IMPLEMENTATION OF MESH GENERATION ALGORITHMS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZGÜR YILDIZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2001

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Tayfur ÖZTÜRK
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. Mübeccel DEMİREKLER
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Dr. Önder YÜKSEL
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Gönül SAYAN	_____
Prof. Dr. Önder YÜKSEL	_____
Prof. Dr. Mustafa KUZUOĞLU	_____
Asst. Prof. Dr. Lale ALATAN	_____
Asım Egemen YILMAZ	_____

ABSTRACT

IMPLEMENTATION OF MESH GENERATION ALGORITHMS

YILDIZ, Özgür
MSc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Önder YÜKSEL
Co-supervisor: Prof. Dr. Mustafa KUZUOĞLU

December 2001, 99 pages

In this thesis, three mesh generation software packages have been developed and implemented. The first two were based on structured mesh generation algorithms and used to solve structured surface and volume mesh generation problems of three-dimensional domains. Structured mesh generation algorithms were based on the concept of isoparametric coordinates. In structured surface mesh generation software, quadrilateral mesh elements were generated for complex three-dimensional surfaces and these elements were then triangulated in order to obtain "high-quality" triangular mesh elements. Structured volume mesh generation software was used to generate hexahedral mesh elements for volumes. Tetrahedral mesh elements were constructed from hexahedral elements using hexahedral node insertion method. The results, which were produced by the mesh generation algorithms, were converted to a required format in order to be saved in

output files. The third software package is an unstructured quality tetrahedral mesh generator and was used to generate exact Delaunay tetrahedralizations, constrained (conforming) Delaunay tetrahedralizations and quality conforming Delaunay tetrahedralizations. Apart from the mesh generation algorithms used and implemented in this thesis, unstructured mesh generation techniques that can be used to generate quadrilateral, triangular, hexahedral and tetrahedral mesh elements were also discussed.

Keywords: Mesh Generation, Isoparametric Coordinate Transformation, Hexahedral Node Insertion Method, Delaunay Tetrahedralization, Constrained (Conforming) Tetrahedralization.

ÖZ

AĞ ÜRETME ALGORİTMALARININ GERÇEKLEŞTİRİLMESİ

YILDIZ, Özgür
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Önder YÜKSEL
Yardımcı Tez Yöneticisi: Prof. Dr. Mustafa KUZUOĞLU

Aralık 2001, 99 sayfa

Bu tezde, üç adet ağ üretme yazılım paketi geliştirilmiş ve gerçekleştirilmiştir. İlk ikisi, yapısal ağ üretme algoritmalarını esas almaktadır ve üç boyutlu alanlar için yapısal yüzey ve hacim ağ üretme problemlerini çözmek amacıyla kullanılmıştır. Yapısal ağ üretme algoritmaları izoparametrik koordinat dönüşümü kavramını temel almaktadır. Yapısal yüzey ağ üretme yazılımında, karmaşık üç boyutlu yüzeyler için dörtgensel ağ elemanları oluşturulmuş ve bu elemanlar daha sonra "nitelikli" üçgensel ağ elemanları elde etmek amacıyla üçgenlenmiştir. Yapısal hacim ağ üretme yazılımı, hacimler için altıyüzlü ağ elemanları üretmek amacıyla kullanılmıştır. Dörtyüzlü ağ elemanları, altıyüzlü kenar elemanı ekleme metodu kullanılarak altıyüzlü elemanlardan oluşturulmuştur. Ağ üretme algoritmalarından elde edilen sonuçlar, çıktı dosyalarına kaydedilmeleri amacıyla istenilen biçime dönüştürülmüştür. Üçüncü yazılım paketi, yapısal olmayan

nitelikli dörtyüzlü ağ elemanları üretecidir ve doğru Delaunay dörtyüzlülemeleri, kısıtlamalı (uyan) Delaunay dörtyüzlülemeleri ve nitelikli uyan Delaunay dörtyüzlülemeleri üretmek için kullanılmıştır. Bu tezde kullanılan ve gerçekleştirilen ağ üretme algoritmalarından ayrı olarak, dörtgensel, üçgensel, altıyüzlü ve dörtyüzlü ağ elemanları oluşturmak amacıyla kullanılan yapısal olmayan ağ üretme teknikleri de ele alınmıştır.

Anahtar Kelimeler : Ağ Üretme, İzoparametrik Koordinat Dönüşümü, Altıyüzlü Kenar Elemanı Ekleme Metodu, Delaunay Dörtyüzlülemeleri, Kısıtlamalı (Uyan) Delaunay Dörtyüzlülemeleri.

To My Family

ACKNOWLEDGEMENTS

I would like to thank Prof. Dr. Önder YÜKSEL and Prof. Dr. Mustafa KUZUOĞLU for their valuable supervision and support throughout the development and improvement of this thesis. Special thanks go to my family, all my friends in the office and A. Egemen YILMAZ for their patience, help and being very kind to me. I would also like to thank Si Hang for publishing his valuable work for academic purposes.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

NMGS-SSMG : Numerical Mesh Generation Software, Structured Surface Mesh Generation.

NMGS-SVMG : Numerical Mesh Generation Software, Structured Volume Mesh Generation.

NMGS-QTMG : Numerical Mesh Generation Software, Quality Tetrahedral Mesh Generation.

N2S : NMGS-SSMG Problem Setup File.

N2R : NMGS-SSMG Result File.

N3S : NMGS-SVMG Problem Setup File.

N3R : NMGS-SVMG Result File.

MoM : Method of Moments.

FDM : Finite Difference Method.

FEM : Finite Element Method.

STC : Spatial Twist Continuum.

RAD : Rapid Application Development.

GUI : Graphical User Interface.

TetGen : TetGen.exe.

CFD : Computational Fluid Dynamics.

CSM : Computational Structural Mechanics.

CEM : Computational Electromagnetics.

PLC : Piecewise Linear Complex.

CHAPTER 1

INTRODUCTION

1.1. Mesh Generation

Mesh generation is defined as the process of breaking up a physical domain into smaller sub-domains (elements), in order to perform a numerical solution for a partial differential or integral equation. Although meshing can be used for a wide variety of applications, the principal application of interest is the finite element method. Surface domains may be subdivided into triangular or quadrilateral shapes, while volumes may be subdivided primarily into tetrahedral or hexahedral shapes. Meshing algorithms ideally define the shape and distribution of the elements.

The finite element method has become a mainstay for industrial engineering design and analysis, in recent decades. It is being used in simulation of complex designs. Its increasing popularity causes automatical meshing algorithms to be improved.

Mesh generation is usually considered as the pre-processing step of numerical computational techniques. Meshes used in numerical solution algorithms must satisfy several conditions depending on the problem. Some of these conditions can be summarized as follows [2]:

i. The mesh must conform to the boundary of the region, which may consist of more than one connected components.
ii. The mesh must be fine enough to produce an adequate approximation to the original problem geometry.
iii. The elements constructing the mesh must be of good quality, because badly shaped elements may cause ill-conditioned matrices [1].
iv. The number of elements in the mesh should not be too large, since the mesh size increases the complexity of solving the finite element problem.

1.1.1. Meshing Technologies

Initially, the finite element method was capable of simulating designs utilizing only tens or hundreds of elements. Very careful and thorough preprocessing was required to subdivide domains into usable elements. Market forces have now pushed meshing technology to a point where users now expect to mesh complex domains with thousands or millions of elements, pushing a "go" button.

When comparing an equivalent number of degrees of freedom, triangle and tetrahedral shaped elements have lower performance than quadrilateral and hexahedra shaped elements, because the use of hex elements can reduce the number of elements and processing times. In addition, hex and quadrilateral elements are more suited for non-linear analysis as well as situations where alignment of elements is important to the physics of the problem.

The mesh generation problem aims to define a set of nodes and elements in order to best describe a geometric domain, subject to various element size and shape criteria, where the geometry is most often composed of vertices, curves, surfaces and solids.

Many applications use a "bottom-up" approach to mesh generation. Vertices are first meshed, followed by curves, then surfaces and finally solids. The input for the subsequent meshing operation is the result of the previous lower dimension

meshing operation. For example, nodes are first placed at all vertices of the geometry. Nodes are then distributed along geometric curves. The result of the curve meshing process provides input to a surface meshing algorithm, where a set of curves define a closed set of surface loops. Decomposing the surface into well-shaped elements (triangles or quadrilaterals) is the next phase of the meshing process. Finally, if a solid model is provided as the geometric domain, a set of meshed areas defining a closed volume is provided as input to a volume mesher for automatic formation of tetrahedra or hexahedra [14].

1.1.2. Meshing Types by Configuration

Meshes can be categorized as structured, unstructured and hybrid meshes by configuration. The choice of the mesh type is clearly related to the application.

1.1.2.1. Structured Meshes

Structured meshes are composed of mesh elements that all interior nodes have an equal number of adjacent elements. They offer simplicity in software development and easy data access. Two-dimensional structured meshes typically use quadrilaterals, while three-dimensional structured meshes typically use hexahedra. Those types of meshes are generated by means of transfinite mapping methods [3]. Structured meshes have been introduced in numerical analysis in the early 1970's after the finite element method became popular [4, 5].

The basic advantage of structured meshes is that they offer simplicity and efficiency in numerical computations. A structured mesh requires significantly less memory than an unstructured mesh with the same number of elements, because array storage can define neighbour connectivity implicitly. A structured mesh can also save computation time to access neighbouring cells when computing a finite-difference stencil, where the software simply increments or

decrements array indices. Compilers can produce quite efficient codes for these operations. A major advantage of structured meshes lies in their compatibility with efficient finite difference algorithms that are utilized in the solution of boundary value problems.

Despite the simplicity and efficiency of structured meshes, it can be difficult or impossible to compute a structured mesh for a complicated geometric domain. In addition, a structured mesh may require more elements than an unstructured mesh for the same problem, because elements in structured meshes have a fixed size whereas it is possible to grade elements in size in unstructured meshes.

1.1.2.2. Unstructured Meshes

Unlike structured mesh generation, unstructured mesh generation allows any number of elements to meet at a single node. When referring to unstructured meshing, triangle and tetrahedral meshes are commonly thought, even though quadrilateral and hexahedral meshes can be unstructured. While there is an overlap between structured and unstructured mesh generation technologies, the main feature that distinguishes the two fields is the unique iterative smoothing algorithms employed by structured mesh generators.

Unstructured mesh generation has been part of mainstream computational geometry for some years. Well-studied geometric constructions such as Delaunay triangulation are central to unstructured mesh generation. The main advantages of unstructured meshes are [2]:

i. Flexibility in fitting complicated domains.
ii. Rapid grading from small to large elements.
iii. Easy refinement and derefinement.

Three main approaches to unstructured mesh generation can be summarized as [2]:

i. Octree based algorithms.
ii. Delaunay triangulation based algorithms.
iii. Advancing front algorithms.

1.1.2.3. Hybrid Meshes

A hybrid mesh is formed by a number of structured meshes arranged in an overall unstructured pattern. Hybrid meshes fall somewhere in between structured and unstructured meshes. These meshes are used in problems with complicated geometries.

1.1.3. Meshing Types by Elements

Meshes can be divided into two main groups by elements. These are Tri / Tetrahedral and Quad / Hexahedral meshes, which are considered in two-dimension / three-dimension, respectively.

1.1.3.1. Quad / Hexahedral Meshes

Surface domains can be subdivided into quadrilateral elements, whereas volumes can be subdivided into hexahedral elements by structured as well as unstructured meshing methods. Isoparametric coordinates can be used to generate both quad / hexahedral meshes, which are considered as structured. As for unstructured quad / hexahedral meshes, they are generated using direct and indirect approaches. Meanwhile, some methods are also available that combine hexahedral and tetrahedral elements in a single three-dimensional domain.

Generally, unstructured mesh generation algorithms use triangle and tetrahedral mesh elements. As a result of this, most of the literature and software are triangle and tetrahedral, although there is a significant group of literature that focuses on unstructured quad and hexahedral methods.

1.1.3.2. Tri / Tetrahedral Meshes

Triangle and tetrahedral meshes are the most common forms of unstructured mesh generation. Most techniques currently in use can be considered in three main categories: Octree, Delaunay and Advancing Front techniques. Tri / tetrahedral meshes can also be constructed from quad / hexahedral mesh elements.

1.1.4. Numerical Methods

The discrete approximation of partial differential equations modeling a physical system can be performed by some numerical methods, such as the method of moments, the finite differences method and the finite element method. The basic steps of the numerical methods can be summarized as follows [13]:

i. Discretization of the geometric domain into meshes.
ii. Symbolic expression of the solution within each sub-domain by a finite number of parameters.
iii. Combination of the local equations obtained for each sub-domain.
iv. Construction of a set of equations describing the whole geometry.
v. Application of the boundary conditions.
vi. Solution of the global equation system to obtain the unknown function.

1.1.4.1. Method of Moments

Probably the most popular method is Method of Moments (MoM). It reduces the functional equations into matrix equations and the analogy between them specifies the field, which is required to be found. The MoM is essentially based on [13]:

i. The projection technique with a symmetric, integral definition of the inner product,
ii. An expansion of the sources, i.e., the current and charge densities,
iii. The integral formulations of the field equations.

MoM is especially used for analyzing perfectly conducting surfaces. It produces so successful results for geometries composed of wires and metal plates. The disadvantages of this approach are the difficulty in modeling dielectric and special magnetic materials of arbitrary shape and getting a resultant system matrix that is not sparse.

1.1.4.2. Finite Difference Method

Usually, the Finite Difference Method (FDM) works on the points of a grid, which can be considered to be edge points of the elements. The main idea is the approximation of the operators occurring in the field equations by the differences of the values of the field on the grid points. These differences are generally finite for finite distances between the grid points.

There are two reasons that FDM approaches usually rely on structured meshes topologically equivalent to regular grids [13]:

i. The resulting linear equation system will be quite sparse, because the finite-difference stencil gives nonzero weight only to neighboring vertices. It is convenient to use the same stencil throughout the mesh. This

restriction simplifies both software development phase and the mathematical analysis of the numerical scheme.

ii. A finite-difference stencil gives a more accurate approximation of a continuous operator when the edges meeting at vertices are nearly orthogonal.

1.1.4.3. Finite Element Method

In the Finite Element Method (FEM), the domains are subdivided into several sub-domains of elements and certain boundary conditions are implied on the boundaries of the elements.

The difference between MoM and FEM is as follows: MoM is based on the expansion of the moments in mechanical elements such as bars, plates, etc., but the FEM expands the deviations.

FEM overcomes most of the limitations of the FDM approach. The essential idea is to replace the unknown function by a finite-dimensional approximation. Then, a variational approach is used to reduce the partial differential equation to a sparse matrix equation. FEMs are typically no more complicated on unstructured meshes than on structured meshes. Furthermore, there is no real advantage in requiring mesh edges to meet orthogonally. Poorly-shaped elements however, can seriously degrade accuracy [1]. In two-dimensional meshes, internal angles must not be small, to guarantee the quality of the mesh.

In two dimensions, the Delaunay triangulation of a point set has the desirable property that it maximizes the minimum angle. Moreover, the Delaunay triangulation gives a FEM matrix, which is diagonally dominant with negative off-diagonal entries, for the Laplacian operator.

1.1.5. Solution Methods

Basic solution methods are direct factorization methods and iterative methods [6, 9]. They are computationally based on the solution of the sparse linear system:

i. These methods exhibit dramatic variations in required storage and computational cost for different problems.
ii. The performance of a solution method is greatly influenced by the mesh generation and discretization steps. For example, although higher-order basis functions in the finite element method allow the use of a coarser mesh, their usage yields a denser linear system. Furthermore, poorly-shaped mesh elements can give an ill-conditioned linear system [1], whose solution is prone to a large error.

1.2. Outline of the Thesis

i. Chapter 1 gives a brief introduction to mesh generation, meshing technologies and numerical methods in solving partial differential equations.
ii. In Chapter 2, structured and unstructured meshing techniques with quad / hexahedral elements are discussed. Formulation of the isoparametric coordinate mapping methods for constructing quad and hex elements is given together with sample outputs of the two of the three software packages developed in this thesis. These two software packages are called Numerical Mesh Generation Software, Structured Surface Mesh Generation (NMGS-SSMG) and Numerical Mesh Generation Software, Structured Volume Mesh Generation (NMGS-SVMG) throughout this text.
iii. Chapter 3 is devoted to the structured and unstructured mesh generation using tri / tetrahedral elements. Unstructured mesh generation algorithms will be classified as: Octree, Delaunay and Advancing Front based methods. Each algorithm uses different approaches to generate high-

quality meshes. Sample outputs of the third software package developed in this thesis are also given in this chapter. The software package is called Numerical Mesh Generation Software, Quality Tetrahedral Mesh Generation (NMGS-QTMG) throughout this text.

iv. In Chapter 4, object oriented design and implementation related aspects, the properties, capabilities and deficiencies of NMGS-SSMG and NMGS-SVMG are described.

v. In Chapter 5, object oriented design and implementation related aspects, the properties, capabilities and deficiencies of NMGS-QTMG are described.

vi. In Chapter 6, the last chapter, performance of the different mesh generation algorithms implemented in the three software packages is discussed.

CHAPTER 2

QUAD / HEXAHEDRAL MESHES

2.1. Structured Quad Meshing

In this thesis, NMGS-SSMG was developed in order to generate structured quad meshes. For this purpose, isoparametric coordinate transformation was used.

2.1.1. Isoparametric Coordinates

Structured meshes can be generated efficiently with isoparametric coordinate mapping [10]. First, the region to be discretized is divided into a number of smaller regions, depending on the geometry. By using this mapping, nodes within each small region are automatically positioned and referenced to a global Cartesian coordinate system, and finally elements are automatically assembled from those nodes. The use of isoparametric transformation has proved to be computationally efficient and reasonably easy to implement in generation of structured meshes.

2.1.1.1. Surface Mesh Generation using Eight-Noded Isoparametric Elements

Eight-noded isoparametric elements have been used to generate structured surface meshes for both two-dimensional and three-dimensional problems. The mesh generator maps the eight-noded quadrilateral to a Cartesian coordinate system ($-1 < \xi < 1$ and $-1 < \eta < 1$). It generates a uniform mesh in this coordinate system and transforms those points back to the curvilinear coordinate system. Fig. 2.1 demonstrates this coordinate mapping.

Considering the case of the eight-noded isoparametric element of Fig. 2.1 in which the x, y and z coordinates of eight nodes are known, these coordinates are expressed in terms of shape functions as

$$x = \sum_{i=1}^{8} N_i \cdot x_i \quad y = \sum_{i=1}^{8} N_i \cdot y_i \quad \text{and} \quad z = \sum_{i=1}^{8} N_i \cdot z_i \tag{2.1}$$

in which x_i, y_i and z_i are the x, y and z coordinates of 8 edge nodes, respectively; N_i is a shape function associated with the i^{th} node and defined in terms of a curvilinear coordinate system with variables ξ and η which has values, ranging from -1 to 1 on opposite sides. Shape functions for the eight-noded isoparametric element are given in Table 2.1.

Table 2.1. Shape functions for an eight-noded isoparametric element

$N_i = \frac{1}{4}(1+\xi\cdot\xi_i)\cdot(1+\eta\cdot\eta_i)\cdot(\xi\cdot\xi_i+\eta\cdot\eta_i-1)$	for corner nodes
$N_i = \frac{1}{2}(1+\xi\cdot\xi_i)\cdot(1-\eta^2)$	for midside nodes with η=0
$N_i = \frac{1}{2}(1+\eta\cdot\eta_i)\cdot(1-\xi^2)$	for midside nodes with ξ=0

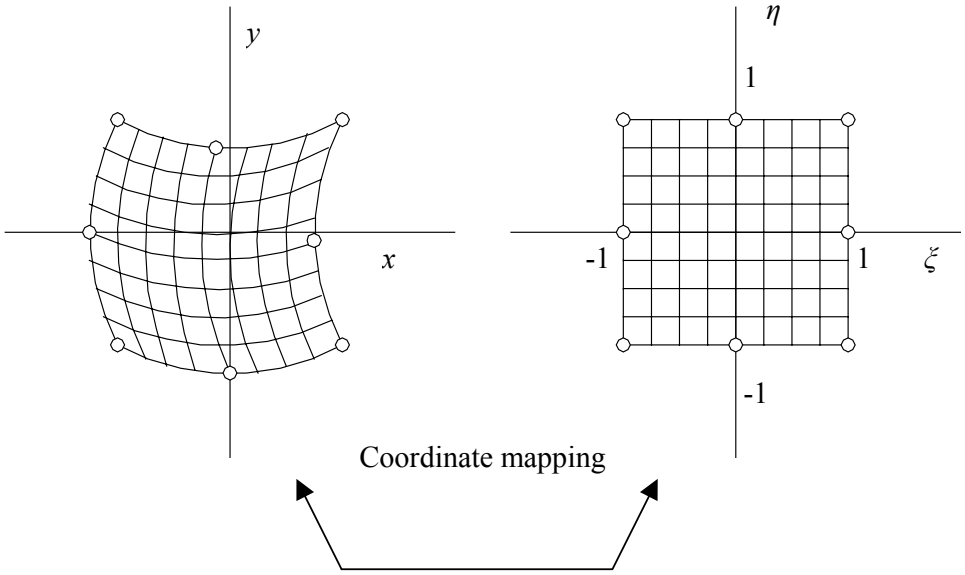

Fig. 2.1. Coordinate mapping for the eight-noded isoparametric element

A mesh generated using one eight-noded isoparametric element is shown in Fig. 2.2.

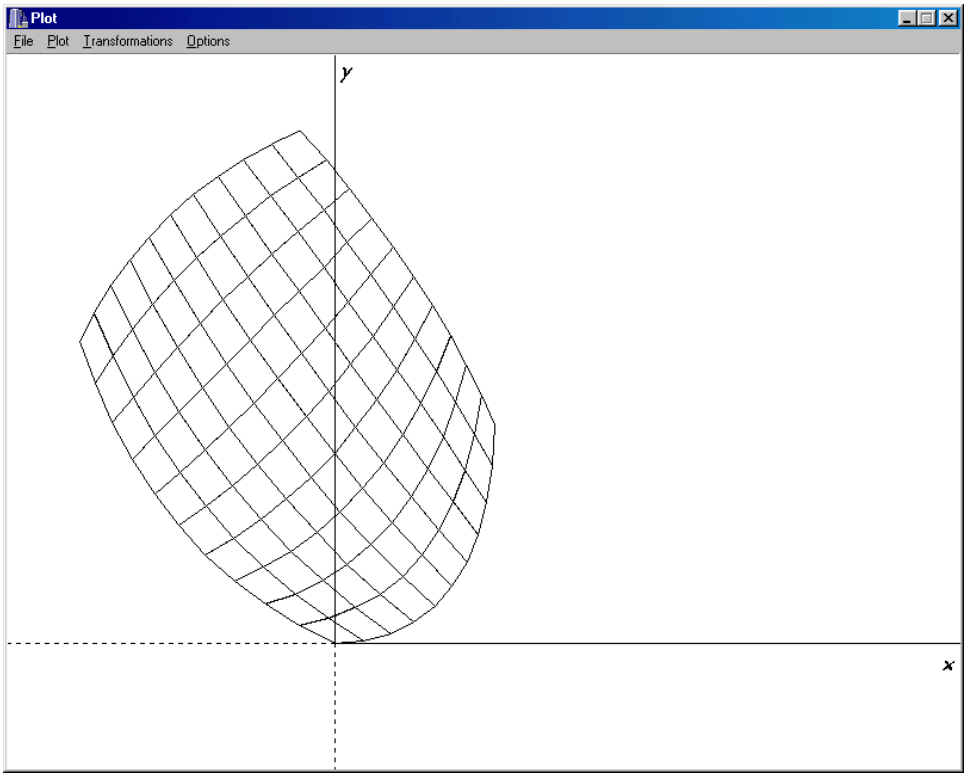

Fig. 2.2. A two dimensional mesh generated using one eight-noded element

Fig. 2.3 is an example of a surface mesh generated using the NMGS-SSMG.

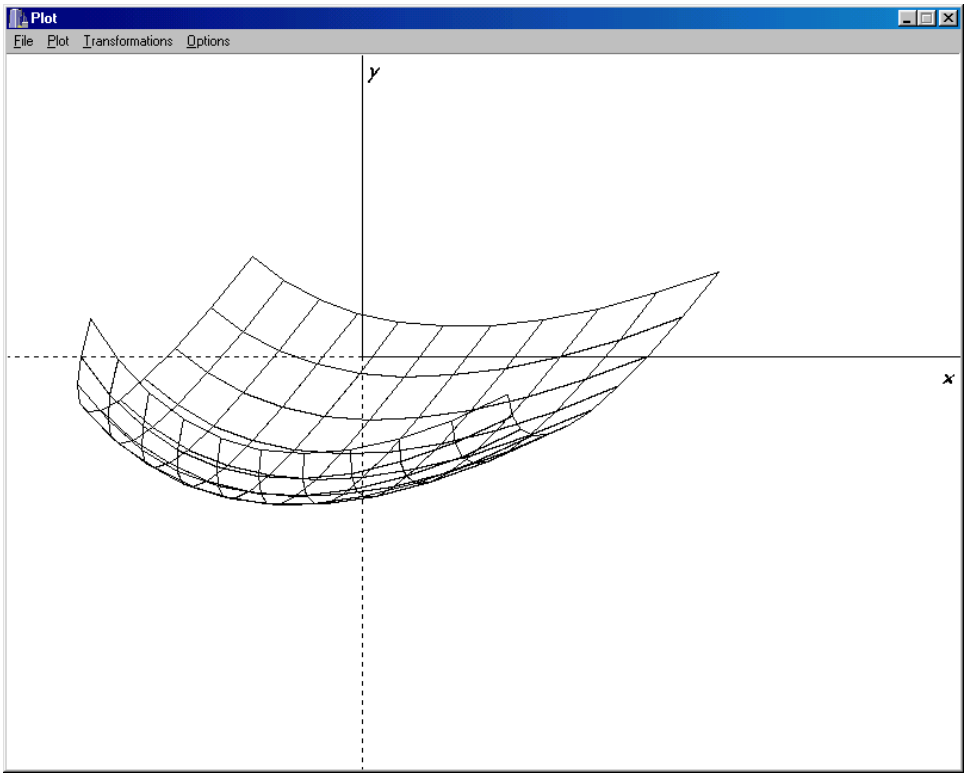

Fig. 2.3. A surface meshed with quadrilateral elements

NMGS-SSMG, was used to generate meshes for complex three-dimensional surfaces embedded in three-dimensional space. A hemispherical surface, which was constructed using five eight-noded isoparametric elements, is given in Fig. 2.4. Top view of the mesh of Fig. 2.4 is given in Fig. 2.5.

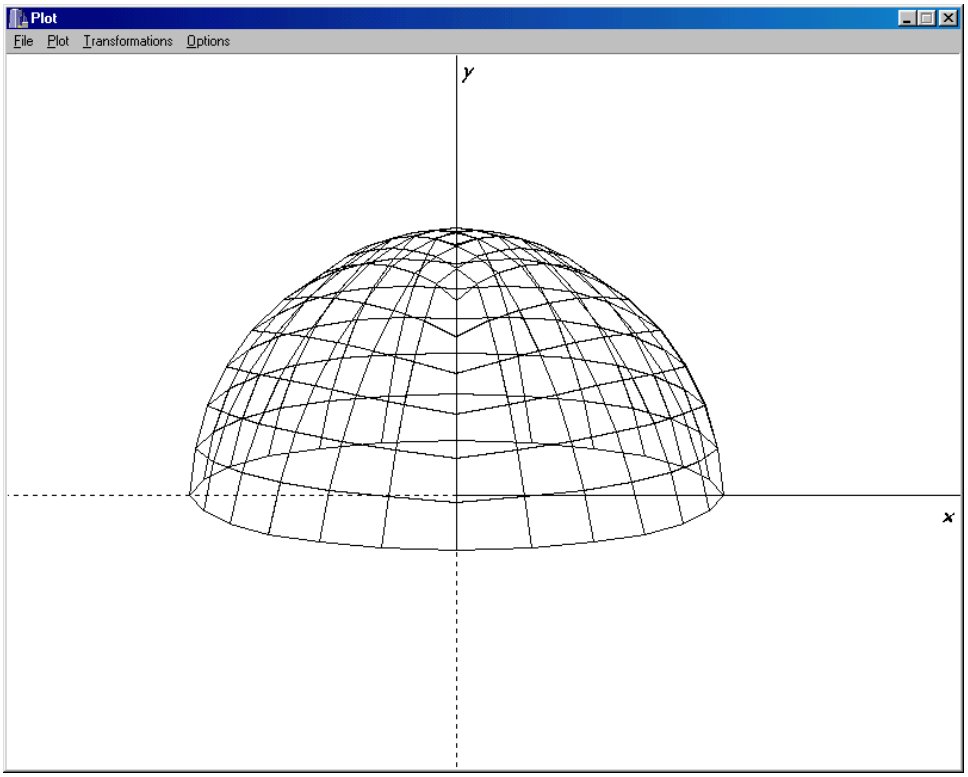

Fig. 2.4. Mesh generated for a hemi-spherical surface using five isoparametric elements

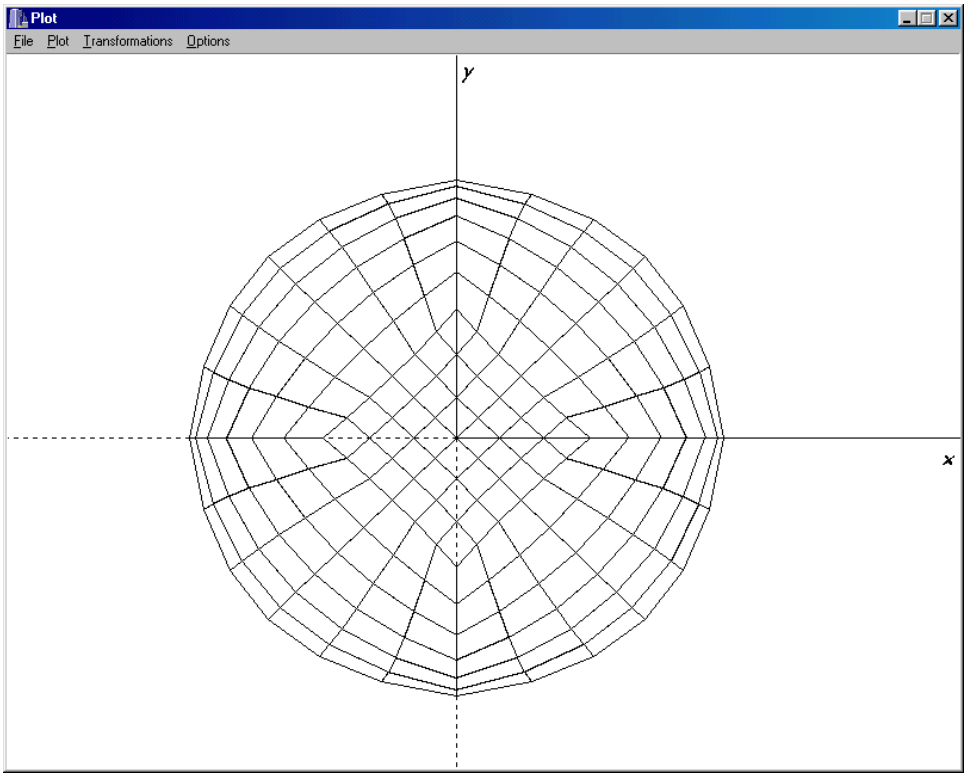

Fig. 2.5. Top view of the mesh given in Fig. 2.4

2.2. Unstructured Quad Meshing

Unstructured quadrilateral meshing algorithms can, in general, be grouped into two main categories as 'indirect' and 'direct' approaches. With the former, the domain is first meshed with triangles and then various algorithms are employed to convert the triangles into quadrilaterals. With the latter, quadrilaterals are placed on the surface directly without first going through the process of triangle meshing.

2.2.1. Indirect Methods

One of the simplest methods for 'indirect quadrilateral mesh generation' includes dividing all triangles into three quadrilaterals, as shown in Fig. 2.6, which

guarantees an all-quadrilateral mesh. However, the element quality of the mesh may be poor, since a number of irregular nodes are introduced. An alternate algorithm is to combine adjacent pairs of triangles to form a single quadrilateral as shown in Fig. 2.7. In this, a large number of triangles may be left while the element quality increases by using this method.

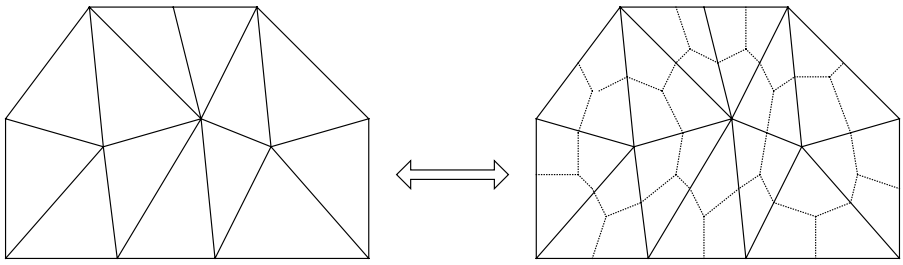

Fig. 2.6. Quad mesh generated by splitting each triangle into three quads

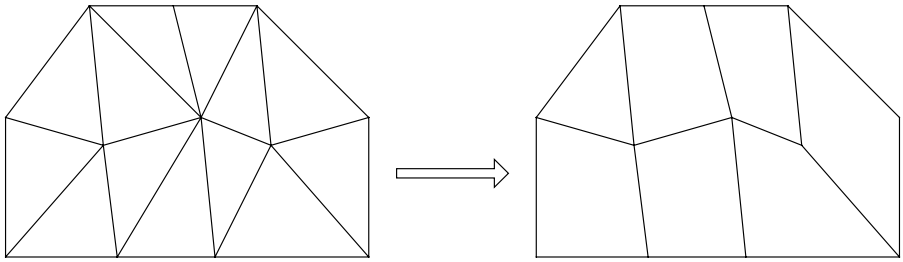

Fig. 2.7. Quad-dominant mesh generated by combining triangles

In these types, it is important to specify which triangles are to be combined in order to maximize the number of quadrilaterals. Lo [11] has defined an algorithm that suggests several heuristic procedures for the order in which triangles could be combined. To increase the number and quality of quads, Johnston [12] proposes additional local element splitting and swapping strategies.

Indirect methods have the advantage of being very fast since all operations are local. Global intersection checks are not necessary as is required with some forms of direct methods. Typically, the disadvantages of indirect methods are that there have been a lot of irregular nodes left in the mesh. Even if few irregular nodes exist, there is no guarantee that the elements will align with the boundary, a desirable property for some applications. Some of the irregular nodes can be reduced, and hence element quality increased by performing topological clean-up operations.

2.2.2. Direct Methods

For direct generation of quad meshes, a lot of methods have been proposed. Of them, two main categories are worth explaining:

i. The methods of the first category are based on some form of decomposition of the domain into simpler regions.
ii. The second ones are those that utilize direct placement of nodes and elements using a moving front approach.

2.2.2.1. Quad Meshing by Decomposition

It was Baehmann [15] who proposed the quadtree decomposition technique, which is among the first methods utilizing decomposition of the area for quadrilateral meshing. Adjusting nodes in order to conform to the boundary, the quadrilateral elements are fitted into the quadtree leaves.

It was Tam [16] who was the first to use the quadrilateral meshing utilizing a medial axis decomposition of the domain. As can be seen in Fig. 2.8, the medial axis is thought of as a series of lines and curves generated from the midpoint of a maximal circle as it is rolled through the area. Having decomposed the area into

simpler regions, sets of templates are then employed to insert quadrilaterals into the domain.

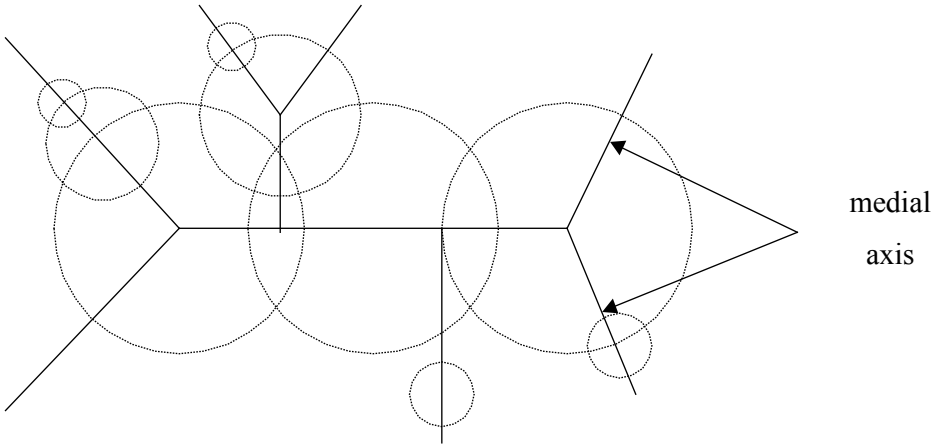

Fig. 2.8. Decomposition of an area using the medial axis

2.2.2.2. Advancing Front Quad Meshing

Zhu [17] is among the first to propose a quadrilateral meshing algorithm using an advancing front approach. Starting with an initial placement of nodes on the boundary, individual elements are formed by projecting edges towards the interior. Two triangles are formed using traditional triangle advancing front methods and then combined to form a single quadrilateral.

2.3. Structured Hex Meshing

In this thesis, structured hexahedral meshes were generated by using NMGS-SVMG. Isoparametric coordinate transformation was used as the meshing algorithm.

2.3.1. Volume Mesh Generation using Twenty-Noded Isoparametric Elements

In the NMGS-SVMG, twenty-noded isoparametric elements have been used to generate structured volume meshes for three-dimensional problems. As shown in Fig. 2.9, the mesh generator maps the twenty-noded cuboid to a Cartesian coordinate system ($-1 < \xi < 1$, $-1 < \eta < 1$ and $-1 < \zeta < 1$), generating a uniform mesh in this coordinate system and transforming those points back to the curvilinear coordinate system.

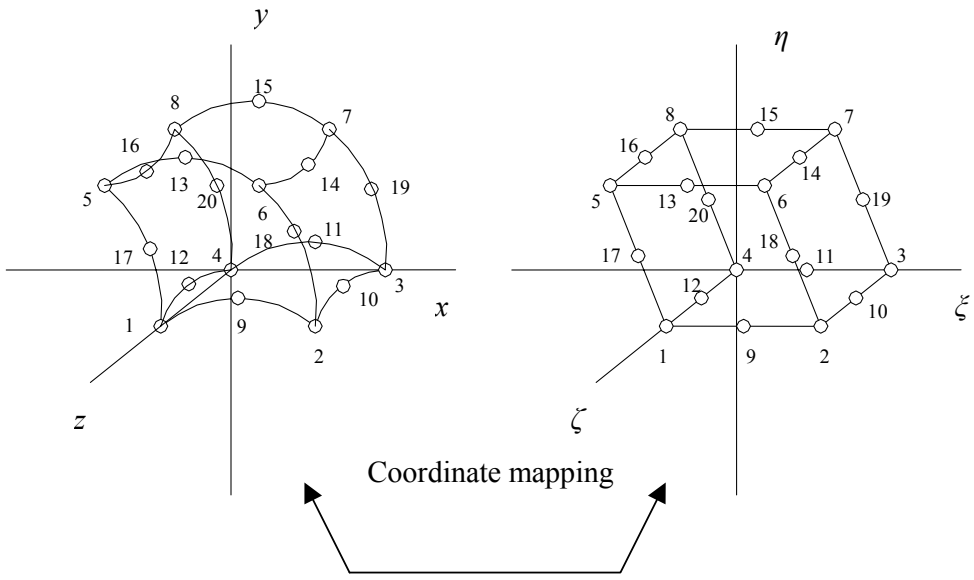

Fig. 2.9. Coordinate mapping for the twenty-noded isoparametric element

Considering the case of the twenty-noded isoparametric element shown in Fig. 2.9, the x, y and z coordinate variations are expressed in terms of shape functions and the nodal coordinates (x_i, y_i, z_i) as follows

$$x = \sum_{i=1}^{20} N_i \cdot x_i \quad y = \sum_{i=1}^{20} N_i \cdot y_i \quad \text{and} \quad z = \sum_{i=1}^{20} N_i \cdot z_i \tag{2.2}$$

in which N_i is a shape function associated with the i^{th} node and defined in terms of a curvilinear coordinate system ξ, η and ζ which has values ranging from -1 to 1 on opposite sides. Shape functions for the twenty-noded isoparametric element are given in Table 2.2.

A twenty-noded element appears in Fig. 2.10, where only the surfaces in x and z directions are shown. Sometimes it is hard to visualize wire frame views of three-dimensional objects, so the NMGS-SVMG, lets the user view the surfaces in the selected directions.

Table 2.2. Shape functions for the twenty-noded isoparametric element

$$N_9 = \frac{1}{4}\left(1-\xi^2\right)\cdot\left(1-\eta\right)\cdot\left(1-\zeta\right) \qquad N_{10} = \frac{1}{4}\left(1+\xi\right)\cdot\left(1-\eta^2\right)\cdot\left(1-\zeta\right)$$

$$N_{11} = \frac{1}{4}\left(1-\xi^2\right)\cdot\left(1+\eta\right)\cdot\left(1-\zeta\right) \qquad N_{12} = \frac{1}{4}\left(1-\xi\right)\cdot\left(1-\eta^2\right)\cdot\left(1-\zeta\right)$$

$$N_{13} = \frac{1}{4}\left(1-\xi^2\right)\cdot\left(1-\eta\right)\cdot\left(1+\zeta\right) \qquad N_{14} = \frac{1}{4}\left(1+\xi\right)\cdot\left(1-\eta^2\right)\cdot\left(1+\zeta\right)$$

$$N_{15} = \frac{1}{4}\left(1-\xi^2\right)\cdot\left(1+\eta\right)\cdot\left(1+\zeta\right) \qquad N_{16} = \frac{1}{4}\left(1-\xi\right)\cdot\left(1-\eta^2\right)\cdot\left(1+\zeta\right)$$

$$N_{17} = \frac{1}{4}\left(1-\xi\right)\cdot\left(1-\eta\right)\cdot\left(1-\zeta^2\right) \qquad N_{18} = \frac{1}{4}\left(1+\xi\right)\cdot\left(1-\eta\right)\cdot\left(1-\zeta^2\right)$$

$$N_{19} = \frac{1}{4}\left(1+\xi\right)\cdot\left(1+\eta\right)\cdot\left(1-\zeta^2\right) \qquad N_{20} = \frac{1}{4}\left(1-\xi\right)\cdot\left(1+\eta\right)\cdot\left(1-\zeta^2\right)$$

$$N_1 = \frac{1}{8}\left(1-\xi\right)\cdot\left(1-\eta\right)\cdot\left(1-\zeta\right) - \frac{1}{2}\left(N_{17}+N_{12}+N_9\right)$$

$$N_2 = \frac{1}{8}\left(1+\xi\right)\cdot\left(1-\eta\right)\cdot\left(1-\zeta\right) - \frac{1}{2}\left(N_{18}+N_{10}+N_9\right)$$

$$N_3 = \frac{1}{8}\left(1+\xi\right)\cdot\left(1+\eta\right)\cdot\left(1-\zeta\right) - \frac{1}{2}\left(N_{19}+N_{11}+N_{10}\right)$$

$$N_4 = \frac{1}{8}\left(1-\xi\right)\cdot\left(1+\eta\right)\cdot\left(1-\zeta\right) - \frac{1}{2}\left(N_{20}+N_{12}+N_{11}\right)$$

$$N_5 = \frac{1}{8}\left(1-\xi\right)\cdot\left(1-\eta\right)\cdot\left(1+\zeta\right) - \frac{1}{2}\left(N_{17}+N_{16}+N_{13}\right)$$

$$N_6 = \frac{1}{8}\left(1+\xi\right)\cdot\left(1-\eta\right)\cdot\left(1+\zeta\right) - \frac{1}{2}\left(N_{18}+N_{14}+N_{13}\right)$$

$$N_7 = \frac{1}{8}\left(1+\xi\right)\cdot\left(1+\eta\right)\cdot\left(1+\zeta\right) - \frac{1}{2}\left(N_{19}+N_{15}+N_{14}\right)$$

$$N_8 = \frac{1}{8}\left(1-\xi\right)\cdot\left(1+\eta\right)\cdot\left(1+\zeta\right) - \frac{1}{2}\left(N_{20}+N_{16}+N_{15}\right)$$

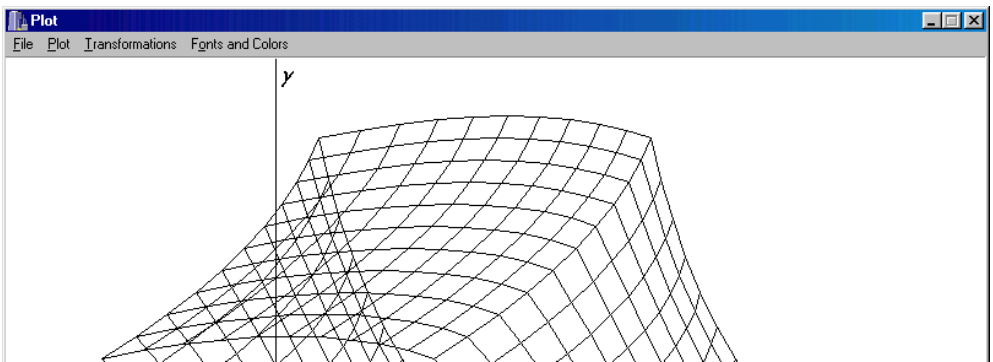

Fig. 2.10. Mesh generated using one twenty-noded isoparametric element

2.4. Unstructured Hex Meshing

Similar to quadrilateral meshing, there are both ‘indirect’ and ‘direct’ methods for unstructured hex meshing.

2.4.1. Indirect Methods

Although indirect methods are not widely used, they have been proposed for some applications [22]. After a solid has been meshed with tetrahedral mesh elements, each tetrahedron can be subdivided into four hexahedra as shown in Fig. 2.11. Due to the poor element quality, most finite element analysts have rejected this solution.

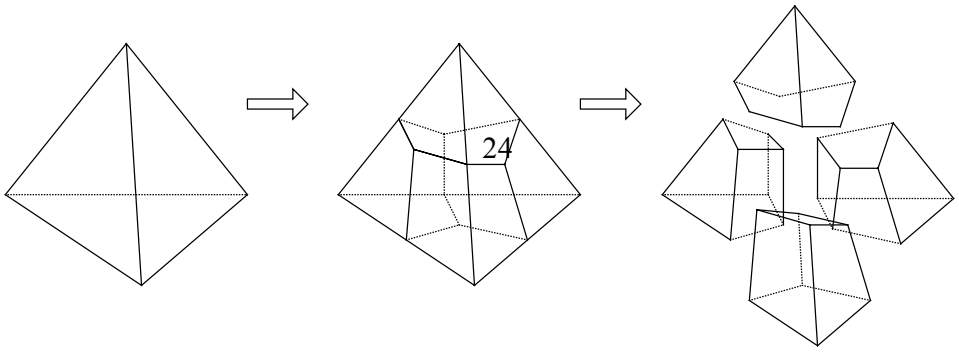

Fig. 2.11. Decomposition of a tetrahedron into four hexahedra

2.4.2. Direct Methods

There are currently four distinct strategies proposed for unstructured hexahedral mesh generation in the literature:

i. Grid-based,
ii. Medial surface,
iii. Plastering, and
iv. Whisker weaving.

2.4.2.1. Grid-Based Methods

The grid-based approach was first proposed by Schneiders [23]. The properties of this approach can be summarized in four steps:

i. It generates a fitted three-dimensional grid of hex elements on the interior of the volume.
ii. Hex elements are added at the boundaries to fill gaps where the regular grid of hexes does not meet flush with the surface.
iii. It tends to generate poor quality elements at the boundary of the volume. Hex elements will in general not be aligned with the boundary.

iv. The resulting mesh generated from the grid-based approach is highly dependent upon the orientation of the interior grid of hex elements. Their sizes must be approximately all the same.

2.4.2.2. Medial Surface

Medial surface method [24] involves an initial decomposition of the volume:

i. As a direct extension of the medial axis method for quad meshing, the domain is subdivided by a set of medial surfaces. They can be thought of as the surfaces generated from the midpoint of a maximal sphere as it is rolled through the volume.
ii. The decomposition of the volume by medial surfaces is said to generate map meshable regions.
iii. A series of templates for the expected topology of the regions formed by the medial surfaces are utilized to fill the volume with hexahedra.
iv. While proving useful for some geometry, this method has been less than reliable for general geometry.

2.4.2.3. Plastering

As shown in Fig. 2.12, plastering [25] is a process in which elements are first placed starting with the boundaries and advancing towards the center of the volume. The main aspects of this method can be summarized as follows:

i. Similar to other advancing front algorithms, a current front is defined consisting of all quadrilaterals. Individual quads are projected towards the interior of the volume to form hexahedra.
ii. Plastering must detect intersecting faces and determine when and how to connect to pre-existing nodes or to seam faces.

iii. As the algorithm advances, complex interior voids may result, which in some cases are impossible to fill with all-hex elements.

iv. Existing elements, already placed by the plastering algorithm must sometimes be modified in order to facilitate placement of hexes towards the interior.

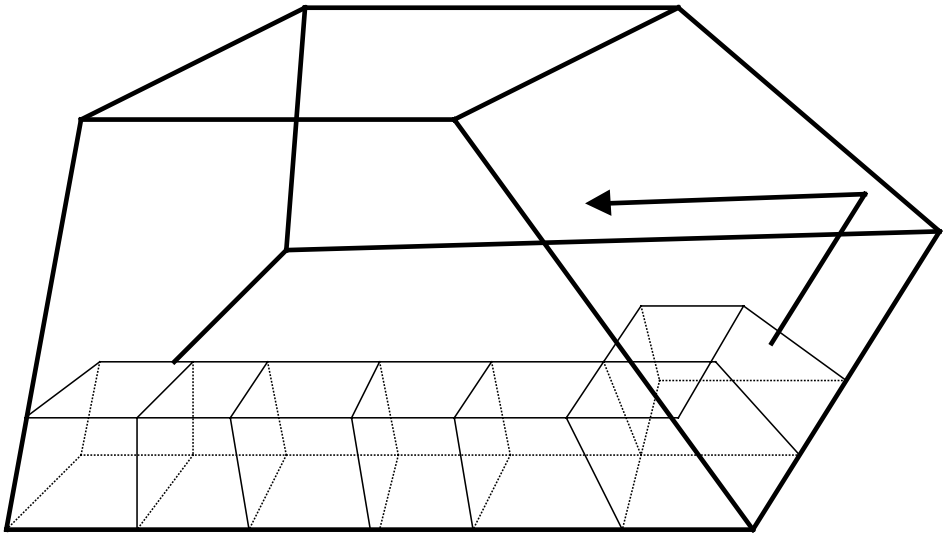

Fig. 2.12. Plastering process forming elements at the boundary

2.4.2.4. Whisker Weaving

Whisker weaving, which was first introduced by Tautges and Blacker [26], is based on the concept of the spatial twist continuum (STC) [27]. They describe the STC as the dual of the hexahedral mesh, represented by an arrangement of intersecting surfaces, which bisect hexahedral elements in each direction. Fig. 2.13 shows a simple representation of the twist planes of the STC defined for a volume composed of only two hexahedra.

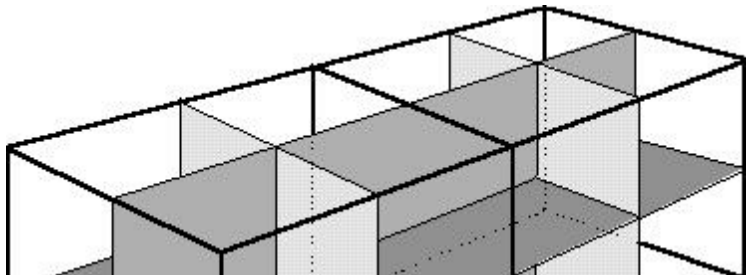

Fig. 2.13. The STC composed of four twist planes, for a solid composed of two hexahedra

The whisker weaving algorithm can be explained as in the following steps:

i. The principal behind this method is to first construct the STC or dual of the hex mesh.
ii. With a complete STC, the hex elements can then be fitted into the volume using the STC as a guide. This is done by beginning with a topological representation of the loops formed by the intersection of the twist planes with the surface.
iii. The loops can be easily determined from an initial quad mesh of the surface.
iv. The objective of the algorithm is to determine where the intersections of the twist planes will occur within the volume.
v. Once a valid topological representation of the twist planes has been achieved, hexes are then formed inside the volume. One hex is formed wherever three twist planes converge.

CHAPTER 3

TRI / TETRAHEDRAL MESHES

3.1. Structured Triangular Meshing

NMGS-SSMG first generates quadrilateral mesh elements and then triangulates the generated mesh. The software package that directly triangulates two-dimensional regions using Octree and Delaunay methods, can be found in [2].

3.1.1. Triangulation of Quad Meshes

Triangulation of quad meshes is done by calculating the length of the two diagonals of the quadrilateral and using the shorter one, in the NMGS-SSMG. That creates a high quality triangulation. The method is explained in Fig. 3.1.

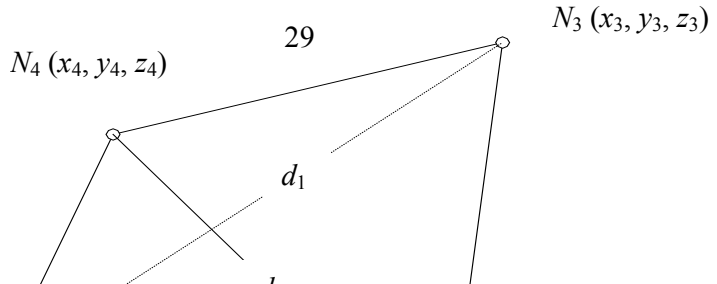

Fig. 3.1. Triangulation of a quadrilateral mesh element

Let the four nodes of the quadrilateral be denoted by N_1, N_2, N_3, N_4 and the two lengths of the diagonals by d_1 and d_2. These two parameters are calculated using

$$d_1 = \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2 + (z_1 - z_3)^2}$$

$$d_2 = \sqrt{(x_2 - x_4)^2 + (y_2 - y_4)^2 + (z_2 - z_4)^2} \qquad (3.1)$$

As $d_2 \leq d_1$, the two triangles created are $N_1N_2N_4$ and $N_3N_4N_2$.

Fig. 3.2 is the triangulated version of the surface shown in Fig. 2.5.

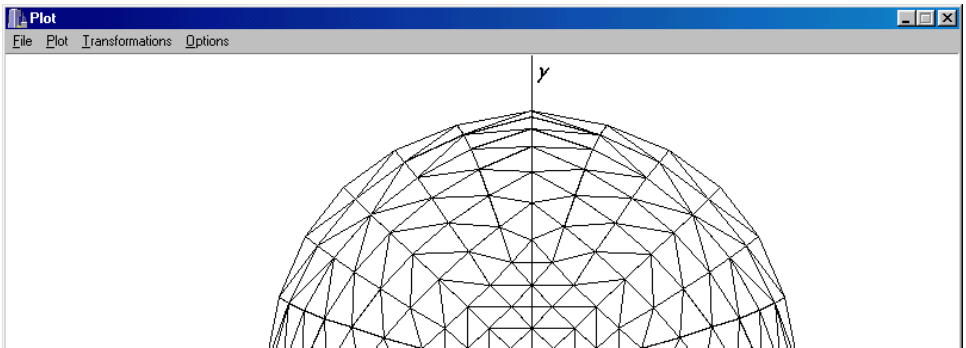

Fig. 3.2. Triangulation of the surface shown in Fig. 2.5

3.2. Structured Tetrahedral Meshing

NMGS-SVMG first generates hexahedral mesh elements and then tetrahedralizes the generated mesh. Hexahedral node insertion method [28] is used in order to obtain a number of tetrahedral mesh elements from a hexahedral element.

3.2.1. Hexahedral Node Insertion

Hexahedral node insertion is simply inserting an additional node at the centroid of a hexahedral element. This process involves splitting the hexahedral element into twelve tetrahedra. Fig. 3.3 shows how hexahedra would be split. All the six quad faces of the hexahedral element are triangulated as explained in Section 3.1.1. For the sake of clarity, only triangulation of six quad faces and single tetrahedron are shown in Fig. 3.3. After inserting a new node, namely node 9, two

tetrahedra are obtained as $N_1N_2N_3N_9$ and $N_3N_4N_1N_9$ by triangulating the quad face $N_1N_2N_3N_4$. Other ten tetrahedra can be specified similarly.

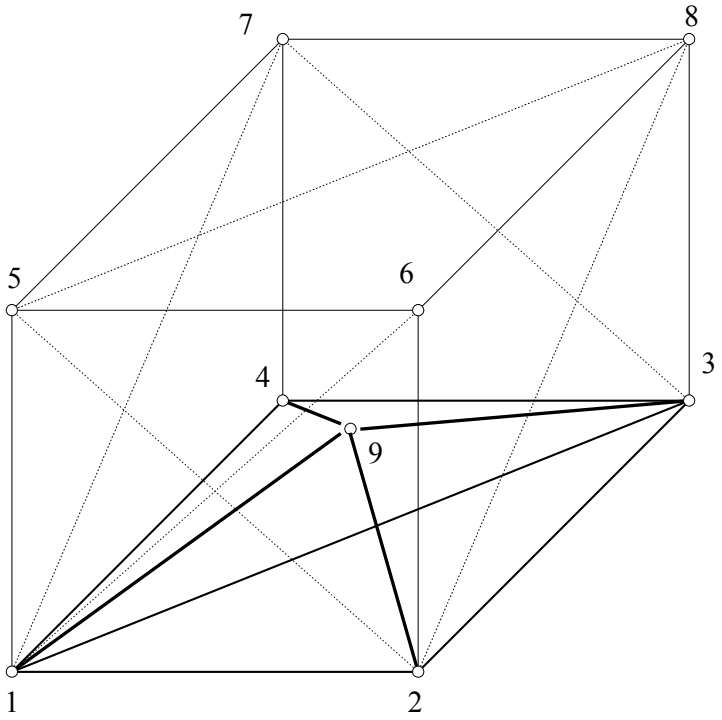

Fig. 3.3. Tetrahedra formed by inserting a single node in a hexahedral element

While this method is relatively simple and tends to produce reasonable quality elements, there are some drawbacks. It is usually the hexahedra in which the user has an additional effort to place within a model. Unless the elements are sufficiently fine that removal of some of the interface hexahedral elements will cause little difference, most would tend to object to the modification of this one layer.

Tetrahedralization of a hemisphere using NMGS-SVMG appears in Fig. 3.4.

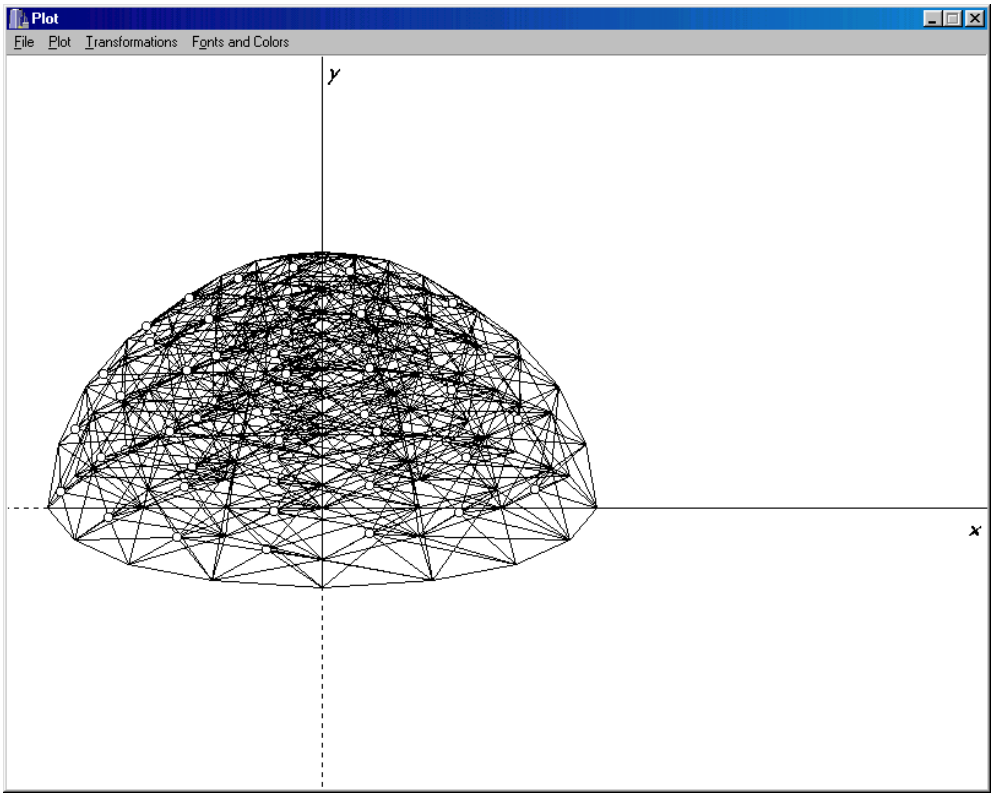

Fig. 3.4. Tetrahedralization of a hemisphere using NMGS-SVMG

3.3. Unstructured Tri / Tetrahedral Meshes

Triangular and tetrahedral meshing are the most common forms of unstructured mesh generation. Currently used techniques can be viewed in three categories: Octree, Delaunay and Advancing Front methods.

Complexity of the methods may differ significantly when moving two-dimensional to three-dimensional problems, but the algorithms are mostly applicable for both triangular and tetrahedral mesh generation.

In this thesis, NMGS-QTMG was developed in order to generate unstructured triangular and tetrahedral meshes. NMGS-QTMG is able to generate exact

Delaunay tetrahedralizations, constrained (conforming) Delaunay tetrahedralizations, and quality conforming Delaunay tetrahedralizations using TetGen [31] which was developed by Hang [29].

Fig. 3.5 is an example of a surface mesh of a sphere generated by NMGS-QTMG. Tetrahedralization of a cube using NMGS-QTMG appears in Fig. 3.6.

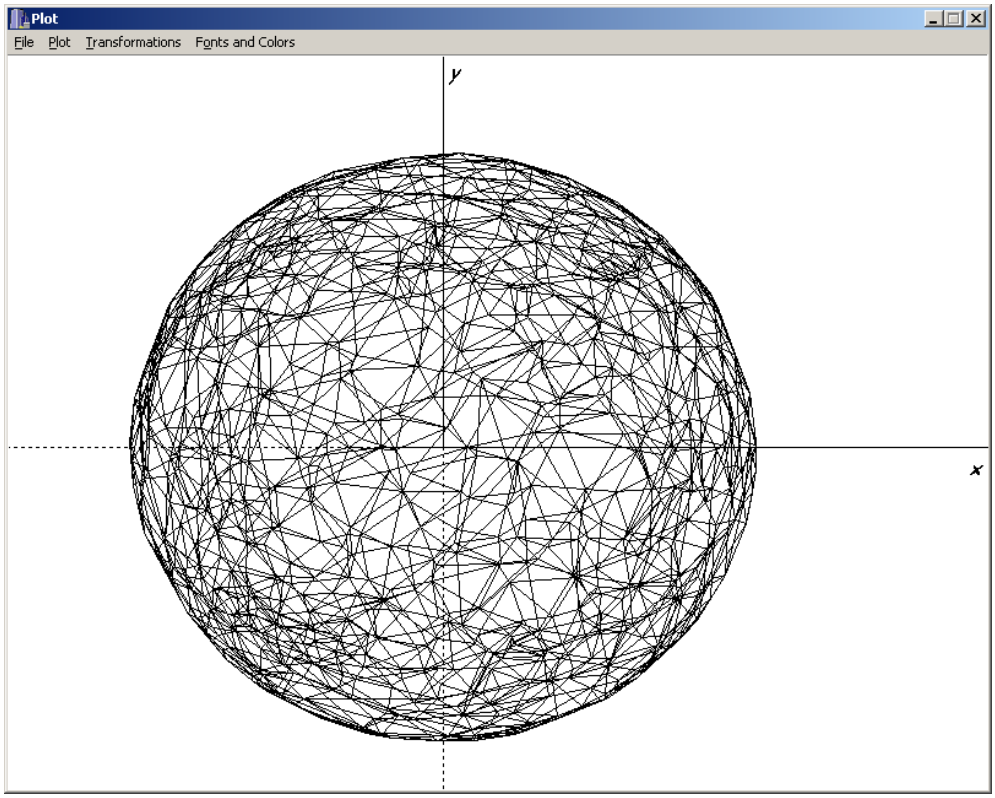

Fig. 3.5. Surface mesh of a sphere generated by NMGS-QTMG

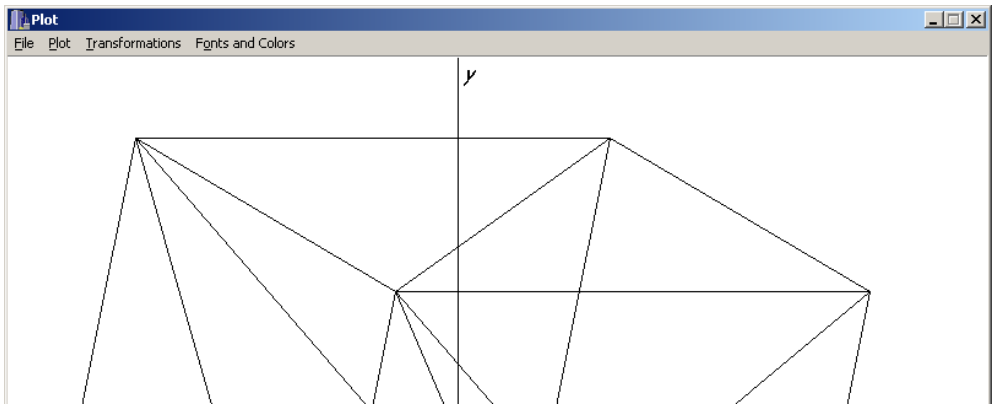

Fig. 3.6. Tetrahedralization of a cube using NMGS-QTMG

3.3.1. Octree

The Octree technique was primarily developed in the 1980s by Mark Shephard's [18, 19] group. Let's see the major properties of this method:

i. With this method, cubes containing the geometric model are recursively subdivided until the desired resolution is reached. Fig. 3.7 shows the equivalent two-dimensional quadtree decomposition of a model.
ii. Irregular cells are then created where cubes intersect the surface, often requiring a significant number of surface intersection calculations.
iii. Tetrahedra are generated from both the irregular cells on the boundary and the internal regular cells.
iv. Unlike advancing front or Delaunay techniques, the Octree technique does not match a pre-defined surface mesh. However, surface facets are formed wherever the internal octree structure intersects the boundary.

v. To ensure element sizes do not change too dramatically, a maximum difference in octree subdivision level between adjacent cubes can be limited to ‘one’.

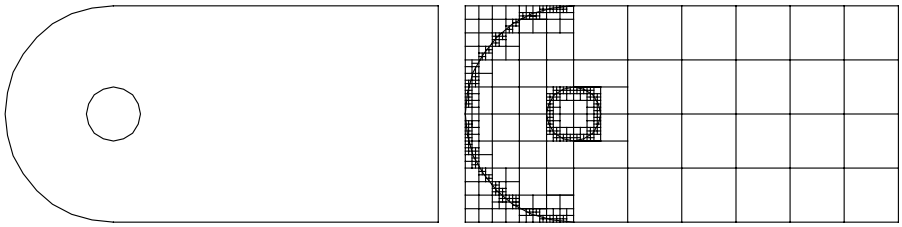

Fig. 3.7. Quadtree decomposition of a simple two-dimensional object

3.3.2. Delaunay

The most popular of the triangle and tetrahedral meshing techniques are those utilizing the Delaunay [20] criterion, which is sometimes called the “empty sphere” property, states that any node must not be contained within the circumsphere of any tetrahedra within the mesh. A circumsphere can be defined as the sphere passing through all four vertices of a tetrahedron. Fig. 3.8 is a simple two-dimensional illustration of the criterion. Since the circumcircles of the triangles in Fig. 3.8(a) do not contain the nodes of other triangle, the empty circle property is maintained.

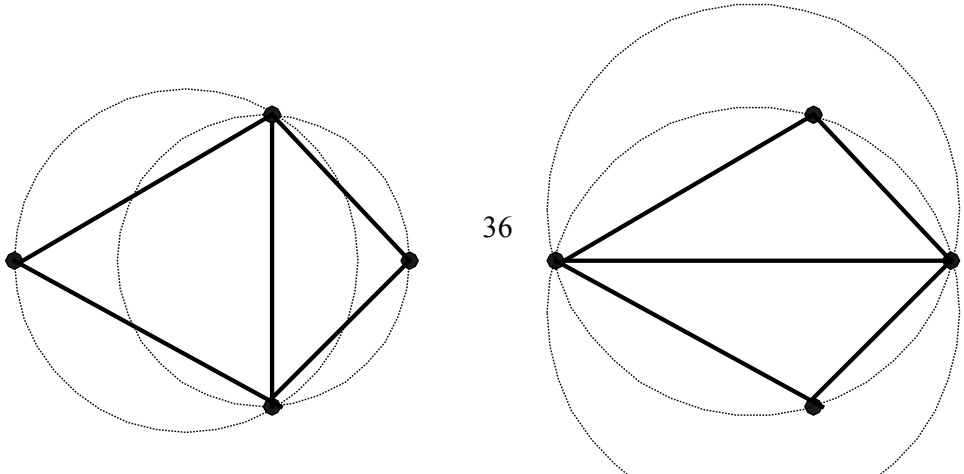

Fig. 3.8. Example of Delaunay criterion (a) maintains the criterion while (b) does not

i. The Delaunay criterion itself is not an algorithm for generating a mesh. It only provides the criteria how the set of existing points in space is connected; nevertheless, it is necessary to provide a method for generating node locations within the geometry.
ii. A typical approach is to first mesh the boundary of the geometry to provide an initial set of nodes.
iii. The boundary nodes are then triangulated according to the Delaunay criterion.
iv. Nodes are then inserted incrementally into the existing mesh, redefining the triangles or tetrahedra locally as each new node is inserted to maintain the Delaunay criterion.
v. It is the method that is chosen for defining where to locate the interior nodes that distinguishes one Delaunay algorithm from another.

3.3.2.1. Point Insertion

It is possible to state the properties of this method as follows [14]:

i. The simplest point insertion approach is to define nodes from a regular grid of points covering the domain at a specified nodal density. In order to

provide for varying element sizes, a user-specified sizing function can also be defined and nodes inserted until the underlying sizing function is satisfied.

ii. Another approach is for nodes to be recursively inserted at triangle or tetrahedral centroids. The nodes are inserted at a tetrahedron's centroid provided the underlying sizing function is not violated.

iii. An alternate approach is to define new nodes at element circumcircle / sphere centers. When a specific order of insertion is followed, this technique is often referred to as "Guaranteed Quality" as triangles can be generated with a minimum bound on any angle in the mesh.

iv. Another technique introduced is called Voronoi-segment point insertion method, similar to the circumcircle point insertion method. A Voronoi segment can be defined as the line segment between the circumcircle centers of two adjacent triangles or tetrahedra. The new node is introduced at a point along the Voronoi segment in order to satisfy the best local size criteria. This method tends to generate very structured looking meshes with six triangles at every internal node.

3.3.2.2. Boundary Constrained Triangulation

In many finite element applications, there is a requirement that an existing surface triangulation be maintained. In most Delaunay approaches, a three-dimensional tessellation of the nodes on the geometry surface is produced before internal nodes are generated. In this process, there is no guarantee that the surface triangulation will be satisfied. In many implementations, the approach is to tessellate the boundary nodes using a standard Delaunay algorithm without regard for the surface facets. A second step is then employed to force or recover the surface triangulation. Of course, by doing so, the triangulation may no longer be strictly 'Delaunay'. For this reason, it is called as 'Boundary Constrained Delaunay Triangulation' .

Edge recovery is done by performing a series of tetrahedral transformations by swapping two adjacent tetrahedra for three, as shown in Fig. 3.9. Where a swap cannot resolve the edge, nodes must sometimes be inserted. After edges have been recovered, in order to recover the face, additional transformations are performed, mostly characterized by swapping three adjacent tetrahedra at an edge for two. More complex transformations or additional nodes can be inserted during the face recovery phase if the transformations do not resolve the surface facet.

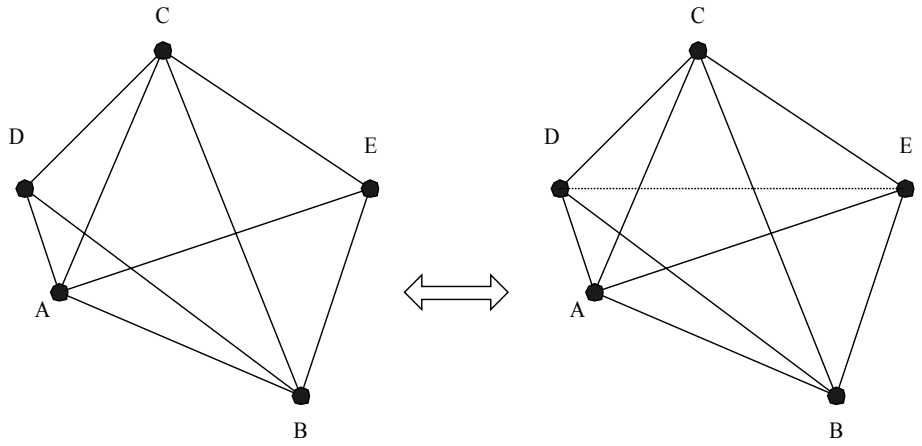

Fig. 3.9. Tetrahedral transformation where two tetrahedra are swapped to three

3.3.3. Advancing Front

Another very popular family of triangular and tetrahedral mesh generation algorithms is the advancing front, or moving front method. The two of the major properties of this method are as follows [14]:

i. In this method, the tetrahedra are built progressively inward from the triangulated surface. An active front is maintained where new tetrahedra are formed. Fig. 3.10 is a simple two-dimensional example of the advancing front, where triangles have been formed at the boundary. As

the algorithm progresses, the front will advance to fill the remainder of the area with triangles.

ii. In three-dimensions, for each triangular facet on the front, an ideal location for a new fourth node is computed. Also the determined ones are any existing nodes on the front that may form a well-shaped tetrahedron with the facet. The algorithm selects either the new fourth node or an existing node to form the new tetrahedron based on which will form the best tetrahedron. Also the required ones are intersection checks to ensure that tetrahedron do not overlap as opposing fronts advance towards each other. A sizing function can also be defined in this method to control element sizes.

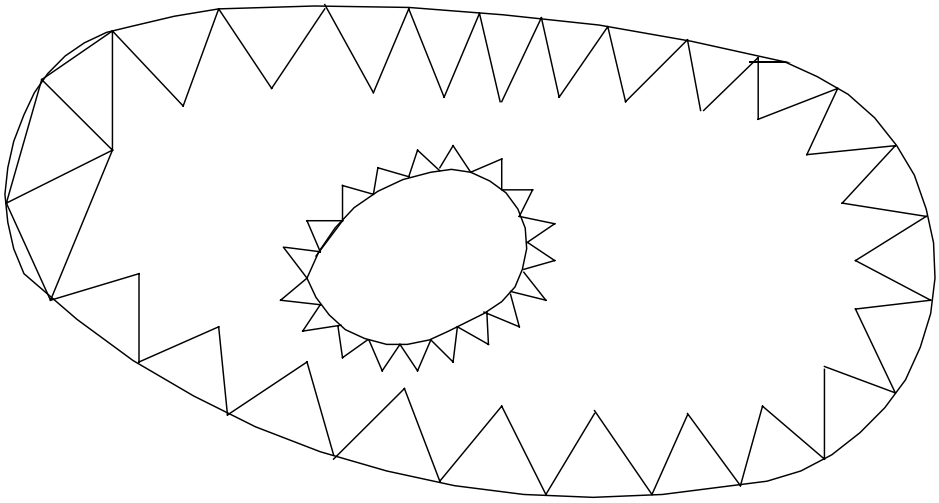

Fig. 3.10. Example of advancing front where one layer of triangles has been placed

CHAPTER 4

DESCRIPTION OF NMGS-SSMG AND NMGS-SVMG

4.1. Definitions of Basic Concepts

These are the definitions of some of the basic concepts that are referenced in this thesis and should be known in order to follow:

i. **Operating System**: A software package installed on a computer equipped with a standard hardware, which lets user operate certain applications by controlling and using all the hardware resources. Popular operating systems now, are capable of running a lot of software packages, such as program development environments, automation, multimedia, games, Internet access and browsing, etc.

ii. **Borland C++ Builder**: C++ Builder is Borland's rapid application development (RAD) product for writing C++ applications. Win32 console applications (32-bit programs that run in a DOS box under Windows 9x or Windows NT) or Win32 GUI (graphical user interface) programs can be created using Borland C++ Builder.

iii. **Visual Component**: It is a self-contained piece of binary software that performs some specific predefined task, such as a text label, an edit control or a list box.

iv. **Dynamic Memory Allocation**: It means that memory required for an object is allocated from the heap. The heap in a Windows program refers to all of the computer's virtual memory.

v. **Project**: A project is a collection of files that work together to create a standalone executable code.

vi. **Unit**: The term Unit is used to refer to source files by Borland C++ Builder.

vii. **Form**: Forms are the main building block of a C++ Builder application. They are the structures that include all the visual components and GUI controls.

viii. **Function**: Functions are sections of code, separate from the main program, that performs a single, well-defined service.

4.2. General Information about the Software Packages

These software packages were developed in 'Borland C++ Builder 5.0' using object oriented programming. Project files were created for each of the software packages in this software development environment. The compiler compiles all the files contained in the projects and finally links the object codes into exacutable files that can be run in Microsoft Windows compatible operating systems, i.e., Windows 9x and Windows NT. Although the compiler specifies and inserts the necessary codes for all the visual components contained in the projects, the source codes were written by the author himself using ANSI C programming language.

As it is in any other Windows programs, also these programs realize their execution by using standard Windows API. In other words, they use many of the standard visual components, such as 'form', 'main menu', 'memo', 'button', etc.

These programs get inputs manually or from certain formatted files which were prepared earlier. In accordance with these inputs, besides they both perform

dynamic memory allocation and solve the problems, that is, they divide the domains defined into regular mesh elements, they plot the geometries and save the mesh information in result files. Both the problem setup and the result files can be edited from any text editor as they are in ASCII format. Since the result files include problem setup, instead of generating meshes using new inputs each time, it is possible to analyze problems that were solved earlier and to perform required changes on the problem setup as well.

4.3. Common Features of the Software Packages

Both programs have some features in common. Hence, they are mentioned within one section.

4.3.1. General Features

The general features of the programs can be summarized under two titles:

i. How to Work
ii. Plot Options

4.3.1.1. How to Work

In order to start the programs, 'NMGS_SSMG_Ver1_2.exe' or 'NMGS_SVMG_Ver1_2.exe' are run. When the programs are first opened, Mesh Generation menu item is disabled, since there will be no data to be processed. At this point, either a setup or result file will be loaded from a fomatted file or a new problem setup will be created manually. If a new valid problem setup has been created, Mesh Generation => Generate item is used to generate mesh for the current problem. In this case, Mesh Generation => Plot item may be used to visualize the generated mesh or the sub-items of the Results menu item can be

used to process the results of the problem solution. On the other hand, if a solution is loaded from a formatted file, since mesh generation is already done, it is not necessary to re-generate the mesh. However, since this formatted result file inlcudes the problem setup as well, it is possible to re-generate another mesh by changing the problem setup inputs. The solutions of the problems may be saved in formatted files by using Results => Save item. In formatted result files, there is a field which is spared to the user for future reference.

4.3.1.2. Plot Options

After generating the mesh, Mesh Generation => Plot item is used to open the plot form, which is shown in Fig. 4.1. The plot is drawn by using Plot => Execute item. Since the plot to be drawn can be seen only as a two-dimensional image on the screen, the axes that can be seen on the form are only *x* and *y*. For this reason, the three-dimensional objects are first seen with a default view angle. Visualization of the third dimension is possible by using the rotation transformation. Hence, the view angle will be changed.

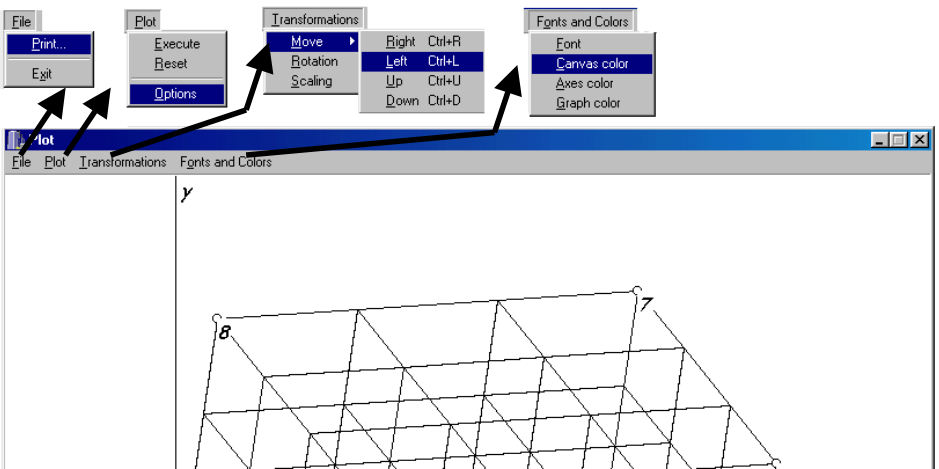

Fig. 4.1. The appearance, the main menu items and their sub-items of the plot form

By using Plot => Options item, the properties of appearance of the three-dimensional objects can be specified. Once a plot has been drawn, Transformations menu item is enabled. This menu item includes Moving, Scaling and Rotation. At the same time, the fonts and colors of the canvas of the plot form can be changed by using the sub-items included in Fonts and Colors menu item.

The information about the transformations implemented in these programs is as follows:

4.3.1.2.1. Moving

Moving is used for sliding the origin so that three-dimensional objects can be best seen on the form. At each moving, the origin slides by only one pixel.

4.3.1.2.2. Scaling

Scaling transformation is used to scale three-dimensional objects. Because of the reason that explained in Section 4.3.1.2, only x and y axes are scaled. As in the case in moving, also scaling transformation provides a flexibility so that the three- dimensional objects can be best seen.

4.3.1.2.3. Rotation

The rotation transformation [21] implemented in these programs is used to improve visualization while displaying three-dimensional objects by looking at those objects from different view angles in three-dimensional Cartesian space. It lets the user specify the rotation angles around x, y and z axes, from 0° to 360°.

4.3.2. Possible Improvements

These programs, in fact, have revealed their application aims correctly, that is, as long as the correct inputs are entered through the programs, correct results can be reached and the required information saved in output files. In spite of this, such possible improvements may be achieved:

i. Instead of entering the inputs that define the certain standard geometric shapes each time, such as squares, cubes, spheres, cones, etc., these can be obtained as ready-made in the programs.
ii. Since it is difficult to divide the complex geometric domains into structured elements and requires more user interactions, these programs are not much appropriate for the solution of such problems. However, new spare programs which break three-dimensional geometries into structural elements and calculate the coordinates of the edge nodes of the elements can be developed.

iii. For the purpose of three-dimensional objects being able to be seen in detail and in various forms, more transformation methods may be improved in addition to the plot options.

4.3.3. Portability Issues

These programs were developed and completed on two personal computers equipped with Intel Pentium-II CPU, 128MB RAM and Intel Celeron CPU, 256 MB RAM, which were installed with Microsoft Windows 98 operating system. However, they are expected to be able to run on all Microsoft compatible operating systems. If some modifications are required to be done on the source code of the programs, Borland C++ Builder 5.0 or a newer version are supposed to be installed on the operating system used. After the required modifications have been done, the projects should be linked again. Hence, the created executable programs will run on that operating system precisely. On the other hand, the programs which linked in a certain operating system should not be run directly on another operating system. If so, some unpredictable and unexpected problems may arise.

For these programs to be able to be moved to any Microsoft Windows compatible operating system, install shield programs were prepared. By running them, the programs will be automatically installed and registered on that operating system.

4.4. Description of NMGS-SSMG

Fig. 4.2 shows the main appereance, the main menu items and their sub-items. The user manual of the program is given in Appendix-A.

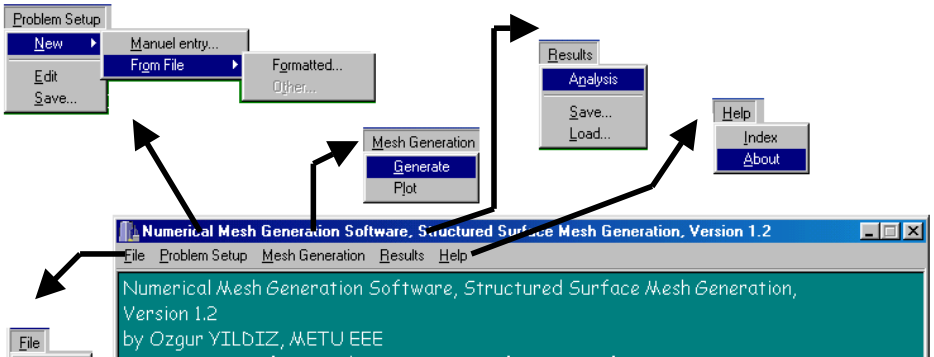

Fig. 4.2. The main appearance, the main menu items and their sub-items of NMGS-SSMG

4.4.1. Implementation

NMGS-SSMG uses isoparamatric coordinate mapping to generate mesh for surfaces represented by eight-noded isoparametric elements. After a problem setup for a number of eight-noded isoparametric elements has been created properly, mesh generation is realized using Mesh Generation => Generate item. Mesh is generated by following these steps:

i. First, mesh is generated for each eight-noded isoparametric element, using coordinate mapping. Each quadrilateral mesh element is then triangulated with the method explained in Section 3.1.1.

ii. Next, the nodes at the boundaries of the elements are checked, if there exist more than one isoparametric elements in the problem setup. This is a very critical step, since some of the nodes at the boundaries may have been calculated twice or more, that is, they may be common to more than one isoparametric elements. As duplication of the same node is not allowed, the common nodes must be behaved as unique nodes in the resultant geometry. Fig. 4.3 illustrates the case: Mesh elements are generated for both isoparametric elements as if they have no common

boundaries as in Fig. 4.3 (a). Then, in Fig. 4.3 (b), duplicated nodes of the second element are excluded.

iii. Finally, the related nodes information is obtained after specifying all the global nodes. Related node concept is explained in Fig. 4.4. It is seen that, node 1 is related with nodes 2, 3 and 4. Similarly, node 3 is related with nodes 1,2,4,5 and 6.

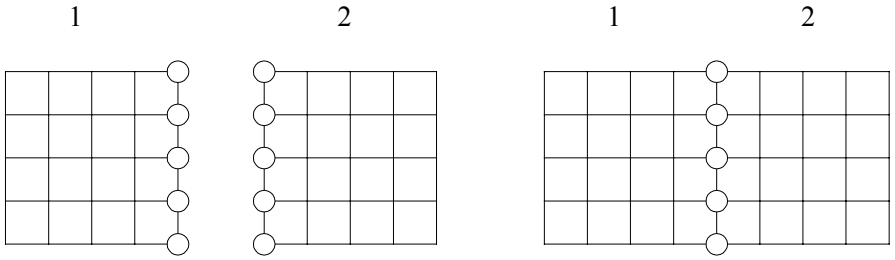

Fig. 4.3. Common nodes concept (a) same nodes are calculated twice (b) duplicate nodes of isoparametric element 2 are excluded.

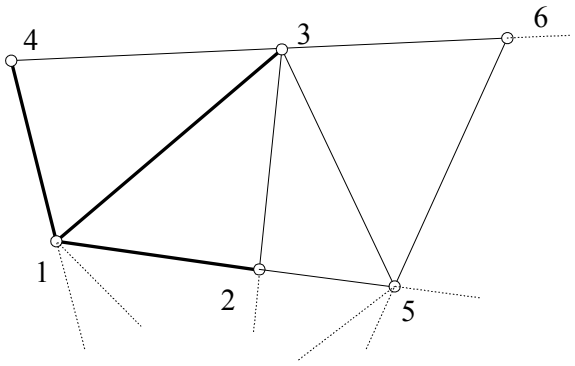

Fig. 4.4. Related nodes concept

4.4.2. Memory Requirements

Let *NE* denote the number of eight-noded isoparametric elements, G_x, G_y denote the resolutions of divisions of each element along *x*, *y* axes, respectively. The maximum values of these constant parameters are shown in Table 4.1. They can also be changed by modifying the source code.

Table 4.1. The maximum values of the constant parameters in NMGS-SSMG

Parameter	Definition	Maximum value
NE	Number of eight-noded isoparametric elements	100
G_x	Resolution of divisions of each element along *x* axis	1000
G_y	Resolution of divisions of each element along *y* axis	1000

The memory used for the execution of the program depends on the code size, stack size, the memory area spent by the visual objects and the operating system. If the program is running on Windows NT operating system, the amount of physical memory used for the execution can be seen on the "Task Manager" tool.

4.4.3. Project Units

The units which are used to create this program within the project are as follows:

i. **SurfaceTriangMainUnit.cpp**: This unit includes all the necessary functions to display the right dialogs when user selects a menu item.
ii. **SurfaceTriangulation.cpp**: The functions of this unit are called to implement the algorithms and arrange the outputs.

iii. **ProbSetupManuelEntryUnit.cpp**: The functions which make this unit up are used to create a problem setup manually.

iv. **SaveFileFromMemoUnit.cpp**: The contents of the files which are opened or saved in this program, are first displayed to the user on a new form. In order to perform this process, the functions in this unit are called.

v. **AboutBoxUnit.cpp**: This unit includes a function which serve to display the information about the version of the program.

vi. **PlotUnit.cpp**: The functions of this unit deal with initializing the graphics device and the viewport and drawing objects on the screen. There are also a couple of private functions to deal with painting the graphics screen, which are called when the forms to be re-drawn.

vii. **PlotOptionsUnit.cpp**: This unit comprises the functions called to display and change the plot options.

viii. **ScalingUnit.cpp**: The functions of this unit help the user to specify the amount of scaling of x and y axes. A three-dimensional object can be scaled from 10 to 200 percent.

ix. **RotationUnit.cpp**: The functions of this unit let the user specify the rotation angles around x, y and z axes.

4.5. Description of NMGS-SVMG

Fig. 4.5 shows the main appearance, the main menu items and their sub-items. The user manual of the program is given in Appendix-A.

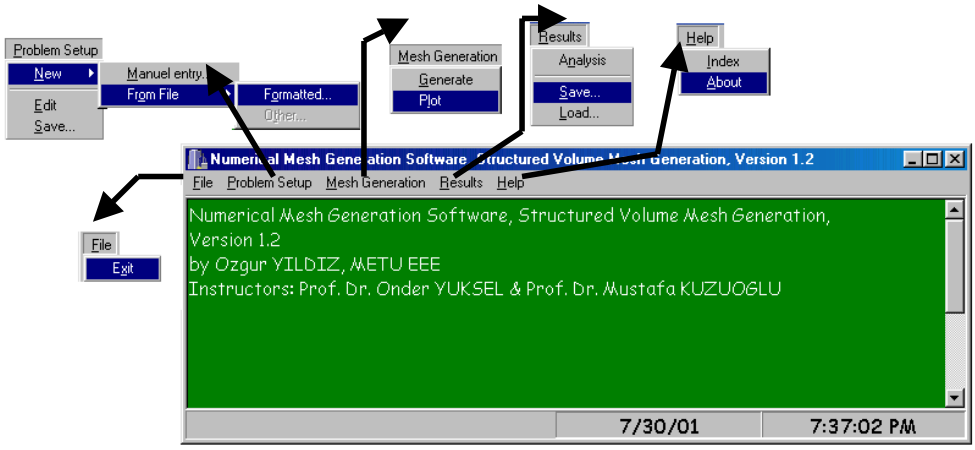

Fig. 4.5. The main appearance, the main menu items and their sub-items of NMGS-SVMG

4.5.1. Implementation

NMGS-SVMG uses isoparamatric coordinate mapping to generate mesh for volumes represented by twenty-noded isoparametric elements. After a problem setup for a number of twenty-noded isoparametric elements has been created properly, mesh generation is realized using Mesh Generation => Generate item. Mesh is generated by following these steps:

i. First, mesh is generated for each twenty-noded isoparametric element, using coordinate mapping. Each hexahedral mesh element is then tetrahedralized with the method explained in Section 3.2.1.
ii. Next, the nodes at the boundaries of the elements are checked, if there exist more than one isoparametric elements in the problem setup. This is a very critical step, since some of the nodes at the boundaries may have been calculated twice or more, that is, they may be common to more than one isoparametric elements. As duplication of the same node is not allowed, the common nodes must be behaved as unique nodes in the resultant geometry. Fig. 4.6 illustrates the case: Mesh elements are generated for both isoparametric elements as if they have no common boundaries as in Fig. 4.6 (a). Then, in Fig. 4.6 (b), duplicated nodes of the second element are excluded.
iii. Finally, the related nodes information is obtained for both the hexahedralization and tetrahedralization cases after specifying all the global nodes. Refering to Fig. 3.3, node 1 is related with nodes 2,4 and 5.

After tetrahedralization, node 1 is going to be related with nodes 2, 3, 4, 5, 6, 7 and 9.

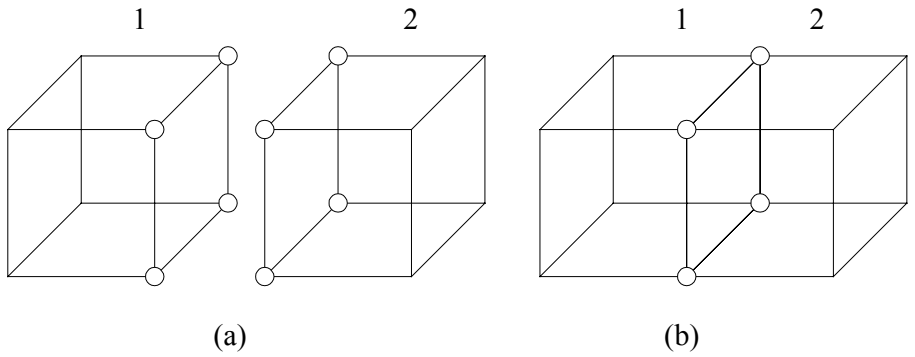

Fig. 4.6. Common nodes concept (a) same nodes are calculated twice (b) duplicate nodes of isoparametric element 2 are excluded.

4.5.2. Memory Requirements

Let *NE* denote the number of twenty-noded isoparametric elements, G_x, G_y, G_z denote the resolutions of divisions of each element along x, y and z axes, respectively. The maximum values of these constant parameters are shown in Table 4.2. They can also be changed by modifying the source code.

Table 4.2. The maximum values of the constant parameters in NMGS-SVMG

Parameter	Definition	Maximum value
NE	Number of twenty-noded isoparametric elements	100
G_x	Resolution of divisions of each element along x axis	1000
G_y	Resolution of divisions of each element along y axis	1000
G_z	Resolution of divisions of each element along z axis	1000

The memory used for the execution of the program depends on the code size, stack size, the memory area spent by the visual objects and the operating system. If the program is running on Windows NT operating system, the amount of physical memory used for the execution can be seen on the "Task Manager" tool.

4.5.3. Project Units

The units which are used to create this program within the project are as follows:

i. **_3DMeshGenMainUnit.cpp**: This unit includes all the necessary functions to display the right dialogs when user selects a menu item.
ii. **_3DMeshGen.cpp**: The functions of this unit are called to implement the algorithms and arrange the outputs.
iii. **ProbSetupManuelEntryUnit.cpp**: The functions which make this unit up are used to create a problem setup manually.
iv. **SaveFileFromMemoUnit.cpp**: The contents of the files which are opened or saved in this program, are first displayed to the user on a new form. In order to perform this process, the functions in this unit are called.
v. **AboutBoxUnit.cpp**: This unit includes a function which serve to display the information about the version of the program.
vi. **ProcessUnit.cpp**: Considering the solution of some problems in this program take a bit long time, the user will wait for a little after the "generate the mesh!" command. Meanwhile, the function that is called for the purpose of the user being shown the message "please wait, processing...", is in this unit.
vii. **PlotUnit.cpp**: The functions of this unit deal with initializing the graphics device and the viewport and drawing objects on the screen. There are also a couple of private functions to deal with painting the graphics screen, which are called when the forms to be re-drawn.
viii. **PlotOptionsUnit.cpp**: This unit comprises the functions called to display and change the plot options.

ix. **ScalingUnit.cpp**: The functions of this unit help the user to specify the amount of scaling of *x* and *y* axes. A three-dimensional object can be scaled from 10 to 200 percent.

x. **RotationUnit.cpp**: The functions of this unit let the user specify the rotation angles around *x*, *y* and *z* axes.

CHAPTER 5

DESCRIPTION OF NMGS-QTMG

5.1. General Information about NMGS-QTMG

The software package was developed in 'Borland C++ Builder 5.0' using object oriented programming. Project files were created for the software package in this software development environment. The compiler compiles all the files contained in the projects and finally links the object codes into exacutable files that can be run in Microsoft Windows compatible operating systems, i.e., Windows 9x and Windows NT. Although the compiler specifies and inserts the necessary codes for all the visual components contained in the project, the source codes were written by the author himself using ANSI C programming language.

As it is in any other Windows programs, also the program realizes its execution by using standard Windows API. In other words, it uses many of the standard visual components, such as 'form', 'main menu', 'memo', 'button', etc.

The program gets inputs from certain formatted files which were prepared earlier. In accordance with these inputs, besides it both performs dynamic memory allocation and solves the problems, that is, it divides the domains defined into regular mesh elements, it plots the geometries and save the mesh information in result files.

NMGS-QTMG uses a free software called “TetGen.exe” (TetGen) [31] as the mesh generator using Delaunay approach. This program is developed by Hang [29] and is free for academic purposes. Hang [29] requires an acknowledgment to be included, if one is to use a mesh generated by TetGen [31] in a publication. Hence, NMGS-QTMG only serves a graphical user interface for using TetGen [31].

Fig. 5.1 shows the main appereance, the main menu items and their sub-items of NMGS-QTMG. The user manual of the program is given in Appendix-B.

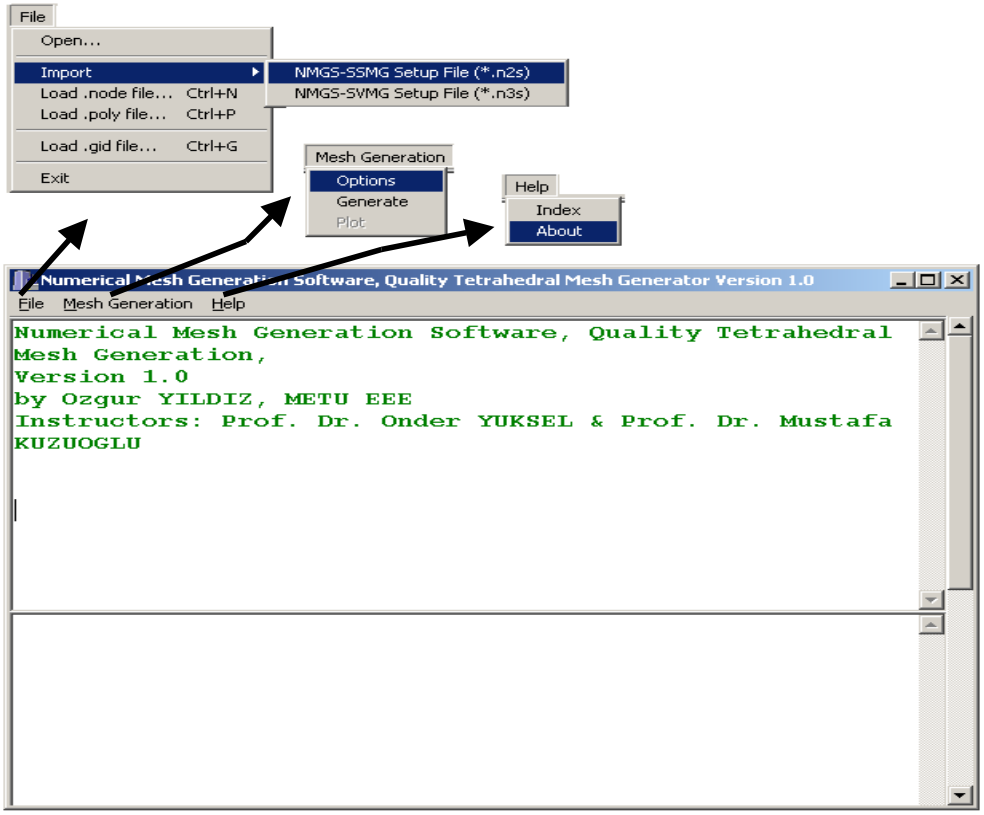

Fig. 5.1. The main appearance, the main menu items and their sub-items of NMGS-QTMG

5.2. Description of TetGen

TetGen [31] was developed by Hang [29]. It is a quality tetrahedral mesh generator and Delaunay triangulator in three-dimension. It is based on the Delaunay method and incorporates face-swapping techniques. It generates meshes composed of tetrahedral elements. The goal of TetGen [31] is to generate meshes for arbitrary three-dimensional domains that are adapted to various problems of scientific computing, e.g. Computational Fluid Dynamics (CFD), Computational Structural Mechanics (CSM), Computational Electromagnetics (CEM), Thermal problems and so on, but it also can perform simpler related tasks such as forming Delaunay tetrahedralizations, solving convex hull problems in three dimension.

TetGen [31] comes as a standalone program or a C++ library to be linked into another application. It is available in source code status and it should run on any computer with a C++ compiler, e.g. cc/gcc under Unix/Linux and bcc32/msvc under Windows 98/NT/2000. In addition, TetGen [31] is a "Triangle"-like program, the coding style and most features were derived from Triangle [30].

TetGen [31] is based on [29]:

i. Using two relative mesh data structures: The tetrahedron-based mesh data strucrure and the triangle-edge mesh data structure.
ii. Using the randomized incremental flip algorithm to construct Delaunay tetrahedralization for three-dimensional point sets.
iii. Using a variant local re-meshing method to construct boundary-constrained conforming Delaunay tetrahedralization of input model.
iv. Using Delaunay refinement algorithm and the radius-edge ratio quality measure to incrementally insert (Steiner) points into the mesh to eliminate bad quality tetrahedra and generate an almost good mesh with good grading.

v. Other algorithms involve using the fast randomized point location algorithm to perform point location in constructing Delaunay tetrahedralization and using gift-wrapping algorithm to construct constrained Delaunay triangulation for triangular faces bounded polyhedra.

vi. Embedding the two-dimensional mesh generator Triangle [30] to generate planar surface mesh for boundary-constrained mesh generation. Optionally using the adaptive exact arithmetic package to improve the robustness of the implementation.

5.3. How to Work

In order to start the program, 'NMGS_QTMG_Ver1_0.exe' is run. When the program is first opened, sub-items of Mesh Generation menu item are disabled, since there will be no data to be processed. At this point, either a problem setup or result file will be loaded from a fomatted file. User can also import problem setup from NMGS-SSMG and NMGS-SVMG problem setup files. If a new valid problem setup has been loaded, Mesh Generation => Generate item is used to generate mesh for the current problem. In this case, Mesh Generation => Plot item may be used to visualize the generated mesh. On the other hand, if a solution is loaded from a formatted file using File => Load *.gid file..., since mesh generation is already done, it is not necessary to re-generate the mesh. Hence, Mesh Generation => Generate menu item is disabled at that case.

5.4. Plot Options

After generating the mesh, Mesh Generation => Plot item is used to open the plot form, which is shown in Fig. 5.2. The plot is drawn by using Plot => Execute item. Since the plot to be drawn can be seen only as a two-dimensional image on the screen, the axes that can be seen on the form are only x and y. For this reason,

the three-dimensional objects are first seen with a default view angle. Visualization of the third dimension is possible by using the rotation transformation. Hence, the view angle will be changed.

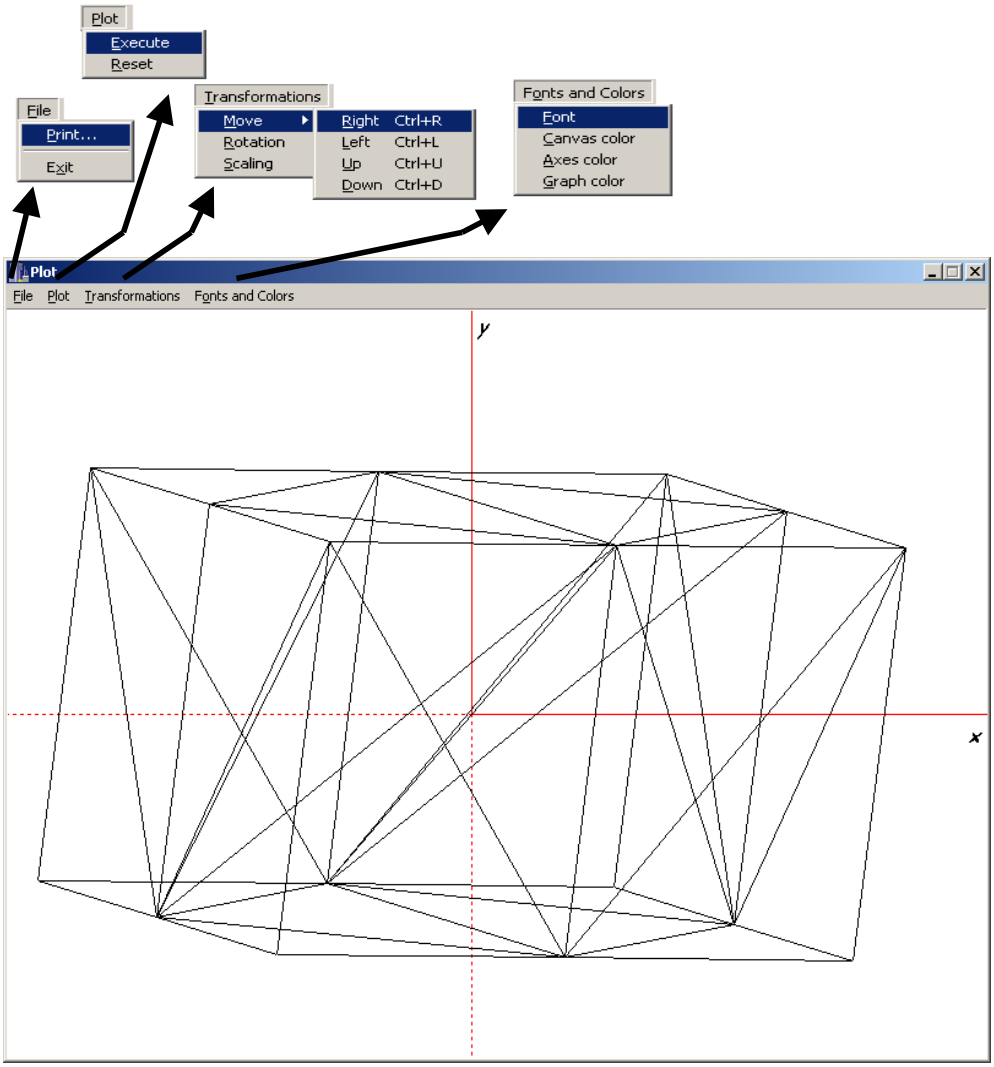

Fig. 5.2. The appearance, the main menu items and their sub-items of the plot form

Once a plot has been drawn, Transformations menu item is enabled. This menu item includes Moving, Scaling and Rotation. At the same time, the fonts and colors of the canvas of the plot form can be changed by using the sub-items included in Fonts and Colors menu item.

The information about the transformations implemented in the program is as follows:

5.4.1. Moving

Moving is used for sliding the origin so that three-dimensional objects can be best seen on the form. At each moving, the origin slides by only one pixel.

5.4.2. Scaling

Scaling transformation is used to scale three-dimensional objects. Because of the reason that explained in Section 4.3.1.2, only x and y axes are scaled. As in the case in moving, also scaling transformation provides a flexibility so that the three- dimensional objects can be best seen.

5.4.3. Rotation

The rotation transformation [21] implemented in these programs is used to improve visualization while displaying three-dimensional objects by looking at those objects from different view angles in three-dimensional Cartesian space. It lets the user specify the rotation angles around x, y and z axes, from 0° to 360°.

5.5. Portability Issues

The program was developed and completed on a personal computer equipped with Intel Pentium-III-800 CPU which was installed with Microsoft Windows 2000 (NT 5.0) operating system. However, it is expected to be able to run on all Microsoft compatible operating systems. If some modifications are required to be done on the source code of the program, Borland C++ Builder 5.0 or a newer version are supposed to be installed on the operating system used. After the required modifications have been done, the project should be linked again. Hence, the created executable program will run on that operating system precisely. On the other hand, the program which linked in a certain operating system should not be run directly on another operating system. If so, some unpredictable and unexpected problems may arise.

For the program to be able to be moved to any Microsoft Windows compatible operating system, install shield program was prepared. By running it, the program will be automatically installed and registered on that operating system.

5.6. Memory Requirements

NMGS-QTMG uses two classes called "Triangles_Class" and "Tetrahedra_Class" to store the triangulation and tetrahedralization information, respectively. It loads the necessary information from "*.face.gid" and "*.ele.gid" files. "*.face.gid" files contain total number of nodes and their x, y and z coordinates; total number of triangles and the node numbers of the three vertices of triangles. "*.ele.gid" files contain total number of nodes and their x, y and z coordinates; total number of tetrahedra and the node numbers of the four vertices of tetrahedra.

Let N_N_F denote the total number of nodes and N_TR_F the total number of triangles. The maximum memory allocated dynamically in bytes, MDA_F, can be calculated as:

$$MDA_F = 24 \,.\, N\_N_F + 12 \,.\, N\_TR_F \quad (5.1)$$

Let N_N_E denote the total number of nodes and N_TT_E the total number of tetrahedra. The maximum memory allocated dynamically in bytes, MDA_E, can be calculated as:

$$MDA_E = 24 \,.\, N\_N_E + 16 \,.\, N\_TT_E \quad (5.2)$$

The memory used for the execution of the program depends on the code size, stack size, the memory area spent by the visual objects and the operating system. If the program is running on Windows NT operating system, the amount of physical memory used for the execution can be seen on the "Task Manager" tool.

5.7. Project Units

The units which are used to create this program within the project are as follows:

i. **NMGS_QTMG_MainUnit.cpp**: This unit includes all the necessary functions to display the right dialogs when user selects a menu item.
ii. **GeneralDefs.cpp**: This unit consists of definitions and initializations of all global variables, visualization classes and function bodies of mathematical calculations.
iii. **GenerateOptions.cpp**: This unit is included in the project in order to display a form that the user will able to set several options about the execution of mesh generator.
iv. **AboutBoxUnit.cpp**: This unit includes a function which serve to display the information about the version of the program.
v. **ProcessUnit.cpp**: Considering the solution of some problems in this program take a bit long time, the user will wait for a little after the "generate the mesh!" command. Meanwhile, the function that is called

for the purpose of the user being shown the message “please wait, processing...”, is in this unit.

vi. **PlotUnit.cpp**: The functions of this unit deal with initializing the graphics device and the viewport and drawing objects on the screen. There are also a couple of private functions to deal with painting the graphics screen, which are called when the forms to be re-drawn.

vii. **ScalingUnit.cpp**: The functions of this unit help the user to specify the amount of scaling of *x* and *y* axes. A three-dimensional object can be scaled from 10 to 200 percent.

viii. **RotationUnit.cpp**: The functions of this unit let the user specify the rotation angles around *x*, *y* and *z* axes.

CHAPTER 6

CONCLUSIONS

In many engineering disciplines, approximate solutions of differential or integral equations play an important role to analyze or design complicated engineering systems. Several examples can be found from various branches such as Computational Fluid Dynamics (CFD), Computational Electromagnetics (CEM), heat transfer applications, Structural Mechanics, ...etc. In all such applications, the spatial domain must be discretized by generating a mesh, which is a collection of elements with simple shapes. Then the operator equations (i.e. partial differential equations or integral equations) are solved by using the well-known methods such as Finite Differences (FDM), Finite Elements (FEM) or Method of Moments (MoM).

Among these, the Finite Element Method is a powerful and useful tool employed in the numerical solution of partial differential equations that arise in different applications. The technique allows for the solution of practical problems that would otherwise be intractable for analytical methods because of non-linearities or complex geometries. However, to achieve the full benefits of considering arbitrary geometries, there must exist simple and efficient means to generate the required meshes. This is especially true for three-dimensional problems where manual or even semi-automatic methods quickly become too tedious to use and,

additionally, are prone to erroneous results. In addition, the cost and accuracy of the numerical analysis is directly tied to the quality of the mesh.

The aim of this thesis is to develop a number of numerical mesh generation software packages, which can be used in Finite Element applications. However, meshes generated on surfaces can be used in MoM applications as well. The main point of the NMGS-SSMG and NMGS-SVMG has been the implementation of the structured mesh generation algorithms based on isoparametric coordinates. It has been observed that structured mesh generation methods do not offer adaptivity and they fail to represent complicated domains unless the complicated domain is divided into many smaller sub-domains. Nevertheless, the two software packages have given good results for geometrical domains, which are applicable to them. Problems of triangulation of quadrilateral mesh elements and tetrahedralization of hexahedral mesh elements have been successfully solved.

Considering the unstructured generation of triangular meshes, the trend is to use the Constrained Delaunay triangulation algorithm. It produces excellent results for domains with complicated two-dimensional geometries. Today, the extension of this algorithm to three dimensions is still an open problem. Direct triangulation of two-dimensional domains using unstructured mesh generation techniques is outside the scope of this thesis.

The ultimate aim of mesh generation is the tetrahedralization of complicated three-dimensional domains. For this purpose, the software package TetGen [31] (obtained from the address http://www.weboo.com/sh/tetgen.htm) has been downloaded and sample geometries have been handled. By means of this software package, we have been able compare the structured mesh generation approach by the unstructured mesh generation algorithms used in TetGen [31]. An interface program has been developed to process the TetGen [31] output files to be used by NMGS-QTMG.

A possible future research subject is the representation of object geometries in the form of sets of points to be used as the input of mesh generation software packages. Another important area is coupling the output files of some popular CAD programs (such as AutoCAD) to the mesh generation routines.

REFERENCES

[1] I. Babuska, and A. K. Aziz, "On the angle condition in the finite element method", SIAM Journal on Numerical Analysis, 13:214-226, 1976

[2] B. B. Dündar, "Development of a mesh generation software for computational Electromagnetics", The Graduate School of Natural and Applied Sciences of METU, 2000

[3] J. F. Thompson, B. K. Soni, N. P. Weatherill, "Handbook of Grid Generation", Lewis Publishers Inc., 1999

[4] O. C. Zienkiewicz, The Finite Element Method, McGraw-Hill, New York, 1983

[5] O. C. Zienkiewicz, and D. V. Phillips, "An automatic mesh generation scheme for plane and curved surfaces by isoparametric coordinates", International Journal for Numerical Methods in Engineering, Vol. 3, 519-528, 1971

[6] M. N. O. Sadiku, "Numerical Techniques in Electromagnetics", CRC Press, 1992

[7] P. P. Silvester, and R. L. Ferrari, "Finite Elements for Electrical Engineers", Cambridge University Press, 1996

[8] J. L. Volakis, A. Chatterjee, and L. C. Campbell, “Finite Element Method for Electromagnetics”, IEEE Press, 1998

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, “Numerical Recipes in C: The art of scientific computing”, Cambridge University Press, 1993

[10] O. C. Zienkiewicz, and D. V. Phillips, “An automatic mesh generation scheme for plane and curved surfaces by isoparametric coordinates”, International Journal for Numerical Methods in Engineering, Vol. 3, 519-528, 1971

[11] S.H. Lo, (1989). “Generating Quadrilateral Elements on Plane and Over Curved Surfaces”, Computers and Structures, Vol.31(3), pp.421-426

[12] Bruce P Johnston, John M. Sullivan Jr. and Andrew Kwasnik (1991). “Automatic Conversion of Triangular Finite Element Meshes to Quadrilateral Elements”, International Journal for Numerical Methods in Engineering, Vol.31, pp.67-84

[13] A. E. Yılmaz, “Analysis of electromagnetic scattering problems with the finite element method”, The Graduate School of Natural and Applied Sciences of METU, 2000

[14] Steven J. Owen, (1998), “Meshing Software Survey”, web page: http://www.andrew.cmu.edu/user/sowen/softsurv.html

[15] Peggy L. Baehmann, Scott L. Wittchen, Mark S. Shephard, Kurt R. Grice and Mark A. Yerry, (1987). “Robust Geometrically-based, Automatic Two-

Dimensional Mesh Generation," International Journal for Numerical Methods in Engineering, Vol.24, pp.1043-1078

[16] T. K. H. Tam and C. G. Armstrong (1991). "2D Finite Element Mesh Generation by Medial Axis Subdivision", Advances in Engineering Software, Vol.13, pp.313-324

[17] J.Z. Zhu, O.C. Zienkiewicz, E. Hinton and J. Wu (1991). "A New Approach to the Development of Automatic Quadrilateral Mesh Generation," International Journal for Numerical Methods in Engineering, Vol.32 pp.849-866

[18] Mark A.Yerry and Mark S, Shephard, (1984) "Three-Dimensional Mesh Generation by Modified Octree Technique", International Journal for Numerical Methods in Engineering, vol 20, pp.1965-1990

[19] Mark S. Shephard and Marcel K. Georges, (1991) "Three-Dimensional Mesh Generation by Finite Octree Technique", International Journal for Numerical Methods in Engineering, vol 32, pp. 709-749

[20] Boris, N. Delaunay, (1934) "Sur la Sphere" Vide. Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk Vol 7 pp.793-800

[21] D. F. Rogers, and J. A. Adams, "Mathematical Elements for Computer Graphics", McGraw-Hill, 1990

[22] Takeo Taniguchi, Tomoaki Goda, Harald Kasper and Werner Zielke, (1996) "Hexahedral Mesh Generation of Complex Composite Domain", 5th International Conference on Grid Generation in Computational Field Simulations, Mississippi State University. pp 699-707

[23] Robert Schneiders, (1996) “A Grid-Based Algorithm for the Generation of Hexahedral Element Meshes”, Engineering With Computers. Vol.12 pp.168-177

[24] T.S. Li, R.M. McKeag and C.G. Armstrong, (1995) “Hexahedral Meshing Using Midpoint Subdivision and Integer Programming”, Computer Methods in Applied Mechanics and Engineering, Vol.124, pp.171-193

[25] Scott A. Canann, (1991) “Plastering and Optismoothing: New Approaches to Automated, 3D Hexahedral Mesh Generation and Mesh Smoothing,” Ph.D. Dissertation, Brigham Young University, Provo, UT.

[26] Timothy J. Tautges, Ted Blacker and Scott Mitchell, (1996) “The Whisker-Weaving Algorithm: A Connectivity Based Method for Constructing All-Hexahedral Finite Element Meshes,” International Journal for Numerical Methods in Engineering, Vol.39, pp.3327-3349

[27] Peter Murdoch, and Steven E. Benzley, (1995) “The Spatial Twist Continuum”, Proceedings, 4th International Meshing Roundtable, Sandia National Laboratories, pp.243-251

[28] Steven J. Owen, Scott A. Canann and Sunil Saigal, (1997) “Pyramid Elements for Maintaining Tetrahedra to Hexahedra Conformability”, AMD-Vol. 220 Trends in Unstructured Mesh Generation, ASME, pp. 123-129

[29] Si Hang, “Tetrahedral mesh generation and refinement”, The Graduate College of Zhejiang University, 2001

[30] J. R. Shewchuck, (1996), “Triangle, A Two-Dimensional Quality Mesh Generator”, web page: http://www.cs.cmu.edu/~quake/triangle.html

[31] Si Hang, (2001), “Tetgen, A Quality Tetrahedral Mesh Generator”, web page: http://www.weboo.com/sh/tetgen.htm

APPENDIX A

NMGS-SSMG AND NMGS-SVMG USER'S MANUALS

A.1. Introduction

NMGS-SSMG and NMGS-SVMG programs both are similar in usage and have the same user menus. Since they are used for solution of different types of problems, they have been developed as two distinct programs. However, from this point, the word "program" is going to be used in place of the two programs. Following sections explain how the menus on the main and plot forms are used.

A.2. Menus on the Main Form

Main form of the program includes menus that enable user to create and edit a problem setup, generate mesh, plot the geometry, analyze results and save input and output information in text files.

A.2.1. File Menu

File menu only consists of Exit menu item. Pressing File => Exit will terminate the execution of the program.

A.2.2. Problem Setup Menu

Problem Setup menu is used to create a new problem setup or to call previously saved input. It includes three menu items.

i. **New Menu Item**: New menu item is used to create new problem setup information. New => Manual entry… menu item lets user construct an input geometry, while New => From File => Formatted… menu item can be used to import a problem setup from a file. New => From File => Other… menu item is disabled by default and left for future use (for possible improvements, look at Section 4.3.2). New menu item is enabled throughout the program execution. This means that user is always able to create a new problem setup.
ii. **Edit Menu Item**: Edit menu item is used to edit the present problem setup. User may change the input parameters of the problem. Edit menu item will remain disabled as long as there exists no problem setup.
iii. **Save Menu Item**: Save menu item is used to save the input of the geometry in a formatted text file. Save menu item will remain disabled as long as there exists no problem setup.

A.2.3. Mesh Generation Menu

Mesh Generation menu is used to generate mesh for a specific problem and plot the mesh in three-dimensions. Mesh Generation menu will remain disabled as long as there exists no problem setup. It includes two menu items.

i. **Generate Menu Item**: Generate menu item is used to generate mesh for the input geometry. It can be used whenever Mesh Generation menu is enabled.
ii. **Plot Menu Item**: Plot menu item is used to plot the resultant geometry after generating the mesh. It is going to be enabled after the mesh has been generated. Pressing Mesh Generation => Plot causes a new form, called the plot form to be shown.

A.2.4. Results Menu

Results menu is used to analyze the resultant geometry, save results in a file and call some output information from a formatted result file. It includes three menu items.

i. **Analysis Menu Item**: Analysis menu item is used to analyze the output geometry. Pressing Results => Analysis causes a new form to appear. In that form, several information lines such as number of nodes, number of zones, etc. are shown. Analysis menu item will remain disabled as long as mesh has not been generated.
ii. **Save Menu Item**: Save menu item is used to save the output information in a formatted result file. Save menu item will remain disabled as long as mesh has not been generated.
iii. **Load Menu Item**: Load menu item is used to import the contents of a formatted result file. Load menu item is enabled throughout the program execution. This means that user is always able to call a set of output information.

A.2.5. Help Menu

Help menu is used to help user to execute the program. It includes two menu items.

i. **Index Menu Item**: Pressing Help => Index causes a new form to appear. In that form, user may find help topics related with the application and file formats.
ii. **About Menu Item**: In order to see the version, author and production date information of the program, About menu item is used.

A.3. Menus on the Plot Form

Plot form of the program includes menus that enable user to view and print the geometry, set the fonts and colors of the form and apply several transformations to the geometry.

A.3.1. File Menu

File menu includes form-specific features. In the File menu, there are two menu items appearing.

i. **Print Menu Item**: If a plot exists in the plot form, it can be printed pressing File => Print.
ii. **Exit Menu Item**: Pressing File => Exit will close the plot form and set focus to main form.

A.3.2. Plot Menu

Plot menu includes menu items that are used to show and clear the three-dimensional graph and set the options for plotting. It includes three menu items.

i. **Execute Menu Item**: In order to show the three-dimensional geometry on the plot form, Execute menu item is used.
ii. **Reset Menu Item**: Pressing Plot => Reset will refresh the plot form.
iii. **Options Menu Item**: Options menu item is used to set the view options for the current geometry. In surface meshes, user may prefer to see quadrilateral or triangular elements, whereas hexahedral or tetrahedral elements in volume meshes.

A.3.3. Transformations Menu

The menu items in the Transformations menu are used to apply several transformations. Transformation menu is disabled as long as the graph is not shown on the plot form. It includes three menu items.

i. **Move Menu Item**: It realizes the translation transformation. Moving is used for sliding the origin so that three-dimensional objects can be best seen on the form. At each moving, the origin slides by only one pixel.
ii. **Rotation Menu Item**: Pressing Transformation => Rotation will open a new form that lets user apply the rotation transformation. It improves visualization while displaying three-dimensional objects by looking at those objects from different view angles in three-dimensional Cartesian space and lets the user specify the rotation angles around x, y and z axes, from 0° to 360°.
iii. **Scaling Menu Item**: Scaling menu item is used to scale the axes of the plot. It provides flexibility so that the three-dimensional objects can be best seen on the plot form.

A.3.4. Fonts and Colors Menu

Fonts and Colors menu includes menu items that help user to change the fonts and colors of the plot form. It includes four menu items.

i. **Font Menu Item**: Using Font menu item user may change the font of the plot form.
ii. **Canvas Color Menu Item**: Color of the surface of the plot form can be changed using this menu item.
iii. **Axes Color Menu Item**: Axes Color menu item is used to change the color of the axes.
iv. **Graph Color Menu Item**: Graph color may be changed by pressing Fonts and Colors => Graph Color.

APPENDIX B

NMGS-QTMG USER'S MANUAL

B.1. Introduction

NMGS-QTMG is used generate triangular and tetrahedral mesh elements for complex three-dimensional domains. It gets inputs from certain formatted files prepared earlier. After the problem setup files are loaded, it generates the mesh and stores the result in ASCII text files. Following sections explain how the menus on the main and plot forms are used.

B.2. Menus on the Main Form

Main form of the program includes menus that enable user to create and edit a problem setup, generate mesh, plot the geometry, analyze results and save input and output information in text files.

B.2.1. File Menu

File menu is used to load problem setup and result files. It includes six menu items.

i. **Open Menu Item**: Open menu item is used to load contents of any file inside the main form.

ii. **Import Menu Item**: It is used to import NMGS-SSMG and NMGS-SVMG problem setup files to NMGS-QTMG. The program reads the contents of the problem setup files and converts them into "*.node" file format. The name of the converted file is "import.node" and the contents of the new file will be shown in the main form.

iii. **Load .node file Menu Item**: This menu item is used to load a "*.node" problem setup file into the program. The contents of the loaded file will be shown in the main form.

iv. **Load .poly file Menu Item**: This menu item is used to load a "*.poly" problem setup file into the program. The contents of the loaded file will be shown in the main form.

v. **Load .gid file Menu Item**: This menu item is used to load a "*.gid" result file into the program. The contents of the loaded file will be shown in the main form. Dynamic memory allocation will be made according to the values read from this file.

vi. **Exit Menu Item**: Pressing File => Exit will terminate the execution of the program.

B.2.2. Mesh Generation Menu

Mesh Generation menu is used to generate mesh for a specific problem and plot the mesh in three-dimensions. Menu items of Mesh Generation menu will remain disabled as long as there exists no data to be processed. It includes three menu items.

i. **Options Menu Item**: Pressing Mesh Generation => Options will cause a new form to be displayed. User can set several options about the execution of mesh generator through the form.

ii. **Generate Menu Item**: Generate menu item is used to generate mesh for the input geometry. It can be used after a problem setup has been loaded from a formatted file.

iii. **Plot Menu Item**: Plot menu item is used to plot the generated mesh. It is going to be enabled after a "*.gid" file has been loaded. Pressing Mesh Generation => Plot causes a new form, called the plot form to be shown.

B.2.3. Help Menu

Help menu is used to help user to execute the program. It includes two menu items.

i. **Index Menu Item**: Pressing Help => Index causes a new form to appear. In that form, user may find help topics related with the application and file formats.

ii. **About Menu Item**: In order to see the version, author and production date information of the program, About menu item is used.

B.3. Menus on the Plot Form

Plot form of the program includes menus that enable user to view and print the geometry, set the fonts and colors of the form and apply several transformations to the geometry.

B.3.1. File Menu

File menu includes form-specific features. In the File menu, there are two menu items appearing.

i. **Print Menu Item**: If a plot exists in the plot form, it can be printed pressing File => Print.

ii. **Exit Menu Item**: Pressing File => Exit will close the plot form and set focus to main form.

B.3.2. Plot Menu

Plot menu includes menu items that are used to show and clear the three-dimensional graph and set the options for plotting. It includes two menu items.

i. **Execute Menu Item**: In order to show the three-dimensional geometry on the plot form, Execute menu item is used.
ii. **Reset Menu Item**: Pressing Plot => Reset will refresh the plot form.

B.3.3. Transformations Menu

The menu items in the Transformations menu are used to apply several transformations. Transformation menu is disabled as long as the graph is not shown on the plot form. It includes three menu items.

i. **Move Menu Item**: It realizes the translation transformation. Moving is used for sliding the origin so that three-dimensional objects can be best seen on the form. At each moving, the origin slides by only one pixel.
ii. **Rotation Menu Item**: Pressing Transformation => Rotation will open a new form that lets user apply the rotation transformation. It improves visualization while displaying three-dimensional objects by looking at those objects from different view angles in three-dimensional Cartesian space and lets the user specify the rotation angles around x, y and z axes, from 0° to 360°.
iii. **Scaling Menu Item**: Scaling menu item is used to scale the axes of the plot. It provides flexibility so that the three-dimensional objects can be best seen on the plot form.

B.3.4. Fonts and Colors Menu

Fonts and Colors menu includes menu items that help user to change the fonts and colors of the plot form. It includes four menu items.

i. **Font Menu Item**: Using Font menu item user may change the font of the plot form.
ii. **Canvas Color Menu Item**: Color of the surface of the plot form can be changed using this menu item.
iii. **Axes Color Menu Item**: Axes Color menu item is used to change the color of the axes.
iv. **Graph Color Menu Item**: Graph color may be changed by pressing Fonts and Colors => Graph Color.

APPENDIX C

FILE FORMATS

C.1. NMGS-SSMG Problem Setup File (N2S) Format

N2S format is given in Table C.1. NMGS-SSMG allows the user to generate a mesh for a surface represented by a number of eight-noded isoparametric elements.

Table C.1. NMGS-SSMG Problem Setup File (N2S) Format

--- start of file
(File Id text)
NMGS-SSMG Problem Setup File
User reference text
--- start of problem setup
(Number of eight-noded isoparametric elements)
NE
(Resolutions of divisions of each element along *x*, *y* axes)
G_{x1} G_{y1}
G_{x2} G_{y2}
…
…
G_{xNE} G_{yNE}
(Coordinates of eight nodes of each element)
$x_1[1]$ $y_1[1]$ $z_1[1]$
$x_1[2]$ $y_1[2]$ $z_1[2]$
…
$x_1[8]$ $y_1[8]$ $z_1[8]$
$x_2[1]$ $y_2[1]$ $z_2[1]$
$x_2[2]$ $y_2[2]$ $z_2[2]$
…
$x_2[8]$ $y_2[8]$ $z_2[8]$
…
…
$x_{NE}[1]$ $y_{NE}[1]$ $z_{NE}[1]$
$x_{NE}[2]$ $y_{NE}[2]$ $z_{NE}[2]$
…
$x_{NE}[8]$ $y_{NE}[8]$ $z_{NE}[8]$
--- end of problem setup
--- end of file

C.2. NMGS-SVMG Problem File (N3S) Format

N3S format is given in Table C.2. NMGS-SVMG allows the user to generate a mesh for a three-dimensional object represented by a number of twenty-noded isoparametric elements.

Table C.2. NMGS-SVMG Problem Setup File (N3S) Format

--- start of file
(File Id text)
NMGS-SVMG Problem Setup File
User reference text
--- start of problem setup
(Number of twenty-noded isoparametric elements)
NE
(Resolutions of divisions of each element along x, y and z axes)
G_{x1} G_{y1} G_{z1}
G_{x2} G_{y2} G_{z2}
…
…
G_{xNE} G_{yNE} G_{zNE}
(Coordinates of twenty nodes of each element)
$x_1[1]$ $y_1[1]$ $z_1[1]$
$x_1[2]$ $y_1[2]$ $z_1[2]$
…
$x_1[20]$ $y_1[20]$ $z_1[20]$
$x_2[1]$ $y_2[1]$ $z_2[1]$
$x_2[2]$ $y_2[2]$ $z_2[2]$
…
$x_2[20]$ $y_2[20]$ $z_2[20]$
…
…
$x_{NE}[1]$ $y_{NE}[1]$ $z_{NE}[1]$
$x_{NE}[2]$ $y_{NE}[2]$ $z_{NE}[2]$
…
$x_{NE}[20]$ $y_{NE}[20]$ $z_{NE}[20]$
--- end of problem setup
--- end of file

C.3. NMGS-SSMG Result File (N2R) Format

N2R format is given in Table C.3. N2R includes resultant number of quadrilateral mesh elements, triangulation parameters, four node numbers and coordinates of each element, total number of nodes and their related nodes and problem setup as well.

Table C.3. NMGS-SSMG Result File (N2R) Format

--- start of file
(File Id text)
NMGS-SSMG Result File
User reference text
--- start of problem setup
(Number of eight-noded isoparametric elements)
NE
(Resolutions of divisions of each element along x, y axes)
G_{x1} G_{y1}
G_{x2} G_{y2}
…
…
G_{xNE} G_{yNE}
(Number of quadrilateral zones is calculated as:
$NZ = NE$ x (G_{x1} x G_{y1} + G_{x1} x G_{y1} +... $G_{x\mathrm{NE}}$ x $G_{y\mathrm{NE}}$)
(Coordinates of eight nodes of each element)
$x_1[1]$ $y_1[1]$ $z_1[1]$
$x_1[2]$ $y_1[2]$ $z_1[2]$
…
$x_1[8]$ $y_1[8]$ $z_1[8]$
$x_2[1]$ $y_2[1]$ $z_2[1]$
$x_2[2]$ $y_2[2]$ $z_2[2]$
…
$x_2[8]$ $y_2[8]$ $z_2[8]$
…
…
$x_{NE}[1]$ $y_{NE}[1]$ $z_{NE}[1]$
$x_{NE}[2]$ $y_{NE}[2]$ $z_{NE}[2]$
…
$x_{NE}[8]$ $y_{NE}[8]$ $z_{NE}[8]$
--- end of problem setup
(Number of nodes)
NN
---start of zones information
(Zone number)

Table C.3, continued

1
(Global node numbers of the four nodes of zone 1)
$GNN_1[1]$ $GNN_1[2]$ $GNN_1[3]$ $GNN_1[4]$
(Coordinates of each node)
$x_{GNN\ 1[1]}$ $y_{GNN\ 1[1]}$ $z_{GNN\ 1[1]}$ …. $x_{GNN\ 1[4]}$ $y_{GNN\ 1[4]}$ $z_{GNN\ 1[4]}$
(Common node parameter, 0: not common, 1:common)
$CN_1[1]$ $CN_1[2]$ $CN_1[3]$ $CN_1[4]$
(Triangulation parameter, 0:nodes 1 and 3 diagonalized, 1: nodes 2 and 4 diagonalized)
TP_1
(Zone number)
2
(Global node numbers of the four nodes of zone 2)
$GNN_2[1]$ $GNN_2[2]$ $GNN_2[3]$ $GNN_2[4]$
(Coordinates of each node)
$x_{GNN\ 2[1]}$ $y_{GNN\ 2[1]}$ $z_{GNN\ 2[1]}$ …. $x_{GNN\ 2[4]}$ $y_{GNN\ 2[4]}$ $z_{GNN\ 2[4]}$
(Common node parameter, 0: not common, 1:common)
$CN_2[1]$ $CN_2[2]$ $CN_2[3]$ $CN_2[4]$
(Triangulation parameter, 0:nodes 1 and 3 diagonalized, 1: nodes 2 and 4 diagonalized)
TP_2
…
(Zone number)
NZ
(Global node numbers of the four nodes of zone NZ)
$GNN_{NZ}[1]$ $GNN_{NZ}[2]$ $GNN_{NZ}[3]$ $GNN_{NZ}[4]$
(Coordinates of each node)
$x_{GNN\ NZ[1]}$ $y_{GNN\ NZ[1]}$ $z_{GNN\ NZ[1]}$ …. $x_{GNN\ NZ[4]}$ $y_{GNN\ NZ[4]}$ $z_{GNN\ NZ[4]}$
(Common node parameter, 0: not common, 1:common)
$CN_{NZ}[1]$ $CN_{NZ}[2]$ $CN_{NZ}[3]$ $CN_{NZ}[4]$
(Triangulation parameter, 0:nodes 1 and 3 diagonalized, 1: nodes 2 and 4 diagonalized)
TP_{NZ}
---end of zones information
---start of related nodes information
(Global node number and number of related nodes)
1 NR_1
(Global node numbers of related nodes of node 1)
$GNN_{RNN\ 1}[1]$
…
$GNN_{RNN\ 1}[NR_1]$
(Global node number and number of related nodes)

Table C.3, continued

2 NR_2

(Global node numbers of related nodes of node 2)
$GNN_{RNN\,2}[1]$
…
$GNN_{RNN\,2}[NR_2]$
…
…
(Global node number and number of related nodes)
$NN\ NR_{NN}$
(Global node numbers of related nodes of node NN)
$GNN_{RNN\,NN}[1]$
…
$GNN_{RNN\,NN}[NR_{NN}]$
---end of related nodes information
---start of nodes information
(Node number)
1
(Coordinates of node 1)
$x_1\ y_1\ z_1$
(Node number)
2
(Coordinates of node 2)
$x_2\ y_2\ z_2$
…
…
(Node number)
NN
(Coordinates of node NN)
$x_{NN}\ y_{NN}\ z_{NN}$
--- end of nodes information
--- end of file

C.4. NMGS-SVMG Result File (N3R) Format

N3R format is given in Table C.4. N3R includes resultant number of hexahedral mesh elements, eight node numbers and coordinates of each element, total number of nodes and their related nodes, resultant number of tetrahedral mesh elements, four node numbers and coordinates of each element, total number of nodes with tetrahedralization and their related nodes and problem setup as well.

Table C.4. NMGS-SVMG Result File (N3R) Format

--- start of file (File Id text) NMGS-SVMG Result File User reference text --- start of problem setup (Number of twenty-noded isoparametric elements) NE (Resolutions of divisions of each element along x, y and z axes) G_{x1} G_{y1} G_{z1} G_{x2} G_{y2} G_{z2} … … G_{xNE} G_{yNE} G_{zNE} (Number of hexahedral zones is calculated as: $NHZ = NE$ x (G_{x1} x G_{y1} + G_{x1} x G_{y1} +... $G_{x\mathrm{NE}}$ x $G_{y\mathrm{NE}}$) Number of tetrahedral zones is calculated as: NTZ = 12 x NE) (Coordinates of twenty nodes of each element) $x_1[1]$ $y_1[1]$ $z_1[1]$ $x_1[2]$ $y_1[2]$ $z_1[2]$ … $x_1[20]$ $y_1[20]$ $z_1[20]$ $x_2[1]$ $y_2[1]$ $z_2[1]$ $x_2[2]$ $y_2[2]$ $z_2[2]$ … $x_2[20]$ $y_2[20]$ $z_2[20]$ … … $x_{NE}[1]$ y_{NE} [1] z_{NE} [1] x_{NE} [2] y_{NE} [2] z_{NE} [2] … x_{NE} [20] y_{NE} [20] z_{NE} [20] --- end of problem setup

Table C.4, continued

--- start of hexahedralization information

(Number of nodes)
NN
(Number of hexahedral zones)
NHZ
---start of zones information
(Zone number)
1
(Global node numbers of the eight nodes of zone 1)
$GNN_1[1]$ $GNN_1[2]$ … $GNN_1[8]$
(Coordinates of each node)
$x_{GNN\ 1[1]}$ $y_{GNN\ 1[1]}$ $z_{GNN\ 1[1]}$ …. $x_{GNN\ 1[8]}$ $y_{GNN\ 1[8]}$ $z_{GNN\ 1[8]}$
(Common node parameter, 0: not common, 1:common)
$CN_1[1]$ $CN_1[2]$… $CN_1[8]$
(Zone number)
2
(Global node numbers of the eight nodes of zone 2)
$GNN_2[1]$ $GNN_2[2]$ … $GNN_2[8]$
(Coordinates of each node)
$x_{GNN\ 2[1]}$ $y_{GNN\ 2[1]}$ $z_{GNN\ 2[1]}$ …. $x_{GNN\ 2[8]}$ $y_{GNN\ 2[8]}$ $z_{GNN\ 2[8]}$
(Common node parameter, 0: not common, 1:common)
$CN_2[1]$ $CN_2[2]$ … $CN_2[8]$
…
(Zone number)
NHZ
(Global node numbers of the eight nodes of zone *NHZ*)
$GNN_{NHZ}[1]$ $GNN_{NHZ}[2]$ … $GNN_{NHZ}[8]$
(Coordinates of each node)
$x_{GNN\ NHZ[1]}$ $y_{GNN\ NHZ[1]}$ $z_{GNN\ NHZ[1]}$ …. $x_{GNN\ NHZ[8]}$ $y_{GNN\ NHZ[8]}$ $z_{GNN\ NHZ[8]}$
(Common node parameter, 0: not common, 1:common)
$CN_{NHZ}[1]$ $CN_{NHZ}[2]$ … $CN_{NHZ}[8]$
---end of zones information

Table C.4, continued

---start of related nodes information
(Global node number and number of related nodes)

1 NR_1
(Global node numbers of related nodes of node 1)
$GNN_{RNN\,1}[1]$
…
$GNN_{RNN\,1}[NR_1]$
(Global node number and number of related nodes)
2 NR_2
(Global node numbers of related nodes of node 2)
$GNN_{RNN\,2}[1]$
…
$GNN_{RNN\,2}[NR_2]$
…
…
(Global node number and number of related nodes)
$NN\ NR_{NN}$
(Global node numbers of related nodes of node *NN*)
$GNN_{RNN\,NN}[1]$
…
$GNN_{RNN\,NN}[NR_{NN}]$
---end of related nodes information
---start of nodes information
(Node number)
1
(Coordinates of node 1)
$x_1\ y_1\ z_1$
(Node number)
2
(Coordinates of node 2)
$x_2\ y_2\ z_2$
…
…
(Node number)
NN
(Coordinates of node *NN*)
$x_{NN}\ y_{NN}\ z_{NN}$
--- end of nodes information

Table C.4, continued

--- start of tetrahedralization information
(Number of nodes)
NNT

(Number of tetrahedral zones)
NTZ
---start of zones information
(Zone number)
1
(Global node numbers of the four nodes of zone 1)
$GNN_1[1]$ $GNN_1[2]$ … $GNN_1[4]$
(Coordinates of each node)
$x_{GNN\ 1[1]}$ $y_{GNN\ 1[1]}$ $z_{GNN\ 1[1]}$ …. $x_{GNN\ 1[4]}$ $y_{GNN\ 1[4]}$ $z_{GNN\ 1[4]}$
(Zone number)
2
(Global node numbers of the four nodes of zone 2)
$GNN_2[1]$ $GNN_2[2]$ … $GNN_2[4]$
(Coordinates of each node)
$x_{GNN\ 2[1]}$ $y_{GNN\ 2[1]}$ $z_{GNN\ 2[1]}$ …. $x_{GNN\ 2[4]}$ $y_{GNN\ 2[4]}$ $z_{GNN\ 2[4]}$
…
(Zone number)
NTZ
(Global node numbers of the four nodes of zone *NTZ*)
$GNN_{NTZ}[1]$ $GNN_{NTZ}[2]$ … $GNN_{NTZ}[4]$
(Coordinates of each node)
$x_{GNN\ NTZ[1]}$ $y_{GNN\ NTZ[1]}$ $z_{GNN\ NTZ[1]}$ …. $x_{GNN\ NTZ[4]}$ $y_{GNN\ NTZ[4]}$ $z_{GNN\ NTZ[4]}$
---end of zones information
---start of related nodes information
(Global node number and number of related nodes)
1 NR_1
(Global node numbers of related nodes of node 1)
$GNN_{RNN\ 1}[1]$
…
$GNN_{RNN\ 1}[NR_1]$
(Global node number and number of related nodes)
2 NR_2
(Global node numbers of related nodes of node 2)
$GNN_{RNN\ 2}[1]$
…
$GNN_{RNN\ 2}[NR_2]$
…
…

Table C.4, continued

(Global node number and number of related nodes)
NNT NR_{NNT}
(Global node numbers of related nodes of node *NNT*)
$GNN_{RNN\ NNT}[1]$

…
$GNN_{RNN\,NNT}[NR_{NNT}]$
---end of related nodes information
---start of nodes information
(Node number)
1
(Coordinates of node 1)
$x_1\ y_1\ z_1$
(Node number)
2
(Coordinates of node 2)
$x_2\ y_2\ z_2$
…
…
(Node number)
NNT
(Coordinates of node *NNT*)
$x_{NNT}\ y_{NNT}\ z_{NNT}$
--- end of tetrahedralization information
--- end of file

C.5. NMGS-QTMG Input File (*.node) Format

"*.node" file format is given in Table C.5. A "*.node" file includes a set of points that represents a complex three-dimensional domain.

Table C.5. NMGS-QTMG Input File (*.node) Format

--- start of file
#Comment line
#Comment line
…
…
NN (Number of nodes) 3 (dimension)
(Coordinates of nodes)
1 $x_1\ y_1\ z_1$
2 $x_2\ y_2\ z_2$
…
…
NN $x_{NN}\ y_{NN}\ z_{NN}$
#Comment line
#Comment line
…
…
--- end of file

C.6. NMGS-QTMG Input File (*.poly) Format

"*.poly" file format is given in Table C.6. A "*.poly" file represents a Piecewise Linear Complex (PLC) (see following), as well as some additional information. The first section lists all the points, and is identical to the format of "*.node" files. <# of points> may be set to zero to indicate that the points are listed in a separate "*.node" file; "*.poly" files produced by TetGen [31] always have this format. This has the advantage that a point set may easily be triangulated with or without segments/facets.

The second section lists the facets. In PLC, facet, however, can be quite complicated in shape. A facet is a planar boundary, may have any number of sides, may be non-convex, and may have holes, slits, or vertices in its interior. However, an immutable requirement is that a facet must be planar. Each facet is represented by a set of polygons and holes. At beginning of each facet, the number of polygons, number of holes and boundary marker of this facet are specified. If the hole number and boundary marker are not provided, TetGen [31] will think there is no hole in this facet and the default boundary mark Zero is used for this facet. If this facet has a specific boundary mark other than Zero,

both the hole number and boundary marker at first line of this facet must be provided.

Then, the set of polygons be listed. Each polygon is specified by giving the number of vertices in the polygon, followed by listing the indices of the endpoints in order. The list of endpoints for polygon is not restricted to a single line. TetGen [31] will automatically read another line until the required numbers of endpoints for a polygon have been read. A segment can be represented by a degenerate polygon with only two endpoints. A degenerate segment with two identical endpoints can represent an isolated point in facet. After listing all polygons, if the number of hole is not zero, holes are specified by identifying a point inside each hole (This hole point does not critically required on this facet, as long as the orthogonal projection of this point onto facet is inside the hole).

Table C.6. NMGS-QTMG Input File (*.poly) Format

--- start of file #Comment line #Comment line … … 1. First line: <# of points> <dimension (must be 3)> <# of attributes> <# of boundary markers (0 or 1)> 2. Following lines: <point #> <*x*> <*y*> <*z*> [attributes] [boundary marker] 3. One line: <# of facets> <# of boundary markers (0 or 1)> (Following lines list all facets, for each facet): One line: <# of polygons> [# of holes] [boundary marker]

Table C.6, continued

Following lines(for each polygon in this facet): <# of polygon…'s vertices> <endpoint1> <endpoint2> … <endpoint #> Following lines (list all hole points in this facet): <hole #> <*x*> <*y*> <*z*> 4. One line: <# of holes>

5. Following lines: <hole #> <*x*> <*y*> <*z*>

6. Optional line: <# of regional attributes and/or volume constraints>

7. Optional following lines: <constraint #> <*x*> <*y*> <*z*> <attrib> <max volume>

#Comment line
#Comment line
…
…
--- end of file

C.7. NMGS-QTMG Output File (*.face) Format

"*.face" file format is given in Table C.7. A "*.face" file includes the total number of triangular faces of tetrahedra and node numbers of the three vertices of the faces.

Table C.7. NMGS-QTMG Output File (*.face) Format

--- start of file
#Comment line
#Comment line
…
…
NF (Number of triangular faces) 3 (dimension)
(Face number, Node numbers of the three vertices of i^{th} triangle)
1 NN_{1_F1} NN_{2_F1} NN_{3_F1}
2 NN_{1_F2} NN_{2_F2} NN_{3_F2}
…
…
NF NN_{1_FNF} NN_{2_FNF} NN_{3_FNF}
#Comment line
#Comment line
…
…
--- end of file

C.8. NMGS-QTMG Output File (*.ele) Format

"*.ele" file format is given in Table C.8. A "*.ele" file includes the total number of tetrahedra and node numbers of the four vertices of them.

Table C.8. NMGS-QTMG Output File (*.ele) Format

--- start of file #Comment line #Comment line … … *NE* (Number of tetrahedra) 4 (dimension) (Tetrahedron number, Node numbers of the four vertices of i^{th} tetrahedron) 1 NN_{1_F1} NN_{2_F1} NN_{3_F1} NN_{4_F1} 2 NN_{1_F2} NN_{2_F2} NN_{3_F2} NN_{4_F2} … … *NE* NN_{1_FNE} NN_{2_FNE} NN_{3_FNE} NN_{4_FNE} #Comment line #Comment line … … --- end of file

C.9. NMGS-QTMG Output File (*.face.gid) Format

"*.face.gid" file format is given in Table C.9. A "*.face.gid" file includes the total number of triangular faces of tetrahedra, node numbers and the x, y, z coordinates of the three vertices of the faces.

Table C.9. NMGS-QTMG Output File (*.face.gid) Format

--- start of file #Comment line #Comment line … … (Reference text)

“mesh dimension = 3 elemtype triangle nnode = 3”
(Reference text for the start of coordinates information)
“coordinates”
1 $x_1\ y_1\ z_1$
2 $x_2\ y_2\ z_2$
…
…
NN $x_{NN}\ y_{NN}\ z_{NN}$
(Reference text for the end of coordinates information)
“end coordinates”
(Reference text for the start of elements information)
“elements”
(Triangle number, Node numbers of the three vertices of i^{th} triangle)
1 $NN_{1\_F1}\ NN_{2\_F1}\ NN_{3\_F1}$
2 $NN_{1\_F2}\ NN_{2\_F2}\ NN_{3\_F2}$
…
…
NF $NN_{1\_FNF}\ NN_{2\_FNF}\ NN_{3\_FNF}$
(Reference text for the end of elements information)
“end elements”
#Comment line
#Comment line
…
…
--- end of file

C.10. NMGS-QTMG Output File (*.ele.gid) Format

“*.ele.gid” file format is given in Table C.10. A “*.ele.gid” file includes the total number of tetrahedra, node numbers and the x, y, z coordinates of the four vertices of tetrahedra.

Table C.10. NMGS-QTMG Output File (*.ele.gid) Format

--- start of file
#Comment line
#Comment line
…
…
(Reference text)
“mesh dimension = 3 elemtype tetrahedron nnode = 4”

(Reference text for the start of coordinates information)
"coordinates"
1 $x_1\ y_1\ z_1$
2 $x_2\ y_2\ z_2$
…
…
NN $x_{NN}\ y_{NN}\ z_{NN}$
(Reference text for the end of coordinates information)
"end coordinates"
(Reference text for the start of elements information)
"elements"
(Tetrahedron number, Node numbers of the four vertices of i^{th} tetrahedron)
1 $NN_{1\_F1}\ NN_{2\_F1}\ NN_{3\_F1}\ NN_{4\_F1}$
2 $NN_{1\_F2}\ NN_{2\_F2}\ NN_{3\_F2}\ NN_{4\_F2}$
…
…
NE $NN_{1\_FNE}\ NN_{2\_FNE}\ NN_{3\_FNE}\ NN_{4\_FNE}$
(Reference text for the end of elements information)
"end elements"
#Comment line
#Comment line
…
…
--- end of file

APPENDIX D

DATA DISK

D.1. Introduction

A CD-ROM device is provided in an envelope attached to the back cover of this book, which contains all the source codes and setup programs of the software packages, this text in Microsoft® Word 2000 document (Master Thesis.doc) format, setup programs of several useful software packages used for opening files in special formats such as *.pdf, *.ps, *.zip, etc., and all the electronic documents downloaded from Internet and used throughout the development of this thesis.