and surgical simulation, we choose to model global deformations using the exact strain instead of linear strain. To emphasize the nonlinear nature of the exact strain, we refer to it as the *quadratic strain* in the rest of the thesis.

The quadratic strain models arbitrarily large rotations exactly. Furthermore, since no small motion assumption is needed in applying the quadratic strain, we can use a fixed global frame for the entire simulation. In this thesis, all equations are derived using a fixed global coordinate system.

2.3 Constitutive Equations and Linear Elasticity

The properties of materials are specified by *constitutive equations.* In particular a stress-strain relationship describes the mechanical properties of a material and is therefore a constitutive equation.

In this thesis, we simply model the deformable objects as damped linear elastic material. Namely there is a linear relationship between the stress vector and the strain vector as following:

$$\sigma = \mathbf{\Lambda}(\epsilon - \epsilon_0) + \sigma_0 \tag{2.16}$$

where $\mathbf{\Lambda}$ is an elasticity matrix containing the appropriate material properties, and σ_0 and ϵ_0 are the initial strain and stress, respectively. Usually we assume both initial strain and stress are zero. Therefore we have the following *linear strain-stress* relationship:

$$\sigma = \mathbf{\Lambda}\epsilon \tag{2.17}$$

If the material is isotropic, the elasticity matrix $\mathbf{\Lambda}$ only has two degrees of freedom: Young's modulus E and Poisson's ratio ν.

The elasticity matrix $\mathbf{\Lambda}$ for isotropic elastic material has the following form:

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda + 2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{pmatrix} \tag{2.18}$$

where μ and λ are called *lamé's constants*, and

$$\begin{aligned} \lambda &= \tfrac{E\nu}{(1+\nu)(1-2\nu)} \\ \mu &= \tfrac{E}{2(1+2\nu)} \end{aligned} \tag{2.19}$$

2.4 Geometrically Nonlinear Finite Element Method

The theory of elasticity is a fundamental discipline in studying continuum material. It consists of a consistent set of differential equations that uniquely describe the state of stress, strain and displacement of each point within an elastic deformable body. It consists of equilibrium equations relating the stresses; kinematics equations relating the strains and displacements, constitutive equations relating the stresses and strains; and boundary conditions relating to the physical domain. The theory was first developed by Louis-Marie-Henri Navier, Dimon-Denis Poisson and George Green in the first half of the 19th century [43].

Synthesizing those equations allows us to establish a relationship between the deformation of the object and the exerted forces. However an analytic expression of such relationship is impossible, except for a small number of simple problems. *Finite element methods (FEM)* are one way to solve such a set of differential equations. From now on, we will discuss elasticity within the context of finite element methods.

When the geometry of the deformable object is complicated, it is impossible to obtain an analytic solution of an elastic deformation. FEM solves this problem by subdividing the object into small sub-domains with simple shapes (tetrahedra, hexahedra, etc.), called finite elements (Figure 2.5). The sub-division (mesh) does not only approximate the original geometry, but also leads to a discrete representation of the deformation.

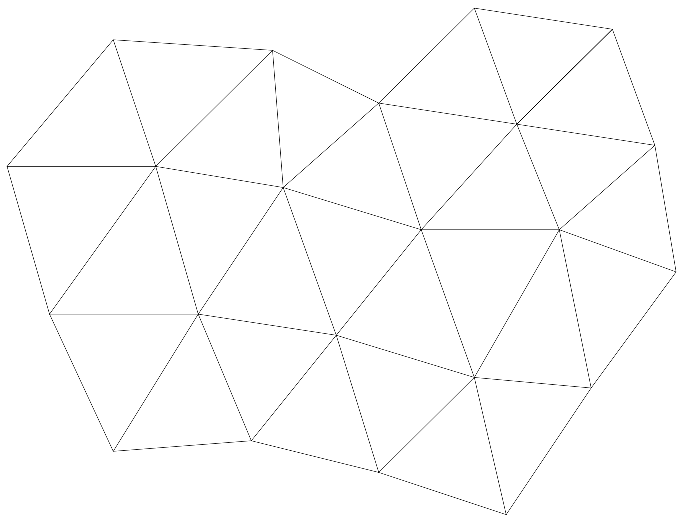


Figure 2.5: A 2-dimension example of finite element mesh of a region. Each triangle is a finite element.

In particular, we apply a *displacement based* finite element method to simulate such deformation. Namely displacements at vertices of the mesh, called nodes, will be calculated. The values at other points within the element are interpolated by a weighted sum of all the nodal displacements within the element. The global equations (the relationship between all the nodal values) are obtained by assembling elementwise equations by imposing inter-element continuity of the solution and balancing of inter-element forces.

We show the finite element formulation of a linear viscous material. In particular, we use linear hexahedral element and quadratic strain. Note that the procedure can be easily generalized to elements of different shapes, such as tetrahedral elements, and higher order elements.

2.4.1 Elementwise FEM Formulation

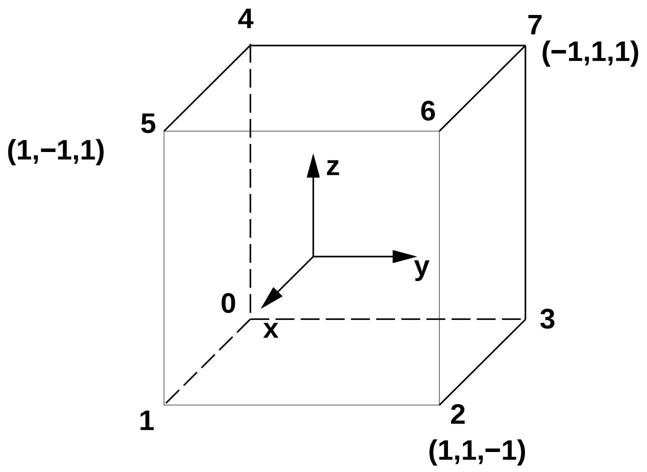


Figure 2.6: A cubic element of size 2, centered at the origin.

For the simplicity of the mathematical presentation, we further assume that the hexahedral element in consideration is a cube centered at the origin and that each side of the cube has a length of 2 (Figure 2.6). Later we will show how to generalize this to hexahedral elements of general sizes and shapes.

To emphasize the fact that our discussion is now localized to a particular element, we use a super-scripted "e" to all the quantities within the given element. For example, we refer to the the entire body as Ω, while referring to a particular element as Ω^e.

Let $\mathbf{u_i}^e$, $i = 0, \ldots, 7$ be the nodal displacements of the corresponding nodes in figure 2.6. The displacement of any point (x, y, z) within this element is approximated by weighted sum of the nodal displacements as following:

$$\mathbf{u}(x, y, z) = \sum_{i=0}^{7} \phi_i^e(x, y, z)\mathbf{u}_i^e \tag{2.20}$$

where ϕ_i^e's are the interpolation functions (*weight function*). Particularly in our case, ϕ_i^e's are tri-linear functions as following:

$$\phi_0^e(x, y, z) = \frac{1}{8}(1 - x)(1 - y)(1 - z) \tag{2.21}$$

The other 7 weight functions are defined similarly with the same key property: ϕ_i *evaluates to 1 at node i and evaluates to 0 at other nodes.*

Let $\mathbf{I}_3$ be the 3-dimensional identity matrix. The point-wise displacement within the element Ω^e (2.20) can be written in the following matrix form:

$$\mathbf{u}(x,y,z) = \mathbf{\Phi}^e \mathbf{u}^e \tag{2.22}$$

where

$$\mathbf{\Phi}^e = (\phi_0^e \mathbf{I}_3, \phi_1^e \mathbf{I}_3, \ldots, \phi_7^e \mathbf{I}_3)_{3\times 24} \tag{2.23}$$

and

$$\mathbf{u}^e = \begin{pmatrix} \mathbf{u}_0^e \\ \mathbf{u}_1^e \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{u}_7^e \end{pmatrix}_{24\times 1} \tag{2.24}$$

Next, we apply virtual work to derive discrete system of equations. Let $\mathbf{b}^e$ be the distributed body force acting on the element, per unit volume. Note that $\mathbf{b}^e$ is a 3×1 vector. Let $\mathbf{q}^e$ be the equivalent nodal forces, which are equivalent to the boundary stresses and distributed loads on the element. Particularly in our case, $\mathbf{q}^e$ is a 24×1 vector, where the first 3 components are corresponding to node 0, the next 3 components are corresponding to node 1, and so on.

Since we are interested in the dynamic behavior of elastic body, we have to consider the inertia force and damping force which opposes the motion. The static equivalent of the inertia force per unit volume is:

$$-\rho\ddot{\mathbf{u}} \tag{2.25}$$

The static equivalent of a simple linear viscous damping force per unit volume is:

$$-\mu\dot{\mathbf{u}} \tag{2.26}$$

Therefore the equivalent static distributed force acting on a unit volume within the element is as following:

$$\mathbf{b}^e - \rho\ddot{\mathbf{u}} - \mu\dot{\mathbf{u}} \tag{2.27}$$

To make the nodal forces equivalent to the actually boundary stresses and distributed loads, we impose an *arbitrary virtual* displacement to the element and equate the external and internal work done by the various forces and the stresses. This approach is called *virtual energy principle.*

Given $\delta\mathbf{u}^e$ as the virtual displacement at the nodes, the displacement within the element is

$$\delta\mathbf{u} = \mathbf{\Phi}^e\delta\mathbf{u}^e \tag{2.28}$$

The strain is as following ([50]):

$$\delta\epsilon = \mathbf{B}^e\delta\mathbf{u}^e \tag{2.29}$$

The virtual work done by the nodal forces is equal to the sum of the products of the individual force components and corresponding virtual nodal displacement. This can be written in matrix form as following:

$$\delta \mathbf{u}^{eT} \mathbf{q}^e \tag{2.30}$$

The virtual internal work per unit volume done by the stresses, distributed body force, inertia force and damping force is

$$\delta \epsilon^T \sigma - \delta \mathbf{u}^T (\mathbf{b}^e - \rho \ddot{\mathbf{u}} - \mu \dot{\mathbf{u}}) \tag{2.31}$$

By (2.28) and (2.29), we have the internal virtual work per unit volume as:

$$\delta \mathbf{u}^{eT} [\mathbf{B}^e \sigma - \mathbf{\Phi}^{eT} (\mathbf{b}^e - \rho \ddot{\mathbf{u}} - \mu \dot{\mathbf{u}})] \tag{2.32}$$

Virtual works on the element can be obtained by integrating (2.30) and (2.32) over the volume of the entire element, denoted by Ω^e. According to the principles of energy, the virtual internal work have to be equal to the virtual external work (virtual work done by the equivalent nodal forces). Therefore we have the following:

$$\delta \mathbf{u}^{eT} \mathbf{q}^e = \delta \mathbf{u}^{eT} \int_{\Omega^e} [\mathbf{B}^e \sigma - \mathbf{\Phi}^{eT} (\mathbf{b}^e - \rho \ddot{\mathbf{u}} - \mu \dot{\mathbf{u}})] d\Omega^e \tag{2.33}$$

Since $\delta \mathbf{u}^{eT}$ is arbitrary virtual displacement, it can be canceled out. Therefore we have the following equality:

$$\mathbf{q}^e = \int_{\Omega^e} [\mathbf{B}^e \sigma - \mathbf{\Phi}^{eT}(\mathbf{b}^e - \rho \ddot{\mathbf{u}} - \mu \dot{\mathbf{u}})] d\Omega^e \tag{2.34}$$

By (2.22), we have

$$\begin{aligned} \dot{\mathbf{u}} &= \mathbf{\Phi}^e \dot{\mathbf{u}}^e \\ \ddot{\mathbf{u}} &= \mathbf{\Phi}^e \ddot{\mathbf{u}}^e \end{aligned} \tag{2.35}$$

Applying (2.35) to (2.34), we have

$$\mathbf{q}^e = \int_{\Omega^e} \mathbf{B}^e \sigma d\Omega^e - \int_{\Omega^e} \mathbf{\Phi}^{eT} \mathbf{b}^e d\Omega^e + \int_{\Omega^e} \mathbf{\Phi}^{eT} \rho \mathbf{\Phi}^e \ddot{\mathbf{u}}^e d\Omega^e + \int_{\Omega^e} \mathbf{\Phi}^{eT} \mu \mathbf{\Phi}^e \dot{\mathbf{u}}^e d\Omega^e \tag{2.36}$$

Finally we obtain the following matrix differential equation for the finite element in figure 2.6:

$$\mathbf{M}^e \ddot{\mathbf{u}}^e + \mathbf{D}^e \dot{\mathbf{u}}^e + \mathbf{R}^e(\mathbf{u}^e) = \mathbf{F}^e + \mathbf{q}^e \tag{2.37}$$

where

$$\mathbf{M}^e = \int_{\Omega^e} \mathbf{\Phi}^{eT} \rho \mathbf{\Phi}^e d\Omega^e \tag{2.38}$$

$$\mathbf{D}^e = \int_{\Omega^e} \mathbf{\Phi}^{eT} \mu \mathbf{\Phi}^e d\Omega^e \tag{2.39}$$

$$\mathbf{F}^e = \int_{\Omega^e} \mathbf{\Phi}^{eT} \mathbf{b}^e d\Omega^e \tag{2.40}$$

and

$$\mathbf{R}^e(\mathbf{u}^e) = \int_{\Omega^e} \mathbf{B}^{eT} \sigma d\Omega^e \tag{2.41}$$

In equation (2.37), $\mathbf{u}^e$ is the 24-dimensional nodal displacement vector of the element; $\dot{\mathbf{u}}^e$ and $\ddot{\mathbf{u}}^e$, the respective nodal velocity and nodal acceleration vectors; $\mathbf{F}^e$, the equivalent 24-dimensional nodal body force vector; $\mathbf{q}^e$, the equivalent nodal force due to stress and load on the boundary of the element; $\mathbf{M}^e$, the 24×24 element mass matrix; $\mathbf{D}^e$, the 24×24 element damping matrix; and $\mathbf{R}^e(\mathbf{u}^e)$, the 24-dimensional equivalent nodal internal force vectors due to deformation.

We have not given the mathematical expression of $\mathbf{q}^e$. The reason is that due to element continuity, this term will eventually "disappear", except on the boundary of the body, when we have the global equations for the entire deformable body. On the surface of element interface in the interior of the body, this term have the same value and opposite directions between any two adjacent elements, which will be canceled in the global equations. Therefore we will not give the expression of this term until we derive the global equations in chapter 2.4.4.

2.4.2 What is $\mathbf{B}^e$ and $\mathbf{R}^e$

The term $\mathbf{R}^e(\mathbf{u}^e)$ is more complicated than others in equation (2.37). We will derive it in this section.

We derive it using the following definition (2.29) ([50]):

$$d\epsilon = \mathbf{B}^e d\mathbf{u}^e \tag{2.42}$$

A simple case of this is when we use the linear strain approximation (2.11) and (2.12). We can write the vector representation of a linear strain in the following matrix form:

$$\epsilon = \begin{Bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{yz} \\ \gamma_{zx} \\ \gamma_{xy} \end{Bmatrix} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \mathbf{Su} \tag{2.43}$$

where $\mathbf{S}$ is the linear differential operator in matrix form and $\mathbf{u}$ is displacement vector at any given point (x, y, z).

By (2.42), we have

$$d\epsilon = d\mathbf{Su} = \mathbf{S}d\mathbf{u} \tag{2.44}$$

By equation (2.22), we have

$$d\epsilon = \mathbf{S}\Phi^e d\mathbf{u}^e \tag{2.45}$$

Therefore we have

$$\mathbf{B}^e = \mathbf{S}\Phi^e \tag{2.46}$$

Let us rename this $\mathbf{B}^e$ to $\mathbf{B}_0^e$ to reflect the fact that it is a special case of $\mathbf{B}^e$ with linear strain. If we have a linear stress-strain relationship as spesified by (2.17), we get

$$\sigma = \mathbf{\Lambda}\epsilon = \mathbf{\Lambda}\mathbf{S}\mathbf{u} = \mathbf{\Lambda}\mathbf{S}\mathbf{\Phi}^e\mathbf{u}^e = \mathbf{\Lambda}\mathbf{B}_0^e\mathbf{u}^e \tag{2.47}$$

Therefore, given a linear strain, the term $\mathbf{R}^e(\mathbf{u}^e)$ can be simplified to a linear function of $\mathbf{u}^e$ as following:

$$\begin{aligned} \mathbf{R}^e(\mathbf{u}^e) &= \textstyle\int_{\Omega^e} \mathbf{B}^{eT}\sigma d\Omega^e \\ &= \textstyle\int_{\Omega^e} \mathbf{B}_0^{eT}\mathbf{\Lambda}\mathbf{B}_0^e\mathbf{u}^e d\Omega^e \\ &= \mathbf{K}^e\mathbf{u}^e \end{aligned} \tag{2.48}$$

where $\mathbf{B}_0^e$ is given by (2.46) and

$$\mathbf{K}^e = \int_{\Omega^e} \mathbf{B}_0^{eT}\mathbf{\Lambda}\mathbf{B}_0^e d\Omega^e \tag{2.49}$$

This will simplify the equality (2.37) to a linear system of differential equations:

$$\mathbf{M}^e\ddot{\mathbf{u}}^e + \mathbf{D}\dot{\mathbf{u}}^e + \mathbf{K}^e\mathbf{u}^e = \mathbf{F}^e + \mathbf{q}^e \tag{2.50}$$

As discussed earlier, we are only interested in simulating large global deformations that can not be approximated by linear strains. Instead we have to use the quadratic strains defined in (2.9) and (2.10).

First let us define 3 new vectors as following:

$$\theta_x = \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial x} \\ \frac{\partial w}{\partial x} \end{pmatrix} \tag{2.51}$$

where θ_y and θ_z are defined similarly.

By the quadratic strain definitions (2.9) and (2.10), and notation defined in (2.51), we can write the quadratic strain in the following matrix form:

$$\begin{aligned} \epsilon &= \begin{pmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} + \frac{1}{2} \begin{pmatrix} \theta_x^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \theta_y^T & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \theta_z^T \\ \mathbf{0} & \theta_z^T & \theta_y^T \\ \theta_z^T & \mathbf{0} & \theta_x^T \\ \theta_y^T & \theta_x^T & \mathbf{0} \end{pmatrix}_{6\times 9} \begin{pmatrix} \theta_x \\ \theta_y \\ \theta_z \end{pmatrix}_{9\times 1} \\ &= \mathbf{Su} + \tfrac{1}{2}\mathbf{A\Theta} \end{aligned} \tag{2.52}$$

where $\mathbf{A}$ is nonlinear.

By (2.42) and (2.46), we have

$$d\epsilon = \mathbf{B}_0^e d\mathbf{u}^e + \frac{1}{2}d(\mathbf{A}\theta) = \mathbf{B}_0^e d\mathbf{u}^e + \frac{1}{2}d\mathbf{A\Theta} + \frac{1}{2}\mathbf{A}d\mathbf{\Theta} \tag{2.53}$$

It is easy to verify the following property of matrix $\mathbf{A}$ and $\mathbf{\Theta}$:

$$d\mathbf{A}\boldsymbol{\Theta} = \mathbf{A}d\boldsymbol{\Theta} \tag{2.54}$$

Therefore, we have

$$d\epsilon = \mathbf{B}_0^e d\mathbf{u}^e + \mathbf{A}d\boldsymbol{\Theta} \tag{2.55}$$

The matrix $\boldsymbol{\Theta}$ can be written as:

$$\boldsymbol{\Theta} = \begin{pmatrix} \frac{\partial}{\partial x}\mathbf{I}_3 \\ \\ \frac{\partial}{\partial y}\mathbf{I}_3 \\ \\ \frac{\partial}{\partial z}\mathbf{I}_3 \end{pmatrix}_{9\times 3} \mathbf{u} = \mathbf{T}\mathbf{u} \tag{2.56}$$

where $\mathbf{I}_3$ is the 3-dimensional identity matrix.

Apply (2.22) to (2.56), we have

$$\boldsymbol{\Theta} = \mathbf{T}\boldsymbol{\Phi}^e\mathbf{u}^e = \mathbf{G}^e\mathbf{u}^e \tag{2.57}$$

where

$$\mathbf{G}^e = \mathbf{T}\boldsymbol{\Phi}^e \tag{2.58}$$

By (2.55) and (2.58), we have

$$\mathbf{B}^e = \mathbf{B}_0^e + \mathbf{A}\mathbf{G}^e \tag{2.59}$$

Therefore we have the following expression for $\mathbf{R}^e(\mathbf{u}^e)$:

$$\begin{aligned}\mathbf{R}^e(\mathbf{u}^e) &= \textstyle\int_{\Omega^e} \mathbf{B}^{eT}\sigma d\Omega^e \\ &= \textstyle\int_{\Omega^e} \mathbf{B}^{eT}\boldsymbol{\Lambda}\epsilon d\Omega^e \\ &= \textstyle\int_{\Omega^e} \mathbf{B}^{eT}\boldsymbol{\Lambda}(\mathbf{B}_0^e\mathbf{u}^e + \frac{1}{2}\mathbf{A}\mathbf{G}^e\mathbf{u}^e)d\Omega^e \\ &= \mathbf{K}^e(\mathbf{u}^e)\mathbf{u}^e\end{aligned} \tag{2.60}$$

where

$$\mathbf{K}^e(\mathbf{u}^e) = \int_{\Omega^e} \mathbf{B}^{eT}\boldsymbol{\Lambda}(\mathbf{B}_0^e + \frac{1}{2}\mathbf{A}\mathbf{G}^e)d\Omega^e = \mathbf{K}^e + \frac{1}{2}\int_{\Omega^e} \mathbf{B}^{eT}\mathbf{A}\mathbf{G}^e d\Omega^e \tag{2.61}$$

Equality (2.61) shows that the stiffness matrix is now *displacement-dependent*, because the matrix $\mathbf{A}$ is displacement dependent. The local stiffness matrix $\mathbf{K}^e(\mathbf{u}^e)$ has two parts: the first part $\mathbf{K}^{\mathbf{e}}$ is the same as the constant stiffness matrix derived from linear strain; the second term depends on the nodal displacement $\mathbf{u}^e$. Therefore $\mathbf{R}^e(\mathbf{u}^e)$ is a nonlinear function of $\mathbf{u}^e$. This makes the (2.37) a nonlinear system of differential equations of nodal displacement $\mathbf{u}^e$.

2.4.3 Element of General Shapes

In section 2.4.1, we have made a very strong assumption about the shape, size and position of the hexahedral element. In general, the element in the finite element mesh will have different sizes, different shapes and will be at different positions in the coordinate frame. Each "facet" of the hexahedral element is not even necessarily co-planar.

However we would like to point out that none of the derivations in chapter 2.4.1

requires a normalized element as in figure 2.6. The normalized element only buys us two simplification: the weight function has a simple form as defined in (2.21); and the volume has a regular shape (a cube) therefore the volume integration is relatively easier. Besides these two mathematical simplifications, the rest of the derivation applies to elements of any size, any shape with any weight functions. Namely, given weight functions $\phi_i^e(x, y, z)$, and an element Ω^e of arbitrary shape and size, all the derived equalities (2.37) to (2.41) still hold.

Theoretically we are done with the generalization of elementwise finite element equations to arbitrary element, including element of different types such as tetrahedra and element of different number of nodes. However there is a practical difficulty in terms of evaluating the volume integrals in equalities (2.37) to (2.41). It is difficult to evaluate the volume integral over an arbitrary volume. Furthermore, we may not even have the explicit form of the weight functions ϕ_i^e's if the element does not have a nice shape.

This problem can be easily solved by mapping a general hexahedral element to the normalized element in figure 2.6. We can then evaluate the volume integral over the normalized element as shown in section 2.4.1. Finally we can map the result back to the physical element in consideration using a Jacobian.

To simplify the presentation, we change the coordinate frame of the normalized element in figure 2.6 to (ξ, η, γ) and refer to it as the *natural coordinate* frame and refer to the (x, y, z) frame as the *physical coordinate* frame (Figure 2.7). We refer to the

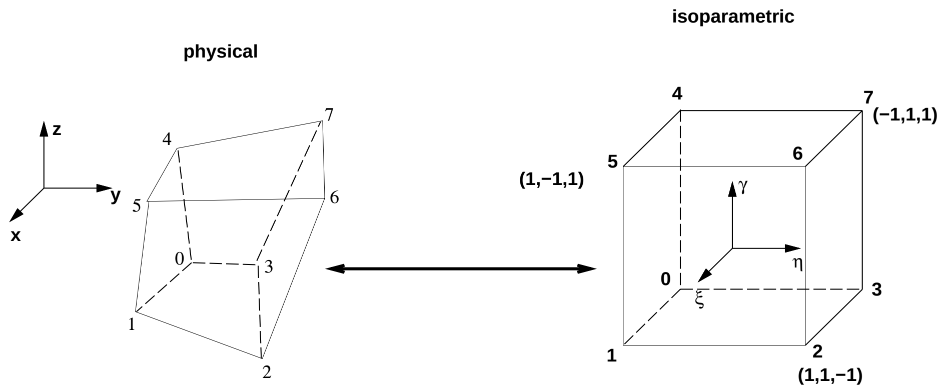


Figure 2.7: Isoparametric transformation.

element in the physical coordinate frame as the *physical element* and the normalized element in natural coordinate frame (as shown in figure 2.6) as the *isoparametric element*. The mapping between a physical element and the isoparametric element is as following:

$$\mathbf{u}^e = \sum_{i=0}^{7} \phi_i^e(\xi, \eta, \gamma)\mathbf{u}_i^e \tag{2.62}$$

where ϕ_i^e's are the same weight function as defined in (2.21).

It is easy to verify that (2.62) gives a one-to-one mapping between an physical hexahedral element and the isoparametric element. Therefore we can evaluate the volume integrals in (2.37) to (2.41) by transforming them into the natural coordinate frame using the Jacobians. For example, the element mass matrix can be evaluated as:

$$\begin{aligned} \mathbf{M}^e &= \int_{\Omega^e} \mathbf{\Phi}^{eT} \rho \mathbf{\Phi}^e d\Omega^e \\ &= \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \mathbf{\Phi}(\xi,\eta,\gamma)^T \rho \mathbf{\Phi}(\xi,\eta,\gamma) d\xi d\eta d\gamma \end{aligned} \tag{2.63}$$

where $\mathbf{\Phi}$ is defined in the natural coordinate frame.

Other volume integrals for general hexahedral element can be evaluated similarly.

For tetrahedral finite element mesh, there is a different isoparametric element to use in the natural coordinate frame. For details, users are referred to [30].

2.4.4 Global FEM Equations

We can indeed consider the entire body Ω as a "gigantic" element with n nodes. We can also assume that weight functions ϕ_i's are defined over the entire body with local support (figure 2.9), namely the weight function ϕ_i has the following property:

$$\phi_i(\mathbf{u}_j) = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \tag{2.64}$$

Figure 2.8 and 2.9 illustrate the relationship between the element-wise weight function and the global weight function defined over the entire body. The weight function N_i corresponding to node $\mathbf{u}_i$ has two parts. Each part is exactly the same as the corresponding local weight function within the element.

All the previous derivations still hold with the exception that we only have to consider surface stress on the surface of the body instead of on the boundary of each element. Therefore we have the following system of global equations:

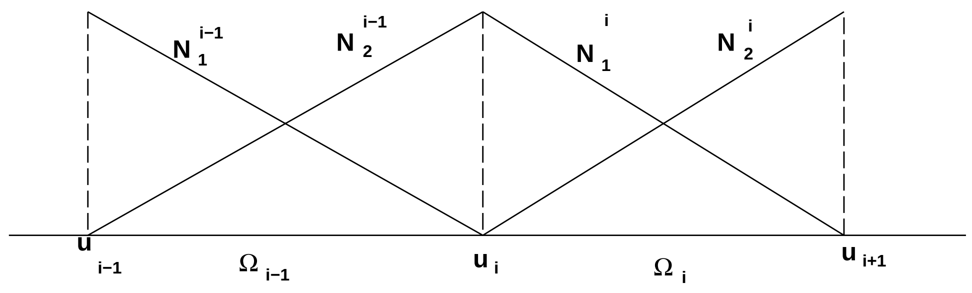


Figure 2.8: 1-dimensional example of element-wise weight functions.

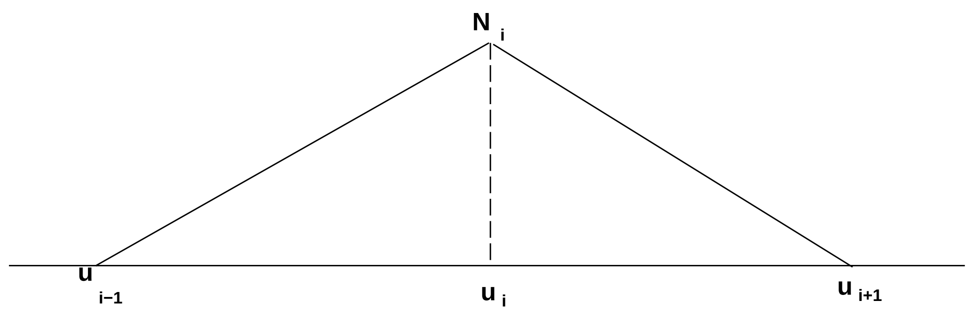


Figure 2.9: 1-dimensional example of Global weight functions.

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{R}(\mathbf{u}) = \mathbf{F} + \mathbf{q} \tag{2.65}$$

where

$$\mathbf{M} = \int_{\Omega} \mathbf{\Phi}^T \rho \mathbf{\Phi} d\Omega \tag{2.66}$$

$$\mathbf{D} = \int_{\Omega} \mathbf{\Phi}^T \mu \mathbf{\Phi} d\Omega \tag{2.67}$$

$$\mathbf{F} = \int_{\Omega} \mathbf{\Phi}^T \mathbf{b} d\Omega \tag{2.68}$$

and

$$\mathbf{R}(\mathbf{u}) = \int_{\Omega} \mathbf{B} \sigma d\Omega \tag{2.69}$$

Note that the equivalent global nodal force $\mathbf{q}$ is due to the stress and load on the boundary of the deformable body and it has the following form:

$$\mathbf{q} = -\int_{\partial\Omega} \mathbf{\Phi}^T \mathbf{t} d\Omega \tag{2.70}$$

where $\partial\Omega$ is the boundary of the deformable body; $\mathbf{t}$ is the surface traction on the boundary.

For convenience, we usually combine the $\mathbf{F}$ term and $\mathbf{q}$ term and rename $\mathbf{F} + \mathbf{q}$ the global external force vector $\mathbf{F}$. Therefore we have

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{R}(\mathbf{u}) = \mathbf{F} \tag{2.71}$$

In the equation (2.71), $\mathbf{u}$ is the $3n$-dimensional nodal displacement vector; $\dot{\mathbf{u}}$ and $\ddot{\mathbf{u}}$, the respective velocity and acceleration vectors; $\mathbf{F}$, the external force vector; $\mathbf{M}$, the $3n \times 3n$ mass matrix; $\mathbf{D}$, the damping matrix; and $\mathbf{R}(\mathbf{u})$, the internal force vectors due to deformation. n is the number of nodes in the FEM model [50].

2.5 Implementation

In terms of implementation, we actually still do the computation locally on each element and then assemble the elementwise matrices into global matrix. For example, we compute $\mathbf{M}^e$ for each element Ω^e and then assemble them into the global mass matrix $\mathbf{M}$. This assembly is valid due to the fact that volume integral over the entire body equals to the sum of that over each element, and that the global weight functions equal to the corresponding local weight function within each element.

The construction of (2.71) requires evaluating volume integrals over each element Ω^e. In general, such a volume integral cannot be evaluated exactly in closed form. We have to evaluate it using numerical integration. In particular, we apply Gauss Quadrature [50] to evaluate the volume integrals.

2.6 Special Cases

We have derived the system of nonlinear differential equations (2.71) for the dynamic behavior of large global deformations. In this section, we would like to point out two special cases, which are familiar to many readers.

One special case is the system of equations for static deformations. It is obtained by setting the nodal accelerations and nodel velocities to zeros. This leads to the following system of equations for static global deformation;

$$\mathbf{R}(\mathbf{u}) = \mathbf{F} \tag{2.72}$$

Another special case is the small deformations. When deformations are small or infinitesimal, we can use the linear strain (2.11) and (2.12). This leads to the following linear system of differential equations for the dynamic behavior of small deformations:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{F} \tag{2.73}$$

The corresponding linear system for static small deformations is:

$$\mathbf{K}\mathbf{u} = \mathbf{F} \tag{2.74}$$

2.7 Conclusion

In this chapter we have presented how to model the dynamic behavior of large global deformations. In next chapter, we will discuss how to solve the *nonlinear* system of differential equations (2.71). In particular, we will discuss how to solve it in real-time.

Chapter 3

Real-time Simulation of Dynamic Global Deformations

In chapter 2, we have derived the nonlinear system of differential equations (2.71) for large global deformations. For clarity of the presentation, let us recall that system of nonlinear equations here:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{R}(\mathbf{u}) = \mathbf{F} \tag{3.1}$$

In this chapter, we discuss how to solve this nonlinear system. Especially we focus on how to solve it in *real-time* for a finite element mesh of reasonable size.

First of all, we will discuss the issue of implicit integration versus explicit integration in chapter 3.1. We argue that implicit integration along the time-dimension makes real-time performance impossible for even problems of moderate size. Therefore we choose explicit integration over implicit integration. In chapter 3.1.1, we show that the computational speed is dramatically improved by using a diagonal mass matrix approximation with the explicit integration scheme.

Secondly, we discuss how we handle collision efficiently. A traditional penalty method usually slows down the system dramatically because the integration time step has to be reduced to prevent object inter-penetration when they are close to each other. In a virtual surgical environment, objects are always close to each other. This makes the penalty method inappropriate. In chapter 3.2, we will introduce a very efficient collision handling scheme which requires only a nominal computational cost in addition to a collision-free dynamic simulation step.

Finally, we discuss the issue of the complexity of the model itself in chapter 3.3. One primary difficulty associated with deformable object simulation in 3D is that the model itself is one order of magnitude more complex than 2D objects, such as clothes. By *one order of magnitude*, we mean the size of the 3D model is $O(n)$ larger than that of a 2D object, where n is the number of nodes in each principle direction. For example, a 2D finite element mesh has size of $O(n^2)$ with n nodes in each principle direction. However a 3D finite element mesh will have a size of $O(n^3)$ with n nodes in each principle direction. To overcome this higher complexity in the size of FEM mesh of a 3D deformable body, we propose *graded meshes* in chapter 3.3 and discuss their properties.

Part of the material presented in this chapter is published by Zhuang and Canny in [46].

3.1 Integration in the Time Dimension

In general there is no closed-form solution to the system of differential equations (3.1). We have to solve it approximately by numerically integrating along the time dimension. Namely given all the information up to the current time step t_n, we have to solve the following system for time step t_{n+1} repeatedly:

$$\mathbf{M}\ddot{\mathbf{u}}_{n+1} + \mathbf{D}\dot{\mathbf{u}}_{n+1} + \mathbf{R}(\mathbf{u}_{n+1}) = \mathbf{F}_{n+1} \tag{3.2}$$

Note that at time t_{n+1}, the information at all the previous time-steps, $t_0, \ldots, t_n$, are considered as known. Namely $\mathbf{u}_i$, $\dot{\mathbf{u}}_i$ and $\ddot{\mathbf{u}}_i$ are known for $i = 0, \ldots, n$.

There are many different integration schemes. In this thesis, we will discuss the issue of explicit integration versus implicit integration using Newmark recurrence scheme ([50, 30]).

Newmark recurrence scheme is a single-step integration method. Namely it only uses the known values at time t_n to solve (3.2), while ignoring the history before t_n. Newmark recurrence scheme is as following:

$$\begin{aligned} \mathbf{u}_{n+1} &= \mathbf{u}_n + \triangle t_{n+1}\dot{\mathbf{u}}_n + \tfrac{1}{2}\triangle t_{n+1}^2 \ddot{\mathbf{u}}_{n+\gamma} \\ \dot{\mathbf{u}}_{n+1} &= \dot{\mathbf{u}}_n + \triangle t_{n+1}\ddot{\mathbf{u}}_{n+\alpha} \end{aligned} \tag{3.3}$$

where

$$\ddot{\mathbf{u}}_{n+\theta} = (1-\theta)\ddot{\mathbf{u}}_n + \theta\ddot{\mathbf{u}}_{n+1} \tag{3.4}$$

and

$$\triangle t_n = t_n - t_{n-1} \tag{3.5}$$

By substituting (3.3) into (3.2), we get a system of equations of only $\ddot{\mathbf{u}}_{n+1}$. For convenience, we represent this system as:

$$\mathbf{\Gamma}(\ddot{\mathbf{u}}_{n+1}) = 0 \tag{3.6}$$

Note that when $\gamma \neq 0$, (3.6) is a nonlinear system of $\ddot{\mathbf{u}}_{n+1}$ because $\ddot{\mathbf{u}}_{n+1}$ appears in the nonlinear term $\mathbf{R}(\mathbf{u}_{n+1})$. This leads to an implicit algorithm, namely at each time step, we have to solve a nonlinear system of $\ddot{\mathbf{u}}_{n+1}$.

Solving a nonlinear system requires a Newton iteration type of algorithm. Namely at each iteration, we have to compute the differential matrix $\frac{\partial \mathbf{\Gamma}}{\partial \ddot{\mathbf{u}}_{n+1}}$ and then solve a different linear system at each iteration. Although we can avoid the cost of inverting a different *differential matrix* at each iteration by applying an iterative solver, such as conjugate gradient solver, the cost of computing the differential matrix alone at each iteration is computationally prohibitive. We have also experienced slow convergence of the conjugate gradient methods. Furthermore, we usually have to go through multiple iterations at each time step t_n. Therefore, an implicit integration algorithm makes real-time simulation impossible for an FEM mesh of even moderate size. Our experiment shows that we can only simulate fewer than half a dozen elements in real-time if we apply this type of integration algorithm.

This leads us to exploring the application of explicit algorithms. When $\gamma = 0$, the unknown $\ddot{\mathbf{u}}_{n+1}$ does not appear in the nonlinear term $\mathbf{R}(\mathbf{u}_{n+1})$. This makes (3.6) a linear system of $\ddot{\mathbf{u}}_{n+1}$. In particular, we choose the central difference method with $\alpha = \frac{1}{2}$. This leads to the following equalities:

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \dot{\mathbf{u}}_n \triangle t_{n+1} + \frac{1}{2}\ddot{\mathbf{u}}_n \triangle t_{n+1}^2 \tag{3.7}$$

$$(\mathbf{M} + \frac{1}{2}\triangle t_{n+1}\mathbf{D})\ddot{\mathbf{u}}_{n+1} = \mathbf{F}_{n+1} - \mathbf{R}(\mathbf{u}_{n+1}) - \mathbf{D}(\dot{\mathbf{u}}_n + \frac{1}{2}\ddot{\mathbf{u}}_n \triangle t_{n+1}) \tag{3.8}$$

$$\dot{\mathbf{u}}_{n+1} = \dot{\mathbf{u}}_n + \frac{1}{2}(\ddot{\mathbf{u}}_n + \ddot{\mathbf{u}}_{n+1})\triangle t_{n+1} \tag{3.9}$$

Now we have converted the nonlinear system (3.1) to 3 linear systems (3.7), (3.8) and (3.9). The order of updating is (3.7), (3.8) and then (3.9). Note that (3.7) and (3.9) are simple algebraic expressions, which are computationally cheap to evaluate.

Before examining (3.8), we would like to point out that the choice of $\gamma = 0$ already avoids the need to solve a nonlinear system by Newton iterations. However we do not have an explicit algorithm unless the matrix $\mathbf{M} + \frac{1}{2}\triangle t_{n+1}\mathbf{D}$ is diagonal. Next we will discuss how to make the central difference scheme an explicit integration algorithm by diagonalizing $\mathbf{M} + \frac{1}{2}\triangle t_{n+1}\mathbf{D}$.

3.1.1 Concentration of Mass

In general the mass matrix **M** and damping matrix **D** are sparse matrix, but not diagonal. Therefore (3.8) requires solving a large sparse linear system. Furthermore, the time step $\triangle t_n$ is, in general, not a constant, therefore it is impossible to preprocess the system by computing the inverse (or the LU decomposition) of this large sparse matrix. Inverting a large matrix at each integration step makes real-time simulation impossible for any problem of reasonable size.

To achieve real-time performance, we approximate the distributed mass with concentrated masses by *lumping* the mass matrix ([49, 30]): each row vector in the mass matrix is replaced by a single value on the diagonal entry, which is equal to the sum of all values in the corresponding row vector.

At the first glance, this approximation may look unacceptable. Actually it is mathematically equivalent to a special type of numerical volume integration algorithm for linear element: *nodal Gauss quadrature.* Numerical volume integration basically means sampling the value of the integrand at one or multiple points within the integration volume and then approximate the volume integration with a weighted sum of the sampled values. Given a linear finite element, if we sample exactly at the nodes of the element, [50] shows that the mass matrix is automatically a diagonal matrix. Furthermore such a diagonal matrix is identical to the one obtained by lumping the original mass matrix. Hence the diagonal approximation of the mass matrix is simply an approximation with a low order numerical integration.

In our implementation, we actually do not construct the original mass matrix and then lump it to a diagonal matrix. Instead we construct it directly using nodal Gauss quadrature.

Intuitively we can also consider this diagonal mass matrix as approximating the mass with concentrated point mass at nodal points of the mesh. The original non-diagonal mass matrix in (2.66) is also an approximation of the inertia property of the continuum, including the total mass and moment of inertia. However this approximation still treats the mass as if it is distributed.

The diagonalization process is equivalent to approximating the mass continuum as concentrated masses at each nodal point of the mesh. By doing this, we basically convert the distributed mass to a particle system. At each integration step, each particle "behaves" independently of the other particles. The forces acted on each particle, at each instance of the simulation, consist of external forces, such as gravity, and internal forces exerted by the neighboring particles (nodes that share the same element). Unlike a mass-spring system, this particle system does not have an explicitly defined spring structure. Instead the equivalent internal forces on each particle (a node of the mesh) is modeled using elasticity, approximated by geometrically nonlinear finite element method.

Similarly we can also diagonalize the damping matrix $\mathbf{D}$. Note that the damping matrix defined by (2.67) has identical structure to the mass matrix given by (2.66).

We have just converted the matrix $\mathbf{M} + \frac{1}{2}\triangle t_{n+1}\mathbf{D}$ to a diagonal matrix. This simplifies (3.8) to a set of independent *algebraic* equations as following:

$$q^i \ddot{u}^i_{n+1} = f^i_{n+1} - r^i_{n+1} - d^i_{n+1} \tag{3.10}$$

where q^i is the $i-th$ component of the diagonalized $\mathbf{M} + \frac{1}{2}\triangle t_{n+1}\mathbf{D}$; $\ddot{u}^i_{n+1}$, f^i_{n+1}, r^i_{n+1} and d^i_{n+1} are the $i-th$ component of $\mathbf{u}_{n+1}$, $\mathbf{F}_{n+1}$, $\mathbf{R}(\mathbf{u}_{n+1})$ and $\mathbf{D}(\dot{\mathbf{u}}_n + \frac{1}{2}\ddot{\mathbf{u}}_n \triangle t_{n+1})$ respectively. Solving this system of equations requires no matrix inversion.

The diagonalization also makes the enforcement of all types of boundary conditions very simple. For natural boundary conditions, we specify the force and compute $\ddot{u}^i_{n+1}$. For essential boundary conditions, we simply ignore equation (3.10) and explicitly set the corresponding displacement and velocity to the given values.

It is worth pointing out that the critical time step for an explicit integration scheme is dictated by the largest stiffness in the material. This is why an explicit integration scheme is appropriate for soft tissues, which are "soft" in all directions (although not necessarily isotropic), while it is not appropriate for cloth simulation [4].

As for any explicit integration algorithm, stability is always a concern. According to [30], the central difference recurrence scheme is *conditionally stable* when the time step satisfies the following:

$$\triangle t \leq \frac{2}{\omega_{max}} \tag{3.11}$$

where ω_{max} is the maximum natural frequency of the dynamic system (3.1). Since we are simulating soft tissues, the natural frequency is usually large. Therefore we can use relatively large time steps to have stable simulations.

In our numerical experiments, we notice that the lumped mass matrix actually improve the stability of the dynamic system. [49] shows that lumping can even improve accuracy of some problem by error cancellation. It can be shown that in transient approximation the lumping process introduces additional dissipation of energy and this can help in cancelling out numerical oscillation.

3.2 Efficient Collision Handling

In this section we address another important issue: *how to proceed with the time-integration when collision occurs?*

Any integration strategy at the moment of collision has to ensure that the colliding objects do not inter-penetrate each other at the point of collision. This non-penetration requirement makes obvious physical sense from our real world experience.

The popular penalty methods [39, 38, 40] model the collision by adding an artificial spring of large stiffness at the point of collision. The stiffness of such a spring is often arbitrary because there is no mathematical foundation to its value. The programmer usually tries several different spring constants until the collision is visually satisfactory.

Furthermore the penalty method adds an extra stiff component to the dynamic system. In chapter 3.1, we have discussed the advantage of an explicit integration algorithm over an implicit one. One reason that it is appropriate to use explicit integration algorithm despite its potential instability is that we are only interested in soft deformable objects. The natural frequency of a soft object is relatively low, therefore we are able to stably simulate the dynamic behavior using relatively large time steps. The artificial spring in a penalty method is an extra stiff component in the dynamic system. A dynamic system is only as stable as its most stiff component. Penalty method requires that we reduce the time step at the moment of collision to meet the stability requirement of the artificial spring attached to the point of collision.

Our experiments have shown that the ratio between a collision free integration time step and that of a penalty collision is on the order of hundreds if not more. Thus even if we are able to simulate the dynamic behavior of a soft object when there is no collision, we lose the real-time performance when collisions occur, by using penalty methods.

This tempts us to develop new collision-handling methods that avoid adding extra artificial stiffness into the system. We will illustrate our collision-handling method, using a special case: collision between a rigid body and a single node of the finite element mesh of the deformable body. (figure 3.1). Later in this section, we will show that it is straightforward to extend this method to handle general collisions of deformable objects.

Consider the collision between a moving deformable body and a moving rigid body (figure 3.1). To simplify the discussion, we use the moving frame attached to the moving rigid body instead of the fixed world frame. Namely all quantities are relative

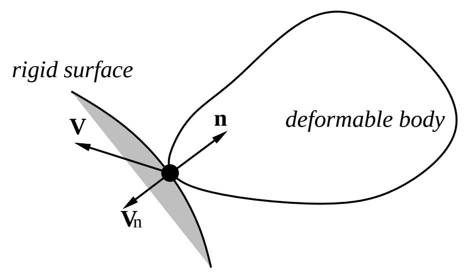


Figure 3.1: A flexible body collides with a rigid body.

to the moving rigid body. Assume that at time t_n, the node p on the deformable object, with *relative* velocity $\hat{\mathbf{v}}(p)_n$, is colliding with the rigid surface of outward normal $\hat{\mathbf{n}}$. The non-penetration constraint requires that the normal component of the relative velocity of point p drops to zero at the moment of collision in the moving frame. Unlike a rigid body collision, the flexible body will maintain contact with the rigid body for a nonzero period of time. We enforce the non-penetration constraint at node p by setting the normal component of $\hat{\mathbf{v}}(p)_{n+1}$ to zero as following:

$$\hat{\mathbf{v}}(p)_{n+1} = \hat{\mathbf{v}}(p)_n - (\hat{\mathbf{v}}(p)_n \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} \tag{3.12}$$

Let $\hat{\mathbf{a}}(p)_n$ and $\hat{\mathbf{a}}(p)_{n+1}$ be the acceleration of point p at time t_n and t_{n+1}, respectively. Apply equation (3.9) to point p, we have:

$$\hat{\mathbf{v}}(p)_{n+1} = \hat{\mathbf{v}}(p)_n + \frac{1}{2}(\hat{\mathbf{a}}(p)_n + \hat{\mathbf{a}}_{n+1})\triangle t_{n+1} \tag{3.13}$$

Solve (3.13) for $\hat{\mathbf{a}}(p)_{n+1}$, we get

$$\hat{\mathbf{a}}(p)_{n+1} = \frac{2\hat{\mathbf{v}}(p)_{n+1}}{\triangle t_{n+1}} - \frac{2\hat{\mathbf{v}}(p)_n}{\triangle t_{n+1}} - \hat{\mathbf{a}}(p)_n \tag{3.14}$$

Apply (3.7) to the point p at time t_{n+2}, we have:

$$\hat{\mathbf{u}}(p)_{n+2} = \hat{\mathbf{u}}(p)_{n+1} + \hat{\mathbf{v}}(p)_{n+1}\triangle t_{n+2} + \frac{1}{2}\hat{\mathbf{a}}(p)_{n+1}\triangle t_{n+2}^2 \tag{3.15}$$

Substituting (3.14) into (3.15), we have

$$\begin{aligned}\hat{\mathbf{u}}(p)_{n+2} &= \hat{\mathbf{u}}(p)_{n+1} + \hat{\mathbf{v}}(p_{n+1})\triangle t_{n+2} \\ &\quad +\tfrac{1}{2}\left(\tfrac{2\hat{\mathbf{v}}(p)_{n+1}}{\triangle t_{n+1}} - \tfrac{2\hat{\mathbf{v}}(p)_n}{\triangle t_{n+1}} - \hat{\mathbf{a}}(p)_n\right)\triangle t_{n+2}^2\end{aligned} \tag{3.16}$$

If we choose the time steps at time of collision such that $\triangle t_{n+2} = \triangle t_{n+1}$, we have

$$\begin{aligned}\hat{\mathbf{u}}(p)_{n+2} &= \hat{\mathbf{u}}(p)_{n+1} + \hat{\mathbf{v}}(p)_{n+1}\triangle t_{n+2} \\ &\quad +\hat{\mathbf{v}}(p)_{n+1}\triangle t_{n+2} - \hat{\mathbf{v}}(p)_n\triangle t_{n+1} - \tfrac{1}{2}\hat{\mathbf{a}}_n\triangle t_{n+1}^2 \\ &= \hat{\mathbf{u}}(p)_n + \hat{\mathbf{u}}(p)_{n+1} + 2\hat{\mathbf{v}}(p)_{n+1}\triangle t_{n+2} \\ &\quad -(\hat{\mathbf{u}}(p)_n + \hat{\mathbf{v}}(p)_n\triangle t_{n+1} + \tfrac{1}{2}\hat{\mathbf{a}}(p)_n\triangle t_{n+1}^2) \\ &= \hat{\mathbf{u}}(p)_n + 2\hat{\mathbf{v}}(p)_{n+1}\triangle t_{n+2} + \hat{\mathbf{u}}(p)_{n+1} - \hat{\mathbf{u}}(p)_{n+1} \\ &= \hat{\mathbf{u}}(p)_n + 2\hat{\mathbf{v}}(p)_{n+1}\triangle t_{n+2}\end{aligned} \tag{3.17}$$

By (3.12), we have

$$\hat{\mathbf{v}}(p)_{n+1} \cdot \hat{\mathbf{n}} = 0 \tag{3.18}$$

Therefore we have the following equality:

$$\hat{\mathbf{u}}_{n+2} \cdot \hat{\mathbf{n}} = \hat{\mathbf{u}}_n \cdot \hat{\mathbf{n}} \tag{3.19}$$

Equality (3.19) shows that the non-penetration constraint is enforced after two time steps, because there is no relative motion of the deformable body normal to the surface of the rigid body. Furthermore, equality (3.17) shows that the local tangential motion, on the surface around the point of collision, is still permitted. The guaranteed non-penetration and permitted local tangential motion is exactly what we need to model collisions.

This collision integration scheme can be generalized to collisions between deformable bodies and collisions that involve multiple point contacts. Multiple point collisions are modeled as a set of simultaneous single point collisions.

This collision-handling integration scheme can be considered a special case of impulse [3]. For rigid body collisions, an impulse requires extremely small time steps for numerical integration because the rigid body collision is considered to occur instantaneously. However, for deformable body collisions, the collision time is finite. By delaying the non-penetration constraint by two time steps, we are able to integrate the impulse using large time steps.

Our collision-handling strategy does not have to distinguish the case that the colliding deformable objects quickly bounce away from each other and that one sticks to or slides on the surface of the other. The bouncing collision, the sticking and sliding contacts, are handled by exactly the same collision integration constraint.

This collision constrain adds little extra cost to the dynamic simulation.

Finally we would like to point out that our collision-handling strategy does lose energy. When we set the velocity normal to the colliding surface to zero, the dynamic system loses some energy. However this loss of energy is too small to affect the simulation outcome in general. The energy loss decreases when we are using finer mesh and this energy loss converges to zero when the number of elements goes to infinity.

3.3 Graded Mesh

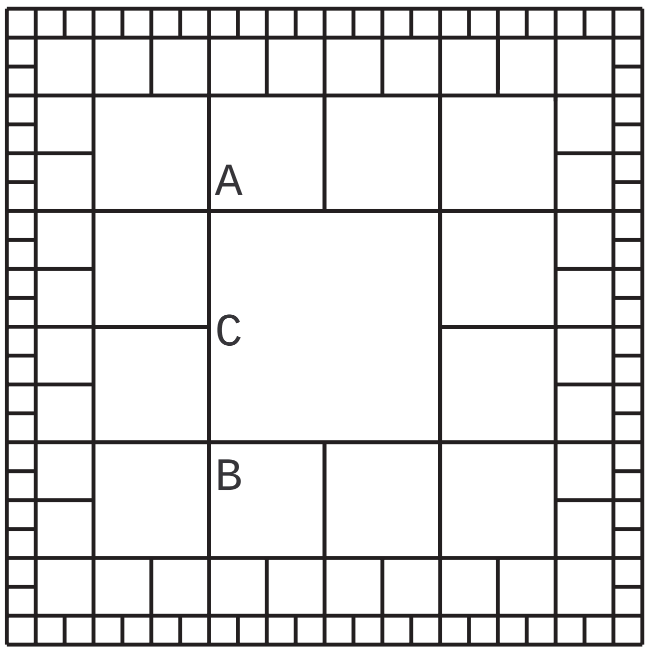


Figure 3.2: A 2D example of graded mesh.

While 2D finite element methods have great success in achieving real time performance in computer graphics applications, the computational cost is much higher for 3D applications, mainly due to the increase in the number of elements in the mesh. In a roughly uniform 2D finite element mesh, the number of elements is about $O(n^2)$, where n is the average number of elements in each principle direction. However a similar 3D mesh would have $O(n^3)$ elements, which leads to a much larger system of equations.

A mesh has to be fine enough to capture the relevant modes. To illustrate this, let us consider using finite element method to simulate a wave in a soft material, which is one special kind of deformation. When the spatial frequency of the wave increases, we need finer elements to simulate the wave. When the spatial frequency decreases, we need fewer elements. Modal analysis shows that deformation can be approximated by model composition, where each mode is corresponding to a deformation of a fixed spatial frequency.

In terms of computational efficiency, it would be desirable to reduce the number of elements in the finite element mesh while being able to satisfy the requirement of accuracy.

Nicolson [25] makes the following empirical observation based on many simulations:

Observation 1 *The cutoff spatial frequency of an object in response to external loads decreases faster than $1/d$ in terms of the distance d away from the surface of the object.*

This means that for a given error bound, we need finer elements on the surface and coarser elements away from the surface. Furthermore Nicolson's [25] result suggests that if the size of the element increases proportionally to d, we will lose little accuracy with respect to static forces exerted on the surface. Based on this observation, we propose using a graded mesh to model 3D objects. A 2D example of such mesh is shown in figure 3.2. Extension of this 2D example to a 3D hexahedral mesh is straightforward. Such a mesh reduces the complexity of 3D models from $O(n^3)$ to $O(n^2)$. Furthermore, the graded mesh has the property that the element size is proportional to the distance from the surface, hence the error is bounded by a constant.

In general, a graded mesh has a severe drawback despite its reduction of the mesh size. It is computationally costly to enforce the compatibility at element interface. To simplify the presentation, we discuss this using the 2D example in figure 3.2. However, note that the discussion applies to 3D hexahedra mesh as well.

For a mesh of linear quadralateral elements, the edge of each element is always a straight line. Therefore the node such as C in figure 3.2 is constrained by node A and B. Namely the displacement of C has to be such that A, B and C are always co-linear. We refer to a node such as C as a *geometrically constrained* node. The general solution is to use a Lagrangian multiplier, which expands the system and therefore adds extra computational cost to the simulation.

However since we have a diagonalized system (3.10), the compatibility at element interface can be easily enforced without a Lagrangian multiplier. For unconstrained nodes, such as A and B, we proceed with the computation as presented in section 3.1. For a constrained node, such as node C, we simply set its displacement to the average of that of A and B. This explicitly enforces the compatibility without any additional computational cost.

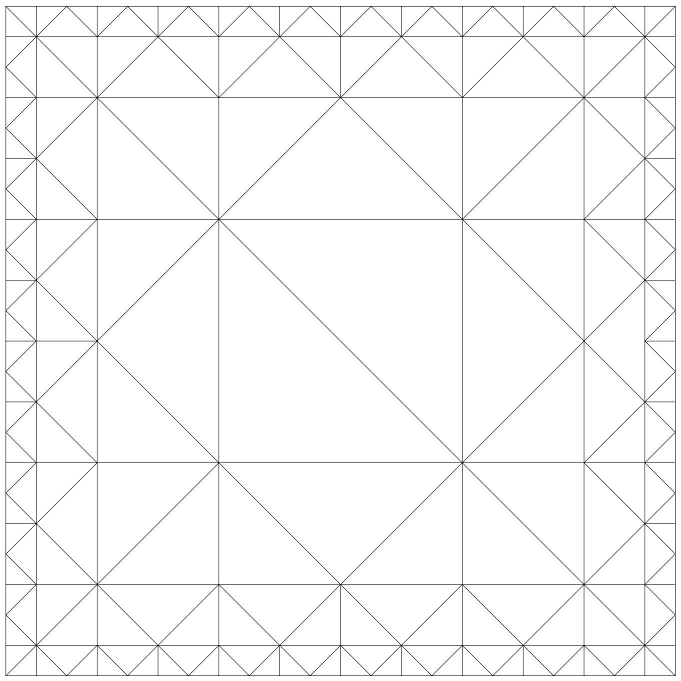


Figure 3.3: Triangular graded mesh.

Figure 3.3 shows a graded triangular graded mesh. Such a triangular graded mesh can also be extended to 3-dimensional tetrehedral mesh. The algorithms discussed in Hebert's ([17]) symbolic local refinement of tetrahedral mesh can be directly applied to generate a graded tetrahedral mesh in 3D.

For triangular mesh, the incompatibility issue on the element interface does not arise.

3.4 Discussion

In chapter 2, we have shown that simulation of global deformations requires solving the system of equations (3.1). In this chapter, we have presented how to solve system (3.1) in real-time.

First we show that it is impossible to solve (3.1) in real-time by implicit integration algorithms. Our experiments show that we can only simulate, in real-time, a mesh of fewer than half a dozen elements, if we apply implicit integration algorithms.

In order to have an explicit integration algorithm for real-time performance, we diagonalize the mass and damping matrix by row lumping. Row lumping is equivalent to gauss quadrature using nodal sampling.

Explicit integration schemes always raise the issue of stability. Fortunately, we are only interested in soft tissues. This allows a relatively large time step for stable simulations.

The diagonalized mass matrix combines the advantages of finite element methods and mass-spring systems. A mass-spring system is less accurate in terms of mathematical formulation. However it is cheap to solve a mass-spring system because it is a decoupled system from the very beginning. By diagonalizing the mass matrix in our FEM simulation of global deformations, we still have the accurate strain modeling provided by finite element methods but we also only have a decoupled system to solve.

Secondly, we have introduced an efficient collision-handling strategy in chapter

3.2, which removes a bottleneck in the real-time simulation of global deformations. Our collision-handling strategy enables us to simulate the collision with norminal extra cost.

Finally, we have proposed a graded mesh in 3.3 to reduce the size of the model itself. Asymptotically the graded mesh reduces the size of the mesh by $O(n)$, where n is the number of elements in each principle direction. If a uniform 3D finite element mesh has $O(n)$ elements in each principle direction, the size of the dense mesh is $O(n^3)$. A corresponding graded mesh with similar static accuracy only has a size of $O(n^2)$ instead.

Implicit integration algorithm only allows us to simulate a mesh of a few elements in real-time. After diagonalizing the mass matrix, the explicit integration algorithm allows us to simulate a mesh of more than 600 elements in real-time. By applying a graded mesh, 600 elements gives the same accuracy of a uniform mesh of more than 1000 elements.

We would like to close this chapter by pointing out one problem with diagonalizing the mass matrix and the damping matrix. Most tissue models are visco-elastic. The visco-elasticity will be reflected in the damping matrix. Although diagonalizing the mass matrix is acceptable, diagonalizing the damping matrix may lose some important material properties. We will address this problem in next chapter.

Chapter 4

Modified Nested Dissection

The bottleneck of Newmark scheme is solving equation (3.8). It requires inverting a large sparse matrix $\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D}$. This matrix is not a constant matrix because the time step $\triangle t_n$ may vary over time. Inverting a different large sparse matrix makes real time performance impossible.

To achieve real-time performance, Zhuang and Canny [46] approximated this matrix with its row-lumped diagonal matrix. This is equivalent to diagonalizing both the mass matrix $\mathbf{M}$ and the damping matrix $\mathbf{D}$. The diagonalization of $\mathbf{M}$ is acceptable because it still preserves the global inertia property of the object, although it does not preserve the local moment of inertia. The diagonalization of the damping matrix may lose important viscous elasticity property of the material. For simulations that require more physical realism, diagonalization of matrix $\mathbf{D}$ is not appropriate.

In this chapter, we propose a different treatment by preprocessing. The matrices

$\mathbf{M}$ and $\mathbf{D}$ are contants. The only variable is $\triangle t_n$. The time step $\triangle t_n$ depends on the stability requirement of the system and the collision handling requirement. Instead of approximating these two matrices as [46] does, we restrict the time step $\triangle t_n$ to a small set of values. Let T be the largest time step allowed, we define the restricted set of allowed time steps as $\{T/2^i | i = 0, 1, \ldots, m\}$. We choose the value m such that $T/2^m < T_{min}$, where T_{min} is the minimum time step in the worst case.

By restricting time steps to such a small set of values, we only have $m+1$ possible matrices needed for the entire simulation. We can therefore pre-compute the $m+1$ inverse matrices before the simulation begins.

Instead of precomputing the inverse of matrix $(\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D})$, we precompute its LU-factorization. Given the LU-factorization, solving equation (3.8) only requires back-substitution. The time for back-substitution is determined by the number of nonzeros in the LU-factorization of $(\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D})$. In section 4.1, we discuss how to reduce number of nonzeros in the LU-factorization.

Part of material presented in this chapter has been published by Zhuang and Canny in [48].

4.1 Nested Dissection

A typical finite element simulation has to solve a large sparse linear system of large number of nonzero entries. For example, a $10 \times 10 \times 10$ linear hexahedral mesh for 3D linear elasticity gives a sparse matrix of 3993×3993 with 242435 non-zeros,

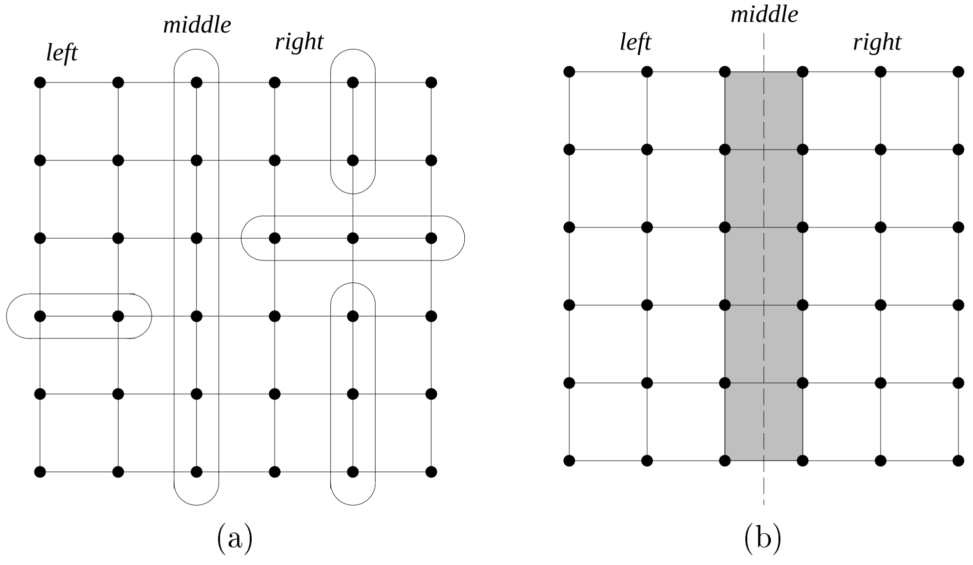
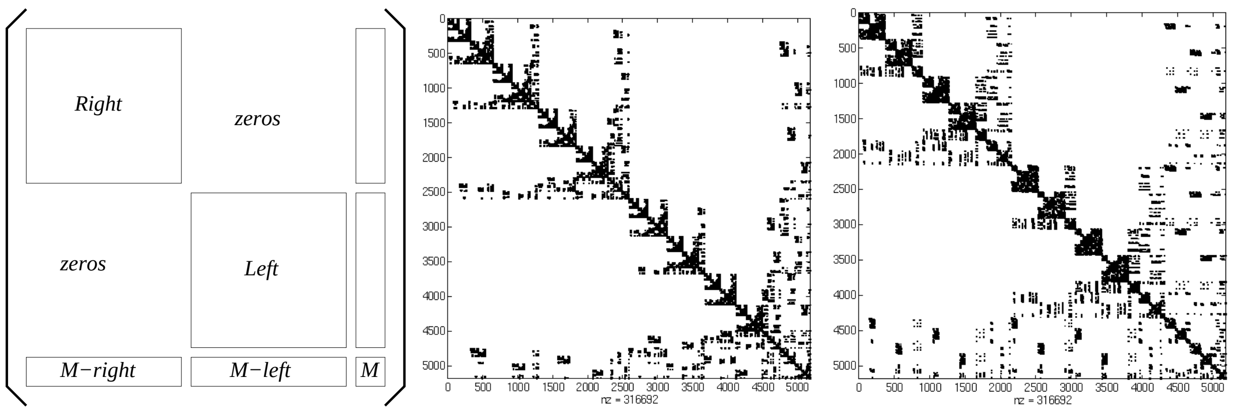


Figure 4.1: (a) The dissector in the regular nested dissection algorithm is a layer of nodes. (b) The simplified version of the modified nested dissection algorithm uses a layers of elements (shaded), cut by a plane (or a line in 2D), as the dissector.

which is about 1.5% of the size of a dense matrix with the same dimensions. To solve such a system efficiently, we have to avoid operating on zeros as much as we can. However how efficiently we can do so largely depends on the sparsity of the matrix: *the position of the nonzero entries.*

Given a finite element model of a physical problem, the values of non-zeros of $(\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D})$ are determined by the underlying model, while the positions of those non-zeros are determined by the indices of the variables. For convenience, let us denote matrix $(\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D})$ by $\mathbf{A}$. The entry (i, j) of $\mathbf{A}$ is nonzero if and only if the variable x_i and x_j are related. Given such a sparse matrix $\mathbf{A} = \mathbf{LU}$, the LU-factorization takes $O(\sum_j d_j)$ space and $O(\sum_j d_j^2)$ time, where d_j is the number of non-zeros in each column vector of $\mathbf{L}$ [13]. If we assume that no exact numerical



(a) (b) (c)

Figure 4.2: (a) The block structure of sparse matrix **A** after the first dissection. (b) The sparse matrix structure generated by regular nested dissection. (c) The sparse matrix using "Simple-Modified-Nested-Dissection".

cancellations can occur, **L** will have non-zeros below the diagonal everywhere that **A** does. We define *fills* to be the below-diagonal entries in which **L** is nonzero and the corresponding entry of **A** is zero.[1]

Different ordering of the row and column vectors of the matrix **A** has no effect on the underlining physical problem that we are solving. However it dramatically change the number of fills. In order to reduce the space and running time for LU-factorization and the time of the corresponding back-substitution, we would like to minimize the number of fills. Unfortunately finding the order that gives the smallest fills is an NP-complete problem [45].

For sparse matrix that arises from *regular* finite element mesh, George [11] proposed a heuristic called *nested dissection* for ordering the variables of the system such that it gives a small number of fills.

[1]**A** is symmetric.

Unfortunately an FEM mesh is often unstructured. In this section, we propose a modified nested dissection that works on any *unstructured* finite element mesh. Later we will compare the performance of our algorithm to Metis [19].

4.1.1 Modified Nested Dissection

For the simplicity of the presentation, let us consider a 2-dimensional finite element mesh, where each node has one degree of freedom.[2] A mesh of n nodes leads to a sparse matrix $\mathbf{A}$ of size $n \times n$. An entry (i, j) is nonzero if and only if the node i and j are in the same element.

For the mesh generated on regular grid (Figure 4.1(a)), the regular nested dissection [11] algorithm recursively divide the unordered nodes into 3 groups: *left, middle* and *right*. The group *middle* is just a set of nodes that completely separate *left* and *right*. This algorithm orders the nodes such that its sparse matrix has fractal sparsity as shown in figure 4.2(b). After the first step of recursion, we immediately get 2 blocks of zeros as shown in figure 4.2(a), because *left* and *right* are *not* directly related.

Unfortunately the regular nested dissection algorithm requires a finite element mesh defined on regular grid. For an unstructured mesh, it would be difficult to find *middle* to dissect the mesh. In computer graphics, most meshes are unstructured. This motivated us to extend regular nested dissection to unstructured finite element meshes.

[2]This can be easily generalized to multiple degrees of freedom and 3-dimensional meshes.

For simplicity of the presentation, we first introduce a simplified version of our modified nested dissection algorithm. Later we will discuss how to apply graph theory to improve it.

4.1.1.1 Simple Modified Nested Dissection

Instead of separating the set of unordered nodes using a layer of nodes, we "cut" the mesh by a axis-aligned plane. It is easy to compute the set of elements cut by this plane. We let *middle* be the set of unordered nodes in the elements cut by this plane and continue recursively as the regular nested dissection.

This *simple* modified nested dissection can be applied to any unstructured finite element meshes, including tetrahedral meshes. Figure 4.1(b) shows one step of such a dissection. It also leads to the block structure as shown in figure 4.2(a), except that the size of the M-block is bigger. At the end, we still get an ordering that gives a sparse matrix with fractal sparsity (Figure 4.2(c)). Due to the larger size of matrix M, the two "wings" ($M - left$ and $M - right$) are wider at each level.

The pseudo code of the modified nested dissection is as following:

```
**Simple-Modified-Nested-Dissection**($\mathbf{E}$, $top$, $perm$)
  **if** $\mathbf{E}.length = 0$ **then**
    **return**.
  **else if** $\mathbf{E}.length = 1$ **then**
    **for** each node $j$ in $\mathbf{E}[0]$ **do**
```

```
        **if** $perm[j]$ != $-1$ **do**
            $perm[j] = top$
            $top - -$
        **endif**
    **else**
        Find-dissector(**E**).
        $middle$ = all the elements cut by the dissector.
        $left$ = elements to the "left" of the dissector.
        $right$ = elements to the "right" of the dissector.
        **Modified-nested-dissection**($middle$, $top$, $perm$).
        **Modified-nested-dissection**($left$, $top$, $perm$).
        **Modified-nested-dissection**($right$, $top$, $perm$).
    **endif**
```

Before calling this function the first time, we initialize each entry of the permutation array *perm* to -1 (order unassigned), and we compute the centroid of each element. The function *Find-dissector* simply computes the 3 medians along x, y and z direction and compare the number of elements cut by the axis-aligned planes thru the medians and return as the dissector the plane with the minimum cut.

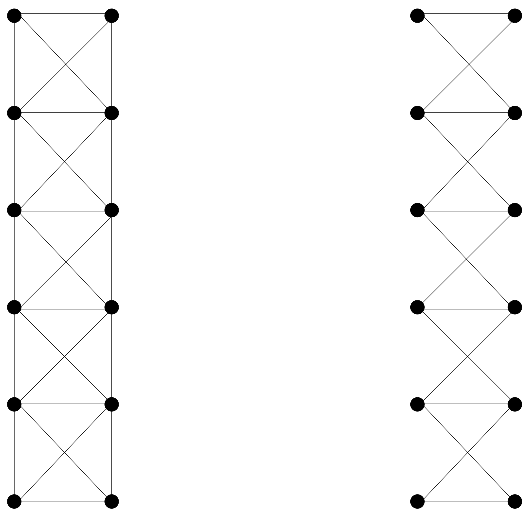


Figure 4.3: (a) The connectivity graph corresponding to the the elements intersected by the cutting plane. (b) The corresponding bipartite graph.

4.1.1.2 Improvement Using Graph Theory

We can actually improve the "Simple-Modified-Nested-Dissection" algorithm significantly by applying graph theory to "middle". Consider the corresponding connectivity graph of "middle" (Figure 4.3(a)). Recall that two nodes are connected if and only if they share the same element. A minimum vertex cover of this connectivity graph will be a separator, which is only about half the size of the separator in the "Simple-Modified-Nested-Dissection" algorithm.

In general, a vertex-cover problem is NP-complete [6]. Fortunately we can further convert the connectivity graph to a bipartite graph. The edges on the same side of the cutting plane are not significant because we only want to separate the "left" from the "right". Therefore, we only need to consider the edges crossing the cutting plane, which leads to the bipartite graph in figure 4.3(b).

Harary [16] shows that a vertex cover problem is equivalent to a maximum matching problem. Namely given the maximum matching of a graph, one can easily convert it to a minimum vertex cover in polynomial time. Cormen *et al* [6] shows how to find a maximum matching for a bipartite graph using Ford-Fulkerson method in time polynomial in $|V|$ and $|E|$, where $|V|$ is the number of vertices in the bipartite graph and $|E|$ is the number edges in the bipartite graph. Therefore we have a simple polynomial time algorithm to find a minimum vertex cover of the bipartite graph in figure 4.3(b). Such a minimum vertex cover is a small separator.

4.1.2 Running time

It takes $O(n)$ time to find the median given a list of n numbers [6]. It also only takes $O(n)$ time to find the elements intersected by the cutting plan. The time to separate the list is also $O(n)$. The depth of the recursion is apparently $O(\log n)$. Thus the total running time for the "Simple-Modified-Nested-Dissection" algorithm is $O(n \log n)$.

Cormen *et al* [6] shows that it takes $O(|V||E|)$ time to compute the maximum matching of a bipartite graph. Since $|E|$ has size $O(|V|)$ in our case, it takes $O(n^2)$ to find the minimum separator at each recursion. Harary [16] shows that it takes $O(|V|)$ time to compute the minimum vertex cover, given the maximum matching of a graph. Therefore the improved version of the modified nested dissection algorithm, using minimum vertex cover, takes $O(n^2 \log n)$ time.

Test	Size(n)	nnz	Fills			
			random	minimum degree	simple modified nested-dissection	nested dissection
1	192	6348	3636	1836	2034	1980
2	375	15285	25722	9342	10269	8586
3	648	30060	89199	31041	28872	26145
4	882	42384	196641	47286	50481	42462
5	1029	52131	254718	74880	73845	57537
6	1176	60360	332766	96381	88785	71685
7	1344	69888	487701	117900	112365	91971
8	1536	82956	657612	149652	153936	116514
9	1728	94266	813186	225801	182817	141741
10	1944	107118	1036521	261072	224280	175743
11	2187	123993	1361250	300312	286929	224973
12	2430	138870	1687527	347985	337383	264609
13	2700	155532		452574	410346	317898
14	3000	176700		550215	519804	385200
15	3300	195630		644067	595296	442053
16	3630	216588		787410	706131	514395
17	3993	242535		1100817	861705	610515
18	4356	266004		1262097	979875	684378

Table 4.1: Number of fills for different ordering.

4.1.3 Numerical Experiments

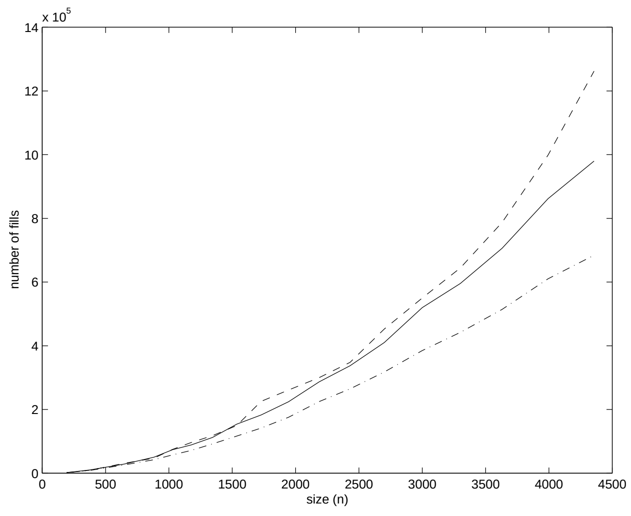
In order to measure the performance of the modified nested dissection algorithm, we compare its fills and LU-factorization time with that of regular nested dissection and that of *minimum-degree* algorithm [21, 12]. All the matrices are derived from a 3-dimension finite element mesh of linear hexahedral elements. The comparison of number of fills is listed in table 4.1 and that of the LU-factorization time is listed in table 4.2. In both tables, n is the number of variables (the dimension of the matrix), and nnz is the number of non-zeros in the original matrix. Recall that "fill" is the

Test	Size(n)	nnz	LU factorization time (seconds)			
			random	minimum degree	simple modified nested-dissection	nested dissection
1	192	6348	0.07	0.05	0.04	0.04
2	375	15285	0.55	0.18	0.17	0.17
3	648	30060	3.38	0.70	0.58	0.53
4	882	42384	10.66	1.07	1.11	0.87
5	1029	52131	15.31	1.98	1.77	1.23
6	1176	60360	22.75	2.67	2.15	1.54
7	1344	69888	40.08	3.54	2.93	2.02
8	1536	82956	61.19	4.65	4.44	2.80
9	1728	94266	86.39	8.86	5.39	3.56
10	1944	107118	123.27	9.88	7.00	4.52
11	2187	123993	192.59	11.31	9.99	6.05
12	2430	138870	274.96	14.15	11.87	7.45
13	2700	155532		21.49	14.91	9.39
14	3000	176700		28.49	21.20	12.51
15	3300	195630		35.26	24.78	15.20
16	3630	216588		46.60	31.96	19.76
17	3993	242535		87.84	41.00	23.56
18	4356	266004		94.44	47.41	27.31

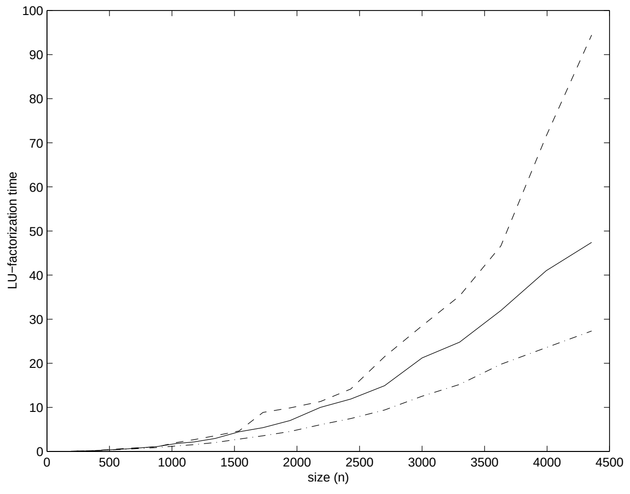
Table 4.2: Time measured on HP9000/715.

number of additional non-zeros in the LU-decomposition.

Minimum-degree ordering is an alternative ordering proposed to handle general matrix. It is an greedy algorithm that does the ordering directly on the connectivity graph defined by the matrix. Our numerical experiments show that even the simplified version of our modified nested dissection algorithm has an apparent advantage over the minimum-degree ordering, in both space and running time. Our "Simple-Modified-Nested-Dissection" algorithm produces an order that has less fills, than the minimum-degree ordering, in 17 of 18 tests, while it has a better LU-factorization time in all 18 test. The modified nested dissection algorithm using vertex cover has performance almost the same as regular nested dissection applied on meshes defined over regular grid. For this reason the numbers for the minimum vertex cover algorithm are not listed in the table. The result is plotted in figure 4.4.



(a)



(b)

Figure 4.4: The result for minimum degree ordering is plotted in dashed line, Simple-Modified-Nested-Dissection in solid line, and the standard nested dissection in dash-dot line. (a) Number of fills. (b) Running time.

We have also run our 18 tests using Metis3.0.3, developed by Karypis and Kumar [19]. Metis3.0.3 first converts the finite element mesh to a graph and then performs graph partition to obtain the fill-reduction reordering of the variables in the mesh. In our comparisons, our modified nested dissection algorithms outperform Metis3.0.3 in all 18 tests. This shows that geometry of the mesh itself gives better heurstics than its connectivity graph.

While the modified nested dissection algorithm using vertex cover performs much better in both time and space, the simplified version of the modified nested dissection algorithm requires significantly less time for LU-factorization while only having slightly less fills than the minimum-degree ordering. This shows that the modified nested dissection leads to better sparsity: non-zeros are more optimally positioned in the matrix.

4.2 Conclusion and Discussion

In chapter 2 and chapter 3, we presented the nonlinear FEM system to model and simulate global deformations. It is in general too computationally expensive to solve such a nonlinear FEM system in real time. In order to achieve real-time performance, without diagonalizing the mass and damping matrix as in chapter 3.1.1, we pre-compute the LU-factorization of a small number of large sparse matrices. Such preprocessing is possible because we restrict the time steps to a small set of values. Our experiments show that usually we only need no more than 3 different values for time steps.

To reduce the time and space for LU-factorization and the time of back-substitution, we apply nested dissection to reorder the vertices in the finite element mesh. Such a reordering does not change the physical model that we are simulating. But it dramatically reduces the number of nonzeros in the LU-factorization.

We modifiy the regular nested dissection algorithm so that it works on unstructured finite element mesh. The ordering produced by the modified nested dissection algorithm using vertex cover has almost the same performance as regular nested dissection algorithm in both space and time. The "Simple-Modified-Nested-Dissection" ordering takes 30% to 50% more time for the LU-factorization than the regular nested dissection, however it is more general than the regular nested dissection: *it is able to handle any unstructured finite element mesh*. This performance gap has been closed when we improve the algorithm by applying vertex cover.

Our current implementation simply uses a cutting plane and separates the mesh using the vertex cover of the bipartite graph derived from the elements intersected by the cutting plane. There seem to be a better algorithm using a topological sweep [8]. A topological sweep may intersect a smaller set of elements of the mesh. This may lead to a modified nested dissection algorithm with better asymptotic bound.

Notice that the non-constant matrix $(\mathbf{M}+\frac{1}{2}\triangle t_n\mathbf{D})$ only has one degree of freedom $\triangle t_n$, while both $\mathbf{M}$ and $\mathbf{D}$ are constant matrices. This suggests that it is possible to efficiently compute an approximate inverse by interpolating for any $\triangle t_{min} < \triangle t_n < \triangle t_{max}$, if we preprocess the exact inverse for $(\mathbf{M}+\frac{1}{2}\triangle t_{min}\mathbf{D})$ and $(\mathbf{M}+\frac{1}{2}\triangle t_{max}\mathbf{D})$. We are currently studying this approach and its error bound.

Chapter 5

Haptic Interaction with Global Deformations

Force feedback coupled with a real-time physically realistic graphic display provides a human operator with an artificial sense of presence in a virtual environment. Furthermore, it allows a human operator to interact with the virtual environment through "touch". In this chapter, we propose a haptic simulation system that allows a human operator to perform real-time interaction with soft 3D objects that go through large global deformations. We model and simulate such a global deformation using *geometrically nonlinear* finite element methods, as discussed in chapter 2 and chapter 3. We also introduce an efficient method that computes the force feedback, in real-time, by simulating the collision between the virtual "*proxy*" and the deformable object. To perceptually satisfy a human operator, haptics requires a much higher update frequency (at least 1000Hz) than graphics. We update the graphics using full simulation and interpolate the fully simulated states at a higher frequency to render haptics. The interpolation is made possible by intentionally delaying the display (both graphics and haptics) by one full simulation cycle.

Part of the material in this chapter has been published by Zhuang and Canny in [47].

5.1 Introduction

The word *haptic* refers to something that is associated with the sense of touch. In a haptic simulation, to achieve a virtual sense of touch, the human operator interacts with an active mechanical device, called a *haptic display.* A haptic simulation system includes the following essential elements: a human operator, a haptic display, a graphic display and a virtual environment. The human operator makes physical contact with the haptic display. The coupling of real-time graphic and haptic displays provides the human operator an artificial sense of kinesthetic presence in a virtual environment. Furthermore, it allows a human operator to interact with the virtual environment through "touch".

A haptic display can take on many forms, most commonly a robotic manipulator with the ability to exert forces on a human. One of the most successful haptic displays is the Phantom. Other haptic displays include Salisbury Hand, mini-WAM, Shah finger, etc. [36].

Applications of haptic simulation include, but are not limited to, surgical training, physical rehabilitation, computer-aided design, and entertainment. A haptic simulation system can also enhance a human operator's ability to perform certain tasks [9, 26].

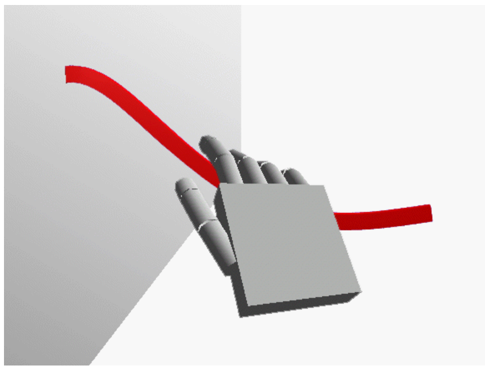


Figure 5.1: A virtual hand interacting with a soft cantilever beam.

In this chapter, instead of addressing a specific application of haptic simulation, we address the bottleneck problem of real-time interaction with large global deformations of 3D soft objects, with physically realistic force feedback. By *global deformations*, we mean deformations, such as twisting and bending of an object, which involve the entire body, in contrast to poking and squeezing, which involves a relatively small region of the deformable object.

To simplify the control of the haptic device, we represent our haptic device by a virtual *proxy* (section 5.3). The force feedback exerted on the human operator by the haptic display is simulated by the collision between this virtual proxy and the deformable object (section 5.4.1). We model and simulate the global deformations of 3D objects using a displacement based *nonlinear finite element method* (FEM) (chapter 2).

While real-time graphic display requires an update rate of only 30Hz, stable haptic display requires an update rate of at least 1000Hz. In this chapter, we propose a simple interpolation scheme (section 5.4.2) that can interpolate force feedback at the required high frequency, while the virtual environment is only simulated at a lower frequency.

5.2 Related Work

Our work involves both real-time realistic visual effects and haptic effects. Computer graphics and haptics share the same goal of evoking the sensation of objects by appropriate sensory stimulation. Graphic rendering techniques seek to provide the perception of an object's color, geometry, surface texture, etc., by rendering an appropriate image. Haptic rendering techniques seek to provide the human operator with the appropriate force feedback to "feel" the geometry, surface and material property of the object.

In the computer graphics domain, our work of modeling and simulating a deformable object falls into the realm of physically based modeling. The related works are discussed in chapter 1.1.

On haptic displays, Salisbury [36] reviews the history of haptic devices. Srinivasan and Salisbury [37] reviews the issues and challenges in haptic feedback. Mark *et al* [22] describes solutions of adding force feedback for static models into computer graphics systems. Adachi *et al* [1] addresses the problem of haptic display of curved surfaces using an intermediate representation. Velula and Baraff [41] discuss the integration of force feedback into their rigid body dynamics simulation system [2]. Minsky *et al* [23] addresses various haptic feedback techniques for surface textures. Ruspini *et al* [34, 35, 33, 31, 32] applies robotic motion planning techniques to haptic interactions in a virtual environment. Furthermore, they describe a new haptic rendering library HL, which enables graphics programmers to add haptics into a graphic virtual environment.

5.3 Haptic Model Overview

The haptic simulation includes a human operator, a haptic device (such as a PHANToM manipulator, a CyberGrasp glove, etc.), a graphic display, and a virtual environment. The human operator makes *physical* contact with the haptic device through pushing, grasping or some other mechanism. The haptic device provides the operator with a kinesthetic sense of presence in the virtual environment through appropriate force feedback.

In this chapter, we describe a system that allows users to virtually interact with objects exhibiting large deformations. The real-time haptic feedback is coupled with a real-time graphic display.

To simplify the control of the haptic device, we simulate the force feedback using a virtual *proxy* similar to that of Ruspini *et al* [35, 33], and the "god object" of Zilles and Salisbury [51]. Our virtual proxy is different from that of Ruspini *et al* [35, 33] because our proxy's motion is guided by the dynamic simulation. Namely upon collision, the motion of the proxy is guided by physics instead of a local minimization.

In chapter 2 we have described how we model dynamic global deformations, using geometrically nonlinear finite element methods. In chapter 3 we have discussed how to simulate such a dynamic global deformation in real-time. In particular, since real-time performance is crucial to force feedback, we choose to diagonalize the mass and damping matrix as in chapter 3.1.1. In section 5.4, we will describe how the simulation can be used to provide haptic feedback to the human operator through the haptic device. In section 5.4.2, we will describe how we display graphics and haptics at different frequencies.

5.4 Haptic Display

We provide force feedback by simulating the collision between the deformable object and the virtual proxy. To provide stable haptic feedback within limited computational power, we run full FEM simulation at a low frequency. The required high frequency haptic feedback is obtained by interpolating between the simulated states.

5.4.1 Collision with the Proxy

A virtual proxy is a rigid object with a piecewise differentiable surface. Usually a proxy has a very regular shape, such as a sphere or a cylinder. However in this section, we will discuss collisions using a proxy of general shape.

The popular penalty methods [39, 38, 40] model the collision by adding an artificial spring of large stiffness at the point of collision. This stiff spring requires tiny integration time steps to stably simulate a collision. Various experiments show that the ratio between a collision free integration time step and that of a penalty collision is on the order of hundreds if not more.

In chapter 3.2, we have developed a new collision-handling strategy that avoid adding extra artificial stiffness into the system. That strategy can be directly applied to simulating the collision between a rigid proxy and the finite mesh of a deformable body.

First we consider the special case: collision between a rigid proxy and a single node of the mesh (figure 5.2). Later in this section, we will show that it is straightforward to extend this method to handle general collisions between the virtual proxy and the deformable body .

Consider the collision between a moving deformable body and a moving rigid virtual proxy (figure 5.2). To simplify the discussion, we use the moving frame attached to the proxy instead of the fixed world frame. Namely all quantities are relative to the moving proxy. Assume that at time t_n, the node p on the deformable object,

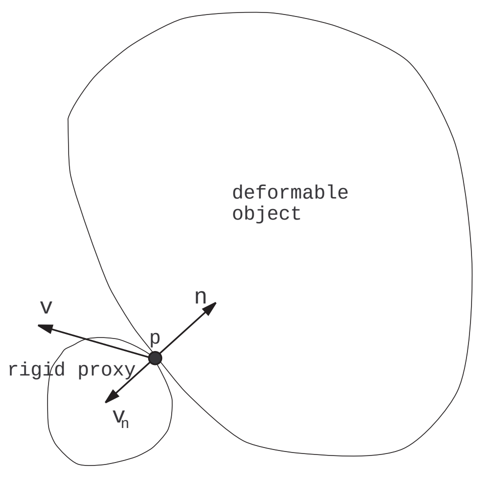


Figure 5.2: A rigid proxy collides with a soft object.

with *relative* velocity $\hat{\mathbf{v}}(p)_n$, is colliding with the rigid surface of outward normal $\hat{\mathbf{n}}$. The non-penetration constraint requires that the normal component of the relative velocity of point p drops to zero at the moment of collision in the moving frame. Unlike a rigid body collision, the flexible body will maintain contact with the rigid body for a nonzero period of time. We enforce the non-penetration constraint at node p by setting the normal component of $\hat{\mathbf{v}}(p)_{n+1}$ to zero as following:

$$\hat{\mathbf{v}}(p)_{n+1} = \hat{\mathbf{v}}(p)_n - (\hat{\mathbf{v}}(p)_n \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} \tag{5.1}$$

If we choose $\triangle t_{n+1} = \triangle t_{n+2}$, by results derived in chapter 3.2, we have

$$\hat{\mathbf{u}}_{n+2}(p) \cdot \hat{\mathbf{n}} = \hat{\mathbf{u}}_n(p) \cdot \hat{\mathbf{n}} \tag{5.2}$$

This shows that the non-penetration constraint is enforced after two time steps, because there is no relative motion of the deformable body normal to the surface of the rigid proxy at node p.

Note that the choice of $\triangle t_{n+1} = \triangle t_{n+2}$ does not mean that the entire simulation has to use a constant time step. Indeed the simulation can still use variable time step. This constraint (choice) is only enforced at collision time.

Equality (3.14) gives the equivalent acceleration at point p. Then we can use equation (3.10) to compute the equivalent force exerted at point p of the deformable object, which is a force normal to the collision surface. The reaction force exerted on the virtual proxy has the same quantity as this force, but in the opposite direction. This equivalent force also enables us to compute the Coulomb friction and simulate a frictional collision, and provide friction feedback.

This collision integration scheme can be generalized to a general haptic interface. A general haptic interface involves multiple virtual proxies (for instance, a virtual hand), therefore multiple point contacts. Since the system is decoupled (chapter 3.1.1), such a collision is modeled as a set of simultaneous *independent* single point collisions.

Our approach is different from that of Ruspini *et al* [35, 33]. Instead of explicitly minimizing the distance between the current configuration and the goal configuration, upon collision, we let physics naturally guide the motion of the virtual proxy. The motion is more physically realistic, compared to that obtained by a purely geometric minimization.

5.4.2 Haptic Interpolation

While the graphic display of the global deformations requires an update rate of only 30Hz, the stable and smooth haptic display requires an update rate of at least 1000Hz. It is impossible to simulate the global deformation at such a high frequency, with a desktop computer. Note that each graphic frame usually requires multiple integration steps, because the explicit integration scheme has to be smaller than the critical time step to be stable. Therefore although the graphic display is at 30Hz, our system actually simulates the global deformations and the collision between the proxy and the deformable object at a slightly higher frequency[1]. To display haptics at 1000Hz or higher, we will interpolate the haptics between two simulated states using the necessary high frequency.

Given the simulated states at time t_n and t_{n+1}, it is straightforward to interpolate the haptics between them. Basically any interpolation scheme, such as a simple linear interpolation, will do. The problem is that at time t_n, we do not have the information about t_{n+1}.

Our proposed solution to this problem is that we simply delay the entire simulation display, both graphically and haptically, by one integration time step. This intentional time lag lasts a few miliseconds. For a virtual interaction with soft objects, such a small lag in time is within the tolerance of human perception. The advantage of such a time delay is that we have already simulated the state at time t_{n+1} when we display

[1]Each explicit integration step is a full simulation step.

the graphics and haptics at time t_n, which makes the haptics interpolation from time t_n to t_{n+1} straightforward.

5.5 Conclusions and Discussions

In this chapter, we proposed a haptic simulation system that allows a human operator to interact with 3D global deformations in real time. Due to the distortion associated with linear strain, we simulate the global deformation using geometrically nonlinear finite element methods. The nonlinear FEM formulation is derived from the application of the nonlinear exact strain (Chapter 2).

It is in general too expensive to solve such a nonlinear FEM system in real time. In order to achieve real-time performance, we diagonalize the mass matrix *approximately* (Chapter 3.1.1). This diagonalization is equivalent to converting the distributed mass to a particle system of concentrated mass.

Since a stable haptic display requires force computation at a much higher frequency than that required by real-time graphics, we proposed a simple interpolation technique by intentionally delaying the display (both graphic and haptic) by one full simulation cycle. This takes advantage of the fact that human perception tolerates a small delay of a few miliseconds. Such a delay turns a complicated extrapolation into a simple interpolation.

We do recognize the possibility of extrapolating haptics without time delay, by estimating a constant local stiffness. However this extrapolation is more computationally expensive than our proposed interpolation technique. The extra computational cost is due to the construction of the local stiffness matrix. Besides the computational cost, there will be either force discontinuity or displacement discontinuity between extrapolation steps.

Currently our system is able to produce real-time graphics and haptics for a mesh of several hundred vertices. We are experimenting with the relationship between the stiffness of soft objects and the maximum time delay that can be tolerated by human operators. Our experience suggests that the softer the object is, the longer delay the human operator can tolerate.

Chapter 6

Conclusion and Future Works

We presented a simulation system that simulates 3D global deformations in real-time. Due to the distortion associated with linear strain, we simulate the global deformation using geometrically nonlinear finite element methods (Chapter 2). The nonlinear FEM formulation is derived from the application of the nonlinear exact strains. To solve such a nonlinear system in real-time, we apply Newmark scheme and diagonalize the mass matrix to avoid matrix inversion (Chapter 3).

We implemented both statics and dynamics for elastic objects, using geometrically nonlinear FEM with both hexahedral and tetrahedral meshes. On a 400MHz Pentium II PC, a uniform mesh of 1331 elements needs about 0.11 seconds per time step. The graded mesh with the same accuracy needs only 0.06 seconds per step.

In some sense this approach combines the best of the linear FEM model and mass-spring model. A mass-spring model is inaccurate in its mathematical formulation, however it is cheaper to solve because it is a diagonal system from the very beginning, and it does not introduce any geometric distortion. Linear FEM model is more accurate in its mathematical formulation of material behaviors, but expensive to solve and has distortion for large motions and deformations. A diagonalized geometrically nonlinear FEM approach models the material behavior with more accuracy than a linear model and it is still cheap to solve and has no distortion.

We also introduced an efficient collision constraint (chapter 3.2). This constraint enables us to simulate the collisions with little extra computation, compared to a collision free simulation step. Our experiments show that this collision constraint handles collision much more efficient than penalty method.

The geometrically nonlinear FEM model (Chapter 2) and collision-handling integration scheme (Chapter 3.2) apply to any types of meshes. Although we presented a graded mesh in terms of a hexahedral mesh (Chapter 3.3), the asymptotic argument applies to tetrahedral meshes as well. Indeed it becomes simpler with a tetrahedral mesh because the issue of geometric compatibility at the element interface does not arise.

The more subtle material properties, such as visco-elasticity, is usually reflected in the damping matrix and the computation of the internal forces. This makes the diagonalization inappropriate for some applications, where it is essential to keep the original damping matrix. However it is in general too expensive to solve the nonlinear FEM system without diagonalization. In chapter 4, we presented an alternative approach to achieve real-time performance. We pre-compute the LU-factorization of a small number of large sparse matrices. Such preprocessing is possible because we restrict the time steps to a small set of values. Our experiments show that usually we only need no more than 3 different values for time steps.

To reduce the time and space for LU-factorization and the time of back-substitution, we apply nested dissection to reorder the vertices in the finite element mesh. Such a reordering does not change the physical model that we are simulating. But it dramatically reduces the number of nonzeros in the LU-factorization.

We modified the regular nested algorithm so that it works on un-structured finite element mesh. The modified nested dissection (using the minimum vertex cover) has a similar performance in terms of both time and space. However it is more general than the regular nested dissection: *it is able to handle any unstructured finite element mesh.*

Out current implementation simply uses a cutting plan and seperate the mesh using the minimum vertex cover of the corresponding bipartite connectivity graph. Another alternative to a cutting plane is a topological sweep [8]. A topological sweep may intersect a smaller set of elements than a cutting plane, which will in turn lead to a smaller separator. We are currently studying this alternative algorithm.

In chapter 5, we have proposed a haptic simulation system that allows a human operator to interact with 3D global deformations in real time.

Since a stable haptic display requires force computation at a much higher frequency than that required by real-time graphics, we proposed a simple interpolation technique by intentionally delaying the display (both graphic and haptic) by one full simulation cycle. This takes advantage of the fact that human perception tolerates a small delay of a few miliseconds. Such a delay turns a complicated extrapolation into a simple interpolation.

Currently our system is able to produce real-time graphics and haptics for a mesh of several hundred vertices. We are experimenting with the relationship between the stiffness of soft objects and the maximum time delay that can be tolerated by human operators. Our experience suggests that the softer the object is, the longer delay the human operator can tolerate.

Our current system only has geometric nonlinearity. In the future, in order to simulate more realistic material behaviors, we are going to extend out system to nonlinear material properties. However many of the techniques presented in this thesis still apply.

Bibliography

[1] Y. Adachi, T. Kumano, and K. Ogino. Intermediate representation for stiff virtual objects. *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 203–210, 1995.

[2] David Baraff. Interactive simulation of solid rigid bodies. *IEEE Computer Graphics and Applications*, 15:63–75, 1995.

[3] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. In *Computer Graphics: Proceedings of SIGGRAPH*, pages 303–308. ACM, 1992.

[4] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Computer Graphics: Proceedings of SIGGRAPH*, pages 303–308. ACM, 1998.

[5] David Chen. *Pump It Up: Computer Animation of a Biomechanically Basded Model of Muscle Using the Finite Element Method.* PhD thesis, MIT, 1992.

[6] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 10 edition, 1993.

[7] Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transcation on Visualization and Computer Graphics*, 5(1):62–73, January-March 1999.

[8] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[9] M. Finch, V. Chi, R. M. Taylor II, M. Falvo, S. Washburn, and R. Superfine. Surface modification tools in a virtual environment interface to a scanning probe microscope. *Proceedings of 1995 Symposium on Interactive 3D Graphics*, pages 13–18, April 1995.

[10] Y. C. Fung. *A First Course In Continuum Mechanics*. Prentice-Hall, Inc, 2nd edition, 1977.

[11] Alan George. Nested dissection of a regular finite element mesh. *SIAM Jounal of Numerical Analysis*, 10(2), 1973.

[12] Alan George and Joseph W. H. Liu. A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Transaction on Mathematical Software*, 6(3), September 1980.

[13] Alan George and Joseph W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Inc., 1981.

[14] Sarah F. Gibson and Brian Mirtich. A servey of deformable models in computer graphics. Technical Report TR-97-19, Mitsubishi Electric Research Laboratories, Cambridge, MA, November 1997.

[15] Phillip L. Gould. *Introduction to Linear Elasticity.* Springer-Verlag, 2nd edition, 1994.

[16] Frank Harary. *Graph Theory.* Addison-Wesley Publishing Company, 1972.

[17] D. J. Hebert. Symbolic local refinement of tetrahedral grids. *Jounral of Symbolic Computation*, 11, 1994.

[18] Doug L. James and Dinesh K. Pai. Artdefo: Accurate real time deformable objects. *Computer Graphics: Proceedings of Siggraph*, pages 65–72, August 1999.

[19] George Karypis and Vipin Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Technical report, University of Minnesota, Department of Computer Science / Army HPC Research Center, Minneapolis, MN 55455, November 1997.

[20] E. Keeve, S. Girod, P. Pfeifle, and B. Girod. Anatomy-based facial tissue modeling using the finite element method. *IEEE Visualization*, 1996.

[21] Joseph W. H. Liu. Modification of minimum-degree algorithm by multiple elemination. *ACM Transaction on Mathematical Software*, 11(2), June 1985.

[22] W. R. Mark, S. C. Randolph, M. Finch, J. M. Van Verth, and R. M Taylor II. Adding force feedback to graphics systems: Issues and solutions. *Computer Graphics: Proceeding of Siggraph*, pages 447–452, August 1996.

[23] M. Minsky, M. Ouh-Young, M. Steele, F. P. Jr. Brooks, and M. Behensky. Feeling and seeing: Issues in force display. *Computer Graphics: Proceedings of 1990 Symposium on Interactive 3D Graphics*, pages 235–243, 1990.

[24] G. Celniker nad G. Gossard. Deformable curve and surface finite elements for free form shage design. *Computer Graphics*, 25(4), 1991.

[25] Edward John Nicolson. *Tactile Sensing and Control of a Planar Manipulator.* PhD thesis, EECS, University of California, Berkeley, 1987.

[26] M. Ouh-Young. *Force Display in Molecular Docking.* PhD thesis, UNC Chapel Hill, Februry 1990.

[27] S. Peiper, J Rosen, and D. Zeltzer. Interactive graphics for plastic surgery: A task-level analysis and implementation. In *Symposium on Interactive 3D Graphics*, 1992.

[28] E. Promayon, P. Baconnier, and C. Puech. Physically-based deformations constrained in displacements and volume. In *EUROGRAPHICS*, 1996.

[29] X. Provot. Deformation constrains in a mass-spring model to describe rigid cloth behavior. *Computer Interface*, 1995.

[30] J. N. Reddy. *An Introduction to the Finite Element Method.* McGraw-Hill, Inc., 2nd edition, 1993.

[31] Diego Ruspini. Adding motion to constraint based haptic rendering systems: Issues and solutions. *Proceedings of the Second PHANToM User's Group Workshop*, October 1997.

[32] Diego Ruspini and Oussama Khatib. Dynamic models for haptic rendering systems. *Advances in Robot Kinematics*, pages 523–532, June 1998.

[33] Diego C. Ruspini, Karsimir Kolarov, and Oussama Khatib. The haptic display of complex graphical environments. *Computer Graphics Proceedings*, pages 345–352, August 1997.

[34] Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib. Rubust haptic display of graphical environments. *Proceedings of The First PHANToM User's Group Workshop*, September 1996.

[35] Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib. Haptic interaction in virtual environments. *The Proceedings of the International Conference on Intelligent Robots and Systems*, September 1997.

[36] Kenneth Salisbury. An overview of haptics research at mit's ai lab. *Proceedings of The First PHANToM User's Group Workshop*, September 1996.

[37] M. A. Srinivasan and J. K. Salisbury. Chapter 4: Haptic interfaces. *Virtual Reality: Scientific Techonological Challenges*, 1994.

[38] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *Computer Graphics*, 22, August 1988.

[39] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics*, 21, July 1987.

[40] D. Terzopoulos and K. Waters. Physically-based facial modeling, analysis and animation. *Journal of Visualization and Computer Animation*, 1990.

[41] Sundar Vedula and David Baraff. Force feedback in interactive dynamic simulation. *Proceedings of The First PHANToM User's Group Workshop*, September 1996.

[42] K. Waters. A muscle model for animating three-dimensional facial expression. *Computer Graphics*, 21(4), July 1987.

[43] H. M. Westergaard. *Theory of Elasticity and Plasticity.* Dover Publications, Inc., 1964.

[44] A. Witkin and *et al* . An introduction to physically based modeling. Course Notes, 1993.

[45] M Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal of Algrbraic Discrete Methods*, 2, 1981.

[46] Yan Zhuang and John Canny. Real-time simulation of physically realistic global deformations. *IEEE Visualization: Late Breaking Hot Topics*, October 1999.

[47] Yan Zhuang and John Canny. Haptic interaction with global deformations. *The International Conference on Robotics and Automations, IEEE*, April 2000.

[48] Yan Zhuang and John Canny. Real-time global deformations. *The fourth International Workshop on Algorithmic Foundations of Robotics*, March 2000.

[49] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method: Basic Formulation and Linear Problems*, volume 1. McGraw-Hill Book Company, 4th edition, 1989. linear finite element method, linear elasticity.

[50] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method: Solid and Fluid Mechanics Dynamics and Non-Linearity*, volume 2. McGraw-Hill Book Company, 4th edition, 1989.

[51] C. B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. *ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems 1994, Dynamic Systems and Control*, 1:146–150, November 1994.