Algorithm Design & Analysis of Multi-Scale Meshing in Surgical Simulators

Zou Chunzhong

(B. Eng., Chongqing University, P. R. China)

(M. Eng., Huazhong University of Science & Technology, P. R. China)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2007

Acknowledgement

The author is grateful for getting a chance to do research with his supervisors: Prof. Teo Chee Leong and Dr. Etienne Burdet. It is great to learn from them: their passion for research, their insights to the key issues of a research topic and their straightforward ways in discussion. Also, the author appreciates their patience.

The author is in great debts to Dr. Tim Poston and Dr. Lim Kian Meng for offering guidance and sharing their expertise. Also, it is helpful to discuss with Liu Bing Feng, Zhang Li and other colleagues. Especially, Wang Fei offered his help in OpenGL programming. The author owes his special thanks to Madam Leow, Mrs Ooi, Miss Tshin, Mr. Yee and Mr. Zhang for offering help and services.

Finally, the author owes very much to his family and friends for their encouragement and financial help: his wife, Zhou Rong, his daughter, Zou Yi, Catherine, his mother-in-law, He Yu Ru and his friends, especially, Yan Li, Lu Yiqiang, Gao Xingsen and Tong Zhipeng.

Table of Contents

Summary

Surgical training simulators with real-time force feedback have become one of the most important applications of robotic systems. It is due to many factors such as 1) the training of an experienced surgeon is time-costly and expensive; 2) situations faced by surgeons are varied and complex, and 3) other constraints such as the availability of subjects and ethical issues involved in vivo-tests. Therefore, computer-based simulations and simulators are of great values. Most importantly, many medical procedures are intrinsically haptic, which means, in addition to visual and auditory feedbacks, it is critical for a surgeon to "feel" the touch, control, manipulation, kinesthetic perception under the effect of forces, torques and other sensory feedbacks resulting from tool-tissue interactions. Thus, the modeled or simulated systems must be physically real and provide surgeons with "true" real-time haptic feedback. Unfortunately, "physically real" and "real-time haptic feedback" are contradictory requirements - a physically real system in surgical simulations is often non-linear and computationally expensive and "true" real-time haptic force feedback always requires a very high update rate (>1000Hz) for an operator to get the "feel" of touch and manipulation in real-time. It is crucial to get a simplified but still sufficiently real simulation model. One method is to use multi-scale meshing. Compared with conventional FEM (Finite Element Method), where all areas are modeled using evenly fine meshes, a so-called multi-scale meshing scheme, where some areas are modeled using fine meshes and other areas are modeled using coarse meshes, seems attractive and promising.

In order to speed up the computation, this thesis investigates three different ways to get multi-scale mesh and their relative efficiency. The primary focus is to investigate if the Dynamic algorithm is used in modeling some simple interaction between tool tip and tissue when only slight change happens. That is to say, if the system evolves slightly from one state to another, thus, can we only need to make relatively small adjustments to the existing mesh to get the new mesh. Using a fineness function to reflect the meshing requirements in different areas and adopting an array-based perfect tree (for 1D case), the author implements three different multi-scale meshing, namely, Division algorithm, Merging algorithm and Dynamic. In great details, the author illustrates how each algorithm is implemented with the same criterion.

The comparison of algorithms and real experiments show that, in 1D case, multi-scale meshing can achieve real-time haptic feedback and the Dynamical algorithm is more efficient when the simulated interaction between tool tip and tissue changes slightly. During the development, the author finds the solver for D tridiagonal linear equations can be divided smaller systems for simplification and an improved solver is implemented. The ideas such as fineness function and Dynamic algorithm developed in this thesis can be served as the base for 2D and 3D cases.

Nomenclature

Δ	Spring deformation (units: length)
f	Spring force (units: force)
$\boldsymbol{f}_{\mathrm{E}}$	Elemental force vector
$\sum f_x$	Summation of external forces in x-axis
F	External force applied at a node
k	Spring constant (units: force/length)
K	Overall stiffness matrix
K_{E}	Element stiffness matrix
h	The height of a perfect binary tree (the number of links from root to the leaf)
m	The number of vertices at the deepest level
m_bQua	A Boolean value to show if an element's division meets the fineness requirements
m_eChg	An integer to reflect an element's new state after the element's fineness value is changed
m_iFLev	The fineness integer of an element in the full element map
m_iDLev	The division level of an element in the full element map
M, N, K	An element's division level in a binary tree
n	The number of vertices in a perfect binary tree
N_n	The index of an element in array-based binary tree
NL	The left child index of an element (N_n)

NP	The parent element’s index of an element (N_n)
NR	The right child index of an element (N_n)
$\boldsymbol{u}_{\mathrm{E}}$	Elemental displacement vector
U	Nodal displacement vector
x	Horizontal displacement (units: length)
X	The leftward division times of post-processing
Y	The rightward division times of post-processing

List of Figures

List of Tables

1 Introduction

1.1 Background and motivation

In the Control and Mechatronics lab of the Department of Mechanical Engineering, National University of Singapore, researchers are trying to work out some surgical training systems to help train new surgeons. The reasons to develop such surgical training systems are as follows.

1. In traditional microsurgery, it takes a long time for surgeons to gain the necessary skills needed for conducting complex operations such as stitching blood vessels.

2. It requires many exercises for novices to master different hand-eye coordination under conditions where angles are rotated and distances are magnified by the microscope.

3. For highly risky operations such as cutting living animals or cadavers, in order to gain real experiences, “physically” real environments are often needed to train surgeons. However, such operations are often constrained by factors such as the availability of the subjects, ethical problems and others.

4. Generally, it is hard for surgeons to apply his experiences gained from training for one situation to another situation. Therefore, different kinds of “physically” real systems are needed for surgeons to practice.

In short, it is of great value if we can develop surgical training systems to overcome the above-mentioned limitations of the real world. And, in fact, much research has been done for the simulation of the physical behavior of deformable tissues and organs [23][28] as well as the virtual surgical training system with the integration of visual feedback, real-time force feedback and computer geometry.

In the simulation of the physical behavior of deformable tissue and organs, many methods have been used. In particular, the Finite Element Method [23], the Mass-Spring-Damper system [28], the Boundary Element Method, and the Finite Difference Method are often preferred. Whichever method is used, the same dilemma should be solved. That is, how to balance contradictory requirements among physical accuracy, real-time constraints and limited computer resources. Since a traditional Finite Element Method is often not able to meet the real-time requirements, researchers have to design sufficiently simplified physical models in order to provide an operator with real-time force feedback.

However, a crucial challenge originates from the intrinsic nature of haptic sense. In a typical robotic surgical system, both visual and haptic loops are involved while the haptic loop imposes a greater challenge for real-time force feedback. In a sharp contrast to human's visual sense which requires only 20 to 30 frame/second or higher update rate [20], haptic sense requires a much higher update rate of at least 200 to 300 frame/second [20] if an operator wants to have a feeling of force. According to R.Chung et al. [53], unlike the 30-60 update rates for visually presented VR systems, haptic simulation places more stringent demands where a rate of 1 kHz is typically cited as desirable for stable and responsive haptic interaction.

In response to the requirements of high haptic update rates, simplified schemes such as multi-resolution, multi-rate and multi-scale methods are designed ([9][11][19][21][25][29][40][50][53]). Though different names are used, all of them focus on simplifying meshing scale in order to provide real-time force feedback.

Compared with a conventional FEM where all areas are modeled with equally high resolution, such multi-scale models can dramatically reduce the order of a physical system and thus greatly reduce the computation burdens.

Thus different algorithms based upon multi-scale FEM methods are developed in order to improve the efficiency of our robotic surgical simulation system. Also, some important new ideas used in multi-scale FEM algorithms are introduced as follows.

Firstly, a so-called fineness function is adopted to assign weights to different areas. In any multi-scale meshing scheme, a basic but difficult problem is how to set reasonable criteria to decide whether fine elements or coarse elements should be used in different areas and situations. For example, in the domain where interaction between a tool and tissue occurs, fine elements must be used while coarse elements might not be acceptable. Meanwhile, many other factors such as curvatures, the geometry of a physical system, loading conditions and visual rendering also need careful consideration. In order to reflect all these different factors involved in the modeling of a physical system, a universal fineness function is used to express the relative importance of different areas.

Secondly, we desire to investigate how the systems evolve from one state to another and how we can save processing time and storage through the understandings of the dynamic natures of robotic systems, which include two types of great interests. One is the dynamic evolution of meshing and the other is the dynamic solving process, by which a system’s previous results can be effectively re-used.

In this study, the one-dimension linear elastic case is used, which can prevent us from being distracted from important issues and pave the way for higher dimensional situations.

1.2 Contributions

The applications of a fineness function and three algorithms are the main contributions of the current study.

a. A fineness function

In order to quantify fuzzy information in meshing and re-meshing, a fineness function is creatively used to reflect all factors involved in decision-making. It is rather general and adaptive for different application occasions.

b. Division algorithm

This algorithm is based on the idea that coarse elements are first used in order to approximate a physical system. If the coarse elements are sufficient to meet preset criteria, a multi-scale mesh can be established using coarse elements. Otherwise, the coarse elements are recursively divided into smaller ones until all criteria are met. This approach is efficient in the situations where coarse meshes are sufficient.

c. Merging algorithm

In contrast to the above-mentioned Division algorithm, Merging algorithm starts from the opposite direction. The finest elements are first used to model a physical system, then, the process will proceed to check whether small elements can be replaced by big elements in re-meshing. To be exact, if two sibling elements can be replaced by their common parent element, then, the parent element will be used. The final multi-scale mesh should be the same as that obtained through Division algorithm. The Merging algorithm is preferred in situations when more fine meshes are needed.

d. Dynamic algorithm

In an interactive real-time haptic system, dynamically changing configurations push us to develop Dynamic algorithm to investigate how a new mesh can be obtained from a previous one. When only minor changes occur in a system, a small portion needs to be re-meshed while the majority of the meshes can remain unchanged. The ideas of Division algorithm and Merging algorithm are combined to form this new algorithm. After re-meshing, some areas become finer and some other areas become coarser.

e. The significance of the algorithms

The significances of the above-mentioned algorithms are presented below. 1) Dynamic meshing can really save computation time under conditions where a system only changes slightly. This algorithm may be suitable for some medical applications. 2) All of the three multi-scale meshing algorithms are implemented and they can be regarded as a skeleton research work. The author proposes and implements these basic ideas and paves the ways for further research and applications in real systems. 3) Algorithms are implemented using object-oriented programming method.

Besides the above contributions, a new and improved solver for tridiagonal equation systems is discovered and implemented. Furthermore, interesting issues related to the array-based binary tree are also discussed

1.3 Organization of the thesis

There are totally eight chapters and the rest of the thesis is organized as follows. A literature review on real-time force feedback systems is given in chapter 2. Topics like the meshing of simulated physical systems, schemes for haptic surgical simulators, and Internet-based tele-operational haptic systems are reviewed. The fundamentals of the process of solving typical FEM problems and the reasons for adopting multi-scale methods are presented in chapter 3. In chapter 4, key issues related to the three proposed algorithms such as the data structure and program schemes are described. In chapter 5, detailed implementations of the three multi-scale meshing algorithms are given. Comparisons of simulation results and the validation of the performances of the algorithms are made in chapter 6. Finally, conclusions for the thesis are made in chapter 7. In the Appendix, A new solver is introduced and the implementation of three algorithms in a real-time workstation is presented.

2 Literature Review

2.1 Introduction

This chapter gives a literature review on historical interests, mainstream research areas, some widely used models, and strategies in haptic systems. Among the current research areas, the following three attract a lot of attention [47].

1. The physiological or anatomical experiments and analysis in order to discover the intrinsic reasons of haptic senses of human being (especially, haptic force sense).

2. The strategies or schemes of real-time force feedback simulation systems (especially medical or surgical simulation systems) in terms of software, hardware and workstation sites (local systems or tele-operation systems over Internet).

3. The meshing and algorithms in modeling physically real systems.

The following paragraphs will give a survey about haptics (section 2.2), challenges and strategies of haptics (section 2.3), and typical surgical systems and models in real-time medical simulation (section 2.4). The summary is presented in section 2.5.

2.2 Haptics: definitions, historical issues and its significance

2.2.1 Definitions of haptics and haptic interface

Many researchers give the descriptions or definitions for haptics and haptic interface [12][17][34]. For some of them, haptics refers to the modality of touch and the associated sensory feedback while others think that it refers to taction and kinesthesia (the sense of body position and motion) [12]. When coupled with a computer, haptics is regarded as the art and science of developing software algorithms that synthesize computer generated forces/torques to be displayed for users to perceive and manipulate through interacting with virtual objects.

The goal of haptic rendering is to enable a user to touch, feel, and manipulate virtual objects through a haptic interface [34]. By now, the technology is relatively mature for net force/torque feedback (the weight of grasped virtual objects, their mechanical compliance, inertia, as well as motion constraints) but not so for tactile feedback. This is because tactile feedback includes other feedbacks (for example, the rugosity of virtual surfaces, their edges, temperature, or slippage) as well as force/torque feedback [20].

A typical haptic system has five major parts [49]: human operator, haptic measurement device, haptic feedback device, virtual environments and virtual coupling. And haptic interface is a human-computer interface with information input and output that allow users to touch, feel, and manipulate objects simulated by virtual environments. With a haptic interface, human-computer interaction is different from its conventional counterpart (Fig. 2.1).

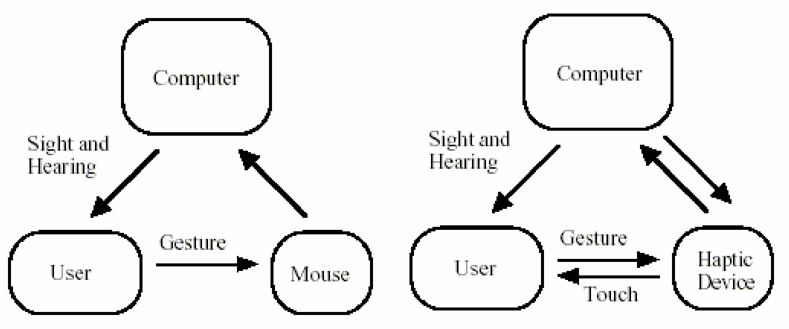


Fig. 2.1 Conventional interaction loop and interaction with a haptic device [17]

2.2.2 Historical origins about Haptics

It is widely accepted that Iwata laid the foundation of modern VR system with visual and haptic feedback through his research in [2]. However, according to Gillespie's comprehensive survey in [12], Haptics as an academic discipline dates back to the time of Aristotle. In Aristotle's view according to his treatise *De Anima* (On the soul), touch was the most essential of human's five senses. By 1749, Denis Diderot published his "Letter on the blind", and thus laid the foundation for our understanding of sensory substitution. He pointed out that one sense gains in power with use or loss of another. In 1834, Ernst H. Weber (widely deemed to be the founder of the field of psychophysics) introduced systematic experimental procedures to the study of haptics. According to the influential book *Der Aufbau Der Tastwelt* published in 1925, David Katz investigated many aspects of the sense of touch and he maintained that touch is a far sense like vision and hearing that not necessarily requires direction impression on the skin by an object. In a real sense, a tool becomes an extension of one's body, and the sensory site moves out to the tool tip, which underlines the claim that understanding haptics has important implications for the effective use and design of tools. Geza Rvesz did research in the development of haptic perception in the blind and the coding of spatial information. James Gibson fostered a more ecological approach to research in sensory processes and perception. Now, people tend to think haptics is not subservient to vision and audition and it has attracted much interests. Compared with visual and audition, Haptics is more complex in that its function is coupled to movement and active participation of the subject.

2.2.3 Applications of Haptics or specifically real-time force feedback

Nowadays, haptic force feedback systems are widely used in many situations. And its typical applications include: surgical simulation, medical training [44] by providing the real-time force feedback, scientific visualization [44] for virtual reality, painting and handwriting [42][45][46], sculpting [24], hologram [15], CAD and mechanical assembly and grasping [16][22] to provide realistic training, micromanipulation [26], nano-manipulation [52], musical learning [12], Tele-monitoring, Tele-master-slave system [3][4][31], Tele-assistance [10][29], Tele-operation or networked robotic systems [3][4][10][14][29][31][33][47][51][53], systems purely for augment [41][48] [52] and Graphical User Interface [39], autonomic robotics [5][10], Java-Based Tele-robotics [14][33][51] and others, among which surgical simulation and medical training have been a primary application area [53]. All these systems are trying to model some kind of force feedback to simulate the physical situations to help operators get realistic training and valuable skills.

2.2.4 Benefits from the use of haptics

Human beings are intrinsically multi-modal systems and Haptics represents the next logical evolution of digit displays [24]. Its significance can be described from the following descriptions.

a) In contrast to vision and hearing, Haptics permits bi-directional information transfer between the user and virtual environments (Fig. 2.1) [47], which is crucial in tasks such as surgical simulation in order to simulate the interaction with the organs.

b) The force feedback is extremely important for medical training, minimally invasive surgery, remote diagnosis and aids for the disabled as the operators must depend on using forces to complete control and make decisions.

c) With the adding of real-time force feedback, it is possible for an operator to feel the properties of simulated objects [53].

d) Distributed users can improve their communication through the use of haptics by improving their awareness of others' activities and intentions over Internet [18] as participants can "sense" the existence of others and "interact" with each other.

e) A graphical user interface integrated with haptic feedback will greatly change the interaction between computer and human being [7] because such interaction models the real-world interaction.

f) Systems [26][43] with real-time force feedback can provide augmented reality and visuomotor skills in nano-manipulation [13][52].

g) Haptics even plays an important role in the process of learning to play an instrument [12].

h) A prototyping system with haptic force-feedback display can save the cost and time and facilitates the modification while the model is still in a computer [11].

2.3 Haptics: challenges and strategies

2.3.1 Challenges

It is challenging to build a haptic system. Firstly, haptic computation is expensive [53] because it involves detecting collision, computing the contact manifold, estimating the penetration depth, and computing restoring forces and torques. Secondly, haptic hardware is difficult to design. If we focus on a typical surgical simulator, there are several challenging issues [53] as follows.

a) Time limitation from real-time requirements.

b) High-fidelity force rendering and self-induced oscillations and instability.

c) Costly modeling of complex objects and computational complexity of simulating the nonlinear dynamics of tool-tissue interaction.

d) Complex haptics hardware and control design of haptic interface [17]: force control, non-linearity of environments such as impedance and admittance characteristics and the inclusion of human operators.

2.3.2 Strategies

In this section, we will investigate the strategies used in some medical simulators to deal with the challenges discussed in section 2.3.1. To meet the requirements of real-time force feedback, the simplification of deformable objects and effective haptic rendering algorithms [34] are indispensable. Also, different software and hardware schemes are used to lessen the computation burden as follows:

a) A server/client scheme is used for haptic rendering framework [6][29][42] in order to distribute the computation burden. It consists of a haptic server for haptic rendering and a graphical client application for visual feedback.

b) Special controllers are designed in master-slave system over Internet for high-fidelity force rendering [3] [4].

c) Physics-based models of soft issues are replaced by simplified models such as: network of point masses model, springs and dampers model, and finite spheres model [28].

d) Efficient algorithms are coded. For example, in minimally invasive surgical procedures, Kim et al. [38] design a collision prediction algorithm in order to achieve real-time force feedback.

e) Special numerical methods are adapted. In [38], a set of finite spheres are used instead of traditional FEM meshes. Thus, computation time is tremendously reduced. A simulation of palpation is reportedly achieved with 1000 Hz for force update and 30 Hz for visual update.

f) Hybrid physical models can be used. In [32], in order to reduce computation, a FEM model is used to simulate the flexible dynamics of the duct and a particle model is used for the catheter.

2.4 Surgical simulation systems and models for deformable tissues

2.4.1 Typical surgical simulation systems

As our main focus is on haptic surgical trainer, it is helpful to know what others have done in the relevant areas, so some representative applications are listed [30][54]:

a) In 1997, Langrana et al. [54] used the Rutgers Master II haptic device in a training simulation for palpation of subsurface liver tumors. Tumors are modeled as comparatively harder spheres within larger and softer spheres. Realistic interaction forces are calculated with FEM and the graphical display showed tissue deformation. Thus, real-time force feedback is achieved.

b) In 1998, Mor [54] in Carnegie Mellon University built up an arthroscopic surgery simulation.

c) In 2000, Balaniuk et al. [54] (Stanford University) simulated fluid-filled objects suitable for interactive deformation such as “cutting”, “suturing” and so on.

d) In 1998, De et al. (MIT) developed simplified models and efficient algorithms for tool-tissue interactions. They modeled soft tissue as thin-walled membranes filled with fluid and the final force-displace response is comparable to that obtained in in vivo experiments.

e) In 2001, Sastry et al. [54] (UC, Berkeley) built complex robots for the surgery of laparoscopic and endoscopic cannulas.

f) In 1999, Aviles et al. [54] developed VR-based dental training simulator.

g) In 1998, Giess et al. [54] integrated haptic volume rendering using the PHANToM into the pre-surgical process of classifying liver parenchyma, vessel trees, and tumors.

h) Surgeons [54] at the Penn State University built a system in which residents passed simulated needles through blood vessels, allowing them to collect baseline data on the surgical skills of new trainees.

i) In 1998, Iwato et al. [54] reported the development of a surgical simulator with “free form tissue” which can be “cut” like real tissues.

j) G. Picinbono et al. [30] presented the minimally invasive haptic surgery simulator prototype. The system is to provide a realistic training test-bed for performing laparoscopic procedures through the simulation of the deformation and cutting of 3-D anatomic model.

2.4.2 Models for deformable objects

In a surgical simulation system, modeling deformable objects and reducing intensive computation are crucial. The idea of using variable multi-scale meshing for modeling and the ways to reduce computation attract much attention. Here, we list some representative schemes.

K.M. Lim et al. [40][50] have developed a fast multi-scale FEM algorithm for a VR-based microsurgery training system to offer novices realistic surgical training. Like other surgical systems, the time saving comes from the reduction of the order of the simulated system because fine and coarse meshes are introduced instead of using evenly fine mesh.

Rynson et al. [9] introduced a multi-resolution method to model arbitrary topology (only triangular elements are used). First, each vertex is given a visual importance value used to determine the visual importance value of each edge and then the visual importance value of each triangle. Then, during runtime, the system will decide how many triangles should be removed from the model according to the render time available and the distance of the object from the viewer at a particular frame. Such a method considers the computation time and also preserves the geometry of the model as much as possible.

Çavusoglu et al. [19] proposed a multi-rate simulation method that uses a local linear approximation. The purpose is to deal with the difference between the high update rate (1kHz) of haptic interfaces and the significantly lower update rates (10Hz) of the physical models being manipulated. First, the inter-sample behavior is first modeled as a linear approximation of the full order model. Then, a local linear model is directly derived from the full-order model. So the system has a full-order model running at the very low rates (10Hz) and a local linear model is running at haptic rate (1kHz) to model the inter-sample behavior of the physical model.

F.Ganovelli et al. [25] introduced another multi-resolution model to simulate a tissue cutting. The system not only uses a multi-resolution triangular mesh, but also allows changing the system's topological structure. So, when the cut is intersected, topological structures are dynamically changed.

Wu et al. [23] presented a nonlinear FEM using mass lumping to produce a diagonal mass matrix that allows real time computation. A dynamic progressive meshing can provide sufficient details where required for the real-time surgical simulation.

K. Hirota et al. [27] used a linear FEM to model the elasticity of an object in VE. In their model, two processes are executed asynchronously. One process is pre-computation of the inverse stiffness matrix, and, the other is the interpolation of forces in order to get a smooth interaction force feedback.

J. Kim et al. [53] used a coarse global model as well as local subdivision and smoothening in their simulation. Such a multi-resolution method is used to simulate laparoscopic surgical procedures when real-time force feedback is needed.

2.5 Summary

This chapter gives a brief literature survey about the haptics: its challenges and strategies, and its typical usage in surgical simulators. Several multi-resolution meshing models are investigated according to the survey. In any interactive VR-based surgical simulation with force feedback (1kHz) and visual feedback (30Hz), it is critical to reduce the model in order to save computation time. The coming chapters will expand around this key issue.

3 Fundamentals of Multi-Scale FEM

3.1 Introduction

As the processes for both multi-scale FEM and conventional FEM are identical, this chapter will introduce only the fundamental issues of finite element analysis. Zhang et al. [60] developed a way to model multi-scale FEM, however, the author wants to develop different algorithms and investigate more fundamental issues involved in multi-scale FEM and explain the difficult research areas.

The introduction of FEM is first presented in section 3.2; the process of FEM and an example are described in section 3.3 and 3.4 respectively. Then, the reasons for using multi-scale FEM are presented in section 3.5. Two important issues relevant to the project are discussed in section 3.6. Finally, a summary for the whole chapter is given in section 3.7.

3.2 Principles of FEM or multi-scale FEM

As a universal engineering analysis method, FEM is widely used in engineering analysis where a continuous problem with countless DOFs is approximated by a discrete model with finite DOFs and interpolation functions (normally interpolation polynomials) are used to describe the behaviors of points between its end points (nodes).

FEM formulae can be established in several ways. For 1D FEM, they can be directly inferred from the approximation of the differential equation with boundary conditions [55]. However, it is also preferred to start from the natural laws governing the behaviors of a system and then to establish their institutional relationships in order to solve the problems. Other more general methods include minimum energy method, variation principle, or, functional analysis and etc. No matter what methods are used, the FEM process means a lot of unknowns and a lot of equations used to solve unknowns.

Because a physical way is often instructive and intuitive, one-dimensional linear elastic mass-spring system is used to illustrate the most basic concepts and the most fundamental issues involved in FEM will be discussed. The specific solvers and skills for solving assembled equations will be discussed in the Appendix.

3.3 FEM: concepts and the process of solving

Here, we will demonstrate the basic ideas in 1D linear elastic cases. Some simple but instructive examples will be used to help expound basic and important concepts.

3.3.1 Basic concepts

Degrees of freedom (DOFs) are defined as the number of independent variables (or here, the coordinates) that are necessary to completely specify the configuration of a system. Figure 3.1 below shows a spring that needs two degrees of freedom at nodes I and J in order to specify its positions.

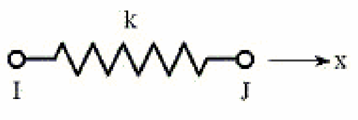


Fig. 3.1 A spring element with two DOFs

A spring element is defined as a deformable element under one-dimensional (x-axis) tensile or compressive loads that produce axial displacements at each node.

A node is defined as a rigid point connecting adjacent elements and bearing forces.

The physical law: for a linear elastic spring, the nodal deflection in a spring is directly proportional to the spring force applied (Fig.3.2).

$$f = k\Delta = k\,(x_I - x_J)$$

f - Spring force (units: force)
k - Spring constant k>0 (units: force/length)
Δ - Spring deformation (units: length)
x_I – Horizontal displacement at node I
x_J - Horizontal displacement at node J

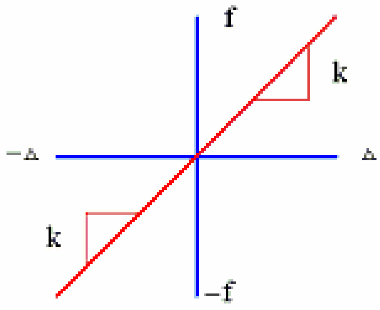


Fig. 3.2 Spring law

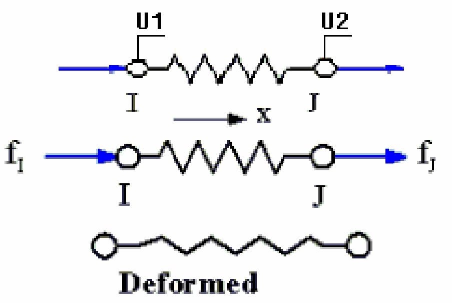


Fig. 3.3 Sign convention

The following sign conventiosn will be assumed (Fig. 3.3):

a) Nodal displacement (*u*): positive to the right.

b) Nodal axial force (*f*): positive to the right.

c) Axial deformation (*Δ*): positive for elongation.

3.3.2 Determination of single element stiffness matrix K_E

For the stress-strain analysis of a statically indeterminate problem, three conditions are stipulated:

a) Equilibrium: forces acting on each separated element must be in equilibrium. That is expressed as: $\sum f_x = 0$.

b) Compatibility: the structure must remain continuous, no geometrical gaps or overlaps must exist;

c) Physical law (stress-strain law): in this case, a linear spring formula that connects the forces in the equilibrium equation to the deformations in the compatibility equation.

The variables defined for the spring element are:

f_I - Axial force of local node I $\quad$ u_I - Axial displacement of local node I.

f_J - Axial force of local node J $\quad$ u_J - Axial displacement of local node J

k - Spring constant

Each element has two nodes and thus has two DOFs. Nodal forces (f_I, f_J) and nodal displacements (u_I, u_J) are all assumed to be positive. We will know that the stiffness matrix will be a 2x2 matrix.

$$\boldsymbol{f}_E = K_E * \boldsymbol{u}_E$$

$$\begin{Bmatrix} f_I \\ f_J \end{Bmatrix} = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \begin{Bmatrix} u_I \\ u_J \end{Bmatrix} \tag{3.1}$$

The task at hand is to determine unknown terms k_{11}, k_{12}, k_{21}, & k_{22}. Fig. 3.4 illustrates a separated spring element.

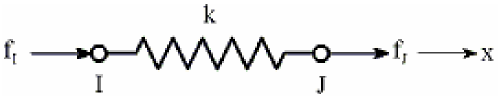


Fig. 3.4 A separated spring element

Applying the physical law at the node J (elongation is assumed.), we have:

$$f_J = k * \Delta = k * (u_J - u_I) = -ku_I + ku_J$$

Applying the equilibrium to the free-body element, we have:

$$f_J + f_I = 0$$

$$f_I = ku_I - ku_J \tag{3.2}$$

$$f_J = -ku_I + ku_J \tag{3.3}$$

Thus,

$$K_{\mathrm{E}} = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \tag{3.4}$$

In a compact matrix form, the equation for the spring element is:

$$\boldsymbol{f}_{\mathrm{E}} = K_{\mathrm{E}} * \boldsymbol{u}_{\mathrm{E}} \tag{3.5}$$

$$\begin{Bmatrix} f_I \\ f_J \end{Bmatrix} = k \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_I \\ u_J \end{Bmatrix} \tag{3.5*}$$

More about the stiffness matrix K_E

We have used a "direct method" to derive K_E, and the characteristics of K_E are listed as follows.

a) Square matrix.

b) Symmetric matrix.

c) Diagonal terms are always positive.

3.3.3 Overall stiffness matrix

After we get the formula (3.5) for each element, we can rearrange both known and unknown variables. Thus, we are able to compute the whole system through specialized equation solvers. Instead of using energy or weighted residual methods, which are deemed complex and general, we use a direct assembly method to get the overall stiffness matrix. That is, all element stiffness matrixes (K_E) are assembled to the overall stiffness matrix (K). The contribution of each (K_E) is directly added into the overall stiffness matrix (K) according to their configurations.

3.3.4 Solution characteristics for 1D linear elastic case

In FEM, a so-called shape function (normally polynomial) is used to describe the deformation of each geometrical point within an element. For any point within an element (1D here), its displacement can be represented by $u\ (x) = a * x + b$ where a and b are constants and can be decided by boundary conditions. At the boundaries I, J, u_I and u_J are both known, so, a and b can be determined. Thus u can be uniquely determined for each given position x. Through this, we can get the distribution field of displacement within an element and consequently, stress distribution. Before we start our case study, theoretical analysis of system behaviors are helpful.

a) At element level

The continuity of axial displacement

Since the axial displacement is described by a linear polynomial, $u\ (x) = a * x + b$ which is a continuous function, the axial displacement must also be continuous for the entire length of the element. This means that a graph of the axial displacement will be continuous and have no gaps (cracks) or overlaps (see Fig. 3.5).

The interpretation of axial displacement distribution

An axial displacement indicates the distance that each point within the element has moved from its non-loaded position along the x-axis. A FEM solution will only give us a value for the axial displacement at each node and displacement distribution can be obtained from the shape function (see Fig.3.5).

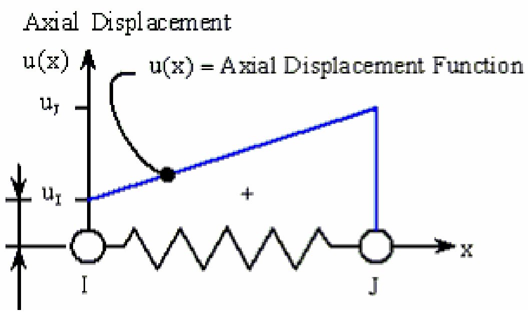


Fig. 3.5 Axial displacement distribution

b) At assembly level

Deformation discontinuities among adjacent elements

For 1D linear case, the deformation ($\varDelta$) is a constant used to describe the overall deformation of the element. At the assembled mesh, the deformations of neighboring elements normally vary because each element may have different spring constant (k), external loading and constraints in spite of the fact that the deformation (Δ) is a constant within each element. Thus, the deformations among adjacent elements will generally not be continuous (see Fig. 3.6).

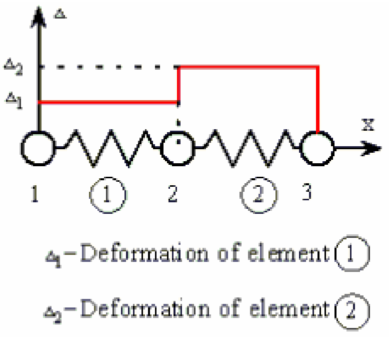

Δ
Δ_2
Δ_1
x
1
1
2
2
3
Δ_1 – Deformation of element 1
Δ_2 – Deformation of element 2

Fig. 3.6 Deformation of neighboring elements

Internal axial force (*f*)

The internal axial force (*f*) within each element is described by a constant. Any two adjacent elements, which have different spring constant, loadings or constraints, will generally have different internal axial force (see Fig. 3.7).

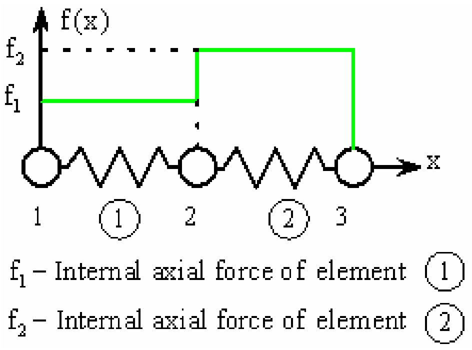

f(x)
f_2
f_1
x
1
①
2
②
3
f_1 – Internal axial force of element ①
f_2 – Internal axial force of element ②

Fig. 3.7 Internal forces among adjacent elements

3.4 The process of FEM - an example

It is helpful to use an example to illustrate the solving process and the analysis of numerical solutions. The configuration of the example is as follows. Three springs are assembled in series as shown in Fig.3.8. An external force (F2) is applied at node 2, assuming *k*1 = *k*3 = 10,000N/m, *k* 2 = 20,000N/m and F2 = 450 N. So, there are 3 elements and 4 nodes (thus 4 DOFs, 1 & 4 are constrained, 2 & 3 are unconstrained).

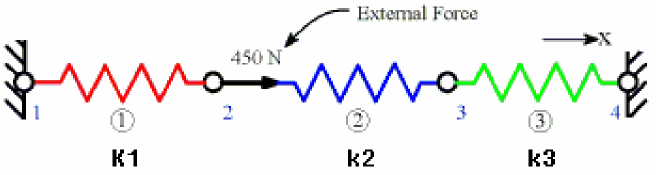


Fig. 3.8 Three linear springs in series

Such a system is statically indeterminate since there are two unknown reactions (both ends) with one equilibrium equation. The unknowns we need to find include nodal displacements (2 & 3), reactions at nodes 1 and 4, element forces, and element deformation. Its solving process is illustrated as follows.

1) Obtain equations

According to the Direct Assembly Method, we can get ***f*** = *K* * ***u***, where ***f*** represents 4 by 1 force vector; ***u*** is 4 by 1 displacement vector; and *K* is 4 by 4 stiffness matrix.

According to the formulae (3.5*), we can write the following three equations for three elements (Fig 3.9).

$$\begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} = 10{,}000 * \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \qquad (\#1)$$

$$\begin{Bmatrix} f_2 \\ f_3 \end{Bmatrix} = 10{,}000 * \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix} \begin{Bmatrix} u_2 \\ u_3 \end{Bmatrix} \qquad (\#2)$$

$$\begin{Bmatrix} f_3 \\ f_4 \end{Bmatrix} = 10{,}000 * \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_3 \\ u_4 \end{Bmatrix} \qquad (\#3)$$

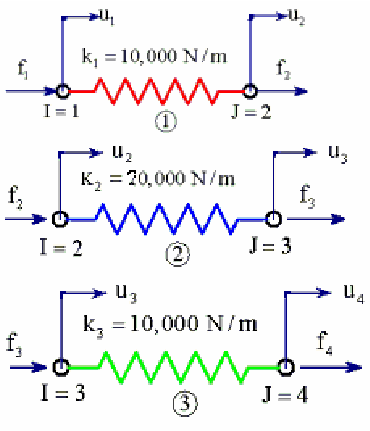


Fig. 3.9 Separate equations of elements

2) Assemble the overall stiffness matrix

We use the direct method to get the overall stiffness matrix (see Fig.3.10). The contributions of each element in the overall matrix are described as follows. The K_E of element #1 will fill a 2x2 sub-matrix (rows 1-2, columns 1-2) in the K. Similarly, element #2 is for rows 2-3 & columns 2-3, and element #3 is for rows 3-4 & columns 3-4. The resulting matrix K is a banded (tridiagonal) matrix with the width of 3 and other terms are all zeros. k_{22} and k_{33} are resulted from two terms since each of node 2 and node 3 connects two elements. After applying boundary conditions (Table 3.1), we get Fig 3.10.

Table 3.1Boundary conditions

Known	Unknown
$u_1 = 0$ (rigid wall);	F_1 = ? (wall reaction)
$u_4 = 0$ (rigid wall);	u_2 = ?
$F_2 = 450$N;	u_3 = ?
$F_3 = 0$ (no external force);	F_4 = ? (wall reaction)

$$\begin{array}{ccc} & \text{Node \#:} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{Bmatrix} = & & \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} \end{array}$$

Column Vector Square Matrix Column Vector

$$\begin{Bmatrix} F_1 = ? \\ F_2 = +450\text{N} \\ F_3 = 0 \\ F_4 = ? \end{Bmatrix} = 10{,}000 \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -2 & 0 \\ 0 & -2 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1 = 0 \\ u_2 = ? \\ u_3 = ? \\ u_4 = 0 \end{Bmatrix}$$

External Forces Displacements

F1, F2, F3 and F4: external nodal load

Fig. 3.10 Overall linear equations of the mesh

3) Solve the system of linear equations

Here, we need to solve unknowns: u_2, u_3, F_1, F_4. In Fig.3.10, since there are four equations for four unknowns, it is a determinate problem unless it is a degenerated system. There are two ways to solve the equations:

1) Re-arranging the known terms and the unknowns, we can get $K^* \boldsymbol{u} = \boldsymbol{f}$**,** where K and $\boldsymbol{f}$ are known, $\boldsymbol{u}$ is an unknown vector.
2) After crossing out rows and columns in K, $\boldsymbol{f}$ and $\boldsymbol{u}$ as illustrated in Fig.3.11, we can solve a smaller system of equations to get u_2 (0.027m) and u_3 (0.018 m).

$$\begin{Bmatrix} \cancel{?} \\ 450 \\ 0 \\ \cancel{?} \end{Bmatrix} = 10{,}000 \begin{bmatrix} \cancel{1} & \cancel{-1} & \cancel{0} & \cancel{0} \\ \cancel{-1} & 3 & -2 & \cancel{0} \\ \cancel{0} & -2 & 3 & \cancel{-1} \\ \cancel{0} & \cancel{0} & \cancel{-1} & \cancel{1} \end{bmatrix} \begin{Bmatrix} \cancel{0} \\ u_2 = ? \\ u_3 = ? \\ \cancel{0} \end{Bmatrix}$$

Fig. 3.11 Cross out terms to solve linear equations

4) Calculate axial displacement distribution

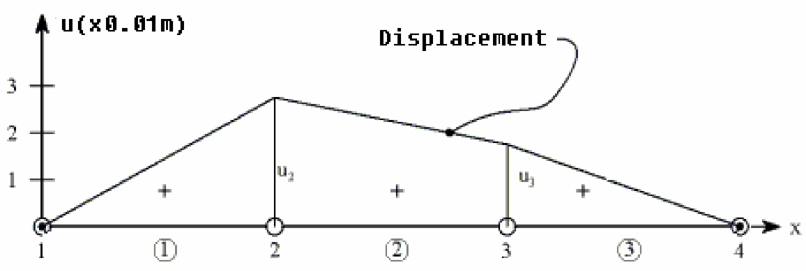


Fig. 3.12Axial displacement verse position along the entire mesh

From the displacement distribution (Fig. 3.12), we find:

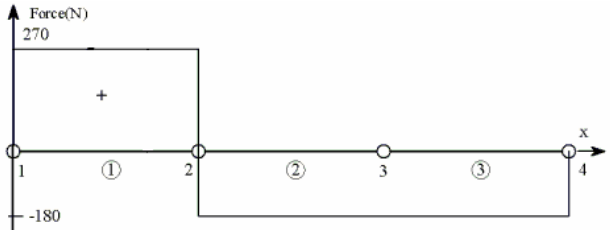
1. Over an element, displacement varies linearly.
2. Over the mesh, displacement is piecewise linear.
3. Displacement is single valued at nodes (compatibility).

5) Compute reaction forces

Theoretically, we can directly replace the known terms in Fig.3.10 to get all the unknowns. However, in actual numerical analysis, the original stiffness matrix is generally destroyed. So, some housekeeping is needed in order to get the final results. The reaction forces F_1 and F_4 can be otherwise numerically computed as F_1= -270N, and F_4 = -180N.

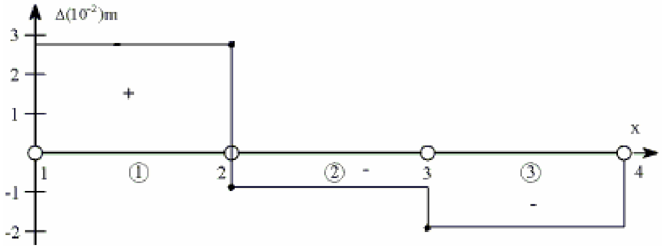
6) Determine spring forces and deformation of elements

Since all nodal displacements are now known, we can calculate elemental forces using the force-displacement relation and deformation of each element (Fig.3.13 and Fig.3.14).



(Above x-axis: tensile; below x-axis: compressive)

Fig. 3.13 Axial elemental forces versus position along mesh



(Above x-axis: elongation; below x-axis: compression)

Fig. 3.14 Axial deformation of elements versus position along mesh

3.5 The motivation for multi-scale FEM

Using the same process of the above-discussed FEM, we can solve a multi-scale FEM problem. However, though the process is identical, the meshing schemes are different. It is necessary to discuss why a multi-scale meshing scheme is a better choice.

Our ultimate aim is to develop a haptic surgical trainer to help novices. A surgical simulation system is complex as many issues are involved. Firstly, we need to model complex tissue and organs. Secondly, we need to simulate real-time tool-tissue interaction which is intrinsically non-linear. Thirdly, we need to compute collisions, the contact manifold, forces and torques. Furthermore, we might need to model complex behaviors such as suturing, cutting and others (Fig.3.15 shows the haptic loop in a surgical trainer). Lastly, the most challenging issue is that, in order to get the "real-time haptic feedback", we must get a very high update rate (>1000Hz). And all of these factors are involved in each loop, thus, the overall simulation is computationally demanding. So, how to simplify the models is crucial to reduce the order of computation. Under current computer resources and schemes, it is hard to model the system using traditional FEM because, in a traditional FEM, all areas are modeled with equally high resolution which demands high computational load, and thus it often cannot meet the real-time requirements required of surgical simulation. In contrast, a multi-scale meshing scheme, where some areas are modeled using relatively high resolution while other areas are modeled using lower resolutions, such a so-called multi-scale meshing scheme ([9][11][19][21][25][29][40][50][53]) can dramatically reduce the order of the simulated system and thus greatly reduce the order of computation. In short, simplified models and efficient algorithms are crucial to develop a surgical simulator. The basic scenarios and further possible speeding up [56] will be discussed in the next section.

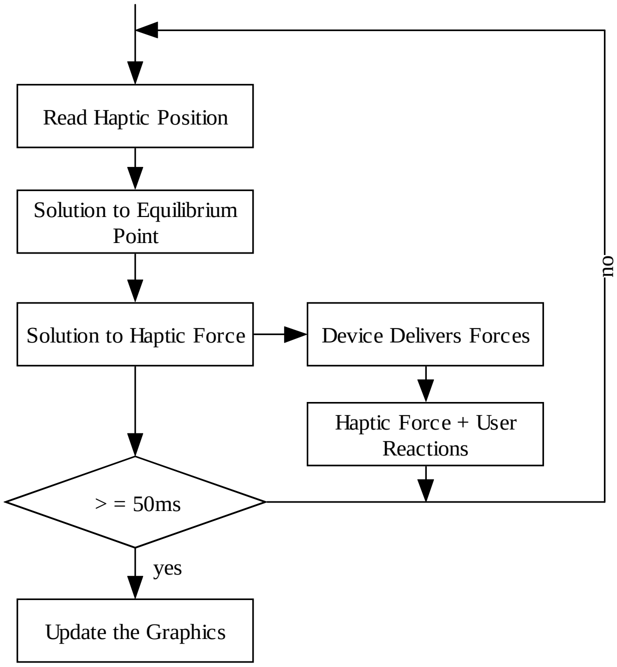


Fig. 3.15 A typical haptic system with real-time force feedback and visual feedback

3.6 Some basic scenarios and further speeding up

In this section, some basic scenarios for our algorithms are first introduced. Then, further reduction of computation overhead is discussed.

In [1], Poston describes tool tip-tissue interaction and suggests that using traditional FEM to model such behaviors would be expensive and inefficient. Therefore, a multi-scale meshing in modeling is suggested. Particularly, since the intrinsic topology of the binary division exists in 1D case, a perfect binary tree data structure is suggested to express the different levels of divisions, which precisely explains why a perfect binary tree data structure is used in our multi-scale algorithms in later chapters. As for the forces, typical force-delivering devices like the PHANToM [57] and torque-delivering haptic devices like the Lausanne system [58] can be used. The following will discuss another important issue: dynamic computation.

Although much computation time can be reduced by linearization and using coarse mesh, further speeding up can be achieved as described by Lim [56].

1) We can reduce repeated computation if we understand the mechanism how the solving process evolves, especially, when only minor changes occur. That means the reduction of the repeated computations of matrices K and $\boldsymbol{f}$ and $\boldsymbol{u}$ incurred in each solution process. Even the previous solutions of $\boldsymbol{u}$ may be used as the initial guess if the iterative matrix equation solver scheme is used.

2) Use parallel computation schemes and let the fine model to share common reusable parts with the coarse model so that their results can update each other mutually.

3.7 Summary

This chapter describes some basic and fundamental concepts. The solving procedures required of a FEM are also given in details. Such procedures are applicable in our multi-scale scheme. Also the motives of using multi-scale mesh and developing efficient algorithms are described. Finally, some scenarios and ideas for possible speeding up relevant to our project are discussed.

4 Key Issues for Multi-Scale Meshing

4.1 Introduction

Some key issues involved in the proposed multi-scale meshing algorithms are introduced in this chapter. They include: 1) object-oriented techniques and basic data structures being used (sections 4.2, 4.3, 4.4, and 4.5); 2) the reasons to use fineness function (section 4.6); 3) classes being used to express the topology of the data structures (section 4.7) and 4) special topics such as initialization and post-processing (section 4.8). All of these issues are indispensable for the proposed multi-scale meshing. However, among them, the applications of the array-based binary tree and the fineness function are crucial and creative. Finally, a summary is given in section 4.9.

4.2 Object-oriented (OO) programming

We design and code the algorithms using OO technology because it can provide advantages for modeling a complex system which includes data abstraction, encapsulation, information hiding and messages-based communication mechanism (Fig. 4.1.)

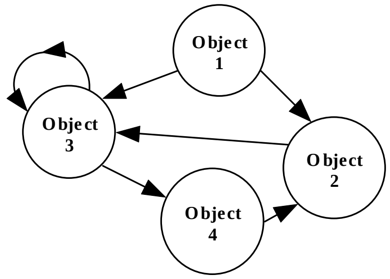


Fig. 4.1 Messages-based communication mechanism

4.3 Standard Template Library

In our program, we borrow the idea of Standard Template Library (STL) and design classes such as iterators and containers. We develop a set of standard traversal operations for standard data structures - doubly linked list and binary tree. Thus, it is necessary to introduce STL here. STL is a set of C++ template classes that provide common programming data structures and functions and it is the most important section of the Standard Library used to reduce the unnecessary programming of standard data structure and algorithms. Among STL techniques, containers and iterators are crucial.

Container classes are classes whose purpose is to contain other objects. That means each container class is a collection class used to contain other objects.

Iterators are the mechanism that makes it possible to decouple algorithms from containers. In STL, algorithms are templates and can be parameterized by the type of iterator. Subsequently, iterators are not restricted to a single type of container.

In [35], Ford and Topp introduce object-oriented data structures and their interfaces using STL. They unify the study of data structures around the concepts of containers and iterators and develop algorithms. The ideas about containers and iterators are directly adopted and necessary alterations are made in the coding.

4.4 Perfect binary tree and doubly linked list

a) Perfect binary tree

The algorithms developed here are based on a so-called perfect binary tree structure, and the discussion of its characteristics will show that such a structure perfectly expresses our world map of nodes.

A binary tree is a tree with at most two children for each node. Fig. 4.2 illustrates concepts such as nodes, root, leaves, left child tree and right child tree.

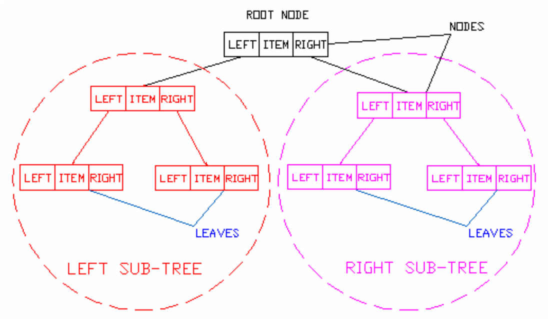


Fig. 4.2 A binary tree

A **perfect binary tree** is a binary tree with all leaf nodes at the same depth. All internal nodes have two children nodes (Fig. 4.3).

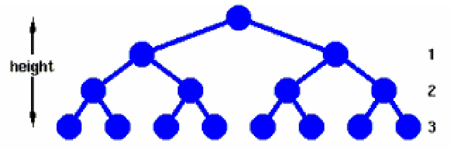


Fig. 4.3 A perfect binary tree

The height of a tree is defined as the number of links from the root to the deepest leaf. A perfect binary tree has an important characteristic about the number of nodes: n. If we set the height of a tree equal to h (Fig. 4.3, h = 3), then, we have:

$$n = 1 + 2^{1} + 2^{2} + \ldots + 2^{h,}$$

So, $$n = 2^{h+1} - 1 \qquad (4.1)$$

And conversely, h = floor (log2n) (4.2)

Typical operations such as deletion or addition of nodes are implemented through the access and modifications of defined node pointers (in C/C++, memory address). Fig.4.4 shows how three nodes are topologically linked.

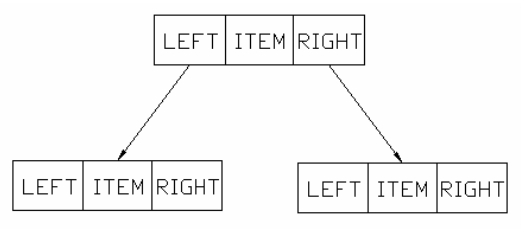


Fig. 4.4 A node of a binary tree and its left pointer and right pointer

b) Doubly linked list

A doubly linked list (see Fig.4.5) is a variant of a linked list in which each item has a link to both its previous item and its next one. This facilitates the convenient access of the linked items backwardly and forwardly.

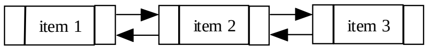


Fig. 4.5 A doubly linked list

A doubly linked list has two basic operations: adding a node and deleting a node. To add a new node (p), four steps are needed (as shown in Fig. 4.6):

1) Let the previous pointer of the new node (p) point to its due predecessor (q).
2) Let the next pointer of the new node point (p) to its due successor.
3) Let the next pointer of its predecessor (q) point to the new node.
4) Let the previous pointer of its successor point to the new node

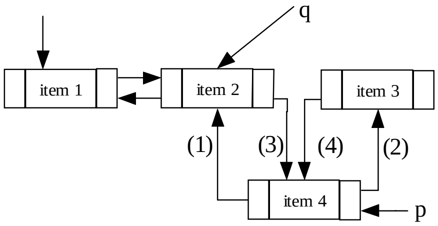


Fig. 4.6 Four steps to add a new node

To delete a node (p), however, only two steps are necessary (shown in Fig. 4.7):

1) Let the next pointer of its predecessor point to its successor.
2) Let the previous pointer of its successor point to its predecessor.

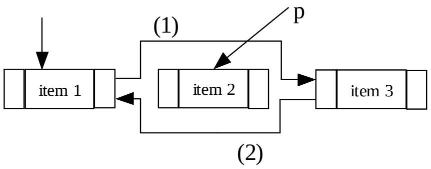


Fig. 4.7 Delete an old node from the list

4.5 Topology of a binary division structure

In order to reduce the order of a full FEM model to a reasonable scale by using a multi-scale mesh, we first need to build up a structure with different levels of elements. For 1D case, we recursively bi-divide the model into different levels of elements and a map of all elements can be formed, which topologically builds up a perfect binary tree (Fig. 4.3). As there are no operations such as addition of new nodes or deletion of old nodes, we can infer from such a structure the following points.

1) The relationships between a node and its children nodes, or, its parent node are fixed, which makes it easy for us to express it using an array.
2) Once we use an array (in C/C++) to express a perfect binary tree (numbering elements in different levels in a suitable order), a fixed mathematical relationship can be inferred between any two nodes.
3) Most importantly, such an array-based expression can completely free us from operations of pointers and thus save us much time and memory.

The following is the process about how we can build up our perfect binary tree

A. First, we sequentially number all the elements as shown in Fig.4.8

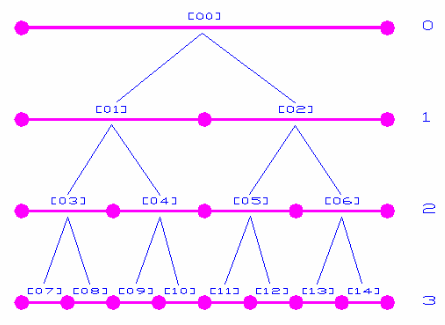


Fig. 4.8 Binary division and its topology

[00]

[01] [02]

[03] [04] [05] [06]

[07] [08] [09] [10] [11] [12] [13] [14]

B. Then, we use an array structure to sequentially store these elements as [00][01][02][03][04][05][06][07][08][09][10][11][12][13][14]

In this way, we successfully use an array to express a perfect binary tree without any redundancy and any memory waste. Thus, we can not only greatly simplify the establishment and initialization of a perfect binary tree, but also tremendously improve the efficiency because we can use array operations instead of pointers. In summary, all elements are topologically organized in a form of a perfect binary tree, but they are stored in an array structure and traversed using indices.

After we know how to build up an array-based perfect binary tree, it is helpful to further list its characteristics. If *n* represents the number of elements, *h* represents division level *h* and m represents vertices. We have:

1) Using formulae (4.1), the relationship between n and h is:

$$n = 2^{h+1} - 1$$

2) The relationship between *m* and *h* is:

At the deepest level, the number of vertices is:

$$m = 2^h + 1 = \text{Number of Leaf Elements} + 1; \tag{4.3}$$

This is the number needed by the leaf elements, however, these vertices are also shared by those elements at higher levels.

3) There exists an explicit parent/children relationship

The following shows how we can access an element's parent and its children or vice versa.

a) For the root node, no parent node.

b) For all the leaf nodes, no children nodes.

c) For internal nodes, if its sequence number in the array is N_n, then, we have:

Left child: $NL = 2 * N_n + 1$ (4.4)

Right child: $NR = 2 * N_n + 2$ (4.5)

Parent node: NP = Integer Part of $(N_n - 1) / 2$ (4.6)

Or: NP = Integer Part of $(N_n - 2) / 2$ (4.6*)

4.6 Fineness function, its calculation and significance

After an array-based perfect binary tree is built, a fineness function is used to reflect relative fineness requirements of different areas in order to get a multi-scale mesh. By this way, fineness requirements are quantified.

In haptic simulation, many factors need to be considered in order to describe relative fineness requirements in different areas. For example, in simple situations, fine meshes are typically used in areas where a haptic tool-tissue interaction occurs or constraints or external forces are applied. However, we also need to consider other factors such as follows:

a) Interactions at several places

b) Different kinds of interactions

c) High curvature for precise visual rendering

d) Complex loading conditions

e) Complex geometric shape and other constraints, etc.

How can we distinguish and quantify (digitize) their relative influences in order to choose whether fine or coarse meshes should be used at different areas? A good answer is to adopt a so-called weight-based method, which often need experts' knowledge accumulated from experiences and lessons. The fineness function plays such crucial role.

4.6.1 Fineness function at the deepest level

How a fineness function is used can be described as follows. Firstly, we assign a fineness function to the whole area of interest. Then, we calculate a fineness value for each element at the finest level. Finally, we use those fineness values calculated at the deepest level to initialize other elements at higher levels. It needs to be pointed out that the fineness values of elements are the results of integer approximation of real values. That is, a fineness value for each element at the deepest level is calculated according to the specified function and then the number will be rounded up to the nearest integer. For example, given the highest division level as 3, the range [0, 80], and the fineness function Y = 0.25 * X (where Y is fineness and X is coordinate), the average fineness for the first element [0, 10] is (0 + 2.5) / 2 = 1.25, which can be rounded to 1. Similar procedure continues untill each element at the deepest level gets its fineness value. Furthermore, all the values have to be normalized within the range [0, *h*] where *h* is the division level. For example, in Fig.4.9, a Gauss function is specified in the meshing area and the division level is five (*h* = 5). There are two cases:

1) For a leaf element, it should be assigned a fineness integer according to its geometric position. However, the fineness function needs to be normalized within the range of [0, 5] if the maximum final fineness value is greater than 5.

2) For a non-leaf element, its fineness value is the largest of its left or right child’s value through an initialization process (in section 4.6.2).

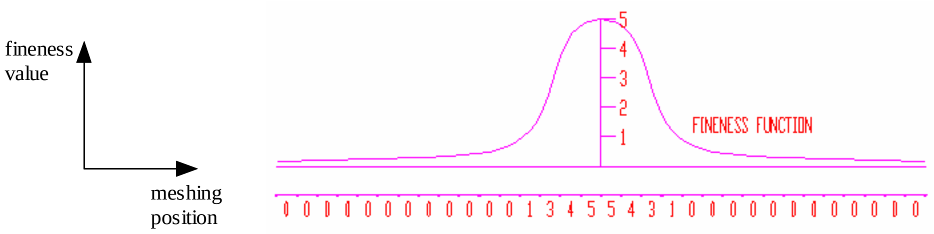
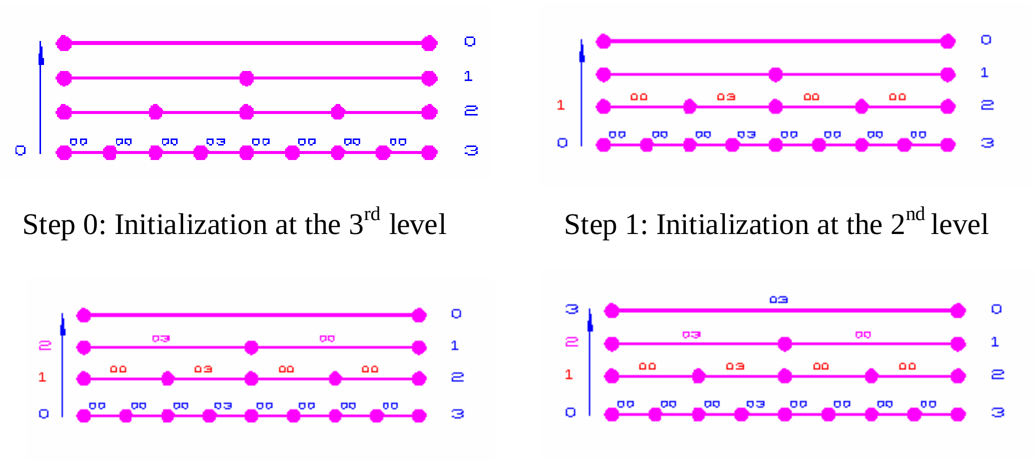


Fig. 4.9 Fineness function of a five-level division

4.6.2 Fineness initialization for the whole perfect binary tree

For all non-leaf nodes at higher levels, a fineness value is assigned according to the following iterative process. From the second deepest level, we “scan” each element and compare the fineness integers of both its left child and right child. The larger of both is chosen as the fineness integer of the current element. The process continues till the root of the tree is traversed. Finally, each element gets a fineness integer that explicitly indicates its fineness requirement. Fig.4.10 shows the whole process.



Step 0: Initialization at the 3rd level

Step 1: Initialization at the 2nd level

Step 2: Initialization at the 1st level

Step 3: Initialization at the 0th level

Fig. 4.10 Fineness initialization of a perfect binary tree

The fineness value is the most important criterion in multi-scale meshing. Once we finish the process of initialization we can start to discuss our algorithms.

4.7 The schematic descriptions of data structures

We will introduce the hierarchical data structures used in representing the perfect binary tree.

Vertex class: used to define the vertices.

Element class: used to define an element.

Doubly linked list class: used in the operations involved in algorithms.

Container class of a doubly linked list: used to maintain a doubly linked list.

Iterator class for doubly linked list container: used to provide a set of standard operations for the objects contained a doubly linked list container.

Binary search tree class: defined as a container class to implement all the operations such as: building up a binary tree, constructing a multi-scale mesh using different algorithms.

4.8 Flow-charts of the execution

4.8.1 Flow chart of initialization: construct a perfect binary tree

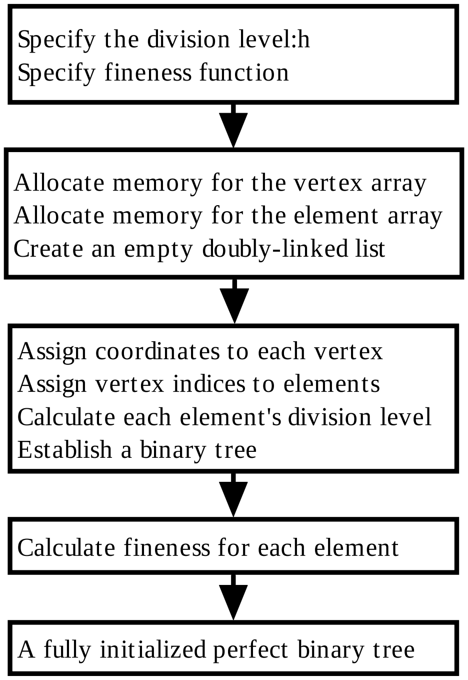
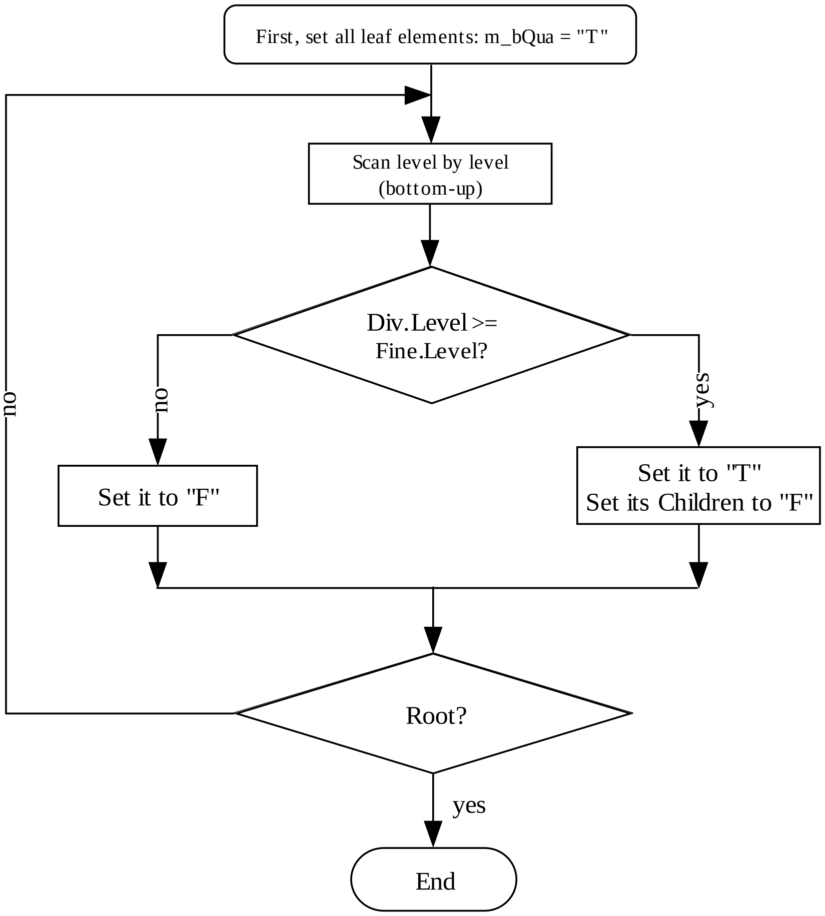

Specify the division level:h
Specify fineness function
Allocate memory for the vertex array
Allocate memory for the element array
Create an empty doubly-linked list
Assign coordinates to each vertex
Assign vertex indices to elements
Calculate each element's division level
Establish a binary tree
Calculate fineness for each element
A fully initialized perfect binary tree

Fig. 4.11 Flowchart of the initialization of a binary tree

4.8.2 Flow chart for generating a rough mesh during initialization

Two very important issues in the proposed algorithms are discussed here. One is on how to quickly get an initial multi-scale mesh and the other is on post-processing. In order to get a rough mesh during initialization, two data members defined in the element class need to be discussed here: m_bQua and m_eChg. m_bQua indicates if the fineness requirement of an element is met and m_eChg is used to reflect the new state after the fineness value of an element is changed, which ultimately determines whether an element is qualified for merging or division.

For all the following discussions and figures, m_iDLev stands for an element's actual division level, and m_iFLev stands for an element's fineness value. Fig.4.12. shows the whole process on how we can get a rough mesh and each of six steps is further explained sequentially from Fig. 4.13 to 4.18. In Fig.4.12, Div.Level (m_iDLev) is the division level of an element, and Fine.Level(m_iFLev) stands for an element's fineness value. The criteria Div.Level >= Fine.Level means the current element is small enough to meet the required precision measured using the current element's fineness value. First, all the leaf elements are labeled as "qualified"(m_bQua = "T"), then, we recursively check each element if an element is too "fine", if it is the case, and the current element is labeled as qualified (m_bQua = "T") and its two children are labeled as "not qualified"(m_bQua = "F"), this process continues till we reach to the root element. That is what Fig. 4.12 shows.



(Div.Level: division level of an element, Fine.Level: an element's fineness value)

Fig. 4.12 Get a rough mesh while initializing

From figures 4.13 to 4.18, the numbers at the left end of each line are the calculation sequence number and each number at the right end of each line stands for the division level of the all elements drawn on that line. The number immediately below an element is the element's division level; the number immediately above an element is the element's final fineness level which can be rounded up and normalized if necessary.

We can properly collect the "qualified" elements as our rough mesh after we recursively (In-Order) "traverse" the whole tree. Then proper post-processing (section 4.8.3) can be carried out to get the initial multi-scale mesh. The detailed process can be demonstrated as follows.

1) Build up and initialize the perfect binary tree.

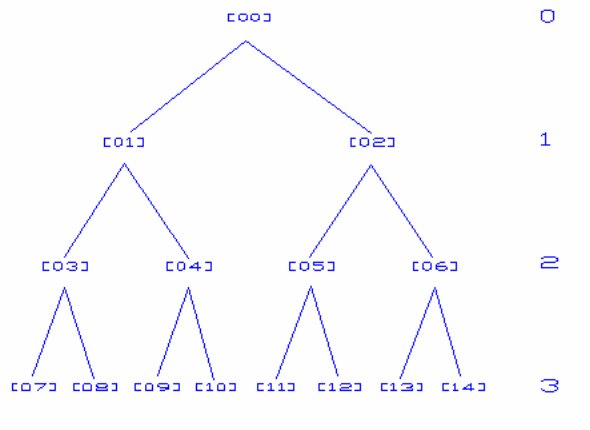


Fig. 4.13 Build up a perfect binary tree

[00] - [14] stand for elements while numbers of 0, 1, 2 and 3 on the right stand for division levels. Lines connecting two nodes stand for parent-children relationships. The perfect binary tree can be initialized as illustrated in Fig. 4.13.

2) Set m_bQua of all leafs at level 3 to "T" (TRUE). See Fig. 4.14, the states of elements [07], [08], [09], [10], [11], [12], [13] and [14] are set to "T".

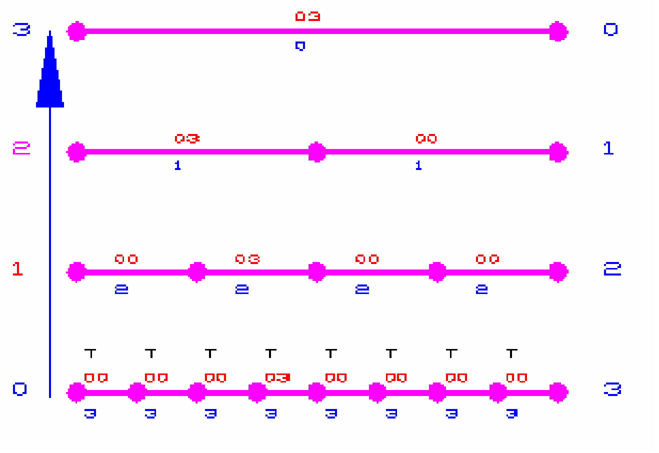


Fig. 4.14 Assign m_bQua = “T” to all leafs

3) Scan the elements at level 2. According to the criterion of m_iDLev >= m_iFLev, [03], [05] and [06] are set to “T”, their corresponding children [07] & [08], [11] & [12], [13] & [14] are set to “F”. However, its children [09] & [10] are untouched since [04] is already set to “F”. The result is shown in Fig. 4.15.

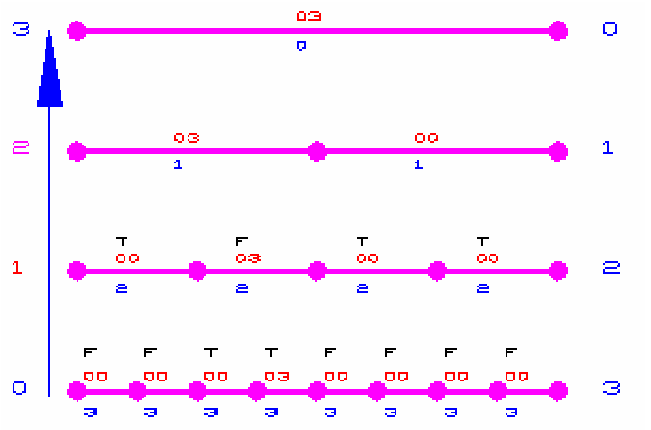


Fig. 4.15 Scan the elements at the second division level

4) Similarly, the elements at level 1 are scanned and [01] is set to “F” with its children untouched; [02] is set to “T” and its children [04] & [05] to “F”. See Fig.4.16.

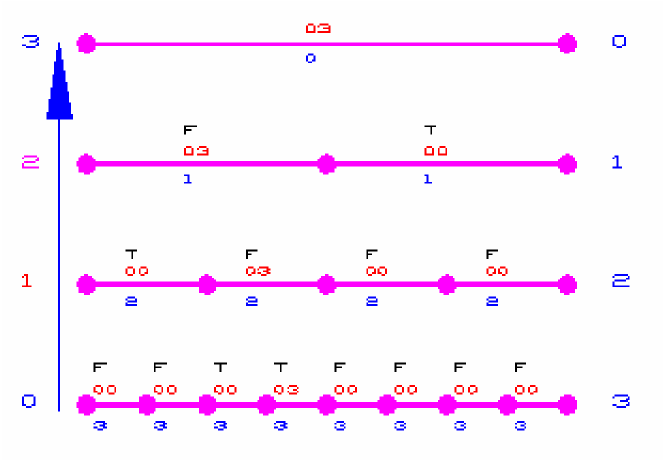


Fig. 4.16 Scan the elements at the first division level

5) We scan the element at level 0 and set its m_bQua = “F”. See Fig.4.17.

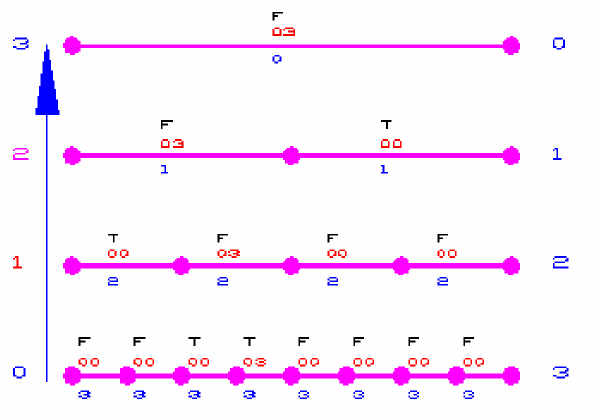


Fig. 4.17 Scan the first element [00]

6) Now if we scan from level 0 to level 3, we can find that all the elements with “T” are just qualified to meet the fineness requirements. Besides, any element (excluding leaf ones) with “T” must have their descendants to be set “F”. With these observations and starting from the first element [00], which is the only element of a double-linked list, the scan procedure stops if element [00] is “T”; otherwise, element [00] should be deleted from the double-linked list and replaced

by its immediate children (from left child to right child). This is a typical recursive process.

After this process reaches the deepest division level, it is guaranteed that all the elements in the double-linked list are with “T” and these elements consist of the mesh area without overlapping. Such a doubly-linked list stands for the roughest mesh because each element in the list stands for the roughest mesh for its mesh area. Thus, the roughest multi-scale mesh for the whole area is obtained as the initial multi-scale mesh. In fact, In-Order Traversal works perfectly to traverse the perfect binary tree and the rough mesh comprised of elements [03][09][10][02] is reaped for further processing.

Fig.4.18 illustrates the post-processing to shorten some big elements according to other requirements. For example, neighboring elements cannot go beyond two levels. As [02]’s m_iDLev is 3 and [10]’s m_iDLev is 1, thus, the final mesh: [03][09][10][05][06] can be obtained after [02] is divided to [05] and [06]. See Fig.4.19.

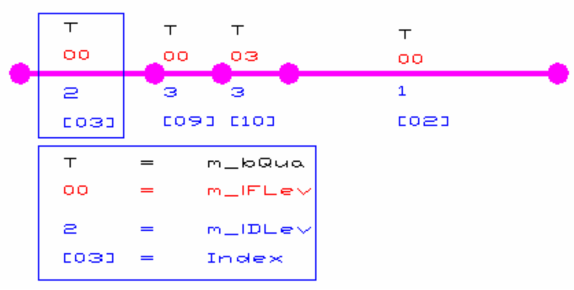


Fig. 4.18 Reap a rough mesh with the In-Order traversal

Numbers (top down): m_bQua, m_iFLev, m_iDLev and the element’s index

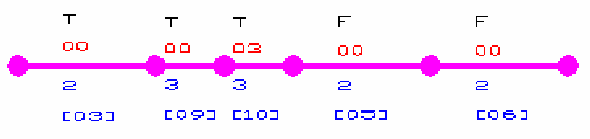


Fig. 4.19 Final multi-scale mesh after post-processing

4.8.3 Post-processing originated from Division algorithm

The rough multi-scale mesh generated from the above initialization should be further processed in order to get the correct multi-scale mesh because there may be "big" level difference between neighboring elements as the results of initializing a rough mesh. For example, big elements appear after division method is applied (Fig. 4.20).

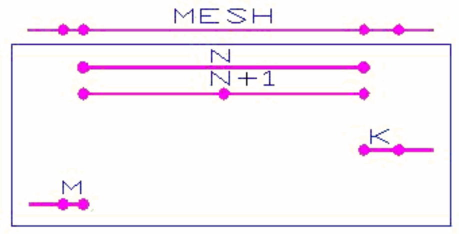


Fig. 4.20 Too-big elements exist

Efficient methods are needed to process such big elements. The post-processing procedure adopted in the current study can be illustrated as follows.

a) The procedures of post-processing

As each element's division level and fineness level are known, the following post-processing steps can be carried out after a rough multi-scale mesh is obtained through the above-mentioned division process.

Check if the level difference between neighboring elements is bigger than 1. If so, the bigger element is divided until the difference is not bigger than one. The formula about how many divisions are needed from a general case can be inferred as follows. Here *M*, *N* and *K* are used to express division levels of elements in a binary tree (Fig. 4.20), we need to know how many sub-division times are needed to meet the boundaries, we assume X times are needed for the left, and Y times for the right.

1. Assume: $M >= K > N$, $M - N > 1$, *and* $K - N > 1$.

2. From the left side of the big element (*N* level), to meet the left boundary, N is sub-divided X times to reach M-1, we have the sequence: *N+1, N+2, …, M-1*, so we reach a formulae: $M\text{-}1 = N+1 + (X\text{-}1) * 1$, thus $X = M – N – 1$.
3. From the right side of the big element (*N* level), to meet the right boundary, *N* is sub-divided *Y* times to reach *K-1*, we have the sequence: *N+1, N+2, …, K-1*, so we reach a formulae: $K - 1 = N + 1 + (Y - 1) * 1$, thus $Y = K – N – 1$.
4. We list the above two formula together, thus:

$$X = M - N - 1 \tag{4.7}$$

$$Y = K - N - 1 \tag{4.8}$$

$$X >= Y$$

5. The division pointing to the element at the *M* level moves leftward (each division from N+1, only the left element is sub-divided) and the division pointing to the element at the *K* level moves rightward (each division from N+1, only the right element is sub-divided). These division procedures at both directions share the same division times: *Y*. Through this division process, the big element is sub-divided into a sub-mesh with bigger elements in the mid and finer elements at both ends. Thus we can get the roughest possible sub-mesh that can still smoothly connect the left and right boundaries.

b) The flowchart of post-processing

The post-processing can be illustrated in a schematic flow-chart (Fig. 4.21), which represents an interesting recursive process.

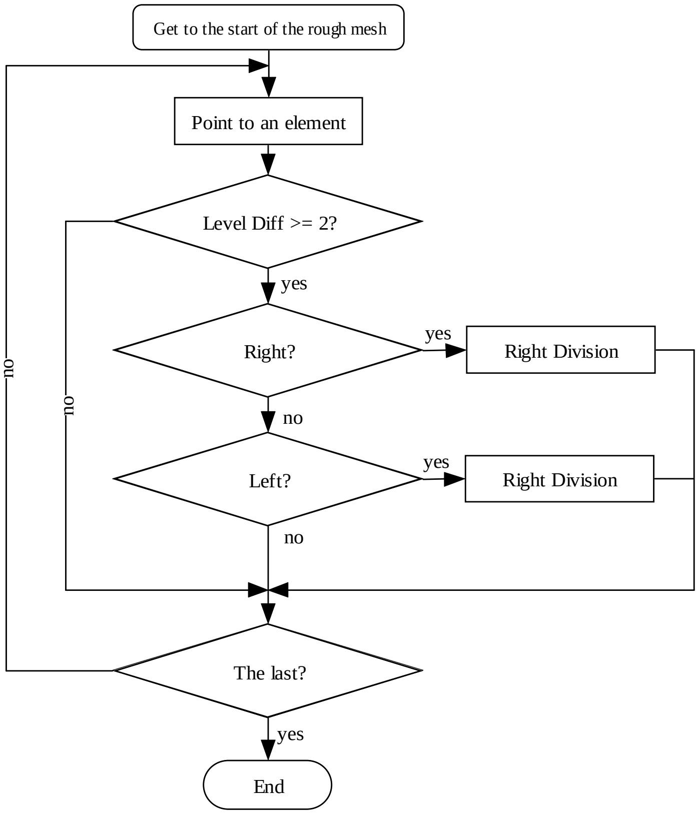


Fig. 4.21 Flow chart of a recursive post-processing

4.9 Summary

Some key issues are introduced in this chapter. They include:

1) Basic ideas about OO programming

2) Standard Template Library

3) Topology of a perfect binary tree

4) The basic characteristics of a doubly linked list

5) The hierarchical topology of the binary division structure used in multi-scale meshing

6) Fineness distribution and its significance in multi-scale meshing

7) Data structure of the proposed algorithms,

8) And schematic flow-charts of key procedures.

These issues lay a solid and necessary foundation for the algorithms that will be developed in the coming chapters.

5 Algorithms of Multi-Scale Meshing

5.1 Introduction

In this chapter, detailed description of the three multi-scale algorithms, e.g. Division, Merging and Dynamic and their comparisons are presented. A 1D example is used to illustrate the most fundamental issues on 1) how a multi-scale mesh can be generated in different ways; 2) how each algorithm can be used for different purposes; and 3) what their fundamental differences are. The chapter is organized as follows. In section 5.2, the criteria used in algorithms are discussed. In section 5.3, Division algorithm and its post-processing are explained. In section 5.4, Merging algorithm is developed. In section 5.5, Merging algorithm and Division algorithm will be compared. In section 5.6, Dynamic algorithm is developed and discussed. And, finally, a summary is given in section 5.7.

5.2 Criteria for Division algorithm and Merging algorithm

In the process of a multi-scale meshing, when an element is divided or two elements are to be merged, the following criteria should be used.

a) When two elements are to be merged, only sibling elements can be combined. That is to say, only the elements that have the same parent element in the perfect binary tree can be merged. Such a criterion can effectively prevent algorithms from creating those meshes that meet the requirements of fineness or level difference but are not optimal in maximally reducing the order of the modeled system. It can be shown in Fig. 5.1. If only sibling elements might be merged, a multi-scale mesh labeled with “GOOD ONE” can be uniquely obtained. Without such a criterion, the mesh might not be unique and a multi-scale mesh labeled with “BAD ONE” in Fig.5.1 could be obtained,

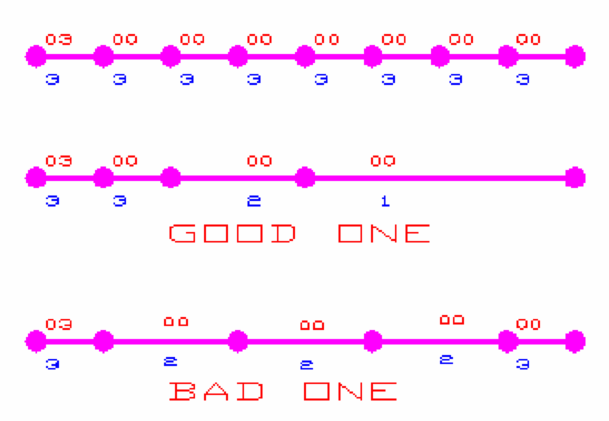


Fig. 5.1 Merge siblings or randomly merge neighboring elements

b) When a multi-scale mesh is built, the level difference of any two neighboring elements cannot go beyond one. That is, the level difference between any two of neighboring elements should not be greater than one. The basic purpose of this constraint is to avoid over-drastic changes. For example, in Fig. 5.2, the mesh in the second row is not good according to criterion (b).

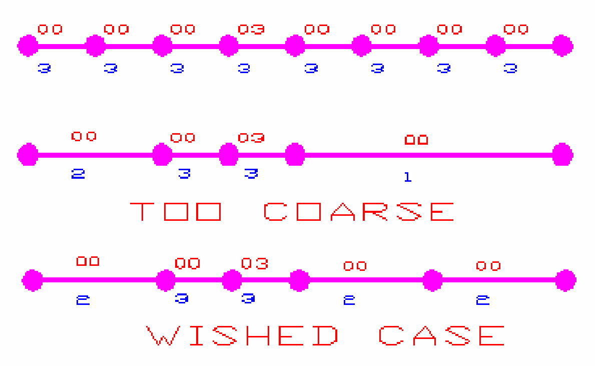


Fig. 5.2 The level difference of neighboring elements should not be greater than 1

With such a criterion, it doesn’t matter whether a continuous or a discrete fineness function is used to describe the fineness requirements of neighboring elements. The smoothness of the final mesh can always be guaranteed by this criterion.

c) The decision to divide an element or merge two sibling elements is based upon the following criteria. Here, m_iDLev stands for the division level of the current element and m_iFLev stands for the fineness value of the current element. Given a fineness function, the value of m_iFLev is calculated according to section 4.6). There are two cases.

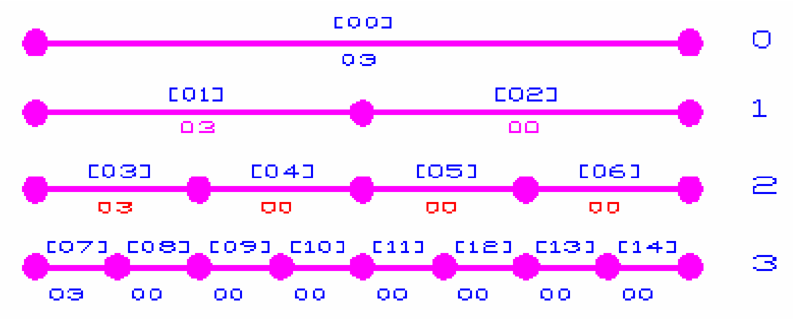
   1) If the division level of the current element is greater than its fineness value, e.g., m_iDLev > m_iFLev, the element is qualified to be merged with its sibling element. The idea can be illustrated in Fig. 5.3, where the number in parenthesis denotes the element index.

      i) At level 0 (the first row), element [00] is not qualified as its division level value (m_iDLev = 0), which is indicated at the end of each line, is smaller than its fineness value (m_iFLev = 3) indicated right below the line.

      ii) At level 1 (the second row), element [02] is qualified as m_iDLev = 1 > m_iFLev = 0; element [01] is not qualified as m_iDLev = 1 < m_iFLev = 3.

      iii) Similarly, at level 2 (the third row), elements [04], [05], [06] are qualified but [03] is not because m_iDLev = 2 < m_iFLev = 3.

      iv) At level 3 (the fourth row), only [07] is not qualified for merging.



[n]: element index, n = 00, 01, …, 14

m: fineness value (m_iFLev), m = 00 or 03

k: division level (m_iDLev), k = 0, 1, 2, 3

Fig. 5.3 Criteria for merging or division

2) If the division level of the current element is less than its fineness value, the element must be divided. For example (see Fig.5.3):

   i) At level 0 (the first row), element [00] must be divided as its m_iDLev = 0 < m_iFLev = 3.

   ii) At level 1 (the second row), element [01] must be divided as its m_iDLev = 1 < m_iFLev = 3.

   iii) Similarly, at level 2 (the third row), element [03] must be divided as its m_iDLev = 2 < m_iFLev = 3.

   iv) At level 3 (the deepest level), no division is possible or needed.

5.3 Division algorithm and its post-processing

5.3.1 Basic procedures

According to the above-stipulated criteria, the Division algorithm is described as follows using a perfect binary tree with a division of four.

1. Specify the division limit and build up the perfect binary tree;
2. Specify a fineness function;
3. Calculate a fineness value for each element in the deepest level;

4. Iteratively calculate and assign a fineness value for all other elements;
5. Push the first element [00] into the doubly linked list as the initial mesh and check if all the criteria are met:
   A. If so, finish the process and get a mesh with only one element;
   B. If not, locate the element [00] and pop it out (delete it) from the list and sequentially (from left to right) insert its two children elements [01] & [02] in the list.
6. Scan the newly updated double-link list from the first element till the last one (from left to right) to check:
   A. If the current element needs no division, move to the next;
   B. If the current element needs division, delete it from the list and sequentially (from left to right) insert its two children elements into the original position of the element.
7. The process of step 6 is repeatedly carried out until neither deletion nor addition operation can be further applied to the list.
8. The element sequence in the doubly linked list constitutes the initial mesh.
9. Post-process the rough multi-scale mesh.
10. Finish the process and get the final mesh stored in the doubly linked list.

5.3.2 Critical criteria and a typical example

The core criteria used here are:

a) If the element is at the deepest level or its division level (m_iDLev) is greater than its fineness value (m_iFLev), no division is needed and the element is left untouched in the list.

b) If an element's fineness value is greater than its division level, the element must be divided and its two children will sequentially replace it in the list.

c) Post-processing should be started if either of the following two situations is met:

   1. The list has been scanned exactly *h* times.

   2. No deletion or insertion occurs during the whole process of scanning. If this happens, it shows that all elements in the list have already met the division requirements even if the scanning times are less than h.

An example of Division algorithm is illustrated in figures 5.4 to 5.8.

a) Put the first element [00] with a fineness value of 03 into the list. An initial mesh with only one element of [00] is thus generated.

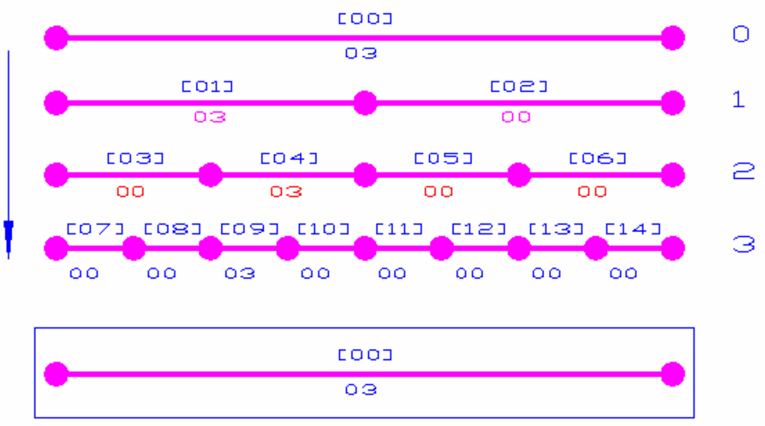


Fig. 5.4 Put [00] into the list

b) Division must be performed because m_iDLev = 0 is less than m_iFLev = 3. Thus [00] is replaced by its two children: [01] & [02]. After the left-to-right scanning is finished, the new mesh with [01][02] is shown in Fig. 5.5.

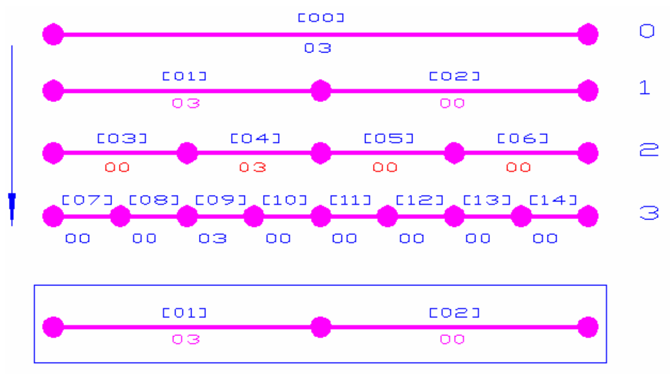


Fig. 5.5 Delete [00] and insert [01] and [02]

c) Scanning procedure starts from the leftmost element of the list. Element [01] is replaced by [03] & [04] because its m_iDLev = 1 is smaller than its m_iFLev = 3; element [02] remains unchanged. The updated mesh [03][04][02] is thus generated as shown in Fig.5.6.

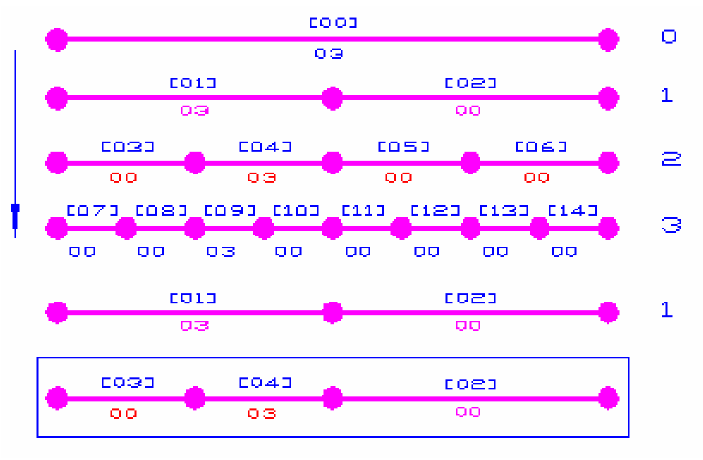


Fig. 5.6 Delete [01], insert [03] and [04], and keep [02]

d) The updated list is scanned again from the leftmost element.

1) Check [03] and it remains unchanged.

2) For [04], there exists m_iDLev = 2 < m_iFLev =3; [04] is deleted from the list and its two children [09] & [10] are sequentially inserted.

3) Check [02] and it remains unchanged. Finally, a new mesh [03][09][10][02] is generated in Fig.5.7.

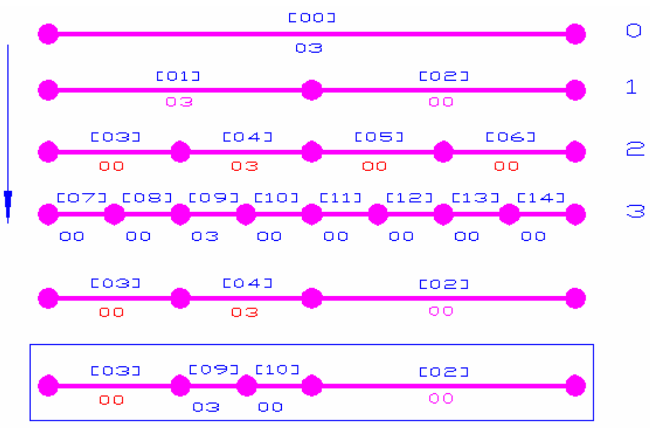


Fig. 5.7 Keep [03], replace [04] using [09] and [10], and keep [02]

Since the next level will be the deepest level, scanning procedure must stop. Thus, the process moves to post-processing.

e) For the post-processing, the generated list is re-scanned. A so-called "Shorten" procedure is applied to all of those elements that might be too "big" when compared with their neighboring elements. For example, since the level difference (absolute value) between [02] and [10] is bigger than one, element [02] is deleted from the list and its children [05] and [06] are sequentially inserted into the list. Post-processing stops when neither deletion nor insertion can be made. Finally, we get the final multi-scale mesh [03][09][10][05][06] as shown in Fig. 5.8.

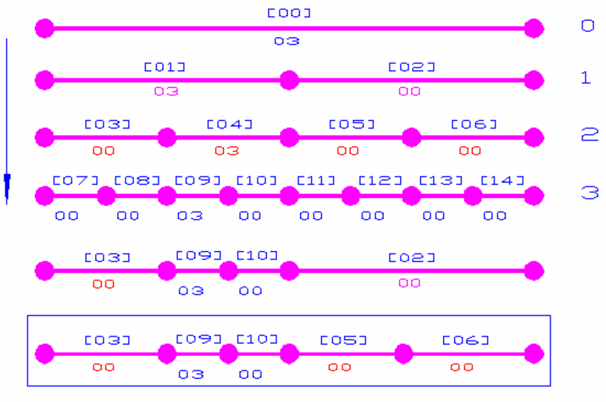


Fig. 5.8 [02] is further divided through a so-called "Shorten" post-processing

5.3.3 Discussion

The Division algorithm reveals the basic idea on how to construct a multi-scale mesh through coarse elements while the smoothness of the mesh is guaranteed by the continuity of the used fineness function and the above-mentioned post-processing. That means two things:

1. The fineness requirement of each element must be met.
2. Big elements are further divided through post-processing.

The post-processing in different situations can be discussed in more details. Let us assume that the scan direction can be carried out either from left to right or from right to left, five situations may occur to a big element.

1) If a big element is constrained between two smaller elements in deeper levels, the division action will proceed in two directions until the level difference requirements of its neighbors are meet. The case is illustrated in Fig. 5.9, where the element: [04] is constrained by two much smaller neighbors, [38] and [23]. In Fig.5.8, the first row stands for the fineness distribution; the second row is the initial mesh obtained before post-processing; the third row is the final mesh after post-processing. The post-processing is described as follows.

   a) Firstly, [04] is divided leftward and rightward.

   b) The leftward division continues until the mesh adapts to [38], and the rightward division carries on until the mesh adapts to [23].

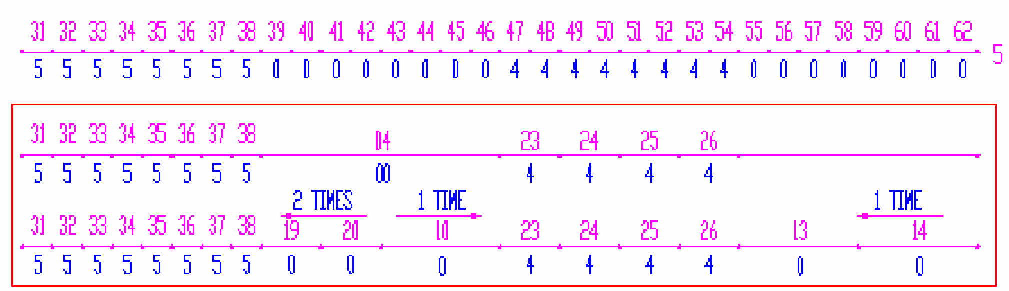


Fig. 5.9 [04] is constrained by [38] and [23]

2) If an element needs division according to its left neighbor but its right neighbor (an even big element) prevents the division, the current element must be divided to meet the requirements of its left neighbor. And the right neighbor will be further divided to meet the newly updated mesh in the same run of the scanning of the post-processing procedure. The case is shown in Fig. 5.10, where [10] is constrained by a much smaller left neighbor [42] and an even bigger right neighbor [02]. Here, the first row stands for the fineness distribution; the second row is the initial mesh before post-processing; the third row is the result before post-processing of [10]; and the fourth row is the result after post-processing [02].

   a) First, [10] is divided to adapt to [42].

   b) Then, [02] is further divided until it adapts to [22].

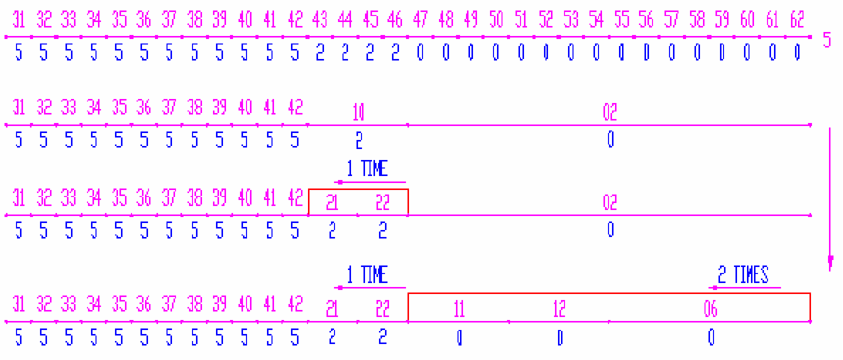


Fig. 5.10 [10] is directly divided according to [42]

3) If an element needs division according to its right neighbor but its left neighbor (an even bigger left element) prevents this division, the current element needs to be divided to meet the requirements of the right neighbor. The left neighbor will be further divided to meet the newly updated mesh through post-processing. It is similar to the case shown in Fig. 5.10.

4) If the leftmost element needs to be divided according to its right neighbor, the division should be carried on from the left to the right. Such a case is shown in Fig.5.11.

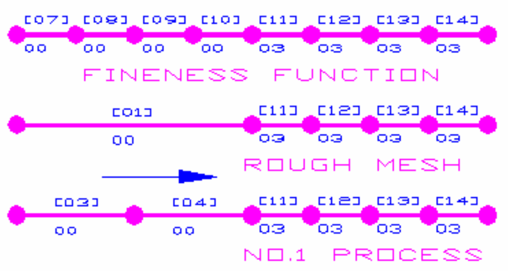


Fig. 5.11 Post-process a big head element

5) If the rightmost element needs division according to its left element, the division should be carried on from the right to the left. The procedure is shown in Fig.12.

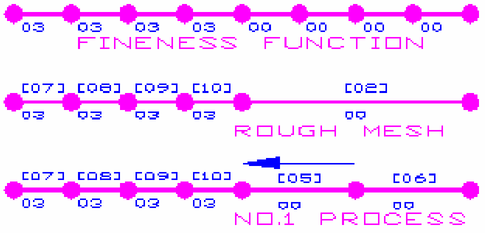


Fig. 5.12 Post-process a big tail element

5.4 Merging

5.4.1 Basic procedures

The Merging algorithm can be described based on the criteria stipulated in section 5.2.

1) Specify the division limit and build up the perfect binary tree
2) Specify a fineness function
3) Calculate a fineness value for each element in the deepest level
4) Iteratively initialize all other elements till the root element [00].
5) Theoretically, all the leaf elements at the deepest level should be sequentially pushed into the doubly linked list and be checked. However, this operation is very time consuming. Thus, the process of merging (from left to right) directly starts from the deepest level. This strategy can save much time and memory because the number of elements at the deepest level is more than half of the total number of the elements in the perfect binary tree. The procedure is described as follows.
   a) If the current element can be combined with its sibling, their parent element should be put into the list instead of the current element and its sibling.
   b) If the current element cannot be merged with its sibling, the current element and its sibling should be sequentially inserted into the list.
6) After the first run of merging, re-check the mesh.
   a) If the current element can be merged with its sibling, the current element and its sibling are deleted and their parent element will be inserted.
   b) If no merging happens to the current element, then, move one position when the next element is not its sibling or two positions if the next is its sibling.
7) Repeat step 6) until no any deletion or addition operation is applicable to the list.
8) Get the multi-scale mesh.

5.4.2 Basic criteria

The following criteria are essential to guarantee the correctness of the final multi-scale mesh.

a) The whole list should be scanned completely from the first to the last element.

b) Only siblings can be merged.

c) At any time, the difference of the division level of the neighboring elements must not go beyond 1. That means that when two siblings are about to be replaced by their parent, the level difference between their parent element and the predecessor of the left sibling or the successor of the right sibling is either zero or 1. Two special cases occur here:

   1. If two siblings are the first two elements (head), only successor is considered.

   2. If they are the last two elements, only predecessor needs to be considered.

   An example is used to illustrate the merging process. After the initialization of a perfect binary tree, the elements at the deepest level are sequentially checked and the merging operation starts simultaneously. With the above-mentioned criteria:

1) [07], [08] are directly inserted into the list;

2) [09], [10] can be merged and thus, their parent [04] is inserted into the list;

3) Similarly, [05], [06] are inserted the list.

4) An initial multi-scale mesh is generated from the first run of merging as shown in Fig.5.13.

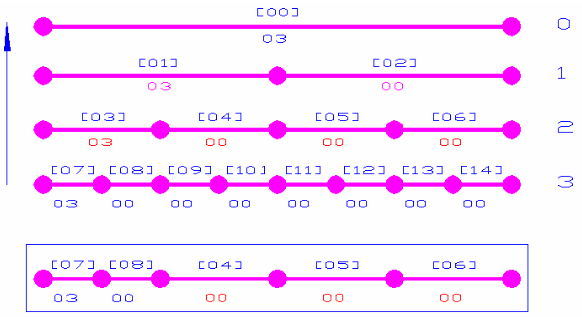


Fig. 5.13 Start merging

5) Scan the newly generated mesh from left to right. [05], and [06] are merged and thus deleted from the list but their parent [02] is inserted. See Fig.5.14.

6) Repeat step 5 until no deletion or insertion is carried out. And the final multi-scale mesh will be generated as shown in Fig.5.14. The stop conditions for merging include 1) neither deletion nor insertion occurs; 2) h (the division limit or the height of the tree) times of merging have been performed.

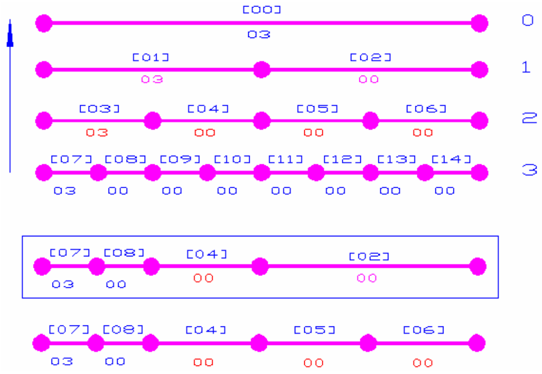


Fig. 5.14 Delete [05], [06] and insert [02]

5.4.3 Discussion

The Merging algorithm is very straightforward as it can guarantees the mesh smoothness and correctness by using all criteria at each step of merging. When merging happens, smoothness and fineness requirements are strictly satisfied. So, any element in the final multi-scale mesh will be the neighbor of either its sibling or an element with division level exactly one level higher or lower.

5.5 Comparisons between Merging and Division algorithms

5.5.1 The different strategies

A crucial criterion in Merging algorithm is that when the condition (m_iDLev = m_iFLev) is satisfied, the merging process must be stopped no matter whether other criteria permit or not. Furthermore, all criteria must be satisfied simultaneously in order to merge two siblings and each element can reach at most to the level where the condition (m_iDLev = m_iFLev) is satisfied.

In Division algorithm, an element must be divided until at least m_iDLev = m_iFLev is satisfied. Division should never be stopped when m_iDLev < m_iFLev maintains. However, division is actually stopped once m_iFLev = m_iDLev is met unless other criteria require for deeper division.

5.5.2 The identical multi-scale mesh

Both algorithms achieve the same purpose - building up a multi-scale mesh to reduce the order of the model. Now that the same function is used to describe the fineness requirements of our domain of interest and other criteria are the same, the final multi-scale mesh obtained using these two algorithms should be identical although their computation time and the required computing resources are different.

5.5.3 Performance comparison

The reason to develop different algorithms for the same problem is that different optimal goals may be required in different situations. For a multi-scale mesh, the dominating factor is the time needed to get and update a multi-scale mesh. And the efficiency of algorithms is greatly influenced by the fineness distribution. Two typical kinds of functions are used for the purpose of comparison.

1. Linear fineness function

Assuming a linear fineness function is required, some special cases which will directly lead to even FEM meshes are illustrated as A, B, C and D in Fig.5.15.

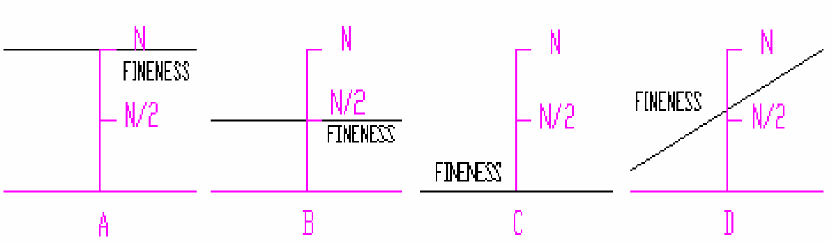


Fig. 5.15 Linear fineness function

a) For case A, all elements require the finest division and a conventional finite element mesh will be obtained. Merging algorithm shows the best performance while Division algorithm performs worst. In this case, a multi-scale mesh degenerates to a conventional FEM mesh. It also reflects the general idea that a fineness function can be adopted for different situations.

b) For case B, an even FEM mesh with middle-scaled mesh can be constructed. There should be no obvious advantages/disadvantages between two algorithms.

c) For case C, an even FEM mesh with only one element will be constructed. It is the best case for Division algorithm and the worst case for Merging algorithm.

d) For case D, it is a general linear case. The efficiency depends upon the density of fine elements. If fine elements dominate, Merging algorithm will be a better choice. Otherwise, Division algorithm will be better.

2. Gauss fineness function

Special cases assuming Gauss fineness function are illustrated in Fig. 5.16 as cases A, B, and C.

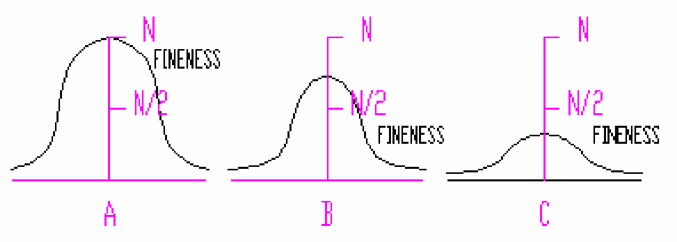


Fig. 5.16 Gauss fineness function

a) For case A, the elements having relatively high fineness requirements constitute the majority, so Merging algorithm is supposed to be of some advantages over Division algorithm.

b) For case B, neither the elements with relatively high fineness requirements nor the elements with low fineness requirements dominate, so two algorithms are supposed to be equally applicable.

c) For case C, Division algorithm should have some advantages over Merging algorithm as the elements with low fineness requirements dominate.

3. Other fineness functions

For other fineness functions, the performance of algorithms depends on real situations although the main criterion is the relative ratio of fine and coarse elements.

5.6 Dynamic algorithm

5.6.1 Introduction

The main reason for developing multi-scale algorithms is to reduce the order of a system and thus save valuable time in real-time haptic simulation. In some surgical training systems, when a surgeon is slowly operating on tissues, a change from one state to another is often minor. This observation inspires a new idea of dynamically generating a new mesh from the previous one according to the changed fineness requirements with less time or resources.

When a system slightly changes from one state to another, two fineness functions are needed to reflect these two states. But, there are two totally different ways to view such a change:

a) Firstly, we might think that now that we have already gotten the old mesh and fineness function changes only slightly, it is natural for us to infer if we can get our new mesh from the previous one. Such an idea leads to our Dynamic algorithm: a new mesh evolves from a previous mesh.

b) Also, we could think that two situations have nothing to do with each other. So we could get our mesh by Division algorithm or Merging algorithm, and a new mesh is solely generated out of the new fineness function and has nothing to do with its previous one.

5.6.2 Assumptions and performance measures

Some assumptions are made to validate the proposed Dynamic algorithm.

1. The algorithm is applied to the situation where the change from one state to another is minor (less than 1% or 5% or 10%).
2. An algorithm can be validated through comparing its results to the ones from known correct algorithms. That is, the meshes obtained using Dynamic algorithm should be the same as the ones obtained using either Merging algorithm or Division algorithm.
3. The advantages of the new algorithm should include its efficiency in computation time, at least in the applied situations.

5.6.3 Validation of Dynamic algorithm

Since a new mesh evolves from the old one, it typically involves the process of merging or dividing some elements in the old mesh. Ultimately, such Dynamic algorithm should combine the ideas that are used in both Division and Merging algorithms.

Whichever algorithm is used, under the stipulated criteria, different algorithms should produce the same mesh. The principle of Dynamic algorithm is rather straightforward: if we check the old multi-scale mesh, we will find that with the updates of the fineness values of the elements, some elements are qualified to be merged with its sibling (if exists) and some have to be further divided into smaller elements. For any element in the final mesh, there are only two cases for any element in an old mesh:

1. For a leaf element at the deepest level, its final fineness value should directly be updated according to the new fineness function. For a non-leaf element, its value should be updated to reflect its new fineness values of its two descendants. However, whether the current element needs division or not depends on the following condition. If m_iFLev is still less than m_iDLev, the current element should remain the same; otherwise, it should be further divided. This is determined from the recursive process of fineness value update.
2. If there is an element whose two descendants get new fineness values that are less than their old values, then the new fineness value of the element will get less. Again, whether the current element will be merged depends on other factors.

In summary, if the previous multi-scale mesh cannot meet the new requirements of the updated fineness function, m_iFLev <= m_iDLev will ultimately not be hold for some elements. A new multi-scale mesh can be generated from the old one according to the changed fineness values. The approach is advantageous if it can effectively save computation time.

5.6.4 Basic procedures

The criteria are exactly the same as those discussed in section 5.2. The procedures of Dynamic algorithm are described as follows.

1) Specify the division level and build up the perfect binary tree.
2) Specify a fineness function to be used.
3) Calculate a fineness value for each leaf element.
4) Iteratively calculate a fineness value for all other elements till the first element.
5) Choose Division or Merging algorithm to build up an initial mesh. The choice of algorithms is decided by the actual situations.

6) When the fineness function is slightly changed, we need to calculate the updated fineness values and mark out elements with a new value. Whether an element is qualified for merging or division depends on four situations. Here m_eChg is used to record a new state.

    a) If a new value is less than the old one and m_iDLev is greater than the new value, its m_eChg will be set to COARSER, which means the element could be merged. If it is the element [00], its m_eChg will be set to TOP (no more merging).

    b) If the new value is greater than the old one and m_iDLev is less than the new fineness value, its m_eChg will be set to FINER, which means it must be further divided. However, if the current element is a leaf, its m_eChg will be set to BOTTOM (no more division).

    c) In other cases, though the value is changed, however, if it is neither case A nor case B, its m_eChg should be only set to CHGD which means the value is changed but no explicit decision can be made so far.

    d) If the value doesn't change in two states, m_eChg keeps the value: "SAME".

7) Once each element of the tree is checked and marked, the elements in the old mesh should be checked according to two situations.

    a) If the m_eChg of an element is COARSER, we immediately check whether it can be merged with its sibling (if exists in the mesh). If so, both the element and its sibling should be deleted from the list and their parent is inserted into the same position. In practice, it is usual to merge this element with its sibling first and temporarily ignore other criteria. The post-processing will ultimately be used to treat possible big elements.

b) If m_eChg of an element is FINER, is must be immediately further divided. And, step 7) should be repeated until no insertion or deletion operations occur.

8) Post-process big elements. The big elements should be divided if they are adjacent to small elements.

5.6.5 Examples

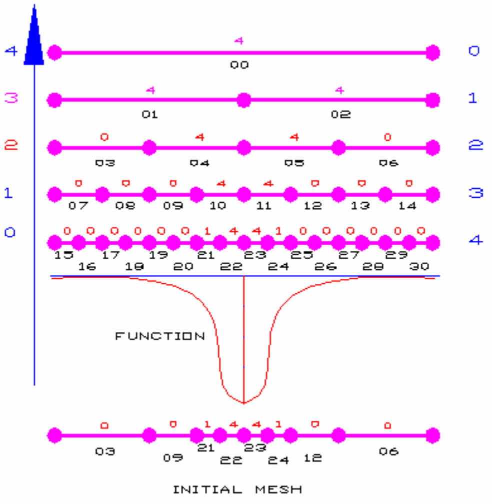
A perfect binary tree with division limit 4 is used to illustrate the intrinsic process of Dynamic algorithm.

A. First, a mesh is generated according to the fineness distribution by Merging or Division algorithm. It is [03][09][21][22][23][24][12][06] as shown in Fig.5.17.

B. Then, the fineness function is slightly changed (we exaggerate the change for the purpose of illustration) and the fineness distribution of the perfect binary tree is updated. The process is reflected in Fig.5.18.

C. After comparing the old and new values of each element, all the affected elements are marked in the whole perfect binary tree, which is shown in Fig.5.19.

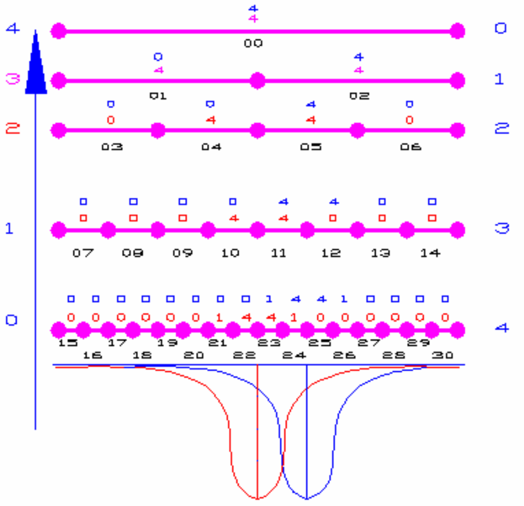
D. If the change of fineness distribution will result in a new multi-scale mesh, this change will be certainly reflected in the fineness values of the initial mesh: some must be further divided and some can be merged. Fig.5.20 is used to show the fineness changes and how these changes are reflected in the values of the initial mesh. In Fig.5.20, [24] is at the deepest level and no further division is possible, so [24] is not qualified for division. Fig.5.21 shows that, in the initial mesh, some elements must be divided and some could be merged.



(The number above en element: fineness value)

(The number below en element: element index)

Fig. 5.17 Original fineness distribution and its initial mesh



(Old value: the number in magenta and above en element)

(New value: the number in black and above the old)

Fig. 5.18 New and old fineness values of each element

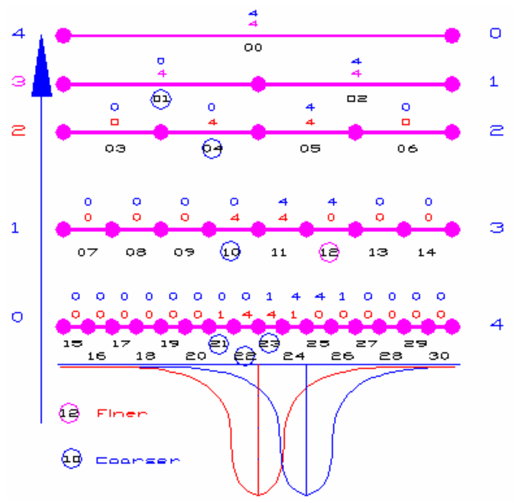


Fig. 5.19 Qualified elements are marked out in the tree

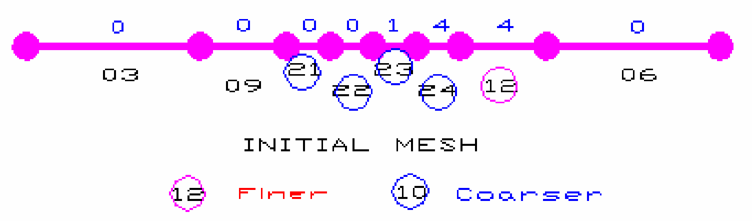


Fig. 5.20 Elements with changed values (the old mesh)

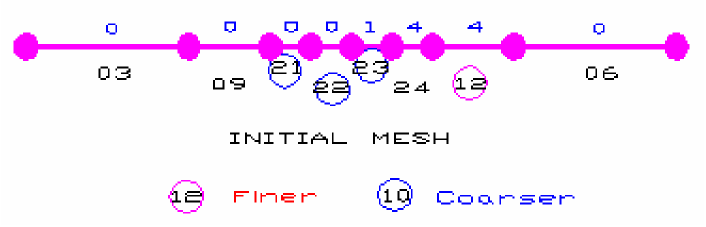


Fig. 5.21 Qualified elements after [24] is excluded (the old mesh)

E. Once the tree is updated with markers. Scanning process starts and merging or dividing operations are applied to appropriate elements.

1) Merge an element with its sibling if it is marked with COARSER. Elements [21], [22] are merged to [10] as shown in the second row of Fig.5.22.

2) Divide an element if it is marked with FINER. The element [12] is divided into [25] and [26] as shown in the second row of Fig.5.22.
3) Repeat steps 1) and 2) until no deletion or insertion operations can be applied (the third row of Fig.5.22).
4) Perform post-processing until there is no deletion or insertion operation. Element [06] is further divided into [13], [14] as shown in the fourth row of Fig.5.22.
5) Finally, a new mesh that reflects the new fineness distribution is generated and the m_eChg should be reset.

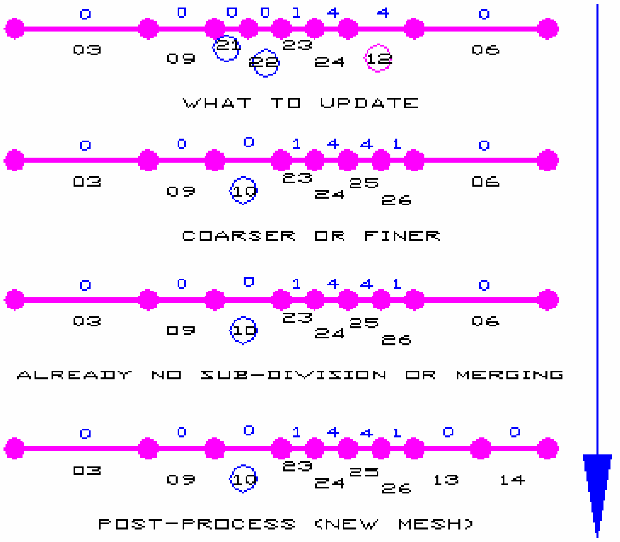


Fig. 5.22 Dynamically evolve from the initial mesh to the new one

From the above-mentioned steps and what is shown in Fig.5.22, it should be noted: 1) for the “COARSER” case, it simply indicates a possibility for merging; 2) but for the “FINER” case, it requires the immediate division operation. This suggests that slight different Dynamic algorithms can be further differentiated. For example, we can also do the following to obtain the same mesh.

1. First, scan the list and divide all elements marked with “FINER” until no deletion or insertion can be applied.

2. Scan the latest list and merge qualified elements marked with “COARSER” and temporarily ignore other criteria.
3. Perform post-processing until no deletion or insertion can be applied.

5.6.6 Discussions

The following situations may be encountered during the above-mentioned process.

a) If the change is very small, it is possible that neither merging process nor division process is needed.

b) Normally, with minor changes to the system, the operations in Dynamic algorithm are less than those if Merging or Division algorithm is used.

c) The correctness of one algorithm can be validated through comparing its results with those obtained using known correct algorithms. The following example shows that the same results can be obtained through either Merging or Division algorithm.

Use Merging algorithm to get the same new multi-scale mesh.

a) Fig.5.23 shows the result of the initialization of the perfect binary tree using a new fineness distribution.

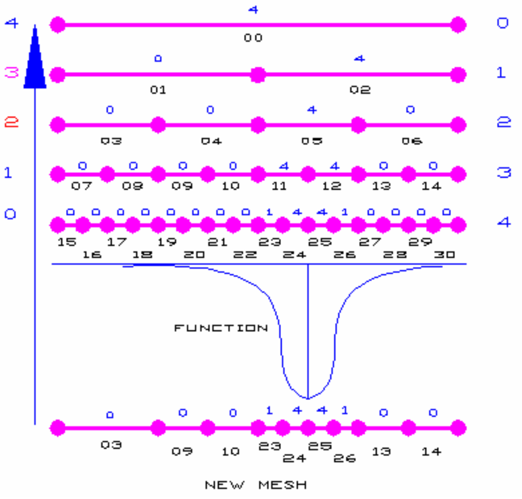


Fig. 5.23 New fineness function & new mesh initialization

b) Fig.5.24 illustrates the merging process by which a new mesh is obtained.

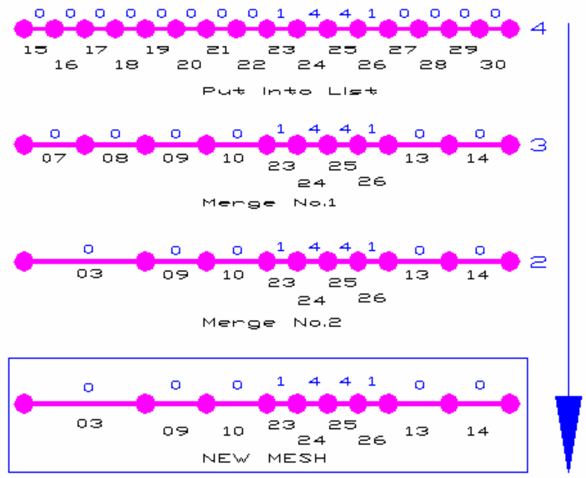


Fig. 5.24 The merging process using the new function

Use Division algorithm to get the same new multi-scale mesh.

1) Firstly, initialize a mesh as shown in Fig.5.23.

2) Apply Division algorithm to obtain a new mesh as shown in Fig.5.25.

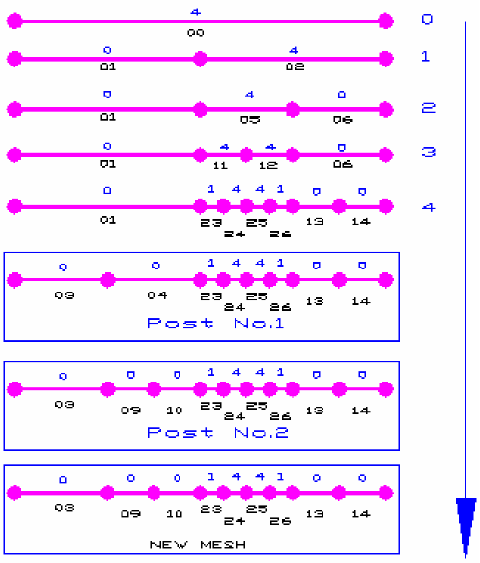


Fig. 5.25 The division process to get the new mesh

From the examples shown above, the same multi-scale mesh can be obtained from three fundamentally different algorithms - Dynamic, Merging and Division algorithms. The comparison of their performance will be presented in Chapter 6.

5.7 Summary

In this chapter, the procedures of three algorithms are illustrated in detail. They are Merging, Division and Dynamic algorithms. The basic ideas and criteria used to obtain a multi-scale mesh are explained and validated. Two special techniques used in the above algorithms are also presented. They are post-processing used in Division algorithm and Dynamic algorithm and fineness function used to reflect the relative fineness requirements. All of the three algorithms can generate the same result.

6 Simulation Results & Discussions

6.1 Introduction

In the previous chapters, the basic ideas of all three algorithms are presented and the detailed processes to get a multi-scale mesh are described. In this chapter, the simulation results are shown to 1) verify whether the multi-scale meshes are identical using merging, division, and dynamic algorithms, 2) investigate the crucial role of fineness function in deciding which algorithm should be applied, and 3) compare computation times used for different fineness functions.

The organization of this chapter is as follows. In section 6.2, the simulation environments and conventions are defined. Then, simulation results and their comparisons are presented in section 6.3. Next, some comments about the significances of all three algorithms are presented in section 6.4. Finally, a summary is made in section 6.5.

6.2 Simulation environments and conventions

All the results are obtained using the computer with a 1.9Ghz Pentium 4 Processor CPU and 256MB memory. Also, we use the following conventions:

1. When a division is denoted as X, it stands for two to the power of X. For example, if the division is 5, that means the number of elements is 32 (2^5) at the deepest level.

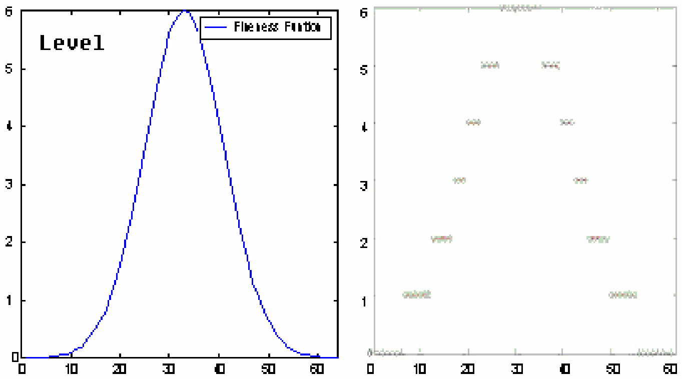
2. A slim and horizontal bar with a length of 1000.0 (mm) is used to illustrate the one-dimension fineness function distribution.

6.3 Simulation results and comparisons

In sections 6.3.1 and 6.3.2, it is verified that Merging and Division algorithms can give identical multi-scale meshes. Their efficiency is compared. The efficiency comparisons of three algorithms are made in section 6.3.3 when the fineness functions are only changing slightly.

6.3.1 Identical meshes obtained using Merging and Division algorithms

In Fig.6.1, a Gaussian fineness function is used and the elements at the deepest level are initialized with fineness integers assuming the division limit is 6. The same multi-scale mesh shown in Fig.6.2 can be generated using Merging and Division algorithms. The result reflects what is anticipated: the distribution of meshes correspond to the function distribution.



(X-axis: element distribution Y-axis: division level)

(Left: exponential fineness function Right: discretized distribution)

Fig. 6.1 Gaussian fineness function & the initialization

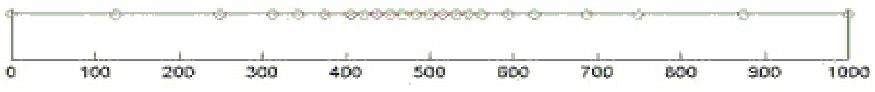


Fig. 6.2 The final multi-scale mesh with Division level: 6

The fineness value will be the greatest both in the middle and at the two ends if a discrete fineness distribution shown in Fig.6.3 is used. Subsequently, the fine elements are used in these areas. The same meshes can be obtained using Merging and Division algorithms (Fig.6.4). The projection in y-axis reflects the size of an element. Fine elements are located at points with greatest fineness value.

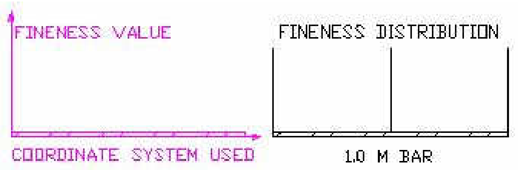


Fig. 6.3 Fineness function with three max positions (X-axis is position)

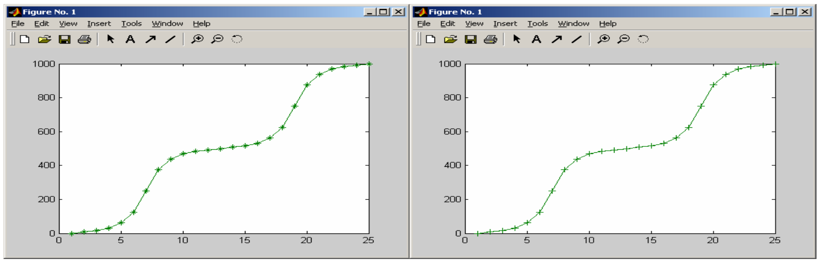


Figure No. 1
File Edit View Insert Tools Window Help
1000
800
600
400
200
0
0
5
10
15
20
25
Figure No. 1
File Edit View Insert Tools Window Help
1000
800
600
400
200
0
0
5
10
15
20
25

(X-Axis: sequential numbering of elements, Y-Axis: the length of each element)

Fig. 6.4 The same mesh results using Division and Merging (Division: 5)

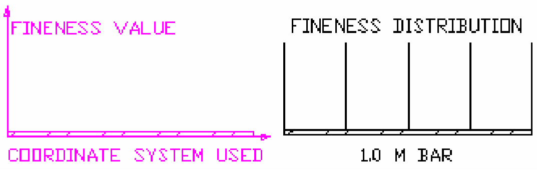
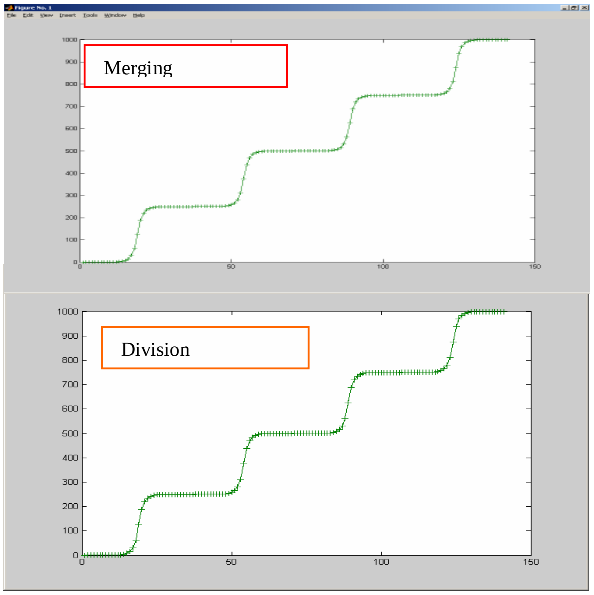


Fig. 6.5 Fineness function with five max positions (X-axis is position)



(X-Axis: sequential numbering of elements, Y-Axis: the length of each element)

Fig. 6.6 Identical meshing results using Division and Merging (Division: 20)

Another discrete fineness function is used as shown in Fig.6.5. Here, five discrete points with greatest fineness value divides the bar into four equal segments. In Fig. 6.6, the same meshes can be obtained using Merging and Division algorithms (The division level is 20. That means there are 1,048,576 elements at the finest level).

6.3.2 Time comparisons between Merging and Division algorithms

The computation time of Merging and Division algorithms is systematically compared in this section. Some extreme cases are used to compare their computation efficiency. From Table 6.1, we list all the abbreviations that will be used in other figures.

Table 6.1 Abbreviations of different fineness functions

Fun. No.	Description	Algorithms	Description
AM	All Max	M.A	Merging Algorithm
AZ	All Zero	D.A	Division Algorithm
MM	Max in Mid		
MR	Max at the Right		
ML	Max at the Left		

1. AM refers to the case where all the elements at the deepest level are required to have the highest fineness. This stands for a typical FEM, which demands finest meshing throughout the whole domain.

2. AZ refers to the case where all elements at the deepest level require the least fineness, which is 0 in fineness value. Division algorithm performs well under this situation, which is only used as an extreme case for comparison.

3. MM refers to the case where only the middle element at the deepest level is required the highest fineness. It is the case when only one haptic interaction is applied in the middle.

4. ML refers to the case where only the leftmost element at the deepest level is required the highest fineness. It is the case when only one haptic interaction is applied at the left end.

5. MR refers to the case where only the rightmost element at the deepest level is required the highest fineness. It is the case when only one haptic interaction is applied at the right end.

These five situations correspond to the cases 1-5 shown in Fig.6.7

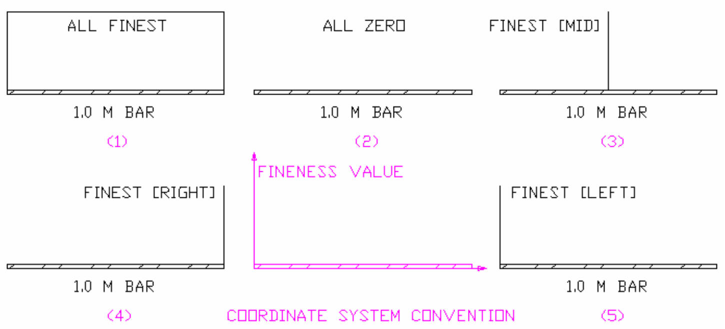


Fig. 6.7 Five different fineness distributions over the bar (X-axis is position)

Table 6.2 Time comparisons between Division and Merging algorithms

Fun. No.	AM (ms)		AZ (ms)		MM (ms)		ML (ms)		MR (ms)	
Finest Div.	M.A	D.A	M.A	D.A	M.A	D.A	M.A	D.A	M.A	D.A
2^{8}	0	0	10	0	0	0	0	0	0	0
2^{9}	0	0	10	0	0	0	0	0	0	0
2^{10}	0	0	10	0	10	0	0	0	0	0
2^{11}	0	0	10	0	10	0	0	0	0	0
2^{12}	10	0	20	0	10	0	20	0	20	0
2^{13}	30	10	30	0	30	0	30	0	30	0
2^{14}	50	20	70	0	60	0	70	0	70	0
2^{15}	111	40	151	0	141	10	140	10	140	10
2^{16}	210	70	311	0	280	10	270	10	290	10
2^{17}	441	150	781	10	590	10	561	10	611	20
2^{18}	901	310	1352	20	1231	20	1172	20	1272	30
2^{19}	4877	580	4867	30	2583	50	2433	40	2654	51
2^{20}							Nil	721		
2^{21}							Nil	103000		

Table 6.2 lists the simulation results. In the table, there are many zeros because when the division level is low, the resolution of CPU clock is not precise enough to reflect that. It is also obvious that Division algorithm is much more efficient. And their comparisons are shown from Fig.6.8 to Fig.6.10. In those figures, the x-axis

represents division limits. For example, the number 12 means the case where there are 2^{12} elements in the finest level.

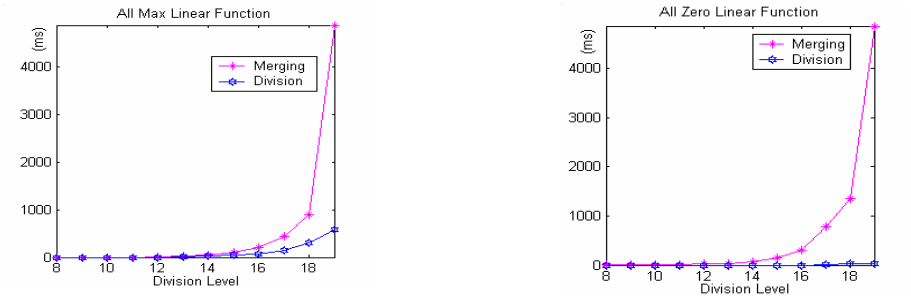


Fig. 6.8 Left (AM): All max case, Right (AZ): All zero case

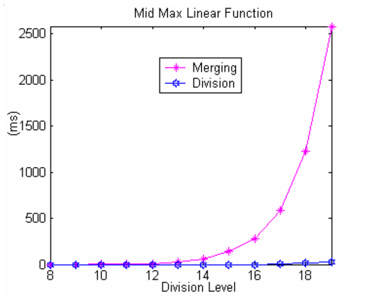


Fig. 6.9 Mid max case

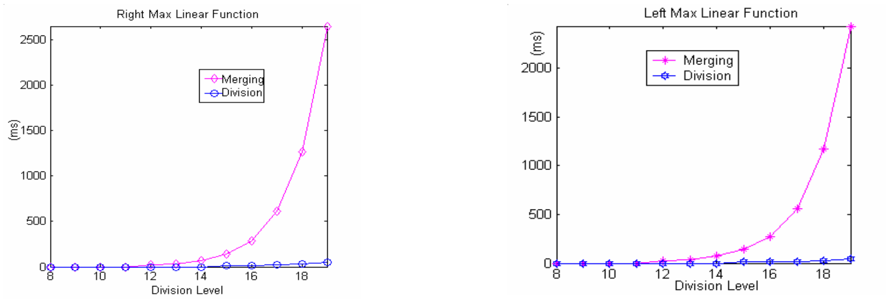


Fig. 6.10 Left (MR): Right max case, Right (ML): Left max case

1) For All Max (AM) case, the result is shown in Fig.6.8 (left). The Division algorithm is more efficient than the Merging algorithm although this superiority is not significant.

2) For All Zero (AZ) case, the result is shown in Fig.6.8 (right). The Division algorithm is significantly more efficient than the Merging algorithm.

3) For Middle Max (MM) case, the result is shown in Fig.6.9. Division is still much more efficient. It can also been inferred that the time needed by early steps takes the majority of the overall computational time.

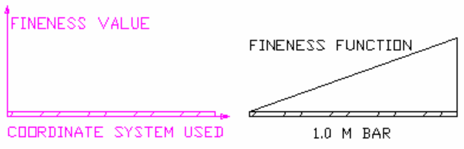
4) For Right Max (MR) case, the result is shown in Fig.6.10 (the left figure). This case and the left MAX case are typical cases and the results show that Division algorithm is much more efficient than Merging algorithm in these situations.

5) For Left Max (ML) case, the result is shown in Fig.6.10 (the right figure) and is explained in case 4).

From Fig.6.8 to Fig.6.10, it is obvious that, in all these five situations, when division moves to 10 levels, Division algorithm consumes significantly less time; but when the division level is below 10, the difference is not so significant.

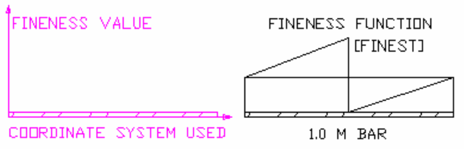
6.3.3 Time comparisons of the three algorithms

From the previous section, it shows that Division algorithm can save more time when the division level moves above 10. We still need to compare the computation time of the three algorithms when the fineness function is slightly changed, which simulates a typical situation where a surgeon slightly moves his tool-tip on tissues and tool-tissue interaction points change only slightly and slowly.



(X-axis is position)

Fig. 6.11 Linear fineness function



(X-axis is position)

Fig. 6.12 Bi-linear fineness function

To represent a “slight” change, three kinds of functions are used and through shifting these functions, the effect of a “slight” change can be achieved.

For the first case, a single linear fineness function (Fig.6.11) is adopted and then shifted step by step. At each small step, only a small proportion of the elements have their fineness values changed. The simulation results are shown in Table 6.3 and the results are also graphically presented in Fig. 6.13.

Table 6.3 Time comparisons (Linear function) of the three algorithms

The Linear Function (Shift)			
Finest Div.	**Dynamic (ms)**	**Division (ms)**	**Merge (ms) ***
		Non-Queue	**Non-Queue**
2^3	20	0	0
2^4	0	10	0
2^5	10	0	50
2^6	10	10	70
2^7	20	20	80
2^8	40	40	250
2^9	80	91	480
2^{10}	170	190	1011
2^{11}	341	400	2250
2^{12}	681	881	1943
2^{13}	1382	1913	5020
2^{14}	2483	4006	9020
2^{15}	4687	8242	14280
2^{16}	9044	17636	24080
2^{17}	17625	39617	43000
2^{18}	34680	Nil	Nil
2^{19}	68760	Nil	Nil

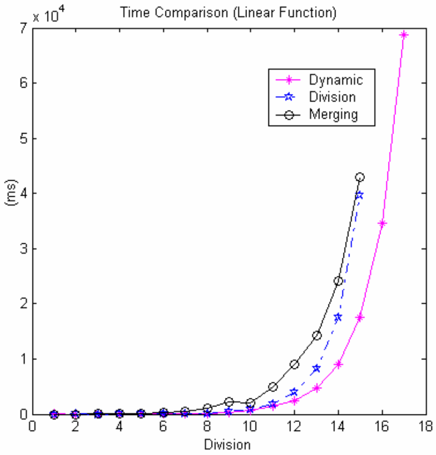


Fig. 6.13 Time comparisons of the three algorithms (Linear fineness function)

Fig.6.13 reveals that Dynamic algorithm is the fastest one in this situation; Division algorithm is the second and merging is the slowest. This shows that Dynamic algorithm consumes relatively less computer resources. If the division level moves to very high (above 18), while Dynamic algorithm is still computing, the other two either run out of memory or run too slowly (see Tables 6.3, 6.4 and 6.5).

In the second case, a bi-linear function (see Fig.6.12) is adopted. To guarantee that only a small proportion of the elements can get changed fineness values, the fineness distribution needs to be shifted with small steps. The simulation result is shown in Table 6.4 and graphically presented in Fig.6.14.

Table 6.4 Time comparisons (Bi-linear function) of the three algorithms

The Bi—linear Function (Shift)			
Div.	**Dynamic (ms)**	**Sub-Division (ms)**	**Merge (ms)**
	Queue	**Non-Queue**	**Non-Queue**
2^1	0	0	10
2^2	0	0	10
2^3	0	10	0
2^4	10	0	10
2^5	10	10	0
2^6	20	10	20
2^7	41	20	20
2^8	70	40	60
2^9	130	100	121
210	360	280	360
211	621	541	741
212	1452	1192	1612
213	2073	1762	2754
214	5248	4627	6149
215	7391	6820	14271
216	17826	21170	30624
217	27440	32738	64313
218	69130	105964	Nil
219	102589	173131	Nil

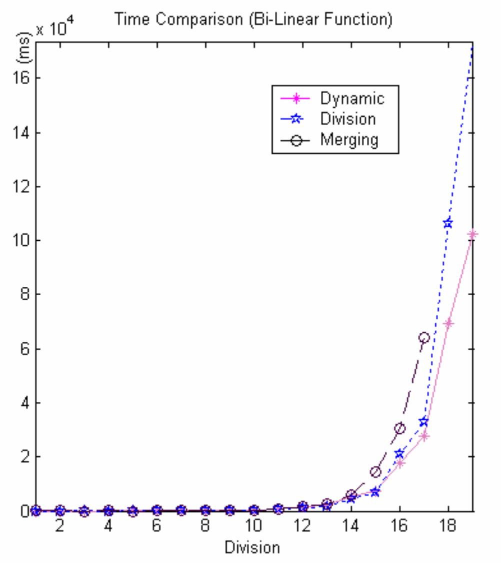


Fig. 6.14 Time comparisons of the three algorithms (Bi-linear fineness function)

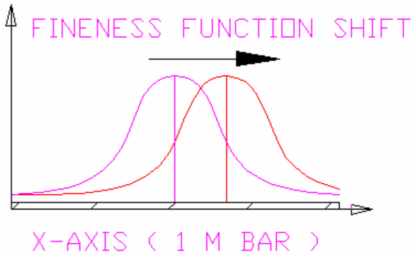


Fig. 6.15 Gaussian fineness function

In the third case, a Gaussian function (see Fig.6.15) is adopted as the fineness function. Again, the fineness function moves rightward along the bar with small steps. The simulation result is shown in Table.6.5 and graphically presented in Fig.6.16.

It needs to be pointed out that all above-mentioned three functions are designed to shift in small steps in order to simulate the realistic surgery environments where the relative position and the interaction between tissue and tool tip change smoothly and slightly. Thus, in such situations, only a small proportion of elements (< 5% elements) have their fineness values changed per re-meshing. Mathematically, this is reflected through some slight shift of the fineness function (so-called small steps) whenever we re-mesh. Such small steps (slight shift of the fineness functions) can be realized through assigning the left element's m_iFLev value to its neighbor on the right side one by one. That is why the procedure is called "shift" (see Fig. 6.15).

Table 6.5 Time comparisons (Gaussian function) of the three algorithms

The Exponential Function (Shift)			
Div.	**Dynamic (ms)**	**Sub-Division (ms)**	**Merge (ms)**
	Queue	**Non-Queue**	**Non-Queue**
2^{1}	10	10	0
2^{2}	0	0	0
2^{3}	0	0	0
2^{4}	10	10	10
2^{5}	10	0	10
2^{6}	20	20	10
2^{7}	30	20	30
2^{8}	60	50	60
2^{9}	140	120	140
2^{10}	371	311	381
2^{11}	951	751	991
2^{12}	1782	1442	1882
2^{13}	3264	2694	3665
2^{14}	6059	6209	8052
2^{15}	11827	14521	18346
2^{16}	22653	25106	35722
2^{17}	46066	86645	84923
2^{18}	87937	170747	314926
2^{19}	165760	Out of Memory	Nil

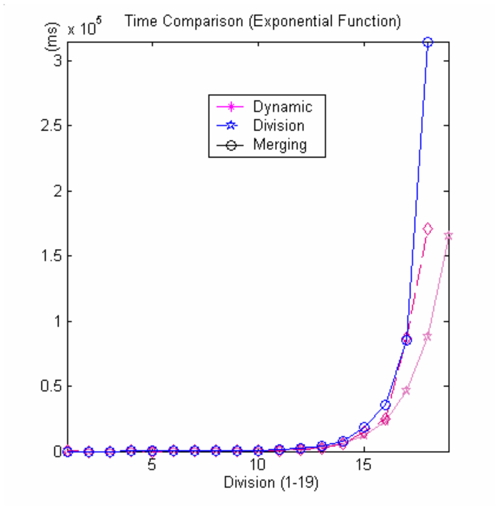


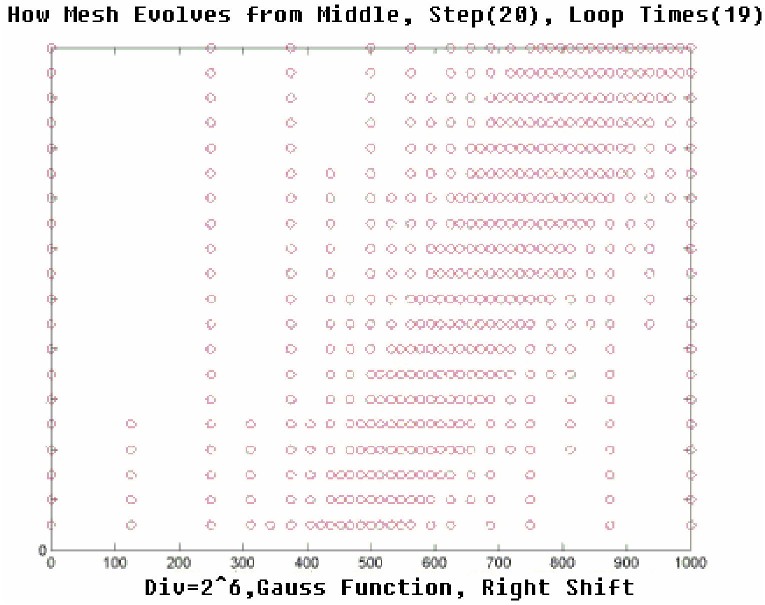
Fig. 6.16 Time comparisons of the three algorithms (Gaussian fineness function)

Discussion

From the three cases above, Dynamic algorithm can save more time in re-meshing. It is true that Dynamic algorithm can also employ merging and division operations. However, Dynamic algorithm is different from Merging and Division algorithms in its way to re-mesh. Dynamic algorithm starts from an almost-finished initial mesh and re-meshes only the areas that changes occur while both Division and Merging algorithms initiate the whole re-meshing procedure from scratch whenever a change occurs; no matter it is big or small. Subsequently, the overall operations for Dynamic algorithm are significantly fewer than those needed in Merging or Division algorithm. That enables Dynamic algorithm to speed up the process of re-meshing. Besides, the updated multi-scale mesh is similar to the previous one in Dynamic algorithm. The result suggests that Dynamic algorithm should be preferred when only minor changes occur to the system. However, Dynamic algorithm will lose this advantage if the system is changing drastically.

6.3.4 Evolution of Dynamic algorithm

How the multi-scale meshes evolve from one stage to the next one and how meshes evolve under minor changes will be shown in this section using the same fineness function shown in Fig.6.15. Generally, the fineness value of an element is replaced by the value of its left neighboring element while the function is directly shifted to the right direction. The value of zero is applied to the leftmost element. Fig.6.17 clearly shows how a multi-scale mesh evolves to a new one (with the division level 6). During the evolution, only minor change occurs between adjacent results sequentially obtained. Two observations can be identified as follows.



X-Axis: nodes coordinates, Y-Axis: multi-scale mesh evolution (bottom-up)
Fineness function drifts rightward with each step: 20

Fig. 6.17 Mesh evolves with fineness function drifting rightward

A. The multi-scale mesh reflects the fineness change;

B. It is true that meshes evolve slowly if only minor changes occur. So Dynamic algorithm should be the first choice in practice in such situations.

6.4 Discussions

After these comparisons, some general conclusions can be reached.

A. Generally, Dynamic algorithm can save time and also achieve a more efficient memory usage if only very small changes occur.

B. Dynamic, Division and Merging algorithms can all serve as general methods of choice in different situations. Whether it is in one-dimensional situation or two- or three-dimensional situations, the basic idea is the same. They represent different kinds of ideas and different ways of thinking.

C. If we include other factors such as reusable intermediate results that are only possible for Dynamic algorithm, it can be expected that the Dynamic algorithm will save even more time because only the changed parts need to be recalculated. That is, if only minor changes occur, there is no need to build a multi-scale meshing from scratch and recalculate a big matrix in real medical simulation.

D. These algorithms are important to mutually verify the correctness of the final multi-scale mesh.

E. The fineness function plays a crucial role in order to decide where to generate fine or coarse meshes. It is a generic method and can be applied in other situations. The method purely based on fineness functions provides us flexibility to simulate and model the real cases. That is to say, fuzzy concepts are described quantitatively.

Further discussions and comparisons:

1. If only one-dimensional mesh is considered, all operations of Division algorithm or Merging algorithm can be implemented through setting and changing the tag of each element in the binary tree (actually in an array). In this way, operations such as deletion or insertion of a node can be avoided. Since all the operations are performed on an array at the sacrifice of a few data items, the computation is fast. Meanwhile, the queue-based algorithm is also coded.

2. For one-dimensional situation, though we can develop queue-based (list) Division algorithm and Merging algorithm and non queue-based Division algorithm and Merging algorithm. However, non queue-based method is not of general interests for two-dimensional problems or for problems that cannot be defined by a perfect binary tree (actually is organized in an array.)

3. For non queue-based algorithms in one-dimensional situation, only a set of rules and communication procedures are used to dynamically update the state of each element in the tree (in the array). More future work is expected.

6.5 Summary

In this chapter, it is verified that any of the three algorithms can be used to arrive at the same multi-scale mesh although their relative efficiencies of time are significantly different. Generally speaking, Division algorithm is more efficient than Merging algorithm. However, Dynamic algorithm is preferred when a system changes slowly. Division algorithm is preferred when a coarse mesh is applicable while Merging algorithm performs well when a fine mesh is needed.

However, it deserves our special attention that the comparisons are only based on these simple fineness functions, thus the graphs and tables listed here cannot be generalized without conditions.

7 Conclusions and Future Work

Based on the previous studies and discussions, several important conclusions are drawn in this chapter. This is followed by the author's recommendations of future works.

7.1 Conclusions and discussions

This dissertation focuses on the algorithm design and analysis of multi-scale meshing in the context of surgical simulators. Division algorithm, Merge algorithm and Dynamic algorithm are investigated, implemented and compared in the one-dimension situation.

Firstly, the idea of a fineness function has been adopted to provide a comprehensive and general expression of criteria in order to reflect the dynamically changing nature of the simulated interaction. All related factors can be taken into consideration in the fineness function, which plays a key role in representing the influences from curvature, force and visual presentation.

Secondly, the process and criteria used to develop three algorithms are also carefully explained. The basic process is to calculate a fineness value for each element through the current fineness function. Then the principles of "as coarse as possible" and "smoothing" are applied to determine the elements for further merging or division. The procedure continues until a final smooth mesh satisfying the fineness requirements is achieved. Three different algorithms are developed basing on this process.

Thirdly, the relative efficiency and applicability of algorithms are compared in order to find the best algorithm for different situations. The experimental results show that Dynamic algorithm always has greater efficiency over the other two algorithms and thus has better potential applicability in surgical simulations where physically real and better visual presentation are always desired. Such requirements are also from the higher resolution of meshing and the faster computation. This is not contradictory. On the one hand, a high resolution of even meshing in traditional FEM is computationally expensive and a multi-scale meshing must specifically be developed to reduce the computational cost; on the other hand, Dynamic algorithm (multi-scale) can be used to improve the meshing resolution with the similar computational cost.

Fourthly, experiments are done through a robot to verify the applicability and efficiency of these three algorithms. People can feel the physically real force through the robot which has haptic feedback. All three algorithms can achieve similar meshing and re-meshing behaviors as expected. Moreover, the experimental results also show that relative efficiencies of three algorithms can be shown from the significantly differences in force feedbacks perceivable by an operator as the meshing resolution increases. This can verify the previous analysis that Dynamic algorithm is more efficient when the meshing resolution reaches to a threshold point.

Furthermore, an array-based data-structure is used in all three algorithms instead of the linked list data-structure to significantly reduce the computational time as the operations for array index are much more efficient than pointer operations in C/C++. A perfect binary tree data-structure is adopted to provide such efficient indexing operations for the one-dimension case considered.

Finally, the author finds that the final linear equations can be divided into several smaller linear equations during and an improved solver for 2D linear elastic system has been developed.

7.2 Future works

As mentioned previously, all the algorithms and experiments in this dissertation are based on the one-dimensional case. However, two-dimensional or three-dimensional models as well as algorithms are inevitable in real surgical simulations where the deformation of real tissues and organs can occur in two or three dimensions. The ideas applied in the current one-dimension modeling and the developments of three algorithms can be extended to two- or three-dimension situations although the data structure, merging or division criteria, and fineness functions may become more complex. Furthermore, It is worthwhile to the investigation on how the intermediate terms evolves and how the intermediate results can be reused in the whole process of solving equations in order to reduce the computational effort of the 3D algorithms in the surgical simulation.

References

[1] Poston, T., 2000. “Geometric curve dynamics”, Digital Medicine Laboratory, Johns Hopkins Singapore (not published).

[2] Iwata, H., 1990. *"Artificial Reality with Force-Feedback: Development of Desktop Virtual Space with Compact Master Manipulator"*. Computer Graphics, Vol. 24, No.4, pp. 165-170.

[3] Hannaford, B., Wood, L., McAffee, D.A., and Zak, H., 1991. “*Performance evaluation of a six-axis generalized force reflecting teleoperator*” IEEE Transactions on Systems, Man, and Cybernetics, 21(3): 620—633.

[4] Lee, S. and Lee, H.S., 1993. “Modeling, *Design, and Evaluation of Advanced Teleoperator Control Systems with Short Time Delay*” IEEE Transactions on Robotics and Automation, 9(5): 607—623.

[5] Buttazzo, G. C., 1993. “*HARTIK: A real-time kernel for robotics applications*” In Proceedings of the Real-Time Systems Symposium, pages 201-205. IEEE Computer Society Press.

[6] Ruspini, D., Kolarov, K, and Khatib, O. 1997. "*Haptic Interaction in Virtual Environments*" *IEEE/RSJ International Conference on Intelligent Robots and Systems: IROS'97,* Genoble, France. http://robotics.stanford.edu/users/ruspini/haptic.html

[7] Münch, S. and Dillmann, R., 1997. “*Haptic output in multimodal user interfaces*” Proceedings of IUI’97: International Conference on Intelligent User Interfaces, 105–112.

[8] Steve, T., 1998. C++ Iostreams Handbook**,** S. l. Addison Wesley Longman, Incorporated.

[9] Lau, R.W.H., Green, M., To, D. and Wong, J. 1998. *“Real-Time Continuous Multiresolution Method for Models of Arbitrary Topology”* Presence, 7(1): 2-35. http://www.cs.cityu.edu.hk/~rynson/papers/presence98.pdf

[10] Charoenseang, S., Srikaew, A., Wilkes, D.M. and Kawamura, K. 1998. *"Integrating Visual Feedback and Force Feedback in 3-D Collision Avoidance for a Dual-Arm Humanoid Robot"* 1998 International Conference on Systems, Man and Cybernetics, California, USA, pp. 3406-3411. http://fibo.kmutt.ac.th/publication/thai/conference/conference_thai.html

[11] Ruspini, D. and Khatib, O. 1999. *“Haptics for Multi-scale Virtual Prototyping”* RECENT ADVANCES IN MECHATRONICS, ICRAM’99, Turkey, Springer Press, Edited by Okyay Kaynak, Marcelo H. Ang Jr., Sabri Tosunoglu.

[12] Perry R.C. (editor), 1999. *Music, cognition, and computerized sound: an introduction to psychoacoustics,* Cambridge, Mass.: MIT Press, Chapters 18 and 19 (by Brent Gillespie)

[13] Yokokohji, Y., Hollis, R.L. and Kanade, T., 1999. *WYSIWYF Display: A Visual/Haptic Interface to Virtual Environment*, PRESENCE, Teleoperators and Virtual Environments, Vol.8, No.4, pp.412 – 434.

[14] Oboe, R. and Piovan, S., 1999. *Sensorless force reflecting teleoperation for low cost web-interfaced systems,* IEEE Symposium on Industrial Electronics – Bled, Slovenia.

[15] Plesniak, W., Pappu, R., 1999. *Spatial interaction with haptic hologram*, Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS'99). http://web.media.mit.edu/~wjp/htmFiles/publications.htm

[16] Namiki, A., Nakabo, Y., Ishii, I. and Ishikawa, M., 1999. *High Speed Grasping Using Visual and Force Feedback*, Proc. IEEE Int. Conf. on Robotics and Automation, Detroit.

[17] Hayward, V., 2000. *Haptics: A Key To Fast Paced Interactivity*, Human Friendly Mechatronics. Selected Papers of the International Conference on Machine Automation, 25-27 Sep, 2000, Osaka, Japan. By M. Takano, E. Arai and T. Arai, Elsevier Science. http://www.cim.mcgill.ca/~haptic/pub/VH-HFM-01.pdf

[18] Oakley, I, Brewster, S., and Gray, P.D., 2000. *Communication with Feeling*, Proceedings of First Workshop on Haptic Human-Computer Interaction, Glasgow, Scotland. http://www.dcs.gla.ac.uk/~stephen/papers/HHCI-ian.pdf

[19] Çavusoglu, M. C., and Tendick, F., *2000. Multirate simulation for high fidelity haptic interaction with deformable objects in virtual environments*, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2000), pp. 2458-2465.
http://itsa.ucsf.edu/~frankt/Cavusoglu_ICRA_2000.pdf

[20] Burdea, G., 2000. "*Haptics Issues in Virtual Environments*" Proceedings of Computer Graphics International 2000, Geneva, Switzerland, pp. 295-302.
http://www.caip.rutgers.edu/vrlab/publications/papers_2000.html

[21] Pajarola, R. and Rossignac, J. 2000. *SQUEEZE: Fast and Progressive Decompression of Triangle Meshes*, Proceedings of Computer Graphics International CGI 2000, Pages 173–182. IEEE Computer Society Press.
http://www.ics.uci.edu/~pajarola/pub/SQUEEZE.pdf

[22] FrÄohlich, B., Tramberend, H., Agrawala, M. and Bara, D., 2000. *Physically-Based Manipulation on the Responsive Workbench*, Proc. of IEEE VR Conference, 2000, pp. 5-12.

[23] WU, X., DOWNES, M. S., GOKTEKIN, T., AND TENDICK, F., 2001. *Adaptive Nonlinear Finite Elements for Deformable Body Simulation Using Dynamic Progressive Meshes*, Computer Graphics Forum 20, 3, 349-358. http://itsa.ucsf.edu/~frankt/Wu_EG2001.pdf

[24] Lewis, F. and Street, R., 2001. Touch: graphic design with tactile appeal, Gloucester, Mass.: Rockport Publishers. Chapter 5: The future of Touch

[25] Ganovelli, F., Cignoni, P., Montani, C. and Scopigno, R. 2001. *Enabling Cuts on Multiresolution Representation*, The Visual Computer, Springer International, Vol. 17 (5), 274-286. http://vcg.isti.cnr.it/publications/papers/enablingcuts.pdf

[26] Eisinberg, A., Scalari, G., Mazzoni, M., Menciassi, A., and Dario, P., 2001. *Force-feedback Sensorized Micrograppers for a Micromanipulation Workstation*, Proceedings of AISEM 2001, 6th National Conference on Sensors and Microsystems. http://www-crim.sssup.it/download/papers/default.htm

[27] Hirota, K. and Kaneko, T., 2001. *Haptic Representation of Elastic Objects"*, Presence: Teleoperators and Virtual Environments, Vol. 10, Iss.5, 525-536.

[28] Mahal, B.S., Clark, D.E.R., and Simmons, J.E.L., 2001. *Mass-Spring Simulation of Deformation in Elastic Sheet Structures*, Presence-Teleoperators and Virtual Environments, Vol. 10, Iss. 3, pp. 331-342.

[29] To, D., Lau, R.W.H. and Green, M., 2001 *"An Adaptive Multiresolution Method for Progressive Model Transmission"* Presence, Vol. 10, No. 1, pp 62–74. http://www.cs.cityu.edu.hk/~rynson/papers/presence01.pdf

[30] Picinbono, G., Lombardo, J.C., Delingette, H., and Ayache, N., 2000. *Anisotropic elasticity and force extrapolation to improve realism of surgery simulation*, IEEE International Conference on Robotics and Automation: ICRA 2000, San Francisco, CA, pp. 596-602.

[31] Elhajj, I., Hummert, H., Xi, N., Liu, Y. H., and Li, W. J., 2001. *Synchronization and control of supermedia transmission via the Internet,* International Symposium on Intelligent Multimedia, Video and Speech, Kowloon, Hong Kong, pp. 320-323.

[32] Basdogan, C., Ho, C.-H., and Srinivasan, M.A., 2001. "Virtual environments *for medical training: graphical and haptic simulation of laparoscopic common bile duct exploration"* Mechatronics, IEEE/ASME Transactions on Vol. 6, Issue 3, Page(s): 269-285.

[33] Oboe, R., 2001. *Web-Interfaced, Force-Reflecting Teleoperation Systems*, IEEE Transactions on Industrial Electronics, 48:6, pp. 1257-1265.

[34] Basdogan, C., Srinivasan, M.A., 2001. *Haptic Rendering In Virtual Environments*, Virtual Environments HandBook, pp. 117-134.

http://network.ku.edu.tr/~cbasdogan/publications02.html

[35] Ford, W. and Topp, W., 2002. Data structures with C++ using STL, Upper Saddle River, NJ: Prentice Hall.

[36] Liu, B. F., 2002. *Design of A Virtual Reality Workstation With 3-Dimensional Visual and Haptic Feedback,* Master Thesis, National University of Singapore, 2002

[37] Rovers, A. F. 2002. Haptic Feedback --*A literature study on the present-day use of haptic feedback in medical robotics September 2002 DCT Report nr. 2002.57.* http://www.bmt.tue.nl/biorobotics/Publications/DCT_2002_57_TReport_-_AFRovers.pdf

[38] Kim, J., De, S., and Srinivasan, M.A, 2002. *Computationally efficient techniques for real time surgical simulation with force feedback,* 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, Orlando, US, p51-57.

[39] Oakley, I, Adams, A., Brewster, S., and Gray, P., 2002. *Guidelines for the design of haptic widgets,* Proceedings of BCS HCI 2002 (London, UK), pp. 195-212. http://www.dcs.gla.ac.uk/~stephen/papers/HCI2002-oakley.pdf

[40] Lim, K.M., Poston, T., Zhang, L., Liu, B.F., Teo, C.L., and Burdet, E., 2002. *"MULTI-SCALE SIMULATION FOR A ROBOTIC SURGICAL TRAINER"* The Bio-Era: New Challenges, New Frontiers, Singapore. http://guppy.mpe.nus.edu.sg/~eburdet/papers/ICME_FULLPAPER5.pdf

[41] Nojima, T., Sekiguchi, D., Inami, M., and Tachi, S., 2002. *The SmartTool: a system for augmented reality of haptics,* Proceedings of IEEE VR 2002, pp.67-72. http://www.star.t.u-tokyo.ac.jp/~tnojima/Achievements/pdf/IEEEVR2002.pdf

[42] Foskey, M., Otaduy, M.A. and Lin, M.C., 2002. *"ArtNova: Touch-Enabled 3D Model Design"* Proceedings of IEEE VR 2002, pages 119–126. http://www.cs.unc.edu/~geom/ArtNova/artnova.pdf

[43] Yamada, T., Tsubouchi, D., Ogi, T., and Hirose, M., 2002. *Desk-Sized Immersive Workplace Using Force Feedback Grid Interface,* VR 2002: 135-142.

[44] Agus, M., Giachetti, A., Gobbetti, E., Zanetti, G., and Zorcolo, A., 2002. *Real–time haptic and visual simulation of bone dissection, IEEE Virtual Reality Conference*, pages 209–216, IEEE Computer Society Press. http://www.crs4.it/vic/data/papers/vr2002-burr.pdf

[45] Teo, C.L., Burdet, E., and Lim, H.P., 2002. *A Robotic Teacher of Chinese Handwriting,* Proceedings of 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'02), pp. 335-341.

[46] Yeh, J.S., Lien, T.Y., and Ouhyoung, M., 2002. *"On the Effects of Haptic Display in Brush and Ink Simulation for Chinese Painting and Calligraphy"* Proc. of the 10th Pacific Conference on Computer Graphics and Applications (PG02), pp. 439-441, Beijing, China, IEEE Press. http://3dsite.dhs.org/pg02/pg02_jsyeh_painting.pdf

[47] Biggs, J. and Srinivasan, M.A. 2002. Haptic Interfaces: Handbook of Virtual Environments, by K. Stanney. London, Lawrence Earlbaum, Inc.: Chapter 5, pp. 93-116. http://www.mit.edu/people/jbiggs/

[48] Kim, S., Hasegawa, S., Koike, Y., and Sato, M. 2002. *Tension Based 7-DOF Force Feedback Device: SPIDAR-G,* Virtual Reality 2002 Conference.
http://sklab-www.pi.titech.ac.jp/~hase/index.en.php

[49] Ye, Z.M. and Auner, G., 2003. *Haptic interface prototype for feedback control on robotic integration of smart sensors,* Proceedings of 2003 IEEE Conference on Control Applications, page(s): 995- 1000 vol. 2.

[50] Lim, K.M., Wang, F., Poston, T., Zhang, L., Teo, C.L. and Burdet, E. 2004. *Multi-Scale Simulation For Microsurgery Train,* 2004 IEEE International Conference on Robotics and Automation, in press.
http://guppy.mpe.nus.edu.sg/~eburdet/VR/vr.html

[51] Oboe, R., 2003. *Force-reflecting teleoperation over the Internet: the JBIT project,*
Special issue on networked intelligent robots through the internet, Vol. 91, No. 3, Page(s):449-462.
http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=26724&puNumber=5

[52] Li, G.Y., Xi, N., Yu M., and Fung, W.K. 2003. *Augmented Reality System for Real-time Nanomanipulation,* Proceeding of the 2003 IEEE Nanotechnology, San Francisco, CA, US.

[53] Kim, J., De, S., and Srinivasan, M. A., 2003. *An Integral Equation Based Multiresolution Modeling Scheme for Multimodal Medical Simulations*, Proc. of the IEEE VR2003 Conference, Los Angeles, California.

[54] McLaughlin, M. L., Hespanha, J. P., and Sukhatme, G. S., 2001. Touch in Virtual Environments: Haptics and the Design of Interactive Systems, PRENTICE HALL, Upper Saddle River, NJ.

[55] Buchanan, G. R., 1995. Schaum's Outline of Theory and Problems of Finite Element Analysis, New York: McGraw-Hill.

[56] Lim K. M., "Multi-scale Simulation for Haptic Applications Preliminary Investigations" (Proposal)

[57] http://www.sensable.com/

[58] http://www.forcedimension.com/

[59] http://www.nist.gov/dads/

[60] http://www.nist.gov/dads/

[61] Zhang L., 2003. “Multi-Scale Simulation For Real-Time Haptic Feedback” Master’s Thesis, National University of Singapore, 2003

[62] Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P., 2003. *Numerical recipes in C++ : the art of scientific computing*, imprint New York : Cambridge University Press.

Appendix: Solver and Real-Time Simulation

I. Introduction

In this appendix, a brief introduction of the solution of linear algebraic equations is first given. Then an improved solver for tridiagonal equations system is illustrated. Finally, proposed algorithms are applied in some real-time situations, and the simulation results are presented.

II. Solution of linear algebraic equations

In the earlier chapters, a fineness function is used for multi-scale meshing (the idea applies to one-dimensional, two-dimensional, and three-dimensional situations). As a result, the conventional finite element meshing becomes a special case, in which the finest division is governed everywhere throughout the whole domain. After meshing, linear algebraic equations can be built up and the system's unknowns should be solved. A discrete fineness function with four constraint points (two fixed ends and two haptic tips) is used to test whether all of the proposed algorithms can result in consistent multi-scale meshes. The performance of algorithms is also studied.

In the current linear-elastic system, $K*\boldsymbol{u} = \boldsymbol{f}$ always maintains no matter whether it is obtained from a pure mathematical analysis, physical laws such as the minimal potential principle or the minimal virtual work principle. While some variables are pre-specified, other unknowns have to be solved. For a one-dimension case, a tridiagonal linear equation set should be obtained. An assembled stiffness matrix K

(Fig.1b) can be obtained for a very small tridiagonal equations system in Fig.1a. As similar but bigger equations system with more DOFs can be obtained.

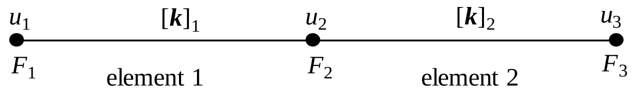


Fig. 1a A simple spring system

$$K = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix}$$

Fig. 1b Assembled matrix of a simple spring system

The numerical analysis for a system with huge DOFs, the inverse matrix of K is never computed. Instead, **LU**-decomposition method is often used to solve linear equation systems. There are many established methods to transform a common matrix into the multiplication between a lower tridiagonal matrix and an upper tridiagonal matrix, which is K **=** L * U**.** For a 4 by 4 matrix K, it can be shown in Fig. 2.

$$\underset{\mathbf{L}}{\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix}} \underset{*}{\cdot} \underset{\mathbf{U}}{\begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix}}$$

Fig. 2 L-U decomposition [62]

Once L and U are known, we have:

$$K = \mathrm{L} * \mathrm{U} \tag{1}$$

$$K * \boldsymbol{u} = (\mathrm{L} * \mathrm{U}) * \boldsymbol{u} = \mathrm{L} * (\mathrm{U} * \boldsymbol{u}) = \boldsymbol{f} \tag{2}$$

$$\mathrm{L} * \mathrm{Y} = \boldsymbol{f} \tag{3}$$

$$\mathrm{U} * \boldsymbol{u} = \mathrm{Y} \tag{4}$$

For (3), a forward substitution is used to get Y; and for (4), a back substitution is used to get ***u*** after Y is obtained. There are two possible cases:

(1) ***f*** is fully specified, only ***u*** need to be solved, or

(2) Both ***u*** and ***f*** are only partially specified.

The first case can be directly solved with a suitable solver. However, some adjustments are needed for solving the second case (our current case). In a system in Fig.3, at certain interaction position, it has unknowns such as $[u_1 \; u_2 \; \dots \; u_{i-1} \; u_{i+1} \; \dots \; u_{M-1} \; u_M]$ and f , but $[f_1 \; f_2 \; \dots \; f_{i-1} \; f_{i+1} \; \dots \; f_{M-1} \; f_M]$ and U are unknown. The original system needs to be adjusted to a final form as shown in Fig. 4 to be solved.

$$\begin{bmatrix} K_{11} & K_{12} & \Lambda & K_{1M} \\ M & M & & M \\ K_{i1} & K_{i2} & \Lambda & K_{iM} \\ M & M & & M \\ K_{M1} & K_{M2} & \Lambda & K_{MM} \end{bmatrix} \left\{ \begin{matrix} u_1 \\ u_2 \\ M \\ u_{i-1} \\ U \\ u_{i+1} \\ M \\ u_{M-1} \\ u_M \end{matrix} \right\} = \left\{ \begin{matrix} f1 \\ f2 \\ M \\ f_{i-1} \\ f \\ f_{i+1} \\ M \\ f_{M-1} \\ f_M \end{matrix} \right\}$$

Fig. 3 The original system

$$\begin{bmatrix} K_{11} & \Lambda & K_{1(i-1)} & K_{1(i+1)} & \Lambda & K_{1M} \\ M & & & & & M \\ K_{(i-1)1} & \Lambda & K_{(i-1)(i-1)} & K_{(i-1)(i+1)} & \Lambda & K_{(i-1)M} \\ K_{(i+1)1} & \Lambda & K_{(i+1)(i-1)} & K_{(i+1)(i+1)} & \Lambda & K_{(i+1)M} \\ M & & & & & M \\ K_{N1} & \Lambda & K_{M(i-1)} & K_{M(i+1)} & \Lambda & K_{MM} \end{bmatrix} \left\{ \begin{matrix} u_1 \\ u_2 \\ M \\ u_{i-1} \\ \\ u_{i+1} \\ M \\ u_{M-1} \\ u_M \end{matrix} \right\} = -U \left\{ \begin{matrix} K_{1i} \\ K_{2i} \\ M \\ K_{(i-1)i} \\ \\ K_{(i+1)i} \\ M \\ K_{(M-1)i} \\ K_{Mi} \end{matrix} \right\} + \left\{ \begin{matrix} f1 \\ f2 \\ M \\ f_{i-1} \\ \\ f_{i+1} \\ M \\ f_{M-1} \\ f_M \end{matrix} \right\}$$

Fig. 4 The transformed system

After all the unknown displacements are known the force at the interaction position can be calculated by (5).

$$\lfloor K_{i1} \quad K_{i2} \quad \Lambda \quad K_{i(M-1)} \quad K_{iM} \rfloor * U = f \tag{5}$$

In more complex cases, several displacements are specified and the forces at those positions need to be calculated while forces are known as zeros by default at other positions. Other unknown displacements can be obtained through systematically adjusting the original linear equation system. While the required forces can be computed using good “house-keeping”, a “divide and conquer” method is adopted here to solve the tridiagonal equation system. Such a solver greatly reduces the necessity of burdensome housekeeping and the whole equation system structure remains unchanged.

Conventionally, the original system in Table 1 can be changed to a final form as shown in Table 2 after matrix transformations. Then the unknown displacements and the forces can be solved. Since all the elements of the matrices are re-shuffled, systematic ways are needed to track and record changes. Furthermore, only five vectors are used to store lower entries, diagonal entries, upper entries, input displacement, and output forces as matrix storage is concerned.

The above-mentioned housekeeping can be made simple, especially, when both some displacements and forces are partially specified. For example, in the system shown in Table 3, [u0, u3, u9] and [F1, F2, F4, F5, F6, F7, F8] are known; but [u1, u2, u4, u5, u6, u7, u8] and [F0, F3, F9] are to be found.

Table 1 The original system of equations (conventional)

A11	a12									0		F1
A21	a22	A23								u2		0
	a32	A33	a34							U3		F3
		A43	a44	a45						u4		0
			a54	a55	a56					u5	=	0
				a65	a66	a67				u6		0
					a76	a77	a78			U7		F7
						a87	a88	a89		u8		0
							a98	a99	a9,10	u9		0
								a10,9	a10,10	0		F10

Table 2 The transformed system of equations (conventional)

a22	0					u2		V2
0	a44	a45	🟩			u4		V4
	a54	a55	a56			u5	=	V5
	🟩	a65	a66	0		u6		V6
			0	a88	a89	u8		V8
				a98	a99	u9		V9

Table 3 The original system of equations (new)

a00	a01									U0		F0
a10	a11	a12								U1		F1
	a21	a22	a23							U2		F2
		a32	a33	a34						U3		F3
			a43	a44	a45	🟩	🟩	🟩		U4	=	F4
				a54	a55	a56	🟩	🟩		U5		F5
				🟩	a65	a66	a67	🟩		U6		F6
				🟩	🟩	a76	a77	a78		U7		F7
				🟩	🟩	🟩	a87	a88	a89	U8		F8
								a98	a99	U9		F9

Table 4 The adjusted system of equations (new)

a00	a01									0		F0		a00		0		0
a10	a11	a12								U1		F1		a10		0		0
	a21	a22	a23							U2		F2		0		0		a23
		a32	a33	A34						0		F3	-u0*	0	-u9*	0	-u3	a33
			a43	A44	a45	🟩	🟩	🟩		U4	=	F4		0		0		a43
				A54	a55	a56	🟩	🟩		U5		F5		0		0		0
				🟩	a65	a66	a67	🟩		U6		F6		0		0		0
				🟩	🟩	a76	a77	a78		U7		F7		0		0		0
				🟩	🟩	🟩	a87	a88	a89	U8		F8		0		a89		0
								a98	a99	0		F9		0		a99		0

If both input force vector and displacement vector are adjusted as shown in Table 4, two smaller tridiagonal systems will be obtained and be solved easily. Thus [u1, u2, u4, u5, u6, u7, u8] is obtained with u0 = 0, u3 = 0, and u9 = 0 being unchanged.

However, house-keeping should be used to obtain forces since displacement vector is not the original one any more. Since the zero positions in the displacement vector are not used during the solving process, they can be used to keep the original values in these places. Finally, required forces [F0, F3, F9] can be solved directly after we get [u1, u2, u4, u5, u6, u7, u8].

In the beginning, the original F0, F3 and F9 can be assigned with random values since they will be overridden later.

In the whole process, the K matrix (three vectors) is kept untouched; the displacement vector and the force vector keep their original forms. Only the values are changed as the system proceeds. Besides, the process of re-assembling the original system is completely avoided by separately solving several smaller tridiagonal systems(see Table 5). Thus, a new solver could be developed through making some minor and wise adjustment to the typical algorithm for a tridiagonal system ([62]).

Table 5 The actual system of equations (new)

a00	a01									U1		F0		a00		0		0
a10	a11	a12								U1		F1		a10		0		0
	a21	a22	a23							U2		F2		0		0		a23
		a32	a33	a34						U3		F3	-u0	0	-u9	0	-u3	a33
			a43	a44	a45					U4	!=	F4		0		0		a43
				a54	a55	a56				U5		F5		0		0		0
					a65	a66	a67			U6		F6		0		0		0
						a76	a77	a78		U7		F7		0		0		0
							a87	a88	a89	U8		F8		0		a89		0
								a98	a99	U9		F9		0		a99		0

In summary, after initialization, the solver first divides a big problem into many smaller ones. Subsequently, it solves all smaller tridiagonal systems with less effort and finally obtains all the unknowns. The whole process can be termed as **divide and conquer.**

The process of meshing and re-meshing is a relatively independent process, which 1) builds up a multi-scale mesh according to the input fineness function and 2) calculates the absolute positions of interaction points (haptic tips) in the finest division and the corresponding positions in a multi-scale mesh. At these interaction points, the displacements are pre-specified or input by the user. Once the mesh has changed, it will pass data to the solver. The solver then starts a new round of solution. As for the positions in the finest level that are not included in the multi-scale mesh, an interpolation method is used to calculate their displacements. The whole process is shown in Fig. 5.

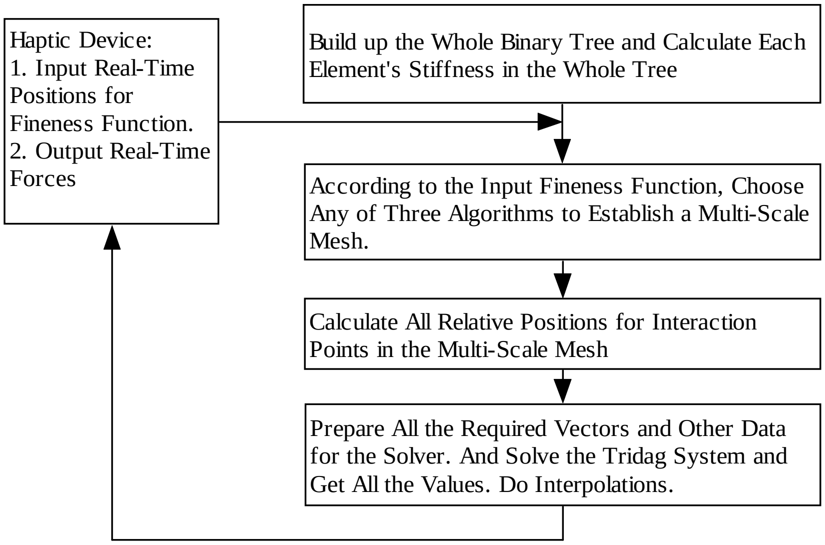
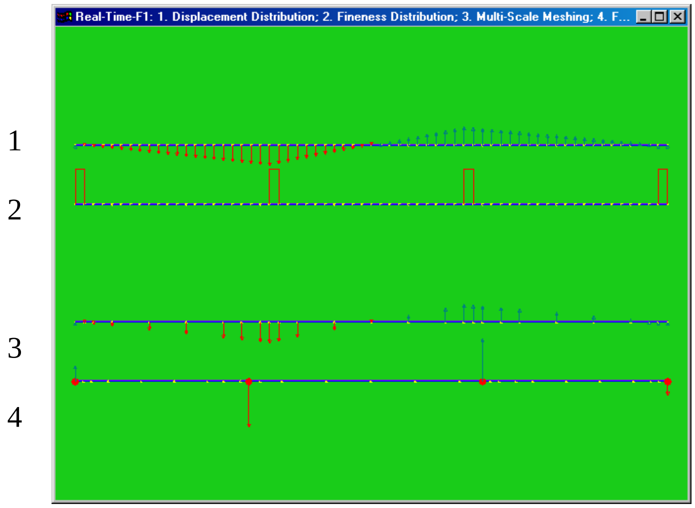

Haptic Device:
1. Input Real-Time Positions for Fineness Function.
2. Output Real-Time Forces
Build up the Whole Binary Tree and Calculate Each Element's Stiffness in the Whole Tree
According to the Input Fineness Function, Choose Any of Three Algorithms to Establish a Multi-Scale Mesh.
Calculate All Relative Positions for Interaction Points in the Multi-Scale Mesh
Prepare All the Required Vectors and Other Data for the Solver. And Solve the Tridag System and Get All the Values. Do Interpolations.

Fig. 5 The whole process

III. Simulation results and discussions

The simulations are conducted to: 1) show the correctness of the final mesh using Dynamic algorithm, Merging algorithm and Division algorithm, 2) show if these three algorithms are efficient enough to let the operator feel the real-time force feedback when an operator changes the interaction point slightly, and thus changes the fineness function, and 3) graphically show the force distribution, displacement distribution, and the elements consisting of a multi-scale mesh.

In Fig.6, both ends are constrained and displacements are specified at the interaction points. All the displacements and forces are in the horizontal direction but are shown in the Y-axis for better presentation. The red arrow stands for negative X-axis and the blue one for positive X-axis.


Real-Time-F1: 1. Displacement Distribution; 2. Fineness Distribution; 3. Multi-Scale Meshing; 4. F...
1
2
3
4

(X-axis: position, Y-axis: magnitude of values)

1: The displacement distribution of all nodes at the finest level

2: The fineness function distribution at the finest level

3: The displacement distribution of a multi-scale mesh

4: The actual positions of a multi-scale mesh and the forces

Fig. 6 Simulation results (Division: 6)

During the simulation experiments, all three algorithms get the same multi-scale mesh as the user moves the intermediate interaction points and the final results are shown in Fig. 6. Here, the part labeled with 1 shows the displacement distribution of all elements in the deepest level; the part labeled with 2 specifies the fineness function (two constraints at both ends and two interaction points in the mid require fine meshing); the part labeled with 3 shows the displacement distribution of a multi-scale mesh where the position is before deformation; and the part labeled with 4 shows the calculated forces at both ends and two interaction points and they are presented at the actual nodes (after deformation).

During the simulation, an operator can use all three algorithms to feel the real-time force computed in the Delta system.

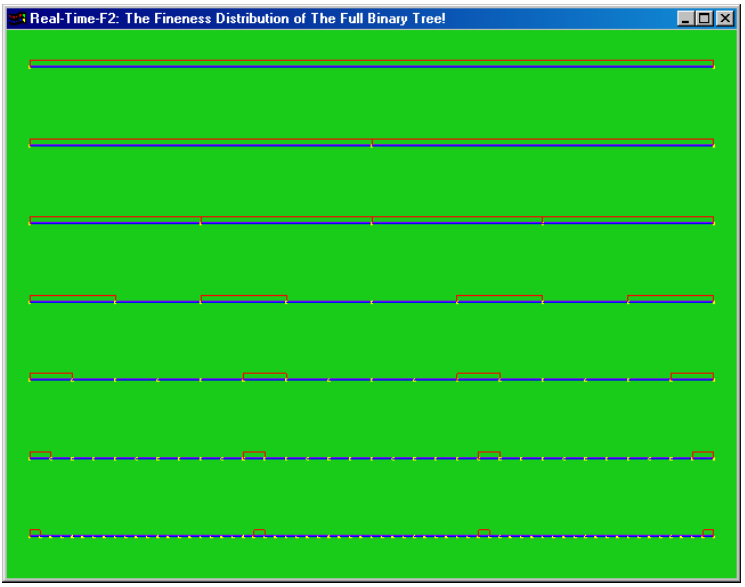

Real-Time-F2: The Fineness Distribution of The Full Binary Tree!

Fig. 7 The fineness function at the finest level

Fig.7 shows the process of how the fineness function is calculated and propagated from the deepest level (6) t the top level (0). At level 6, a fineness value is calculated for each element using the specified fineness function: here, for the whole area of interest, the fineness function consists of four discrete points with the fineness value of 6, then, each element at level 5 is assigned a fineness value: the bigger of the two fineness values of its two children elements. This process continues until all elements get their values. Fig.7 also shows the whole element map of our data-structure.

All the red rectangles in Fig. 8 constitute the multi-scale mesh under the specified fineness function. From left to right, it can be seen that the level difference of any two neighboring elements can go beyond 1.

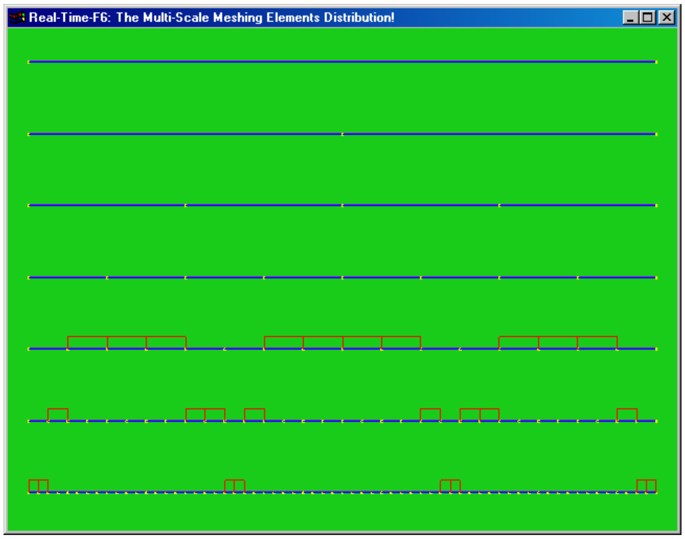

Real-Time-F6: The Multi-Scale Meshing Elements Distribution!

Fig. 8 The distribution of elements consisting of the final multi-scale mesh

IV. Summary

This appendix explained a solver for the tridiagonal equation system and introduced an improved solver. Also, the simulation results in a real-time workstation are presented.