NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

Mesh Generation for Finite Element Analysis

Thi Nhu Hanh Vo

A Thesis

in

The Department

of

Computer science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

September 1988

ISBN 0-315-44860-1

ABSTRACT

Mesh Generation for Finite Element Analysis

Thi Nhu Hanh Vo

The finite element analysis in engineering applications comprises three phases: domain discretization, equation solving and error analysis. Numerous works toward the automation of the analysis with computer technology have been done. The domain discretization or mesh generation is the pre-processing phase which plays an important role in the achievement of accurate solutions. There exist several mesh generators, semi-automatic and automatic, which have been implemented and reported by the authors in research literatures on this topic. The work presented in this thesis is another contribution to this field of research on automatic mesh generation. A brief overview of the finite element analysis and the finite element mesh requirements is given, followed by a complete review and classification of mesh generation methods reported in the literature. One particular and promising technique for producing two-dimensional meshes is described and the successful

implementation and enhancement of the method are presented, showing the advantages and efficiency improvement over some currently available mesh generators.

ACKNOWLEDGEMENTS

I am grateful to my thesis supervisor Professor T.D. Bui for suggesting this topic and for his valuable guidance in the course of this work.

I am indebted to my family and friends for their continuous support and encouragement throughout my studies to complete this work.

I would like to thank the staff of the Computer Center, and the Department of Computer Science of Concordia University for their assistance in using the computing facilities.

My special thanks go to Dr. Z.C. Li for the discussions related to the application of this work, and to W. White for proof-reading of the thesis.

This project was supported in part by the Natural Sciences and Engineering Research Council of Canada and by the Centre de Recherche Informatique de Montréal.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter One

FINITE ELEMENT ANALYSIS AND MESH GENERATION

INTRODUCTION

Many practical problems arising from various fields in engineering are either extremely difficult or impossible to solve by conventional analytical methods. Such methods involve the determination of several mathematical functions to express the relations among the posted variables thus predicting the behavior of some variables in terms of the others. In the past, it was a common practice to simplify complex problems by reducing them to the least form from which an analytical solution that is not too difficult to obtain and bears as much resemblance as possible to the solution of the original problems. Error analysis must always be carried out as a posterior process to determine the degree of accuracy of the approximate solutions. Another form of approximation, the numerical analysis based on the discretization of the problem's domain, has shown to be as much efficient as the analytical simplification methods, and yet they are much easier to

analyze and the solutions can be obtained within acceptable error norms. The accuracy of numerical approximation often depends on how the problems are discretized: the finer the discretization is the greater the accuracy gained. This condition is not difficult to achieve, as the computer technologies provide a powerful tool of analysis for discrete problems.

This chapter presents one of the most popular numerical techniques in engineering: the finite-element method. A brief description of the method as it is applied by means of computer software will show different research topics related to the method, of which mesh generation reveals to be an important and interesting topic.

1. Finite-Element Analysis

The finite-element method was originally developed by engineers in the 1950's to analyze large structural systems. Application of the finite element method to non-structural problems such as elementary flow and electromagnetism in the 1960's has opened the door to a wide class of problems in engineering to which the method has shown to be powerful. Expansion of application of the method to other fields now becomes popular, such as a recent successful use of the method for problems in

pattern recognition [LI88]. As the finite element method matured in applications, the original concept was replaced by a robust theoretical analysis founded on the classical variational calculus and Rayleigh-Ritz methods. There have been since many contributions to the development of the mathematical theory of Finite Elements, leading to several variations of the method, which we shall refer to as finite-element methods or finite-element analysis, in subsequent discussions.

In general, each finite-element method is an approximation procedure for solving differential equations of boundary and/or initial-value type in engineering and mathematical physics. The procedure employs subdivision of the solution domain into many smaller regions of convenient shapes, the so called finite elements, such as triangles and quadrilaterals, and use approximation theory to quantize behavior on each finite element. Suitably disposed coordinates are specified for each element, and the action of the differential equation is approximately replaced using values of the dependent variables at the element nodes. Using a variational principle, or a weighted-residual method, the governing differential equations are then transformed into finite-element equations governing each isolated element. These local equations are collected together to form a global system of ordinary differential or algebraic equations including a proper accounting of boundary

conditions. The nodal values of the dependent variables are determined from the solution of this matrix equation system whose complexity depends largely on the number of finite elements involved in the discretization. It is clear that the finer the domain is subdivided, the greater the accuracy of the solution is. This raises the question: "To what degree the discretization should be in order to obtain a good solution ?" Since the computational cost to carry out analysis on large number of elements is rather high, an adaptive scheme is applied to arrive at an adequate solution within some acceptable error norms: an error analysis must follow the solution of the equation system, which may recommend finer discretization of the domain.

It follows from the above description of a complete finite-element approximation procedure that any computer implementation for finite-element analysis should comprise three distinct but closely related modules whose functions are well-defined (see figure 1.1).

The finite-element pre-processor is responsible for the production of an approximation of the problem's domain by a discrete representation. Basic input required consists of the domain boundary and some parameters defining the variation in size of the finite elements in the discretized domain. The simplex chosen to be the type of the finite elements is often fixed beforehand, but

could also be given as input data to the pre-processor. The ensemble of simplices produced by the pre-processor, which completely covers the domain such that no two simplices intersect, is called the mesh representing the domain in discrete form.

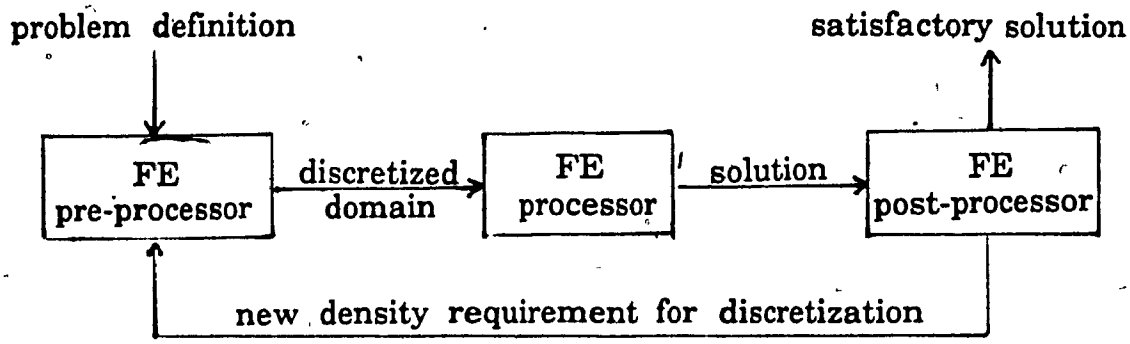


Figure 1.1 Finite-element analysis

The finite-element processor accepts a mesh of the domain and other data such as boundary conditions and materials, then proceeds to find the solution of the differential equation at each node in the mesh. The calculation starts with a computation of the stiffness matrix and load vector associated with each individual finite element. These matrices and vectors are then assembled into a large equation system using information on the adjacency of elements in the mesh. The last step is to find the solution of this matrix system, usually by some numerical method, and thus obtain the value of the solution at each element node in the mesh.

In the last stage of a finite-element analysis, the solution obtained previously from the processor is fed into the post-processor for error analysis. The type of error norm being used by the post-processor depends heavily on the requirement of the problem. The result of this analysis often gives indication on the density distribution, that is the degree of finess, of the treated mesh, and thus provides the mesh generator with new requirements to improve the mesh. Acceptable solutions satisfying all problem conditions would be signaled by the post-processor, terminating the finite-element analysis for the input problem.

2. Finite Element Meshes

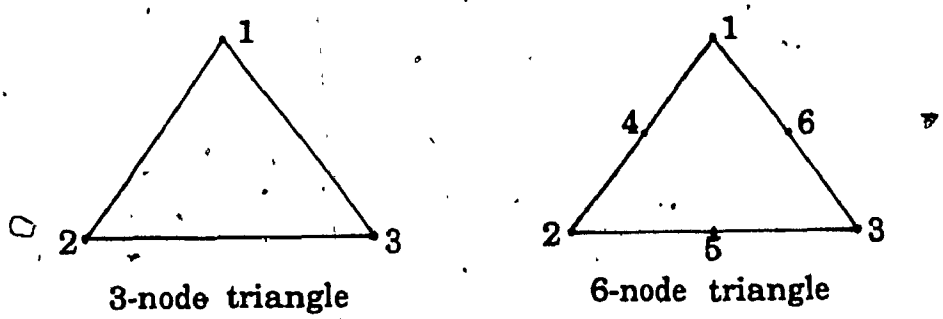
A finite element mesh, by definition, is a discretized form of a given domain (also refered to as region). The discretization is a subdivision of the domain enclosed by a boundary into a number of adjacent smaller subregions of simple shape, which completely fill the region. Such division is not unique for a given domain since it depends on several factors such as element type, number of elements, element size, and element density distribution. However not all subdivisions are suitable for finite-element analysis. This is obviously due to the fact that the local function on each element is itself dependent on the element type and shape, therefore affecting the resulting matrix system and consequently the

solutions. Thus one may ask "How to choose a good mesh suitable for analysis, so that the number of iterations in the analysis is minimum?". Several criteria for selection of finite element meshes satisfying the demand are presented below.

2.1 Finite Element: Type and Shape

The domain discretization in finite-element analysis has no other purpose than that of simplifying the problem. It is therefore obvious that the type of element chosen to fill the domain should have a relatively simple interpolation function. The most commonly used element types are triangles and quadrilaterals in two dimensions, and their counterparts, tetrahedra and bricks in three dimensions.

For each of these geometric element types, variations can be formulated to increase the degree of the associated polynomial function. For example, a three-node triangular element has linear shape function while a six-node triangular element's shape function is quadratic.



3-node triangle 6-node triangle

Figure 1.2 Triangular elements

The choice of an element type depends on the governing differential equations, the number of elements desired, the required accuracy of the solution, and the available computing resources. Firstly the element must represent derivatives up to the order required in the solution procedure. Secondly, if a large number of elements is to be used, a simple form of element would give sufficient accuracy, but more complex elements would be necessary with only a few elements in the subdivision.

Another factor which also affects the final solution is the shape of each element in the mesh. It has been proved [CAR84] that poorly shaped elements lead to unstability of the resulting matrix system. Long and thin elements are to be avoided whenever possible, and elements should have a shape close to the ideal element of the same type, e.g. equilateral triangle is best for triangular elements.

Elements within a mesh can be converted to another type with corresponding changes in the subsequent calculations. Quadrilaterals and bricks are easily converted to well-shaped triangles and tetrahedra of similar sizes. Conversely, triangles and tetrahedra may be subdivided into quadrilaterals and bricks, but the resulting elements may not have good shapes since the angles around the newly introduced nodes are large for a simple subdivision obtained by adding straight lines. This problem can be corrected by moving the interior node points such that all elements have nearly optimal shapes.

2.2 Selection of Finite Element Meshes

In general there exists more than one configuration of the mesh produced for a given domain. The differences are in the type of elements, either a uniform element type or a mixture of several types, and in the shape of each element in the mesh. Comparisons are usually made among meshes of only one element type. The requirements of finite element meshes can be described in terms of boundary fitting, mesh density and conformity, element shapes, and numbering of elements and node points.

2.2.1 Adequate boundary approximation

A finite element mesh should represent a region as close to the original domain as possible, that is the boundary must be well approximated. This requirement can be fulfilled within an acceptable deviation provided that the boundary shape can be replaced by a series of short straight lines which form sides of elements adjacent to the boundary. Sharply curved boundaries require a dense distribution of small elements for a good approximation.

2.2.2 Mesh density and conformity

Analysis on uniform mesh size often reveals to be inaccurate when a coarse mesh is used, or requires too much computational effort when the mesh is dense. In practice a sufficiently accurate solution can be found with less computations by using a small number of elements, but the element sizes vary following the rates of change of the solution. A concentration of relatively small elements is needed in the subregions of the domain where the variables are likely to change rapidly, such as around the singularities of the domain where the solutions are often of greatest interest. Other less important subregions can be accommodated with a more coarse mesh and yet do not seriously affect the accuracy of the final solutions.

A mesh must accommodate changes in element sizes from one subregion to another. This transition may affect the finite-element computation if the mesh is not conform, i.e. adjacent elements do not share a whole edge or face. Mesh conformity is easy to maintain for triangular meshes, but transition from large quadrilaterals to smaller ones often results in a non-conform mesh when the mesh is refined. Non-conformity can be remedied by either modifying the finite-element process to accommodate irregular meshes [SIM79s] or accepting a new element type or shape into the mesh (see figure 1.3).

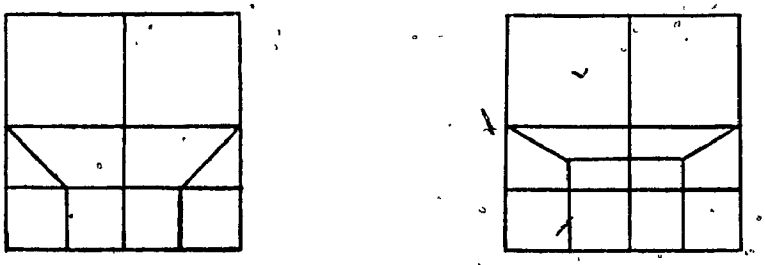
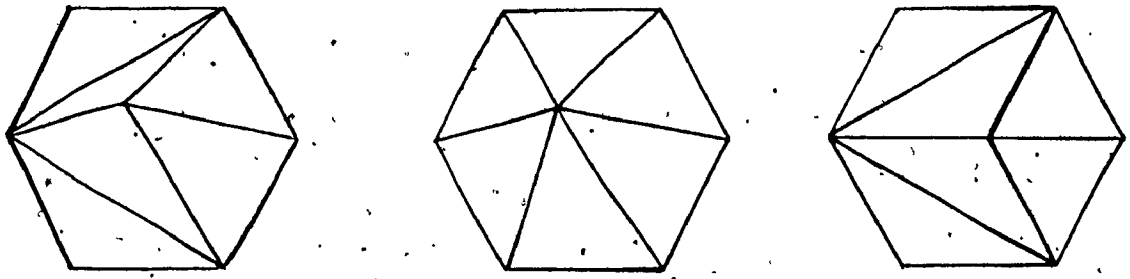


Figure 1.3 Quadrilateral mesh refinement: (a) with triangles (b) with quadrilaterals only

2.2.3 Overall element shapes

It is always desirable to have all elements in the mesh bearing an ideal shape so that the finite-element matrix system is stable. In practice the irregularity of the domain boundary often makes ideal configuration impossible to obtain and so only

relatively good meshes can be produced. Local mesh optimization produces better individual element shape and global mesh optimization attempts to improve element shapes for all elements in the mesh. Local and global optimizations do not always give the same result (see figure 1.4).



(a) (b) (c)

Figure 1.4 (a) Original mesh (b) Locally optimized mesh (c) Globally optimized mesh

2.2.4 Numbering of elements and nodal points

After discretizing the problem's domain, the finite-element methods compute the stiffness matrix and load vector of each individual element and then assemble them into a large matrix system to be solved. The matrix is always sparse, i.e. it contains several zero entries, so that computational cost can be greatly reduced if the non-zero entries are organized into some special form that is simple to solve such as symmetric banded matrix. Since only element adjacency information is used to build up the

matrix, ordering of the elements and nodal points is an important factor for cost saving. In fact, algorithms for element and node re-ordering constitute a well-defined topic in the design of finite-element pre-processor.

2.3 Mesh Information Requirements of the Finite Element Methods

We close our general discussion on finite element meshes with the mesh information required to proceed finite-element analysis, specifically the assembly of the element matrices and vectors. Simpson [SIM79s] outlined three general information requirements for the lists of mesh representation:

a) The need for element descriptions in forming the finite-element equations;

b) The need for geometric adjacency information for vertices or elements; and

c) The need to determine which element of the mesh contains a given point P of the region.

Tables of connectivity and adjacency provide adequate structures to meet the requirements. It is even sufficient to have only connectivity (element definition) since list inversions of the table would give adjacency information.

3. Triangular Meshes

In two-dimensional analysis, the triangle is the most widely used simplex because of its simple shape function and its suitability for approximating boundary curves of different shape complexity, from regular to sharply curved boundaries. Therefore it is not surprising that most of the literatures on the problem of mesh generation is devoted to triangular meshes in two dimensions and tetrahedra in three dimensions [BUE73,SIM79s,THA80,HOL88]. The general criteria for finite element meshes apply to triangular meshes as well. Local and global optimization procedures are well developed and shall be presented in details. The local optimization results from the properties of a special class of triangulation, the Delaunay triangulation, considered to be optimal for convex hull triangulation [WAT83]. Global optimization, commonly known as mesh smoothing, applies to a completely generated mesh while local optimization is applied every time an element is generated. The validity of triangular meshes with respect to finite-element applications can be verified to ensure stability in the finite-element calculations.

3.1 Delaunay Triangulation

Definition 1.1 [CLI84]

A region of the plane is convex if and only if for any two points contained in the region the line segment connecting the two points also lies in the region. The convex hull of a finite set of points in the plane is the smallest convex region which contains the points. Such a convex hull is closed and contains all the points which define it, and the convex hull of a finite set of non-collinear points has at least three of the points on its boundary.

A Delaunay triangulation (also termed as Thiessen, Dirichlet tessellation) of a given set of points is a set of triangles formed by joining these points such that they completely cover the convex hull of the points. Delaunay triangulation is the dual geometry of the Voronoi tessellation defined below.

Definition 1.2 [MAU84]

Let $P = \{p_1, p_2, \ldots, p_n\}$, $n>2$, be a finite set of n different points, not all collinear, in the Euclidean plane, and let $d(p_i,p_j)$ be the Eucliden distance between point p_i and p_j. Then

$$V_m = \{x : d(x,p_m) < d(x,p_k), \forall k \neq m\}$$

is the Voronoi polygon surrounding the point p_m. The set of Voronoi polygons of p_1, p_2, ..., p_n defines the Voronoi tessellation of P.

It is clear that the Voronoi tessellation of a given set of points is unique. The duality was established by Delaunay [DEL34] who showed that the dual of the Voronoi polygons is a triangulation of the n points.

Definition 1.3 [MAU84]

An edge of a set P of n points is a line segment between two points p_i and p_j in P. A triangulation T of P is a set of edges such that:

(a) No two edges intersect (except possibly at the endpoints), and

(b) It is not possible to add another edge to T without violating (a).

The interior of a triangulation are triangles and the outer edges form the convex hull of P.

Definition 1.4 [MAU84]

A Delaunay triangulation is a triangulation where the circumscribed circle of any triangle contains no points of P in its interior. Given a Voronoi tessellation of P, the dual Delaunay triangulation is obtained by joining every two points in P whose Voronoi polygons have a common edge, assuming that no four points of P are cocircular. When four points are on the same circle, connection of these points forms a quadrilateral and either

of the diagonals is chosen to be an edge. This phenomenon is called degeneracy in the triangulation [BOW81,WAT81].

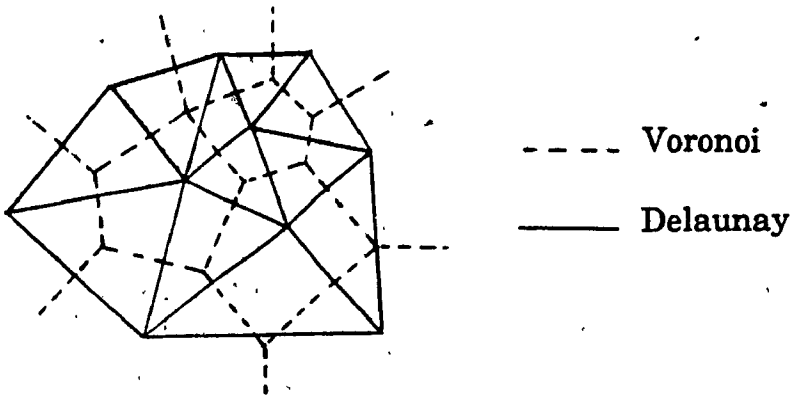


Figure 1.5 Voronoi tessellation and Delaunay triangulation

There exist several triangulations of a set of points P but the Delaunay triangulation of P is unique, and the number of triangles and edges is constant for all the triangulations of P [LAW77].

Theorem 1.1 [MAU84,CLI84]

Let there be M points on the convex hull and N interior points in P. Then there are t triangles and e edges in any triangulation of a planar set P of n points, where

$$t = M + 2N - 2$$

$$e = 2M + 3N - 3$$

$$M + N - 2 \leq t \leq 2M + 2N - 5$$

and

$$2M + 2N - 3 \leq e \leq 3M + 3N - 6$$

Proof: The Euler-Poincaré theorem

$$\text{regions} + \text{vertices} - \text{edges} = 2$$

holds for any connected planar graph.

There are M boundary edges in the triangulation. Each interior region has 3 edges which gives 3t+M edges, each being counted twice. Therefore $2e = 3t + M$.

The number of regions is t+1 (t interior regions and the convex hull), and the number of vertices is M+N. The equation is

$$(t+1) + (M+N) - e = 2$$

Substituting $e = \frac{1}{2}(3t+M)$ and solving for t gives

$$t = M + 2N - 2$$

and hence $e = 2M + 3M - 3$

The inequalities follow from simple observations.

Q.E.D.

Delaunay triangulation is optimal [WAT83] and its empty circle property is in fact used as a criterion for the local optimization of triangular meshes.

3.2 Local and Global Triangular Mesh Optimization

The local optimization procedure (LOP) attempts to improve each element shape individually. The test is made on the basis of two adjacent triangles at a time. If the quadrilateral formed by the two triangles is convex, the test would make a decision on

whether or not the diagonal (the common edge) should be swapped to satisfy some criteria of Delaunay triangulation. The procedure is formally described below [CLI84].

Definition 1.5

A pair of triangles (p_1,p_2,p_3) and (p_2,p_1,p_4) of a triangulation T, which share a common edge form a quadrilateral of T denoted (p_4,p_2,p_3,p_1). The quadrilateral is said to be strictly convex if the diagonals p_1p_2 and p_3p_4 intersect at a point inside the segments p_1p_2 and p_3p_4. A swap in this case is the replacement in T of (p_1,p_2,p_3) and (p_2,p_1,p_4) by (p_3,p_4,p_2) and (p_4,p_3,p_1). Note that the swap leaves the number of triangles and edges unchanged.

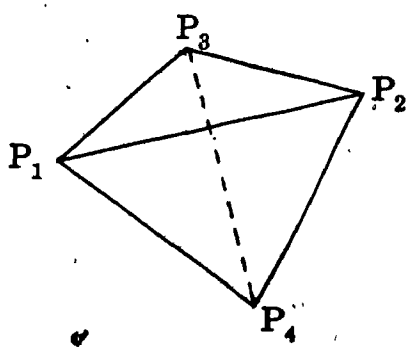


Figure 1.6 Diagonal swap test

Definition 1.6

Given an interior edge p_1p_2 with corresponding quadrilateral (p_1,p_4,p_2,p_3), the local optimization procedure performs a diagonal swap based on either of the following criteria:

a) The *max-min angle criterion* selects the pair of triangles which

maximizes the minimum of the six interior angles when (p_1,p_4,p_2,p_3) is strictly convex. A swap is made on the diagonal only when the quadrilateral is strictly convex and the new pair of triangles satisfy the max-min angle criterion.

b) The *circle criterion* selects the pair of triangles whose circumscircles do not contain the remaining vertices in their interiors. The swap is performed only when the fourth vertex of the quadrilateral is interior to the circumcircle of the other three vertices.

The above criteria can be shown to be equivalent [LAW77]. There is another criterion employed by some authors [SIM79], choosing to minimize the maximal angle. Equivalence to the max-min angle has not been proved, although it does produce adequate meshes.

In addition to the local optimization, a triangular mesh can be improved globally by a procedure called mesh smoothing. Several techniques are available, the most popular one is Laplacian smoothing which seeks to re-position each node to the centroid of the polygon formed by triangles sharing the point. This re-positioning can be done iteratively or simultaneously by setting up a set of equations for each interior point using the Laplace equation for point i,

$$P_i = \frac{1}{n_i} \sum_{j=1, j\neq i}^{n_i} P_j$$

where P_j are the coordinates of the vertices of the surrounding polygon and n_i is the number of such vertices. Figure 1.4(c) illustrates the result of Laplacian smoothing on a triangular mesh.

In some cases, the Laplacian scheme does not work well. A correction is proposed by Hermann [HER76] which adds a weighting factor w to the Laplace equation of a point i:

$$P_i = \frac{1}{e_i(2-w)} \sum_{n=1}^{e_i} (P_{nj} + P_{nl} - wP_{nk})$$

where e_i is the number of elements around node i and $0 \leq w \leq 1$. The positions of p_{nj}, p_{nl}, and p_{nk} are as in figure 1.7. This isoparametric smoothing resumes Laplacian when w = 0.

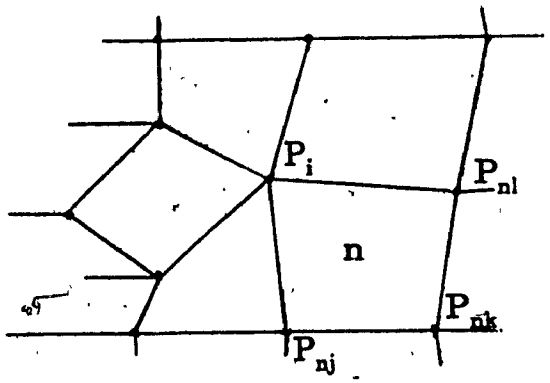


Figure 1.7 Isoparametric smoothing

Isoparametric smoothing is mostly used for quadrilateral meshes; for triangular meshes the Laplacian scheme is sufficient.

3.3 Mesh Validity in the Context of Finite Element Analysis

This section discusses the nodal numbering for a triangular mesh so that the matrix system obtained from the assembly of element matrices and vectors yields correct solutions. Simpson [SIM81] defines the order of local nodal numbering within each element to be counterclockwise, although it is immaterial which node is defined first. If a clockwise order of numbering is used, the element area will be negative, resulting in serious errors in subsequent analysis. The author has devised a two-dimensional mesh verification algorithm for triangular meshes [SIM81] which can be described in terms of the following conditions for a given planar triangulation:

a) The triangle vertices are specified in counterclockwise order;

b) Either the i^{th} edge of the k^{th} element is the only edge joining its endpoints (i.e. the k^{th} element is a boundary element), or there is exactly one element number n having the same edge. In the later case, the direction of this edge in k and n must be different;

c) No boundary edge intersects more than one element except at its endpoints; and

d) A vertex can have at most one boundary edge directed away from it.

It is clear that this test only applies to conforming triangular meshes.

4. Mesh Generation

At the early stage of finite-element analysis, meshes were produced manually by the analysts who also prepare domain description and node numbering. Coarse meshes are not too difficult to obtain but they usually cannot satisfy the accuracy requirement of the analysis. Generating finer meshes is tedious and requires large amount of man-work which makes finite-element methods not recommendable despite its simplicity and accuracy. A solution to this problem is to let the computer do the discretization work, much faster and more accurate, suitable for the analysis. Before presenting any mesh generation algorithm, we shall first address the problem in its general aspect, namely the information and desirable features for automated meshing techniques.

4.1 Mesh Generation: Input and Output

Since computer instructions can only deal with discrete data, it must be possible to specify domains in some suitable form such as functions or sequences of discrete data points. A domain is defined to be the region enclosed by a set of boundary curves from which

we identify one external boundary defining the largest region enclosing all other internal boundaries which represent holes. In two dimensions, boundaries are closed curves, and in three dimensions they are surfaces that defines a closed region. A mesh can be automatically produced solely from the definition of boundaries. Simple and regular boundary shapes can be described by functions of two or three variables, while functions of arbitrary, irregular shapes are not easy to find. In this case, a curve in 2D can be approximated by a set of consecutive line segments. Mesh generators ususally take more than just domain definition as input, but also control parameters on the final mesh such as density distribution of elements, number of elements, etc. Representations of these parameters vary from one implementation to another.

Meshes generated by computer generators always have at least the connectivity information about the mesh, i.e. element definition and the number of elements in the mesh. As discussed previously, this connectivity information relies on a specific numbering scheme of the nodal points, which can be stored in lists. Another useful but optional output is the table of element adjacency which gives the elements adjacent to each element. Of course this information can be visualized by means of some graphics device since the vision of the mesh is always more desirable to the users of the mesh generators. In fact, any mesh generator should have built-in

interactive graphical interface to make mesh generators truly convenient and efficient tools in computer-aided finite-element analysis.

4.2 Features of Automatic Mesh Generation

With the advances of computer technology and the outgrowth of research on the topic, automatic mesh generation is becoming an integral part in almost any finite-element analysis software package. Due to the wide applicability of finite-element methods, some mesh generators are developed specifically for certain fields of application, e.g. [COR87]. However there are also general-purpose mesh generation algorithms devised by several authors using different approaches. These algorithms when implemented show to work well, but correctness proof is often omitted, and thus cannot guarantee absolute generality, e.g. [CAV74,NEL78]. Nevertheless, the design of algorithms producing finite element meshes should, in principle, achieve the following features of automated meshing techniques [HAB81].

a) Precise modelling of boundaries

No error beyond the discretization error inherent to the chosen finite-element model should be introduced by the mesh generation process. Boundary nodes should lie precisely on the boundary of the structure. In two-dimensional structure the location

of interior nodes is less critical provided that acceptable element shapes are obtained. Meshing must not be limited to certain shapes of boundary curves and should support highly complex boundary shapes.

b) Good correlation between the interior mesh and information on the mesh boundary

The curvature and node spacings on the boundaries of the region should be well reflected to the interior of the mesh to allow control on the element size in any region of the domain in a predictable fashion. This would ease the refinement of the mesh and avoid unnecessary refinement leading to wasted computations.

c) Minimal input effort

The amount of data provided by the user should be reduced to a minimum without affecting the generated mesh. This will also reduce the chances of introducing human error into the analysis. The input information should be in a convenient form and easy to communicate to the computer.

d) Broad range of applicability

It is desirable to have a small set of mesh generation techniques that are applicable to a wide variety of structural topologies rather than to use a large set of special-purpose mesh generators. This would create a convenient user interface in that users don't have to learn to use part of the system every time

they come up with new structural problems.

e) General topology

No restriction should be imposed on the topology of the mesh within a region. This maintains the regularity and good shapes of the mesh elements.

f) Automatic topology generation

Element connectivity should be created without user intervention. This reduces the amount of input required and automates the process of generating meshes.

g) Favourable element shapes

Any element in the automatically generated mesh should possess shapes that are as close to ideal as possible to avoid ill-conditioning in the finite-element model and reduce subsequent errors in the analysis.

h) Optimal numbering patterns

The arrangement of node numbers should be in such a way that the resulting matrix system is favourable to the technique used to solve the system. For example, a matrix with minimum bandwidth requires much less computations than an equivalent but sparse matrix.

i) Computational efficiency

Good response and minimum use of resources are common requirements to all software tools, with no exception to finite

element mesh generators. These are expressed in terms of storage used, response time and computer time to generate mesh for arbitrary domains.

Not all of the above features are present in currently available mesh generators. Precise boundary fitting and minimal user input are well incorporated in any developed mesh generator, as well as additional improvement on element shapes. Not all boundary topologies could be tested so that the question of generality cannot have a definite answer. Optimal numbering patterns represent a post-process for mesh generation; independent algorithms exist for this purpose. The efficiency question, especially in terms of time complexity, is difficult to discuss due to the lack of standard analysis procedure for mesh generation algorithms, and is often omitted for heuristic algorithms described in most research publications on mesh generation.

In the next chapters we shall explore the design and implementation of a mesh generator in greater details than the general description in this chapter. We will focus our attention on general-purpose mesh generators keeping in mind the above characteristics.

Chapter Two

METHODS FOR FINITE ELEMENT MESH GENERATION

INTRODUCTION

The first survey of computer application in finite-element pre-processing appeared in 1973 by Buell and Bush [BUE73] with reference to 30 papers related to finite-element analysis and mesh generation. The survey covers schemes for node and element generation as two independent classes of algorithms. These techniques use the idea of element transformation from the variational formulation of finite-element models, in which calculations are carried on a master element and transferred to the actual element in the model. The second survey, presented by Simpson [SIM79s] only six years later, reveals considerable progress in the field. The review focuses on ideas pertinent to the design of software, namely basic representations of meshes with selected data structures, requirements and verification of automatically generated meshes. The classification scheme includes four distinct approaches restricted to the generation of triangular meshes. The methods are

classified by their basic concepts, which are completely different between the defined classes. The class of coordinate mapping algorithms uses mathematical transformation from a simply-shaped region to the actual domain. The local mesh refinement class uses repeated subdivision of a given coarse mesh until the mesh meets some specified analysis requirements. The class of vertex triangulation methods separates the generation into two phases: node generation and element generation. The fourth class contains boundary contraction methods which generate nodes and elements simultaneously by shrinking the boundary. This survey by Simpson emphasizes on the software side and presents typical method for each class. The brief review given by Thacker in 1980 [THA80], by contrast, includes a more complete enumeration of the techniques available up-to-date, with a brief description of a few approaches, but no classification is explicitly proposed. Other computational aspects such as boundary specification, modes of operation (interactive, batch) and post-processing to reduce matrix bandwidth by re-ordering of nodes and elements are also mentioned. Another survey by Watson and Philip [WAT83] touches only a particular problem in triangular mesh generation, namely the connection of nodal points to obtain the mesh. The authors define three different systematic triangulation methods: Optimal, Greedy, and Delaunay triangulations, each using different criterion to select suitable

elements. Some comments on the efficiency of these methods are also given: $O(N^3)$ for Optimal triangulation, $O(N^2 \log N^2)$ for Greedy triangulation, and $O(N^2)$ fot Delaunay triangulation. These bounds have not been mentioned in any other publications except for the class of Delaunay triangulation. The latest review on finite element mesh generation methods by Ho-Le [HOL88] contains an up-to-date reference list and a broader classification scheme applying to two-dimensional mesh generation as well as three-dimensional methods. The review is complemented by a comparison of the various approaches based on criteria described in chapter one: element type and shape, mesh density, and time efficiency.

In this chapter, we present our review of finite element mesh generation techniques focussing on triangular meshes. The order of presentation is: classification of methods, description of 2D methods, performance evaluation of 2D methods, and a brief description of 3D methods.

1. Classification of Mesh Generation Methods

In this section, the mesh generation methods, semi-automatic and automatic, are classified. The classification scheme is similar to [HOL88] with some modifications. There are basically four main classes, arranged by their different nature of generation techniques:

mesh topology first, node first, adapted mesh template, and simultaneous generation of nodes and elements. Each of these classes has one or more subclasses. The complete classification is presented in figure 2.1. The mesh generation methods are classified based on the order of creation of the two basic output sets, the set of nodes and the set of elements.

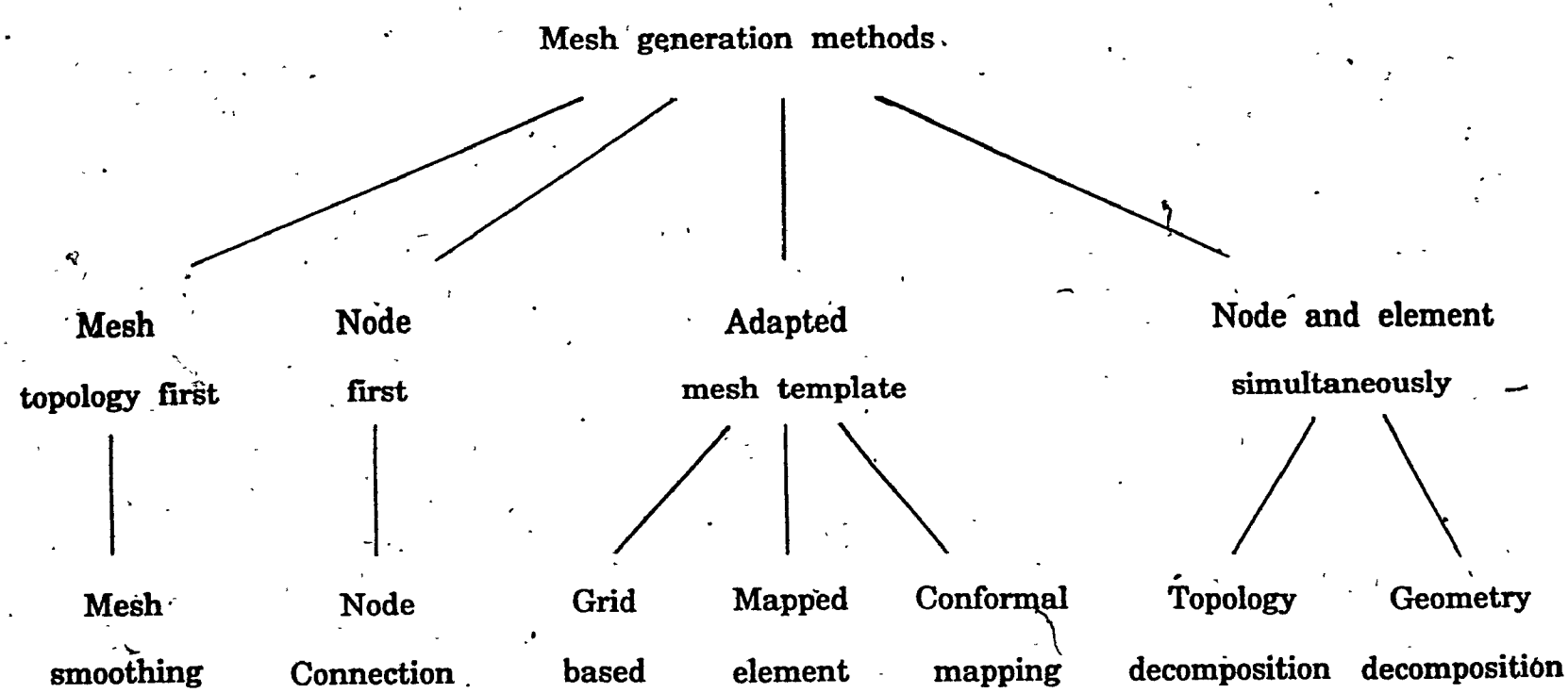


Figure 2.1 Classification of Mesh generation methods

1.1 Mesh Topology First

In this approach, the configuration of the mesh is determined first. This topology consists of the number of elements, the number of nodes, and the interconnection between nodal points to form the elements. Once the mesh topology has been determined, the mesh smoothing techniques used in the global optimization [HER76] can be used to find the exact nodal positions. A typical and generally used technique is to solve a set of Laplacian equations created from the internodal connection previously defined, under the constraint that boundary nodes are fixed. This is the same scheme which was presented in chapter one under the local and global mesh optimization section. The problem with this approach is that there is no known algorithm for creating the mesh topology for arbitrarily-shaped domain, thus the scheme can only be used as complementary post-processing for other automatic mesh generation schemes.

1.2 Node First

The meshes are created in two phases: generation of nodes and creation of element connectivity. The first step produces additional nodes inside and on the boundary of the domain according to the density distribution parameters given by the user

or determined by the processor. The next step is to establish the connection between the nodes to form triangular or quadrilateral elements. These elements are formed in such a way that they do not overlap and do satisfy the criteria associated with subsequent analysis. Enhancement of the generated meshes by some smoothing technique is optional but recommended for the meshes generated by these methods. This class has only one subclass, the node connection or vertex triangulation approach. Numerous algorithms for the triangulation phase are available in research literature.

1.3 Adapted Mesh Template

The mesh for the object of interest is adapted from some predefined mesh template. The mesh template is usually a regular and optimal mesh of simple geometry such as triangle, rectangle, unit cube, etc. Such geometry does not give any difficulty in the generation of the mesh since simple formulas can be derived to calculate the nodal positions and element connectivity [FEN75]. There are three subclasses identified by the template used to create the meshes: grid-based approach, mapped-element approach, and conformal mapping approach.

1.3.1 Grid-Based Approach

An infinite rectangular or triangular grid is superimposed on the domain to be meshed. The grid elements that fall completely outside the domain are discarded, and the ones on the boundary are adjusted to fit into the domain. Meshes created in this manner always have good shapes for interior elements but boundary elements may be very irregular.

1.3.2 Mapped-Element Approach

The domain is subdivided into a set of four-sided (or three-sided) regions, each of which is then mapped to a rectangular mesh in the unit square (or a triangular mesh in an equilateral triangle) via a mathematical function. Each region in the subdivision is called a macro element. The subdivision may be carried out manually or automatically. Automatic subdivision may be difficult to obtain for complex boundary shapes.

1.3.3 Conformal Mapping Approach

The mesh template is a polygon P that has the same number of vertices as the simply-connected region R to be meshed. The polygon P is constructed in such a way that it can be easily meshed, and a conformal mapping F from P to R is found based

on the correspondence between the vertices in P and R. The mesh in P is then mapped onto R using this mapping function. Again the effectiveness of the method depends on the shape of the boundary as in the mapped-element approach.

1.4 Simultaneous Creation of Nodes and Elements

In this class, there is no distinction between the node and element generation phases, and yet no mesh template is used to obtain the mesh. The two different subclasses are mesh refinement and geometry decomposition approaches. In the mesh refinement approach, an initial mesh is constructed from the boundary nodes and refined by subdivision of elements into smaller ones until the desired density is met. Oddly shaped elements may be introduced as the mesh is refined. The geometry decomposition approach attempts to generate good elements by considering the object geometry while decomposing the object into elements or simple regions, and then generating elements. The efficiency of this approach also relies on the geometrical complexity of the object's boundary.

We have described the four classes of mesh generation techniques in general terms. The next sections present the distinct methods in two dimensions and three dimensions with a

performance evaluation for 2D methods.

2. Two-Dimensional Mesh Generation Methods

This section reviews the methods published in the literature on 2D mesh generation, in particular for triangular meshes. The methods will be presented using a descriptive, non-algorithmic language because the diversity of these techniques makes it difficult to give the algorithms for each of them since they are not available in the referenced publications. We shall present these methods using the classification scheme given previously at the subclass level. The performance evaluation will not be given for each individual subclass but rather at the end of this review since it would be easier to have an idea of the relative efficiency of the various methods. More than one representative methods may be described within a subclass depending on their properties and significant differences with other methods of the same category.

2.1 Mesh Smoothing Approach

Given a boundary description in discrete form, that is by point coordinates and a layout (connectivity) of elements in the mesh, the generation procedure consists of establishing a set of equations whose unknowns are the nodal positions of interior points and

then solving for these coordinate values. The most commonly employed generation procedure for arbitrary geometries and element layouts is due to Wilson as cited in [HER76]. This procedure is called the Laplacian scheme which uses equations on the connectivity of the nodes,

$$x_i = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}$$

and

$$y_i = \frac{1}{n_i} \sum_{j=1}^{n_i} y_{ij}$$

for points in two dimensions, where x_{ij} and y_{ij} are coordinates of the points directly connected to point i. Figure 2.2 illustrates this relation for $n_i = 4$.

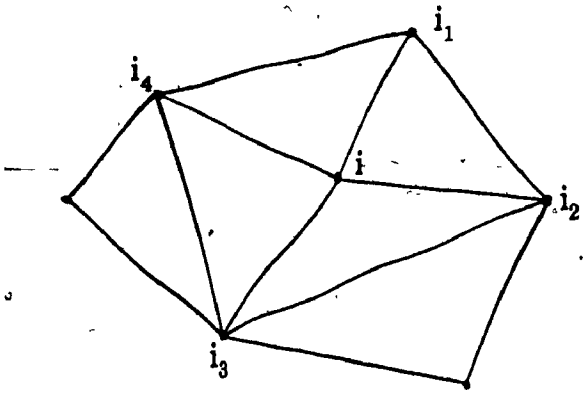


Figure 2.2 Connectivity in the Laplacian equation

The construction of the set of equations for interior nodal positions is performed by establishing the above equation for each point inside the domain, replacing fixed boundary nodes by their known coordinate values. An illustration of the method on a simple

square with 8 nodes on the boundary and only one interior node is in figure 2.3.

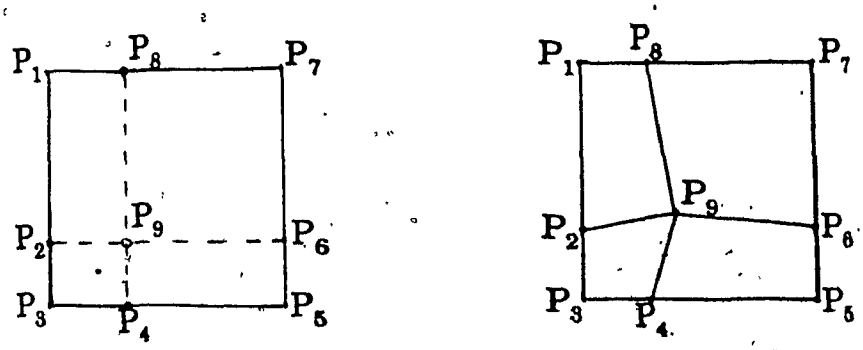


Figure 2.3 Mesh smoothing technique

(a) Mesh topology (b) Laplacian smoothing

For this simple mesh, we only need to solve the equation for point P_9,

$$x_9 = \frac{1}{4}(x_2 + x_4 + x_6 + x_8)$$
$$y_9 = \frac{1}{4}(y_2 + y_4 + y_6 + y_8)$$

Other schemes to improve the Laplacian scheme exist as variations of the original one [HER76]. Despite its simple formulation, mesh smoothing is not very attractive because it is not very efficient. The problem is that there is no way to predict the mesh topology except by using other automatic mesh generation techniques. Therefore mesh smoothing is often used as a global optimization procedure for meshes generated by other methods.

2.2 Node Connection Approach

This approach divides the mesh generation process into two distinct phases: addition of new nodal points and triangulation of a set of points. New points are inserted in such a way that they satisfy given density control parameters or functions in different regions of the domain to be meshed. In the connection phase, nodes are joined by edges which form generally good elements suitable for analysis. Criteria to determine good elements have been described in chapter one on finite element meshes.

2.2.1 Node Generation

Essential input to a mesh generator is a set of boundary node coordinates, or a set of functions describing the domain boundary. Boundary nodes are sufficient to form elements, but a mesh obtained in this way usually has no use in the analysis. Nodes must be introduced inside the domain and sometimes on the boundary to satisfy density requirement. Some mesh generators, by contrast, use the node spacing on the boundary to determine the density of interior nodes [TAN87]. Interior nodes can be generated manually through some interactive device as in [FRE70] or automatically by the mesh generator as most of recent

implementations. There exist several algorithms for automatic node generation. These can be classified into two types: random and non-random insertion algorithms.

a) Random insertion

Nodes are generated through the use of a random number generator inside each small area of the domain to be meshed. The first scheme was developed by Suhara and Fukuda [FUK72] and followed by a number of authors [CAV74,MOS83]. In this scheme, a square grid is superimposed on the domain, each grid cell has size proprotional to the density parameter which also serves as a condition to accept or reject a point randomly positioned within the cell. To avoid the formation of acute angle, an imaginary boundary is used in place of the input boundary during the node generation process. This imaginary boundary is obtained by shrinking the original domain by a certain factor determined from the required density. A new node P in a grid cell is recorded only if it satisfies the conditions:

i) P is inside the region bounded by the imaginary boundary, and

ii) P is not too close to any previously generated nodes.

The first check can be done by determining whether P is to the left of all boundary segments traversed in counterclockwise order. However this condition breaks down when the domain is not

convex. In this case alternative test must be used [MQS83]. The distance between P and other points must not be smaller than the density allowed. If a circle centered at P of radius equal to the square size does not contain any other points, P is accepted as a new node. Otherwise it is rejected.

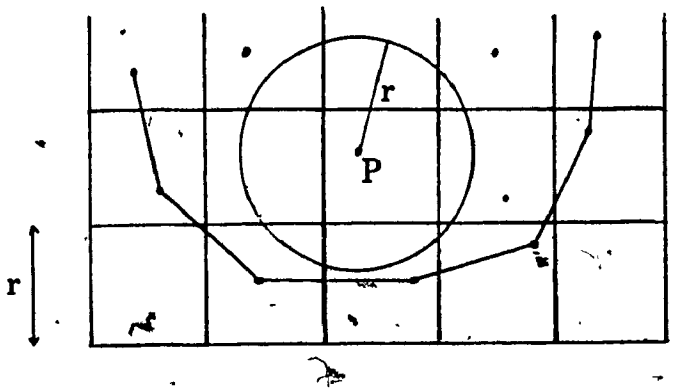


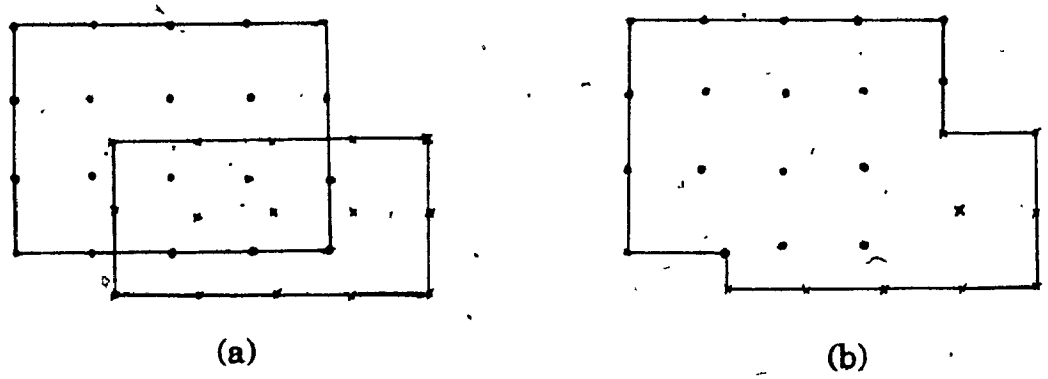
Figure 2.4 Circle test for a point P

If P is rejected, another position within the cell is randomly generated and tested as above. Once the number of attempts has passed a predefined value, that cell is dropped from consideration and the process is repeated for the next cell.

For different zones of different density, the nodes are generated inside each zone independently using the associated density parameter to determine the grid cell dimension.

b) Systematic (non-random) insertion

Non-random methods generate nodes at fixed calculated positions. This scheme seems to be preferred over random insertion since the method can be devised in such a way that density requirement is guaranteed with a lowest number of validity tests for each point. In [SHA78] the authors use the same grid cell concept as Suhara-Fukuda and others, but take rectangular grid cells and choose only two fixed positions in each cell to be two new points. These points are guaranteed to be well displaced from each other and only a test against the boundary needs be done. McGirr and Kraulis [McG84] use circle test on a fine grid for each intersection of grid lines. A simple and faster scheme devised by Lo [LOH85] generates points on the horizontal lines crossing the domain. These lines are equidistant and points are generated in a zig-zag manner on consecutive lines. This method reduces the number of validity checks between points as above. In [JOE86] the author generates nodes by combining the domain shrinking idea and zig-zag node placement on horizontal lines. More checks can thus be eliminated.



(a) (b)

Figure 2.5 CSG point generation

(a) Two overlapping sets of points

(b) Combined set of points

Figure 2.5 illustrates the constructive solid geometry (CSG) scheme employed by Lee & al. [LEE84]. The domain is considered to be a combination of several primitives such as triangles, rectangles, and circles. Each primitive has a predefined set of interior nodes. When combining the primitives, points can be eliminated so that no two points are too close to each other. Other remaining points may be moved to fit the boundary of the domain.

2.2.2 Triangulation

There exists a large number of triangulation methods in the literature on two-dimensional mesh generation. All methods attempt to arrive at an optimal configuration for the mesh produced, and

usually Delaunay triangulation is used as model. A difference between each individual method is on the complexity of the domain to which they apply. Some methods only work on convex regions, others are designed to handle any type of shapes. However there is another, more logical way, to classify triangulation methods by considering the idea employed to perform the triangulation. We have three distinct schemes: the fully-surrounding scheme, the problem-reduction technique, and the iterative insertion.

a) Fully-surrounding scheme

The two basic primitives of the mesh topology, node and edge, play the key role in this scheme. A node-based method can be found in [FRE70] and an edge-based method is devised by Suhara and Fukuda [FUK72] and widely used [CAV74,SHA78,NEL78,MOS83].

Frederick & al. made the following observation: A node which is shared by a number of triangles cannot be used in any other triangle if the angle surrouding it by the sharing triangles is 360°. Such node is said to be fully surrounded. Thus triangulation can be done by fully surrounding each node. When all points are fully surrounded the triangulation terminates. Obviously several tests must be performed to ensure no crossing of triangles and best suitable triangles are obtained. For boundary nodes which cannot

be fully surrounded, imaginary points are added to maintain the consistency of the method. These "ghost" points are deleted once the triangulation is completed.

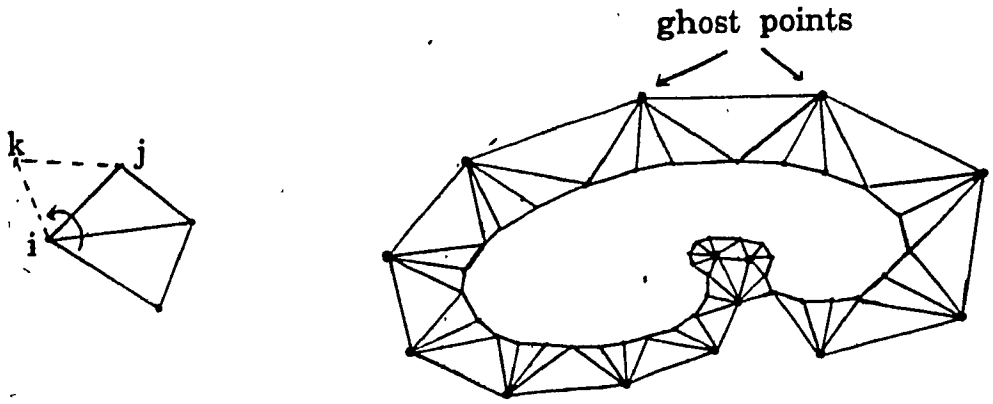


Figure 2.6 Fully-surrounded nodes for triangulation

Similarly, the edge-based methods use the fact that an edge cannot belong to more than two triangular elements in the mesh. Thus a fully surrounded edge has exactly two elements sharing it for any interior edge, while a boundary edge belongs to exactly one element. Suhara and Fukuda's method takes a predefined base from existing elements and forms a new element by choosing an appropriate point among the nodes. This method requires several tests against the boundary and the existing elements. Improvements were made by Nelson [NEL78] to reduce the number of tests for element crossing. The advancing-front method of Lo [LOH85] completely eliminates the element crossing test. Similar method is used by Lee & al. in their constructive solid geometry approach to mesh generation.

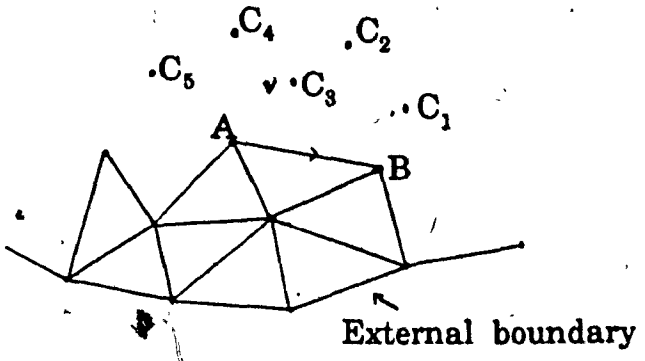


Figure 2.7 Fully surrounded edge

Other methods in this same category are the fully surrounded nodes and edges in [MAU84] to give both the Delaunay triangulation and the convex hull of a set of points. McGirr & al. construct the Voronoi diagram then find the dual Delaunay triangulation [McG84]

b) Problem-reduction technique

Lewis and Robinson [LEW78] use the Quicksort's divide-and-conquer idea to design their triangulation scheme. The triangulation of a region R can be achieved by

i) splitting R into two sub-regions, R_1 and R_2, by creating a new boundary across the region; and

ii) solving the triangulation problems for R_1 and R_2 separately.

Figure 2.8 illustrates such region cutting operation.

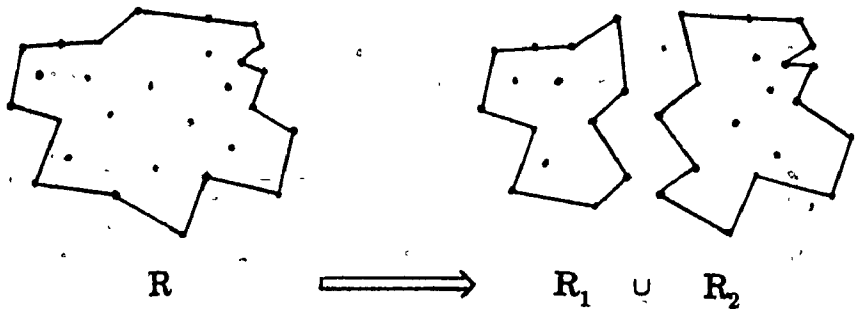


Figure 2.8 Splitting a region

Sub-regions are divided until triangles with no interior points are found, these being the elements of the triangulation. Triangles containing interior points are splitted by two lines joining an interior point to two vertices. The same idea is found in [LEE80], algorithm 1.

c) Iterative insertion

In contrast with the divide-and-conquer technique, the iterative insertion technique builds a new triangulation from an existing triangular mesh by inserting a new point and update the mesh so that it remains optimal, usually equivalent to a Delaunay triangulation. The commonly used and representative scheme can be attributed to Watson [WAT78] as well as to Bowyer [BOW78] who gives a similar result at the same time. The scheme works as follows. Given a set of points, the construction starts by finding a

super-triangle that completely encompasses all the data points to be triangulated. This is equivalent to an initial mesh. When a point P is introduced into the triangulation, we first find an existing triangle which encloses P, and form three new triangles by connecting P to each of its vertices, with the original enclosing triangle then deleted. After the insertion of P the triangulation is optimized by applying the swap test [LAW77] described in chapter one. Once all points have been inserted, the triangulation of the convex hull of the set of data points is completed. Since this algorithm is applicable only to simply-connected convex domains, some pre-processing must be done for arbitrary input domain such as subdivision into convex regions [DEF85,JOE86s].

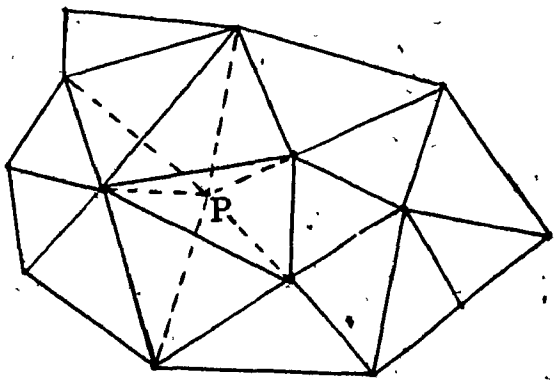


Figure 2.9 Iterative insertion

2.3 Mapped-Element Approach

The widely used method in this approach is first described by Zienkiewics and Phillips [ZIE71]. The essence of this scheme is the use of isoparametric curvilinear mappings of quadrilaterals. A unique coordinate mapping of curvilinear coordinates (ξ,η) and Cartesian coordinates (x,y) for an eight-node parabolic quadrilateral of figure 2.10 is given by

$$x = \sum_{i=1}^{8} N_i(\xi,\eta)x_i$$
$$y = \sum_{i=1}^{8} N_i(\xi,\eta)x_i \qquad (1)$$

in which $N_i(\xi,\eta)$ is a shape function associated with each node i and (x_i,y_i) are coordinates of the eight boundary nodes.

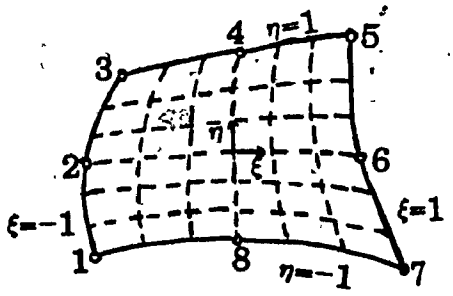


Figure 2.10 Isoparametric mapped element

Typical shape functions for a variety of elements can be found in [ZIE69]. If the coordinates (x_i,y_i) are known, then the Cartesian coordinates of any specified point (ξ,η) in the quadrilateral can be found by using equations (1).

Therefore the mesh generator operates in two phases:

i) Subdivision of the domain into n-node polygons, n = 3, 4 depending on the type of the mapped element used.

ii) Calculation of the Cartesian coordinates for nodes corresponding to the nodes inside the mapped element which already have a mesh defined.

There exist other methods using different mapping schemes such as transfinite mapping, discrete transfinite mapping [HAB81], and composite mapping [CRA87].

Mapped element approach is simple but has some drawbacks, such as a restriction on the mesh topology, e.g. the number of elements along opposite sides for quadrilateral mesh must be the same which propagate throughout the subdivision.

2.4 Conformal Mapping Approach

This approach uses the same mathematical concept as in the mapped-element approach, but employs polygonal mesh template in general. A scheme developed by Brown and Hayhurst [BRO82]

using the Schwarz-Christoffel transformation works as follows. A two-dimensional simply-connected region to be meshed can be modeled with a straight-sided polygon P by approximate discretization of the boundary curve. A polygon Q is associated with P such that Q and P have the same number of vertices and a mesh can be easily produced in Q. Since the Schwarz-Christoffel transformation maps an upper half infinite plane onto the interior of a general polygon, two mappings F and G are defined to map the upper half e-plane onto the polygons P and Q. Then the mesh in Q is mapped to P by the composite mapping $H = F \circ G^{-1}$. Figure 2.11 illustrates this strategy.

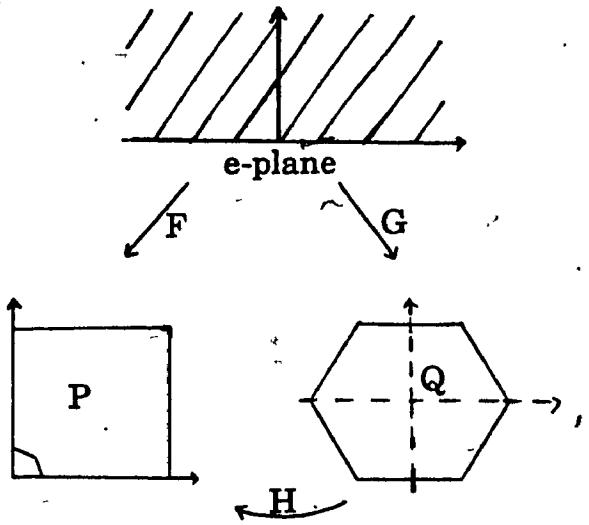


Figure 2.11 Conformal mapping approach

Conformal mapping has an advantage of generating good elements because of the angle-preserving nature of the mapping used. However the scheme is restricted to two-dimensional regions

that are simply connected, and it may not be easy to find the inverse mapping G^{-1} used in the composite mapping H.

Multiply-connected regions must be subdivided into simply connected subregions in order to apply the method. Other mapping scheme can be used, such as in [DEN78] the author uses only one mapping and the finite-element equation assembly to find the mesh in P from an ideal mesh in Q.

2.5 Grid-Based Approach

The idea is to superpose a rectangular grid onto a planar domain and adjust the boundary cuts to obtain the actual mesh. The first published work is due to Thacker & al. [THA80s]. There are two main considerations when using the grid-based approach:

i) Choice of the superposing grid, and

ii) Boundary approximation.

In fact methods belonging to this category differ mainly by these two factors. In [HEI82] the authors use a rectangular mesh geometrically distorted to match the node separations of the polygon bounding region to be meshed (see figure 2.12). Using only the nodes of the superposed mesh which are inside the region, a set of interior triangular elements is formed by appropriate diagonal bisection of the mesh rectangles. This results in one or

more areas of meshed elements isolated from the region boundary by an un-meshed strip. The boundary elements to fill this strip are formed by connecting the sides of the interior mesh elements that are exposed to the un-meshed area to the boundary nodes. The final step is to examine the shapes of the elements, especially those containing boundary nodes. Boundary crossing must also be checked to ensure no invalid elements are included in the final configuration.

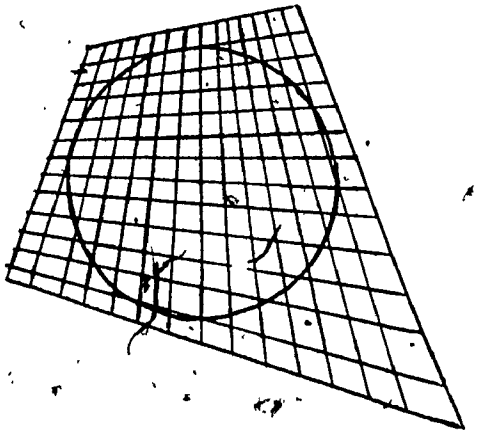


Figure 2.12 Grid-based approach

Another popular scheme pioneered by Yerry and Shephard [YER83,SHE84] uses the quad-tree model as the mesh template. The quad-tree structure of a two-dimensional object corresponds to a set of non-overlapping squares, referred to as quadrants, that are stored in a hierarchical tree [KLI76]. The object to be meshed is placed in a square universe that entirely encloses it. This square represents the root of the tree, and is then subdivided into four

quadrants corresponding to the children of the root. The subdivision is repeated for each quadrant until all quadrants are at some satisfactory level to model the boundary curvature. The next step is to classify each quadrant as being inside, outside, or partially inside the object. Quadrants partially inside the object have their cut points with the boundary joined up to eliminate re-entrant corners of these squares. To allow smooth transition for non-uniform subdivision no two adjacent quadrants can differ in more than one level in the tree. From this stage, quadrilateral or triangular elements, or a mixture of them can be formed. The last adjustment is made on the boundary elements where nodes may be slightly displaced or merged to model the boundary shape. Figure 2.13 gives the quad-tree model of a circle.

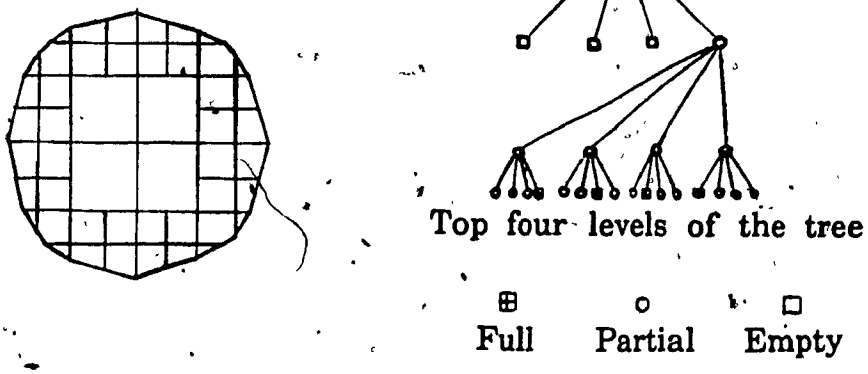


Figure 2.13 Quad-tree representation

The quad-tree approach shows a structural solution to mesh generation and is becoming a popular scheme, improved by several authors, e.g. [BAE87,KEL86]. In fact this method is highly suitable for use in an integrated geometric modelling environment for finite element modelling [SHE85,SHE87].

2.6 Topology Decomposition

The essence of the topology decomposition approach, or local mesh refinement, is to subdivide elements in an existing mesh into smaller elements of the same or different type until the mesh meets a certain density required. Preparation of the initial coarse mesh can be done manually or automatically by some method which performs nodal connection for boundary nodes only, e.g. see figure 2.14.

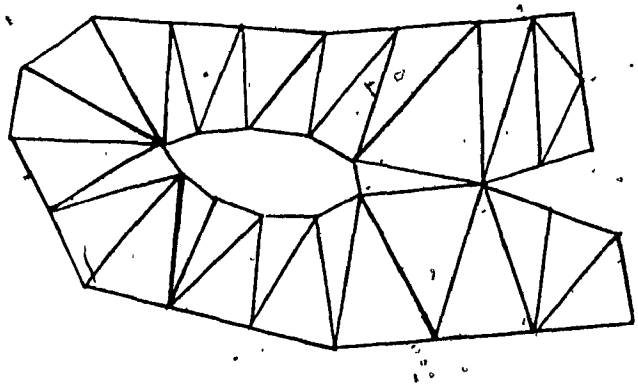


Figure 2.14 Boundary triangulation

The remaining task is how to process the subdivision of each element in the mesh so that new elements have the same type, if a uniform type is required, and the mesh is regular. Obviously no complication arises for rectangular grids where the only possibility is to add two crossing segments that cut the four sides of a rectangle, giving four smaller rectangles. However analytical problem could occur because of inappropriate (non-conforming) neighbouring between elements sharing a common edge (see figure 2.15). There exist different solutions to this problem, for example by modifying the mesh or by making special conditions in the finite element equations [SIM79s].

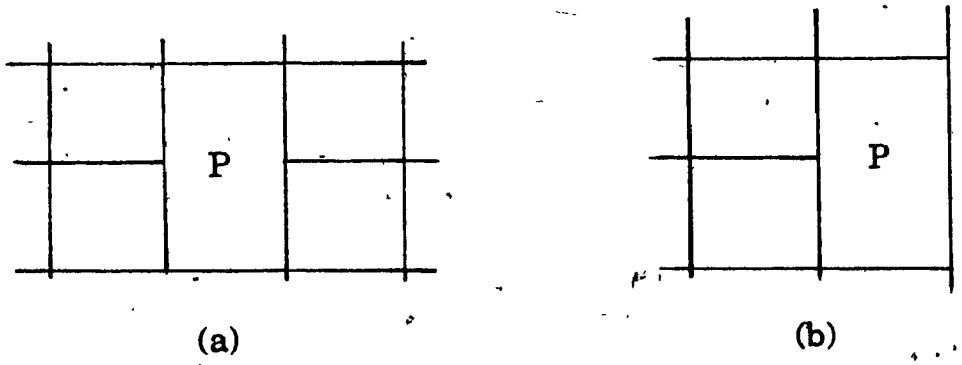
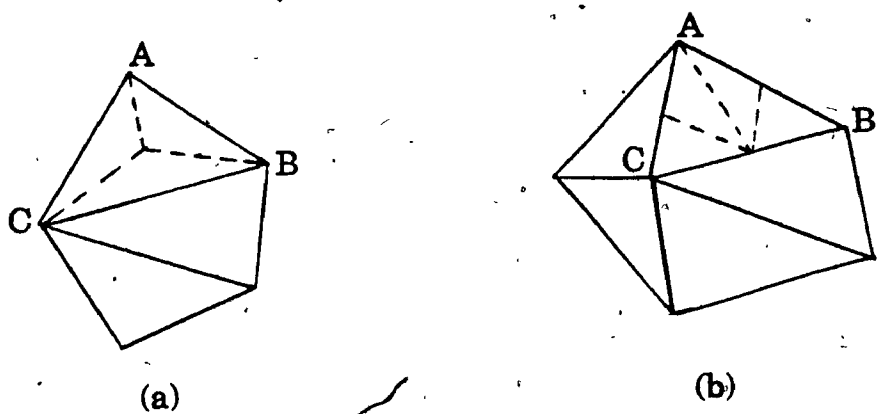


Figure 2.15 Invalid neighbor's configuration for P

Refinement of triangular elements can be done in several ways. The simplest way is to insert a node inside a triangle and connect it to the three vertices to obtain three smaller triangles. The choice of the new node is usually the triangle centroid [KLE80] or alternatively the incentre of the triangle [FRE87].

Rivara [RIV84,RIV87] bisects a triangle by the mid-point of the longest side. Non-conformity occurs if one triangle is bisected along a common edge with another triangle that is not refined (see figure 2.16b). Hence after the individual bisections, a correction procedure must follow; or it may be incorporated at each bisection step.



(a) (b)

Figure 2.16 Triangular refinement: (a) Using the centroid (b) Bisection with non-conformity

2.7 Geometry Decomposition

Most methods in this subclass are designed to deal with simple convex polygonal regions. As a consequence the triangulation must be preceeded by a preliminary subdivision of the input domain into convex parts. The mesh generation procedures are recursive or iterative. In Bykat's method [BYK83] a convex region is subdivided into two halves, also convex. Then nodes are inserted

along the division line as necessary to match the density requirement. The same operations are repeated for each of these subregions until the subregions are triangles and the process terminates. Iterative algorithm works by removing one or two elements at a boundary strip until the region remaining to be meshed is null. In the element removal schemes [BYK76,SAD80], two non-collinear edges are taken and one node is added on each edge such that the two segments with one common endpoint have equal length. If the internal angle α between the two segments is not greater than ϕ for some predefined value ϕ, then only one triangle is removed by joining the other two ends. If α is greater than ϕ, two triangles are removed by choosing a point within the region such that it is equi-distant from the non-common endpoints and the triangles are close to equilateral.

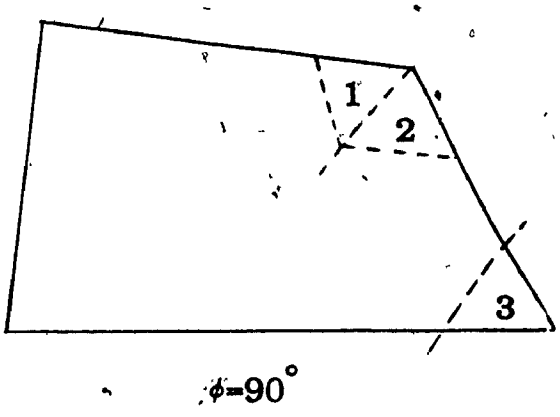


Figure 2.17 Geometry decomposition

Lindholm's method [LIN83] operates on boundary segments by shrinking the boundary of the domain and triangulating the region between the old and new boundaries, one segment at a time. Taniguchi [TAN87] triangulates rectangles by horizontal cutting of the region and forming triangles inside each band subregion.

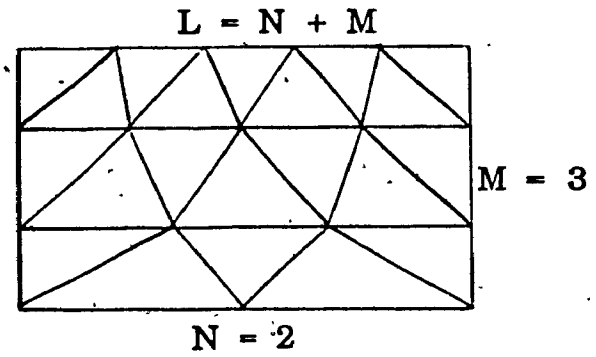


Figure 2.18 Mesh patterns for transition region

2.8 Performance Evaluation

When mesh generation methods are introduced in the finite-element analysis literature, the purpose is to relieve the engineers from the tedious work of manually producing the mesh of the domain of interest. Hence the lack of algorithm analysis in most of published works is easy to understand. And since methods are often described in speaking language rather than algorithmic language, precise comments on their performance are difficult to formulate. In this section, we will present an overview on the space and time complexities reported by some authors or resulted

from our observation for the methods described in previous sections.

2.8.1 Space Complexity

Memory requirement from most mesh generators is linearly proportional to the number of points to be triangulated including both sets of boundary and interior nodes. The data structures involved in the generation are the key factors to determine the space complexity of the methods. Watson [WAT81] claims for N points, $O(N^{(n-1)/n})$ for his n-dimensional triangulation by insertion algorithm hence $O(N^{\frac{1}{2}})$ for two-dimensional case. This figure is evaluated to 16N by Correc and Chapuis [COR87]. There are some other bounds on the space requirement given by different authors for their methods or their implementation of available methods. These upper / lower bounds are summarized in table 2.1.

Table 2.1 Storage requirement of triangulation methods

Sloan [SLO87]	Lawson [LAW77]	Watson [COR87]	Shamos [CLI84]
14N+6	18N	16N	30N

Cline-Renka 1 [COR87]	Cline-Renka 2 [COR87]	Green-Gibson [CLI84]	Correc-Chapuis [COR87]
9N	15N	≥11N	23N

In most of the methods in table 2.1 the core memory needed is used to store list structures. These lists define a tabular form of nodal connectivity and coordinates which is suitable for the algorithm being used.

It should be noted that the above upper / lower bounds are from the methods in the class of node first mesh generation. For the remaining three classes no comments on the space complexity have been mentioned in published works and it is not easy to predict exactly their space requirement.

2.8.2 Time Complexity

No standard time analysis exists for mesh generation methods since these are in a way descriptive and heuristic except for a small number of algorithms that are based on known algorithms such as sorting, searching and merging [LEE80,WAT81]. Our discussion will be restricted to the time claimed or disclaimed by

authors referenced in this review, but when only empirical results are given shall we give our remark or estimate of the time required. It should also be noted that the figures quoted in this presentation are not based on any standard operations since comparisons are done with different criteria hence different set of standard operations.

For the mesh smoothing approach, as well as adapted mesh template methods, the time efficiency is unknown and does not seem to have any interest to their authors. The figures are not easy to estimate since it depends on several factors such as mathematical complexities for methods using mapping techniques, or geometric tests for grid-based methods. Most time analysis can be found for the node connection (vertex triangulation) approach. There exist different figures even for mesh generators which use the same basic scheme. This is the case of the iterative insertion method for triangulation where Lawson gives $O(N^{4/3})$ [LAW77], Lee and Schachter $O(N^2)$ but $O(N^{3/2})$ empirically [LEE80] and observed $O(N^{1.4})$ by Shapiro [SHA81]. Construction of Delaunay triangulation by this method is primarily based on sorting and searching, hence has lower bound O(NlogN) as claimed by Shamos [PRE85] but disagreed by Maus [MAU84] who says that radix sort requires only O(N) time. We summarize the various figures of some iterative insertion algorithms in table 2.2.

Table 2.2 Time complexity of Iterative insertion triangulation

[BOW81]	[CLI84]	[COR87]	[DEF85]
$O(N^{3/2})$	$O(N^{3/2})$	O(NlogN)	$O(N^2)$ - $O(N^3)$

[LAW77]	[LEE80]	[MAU84]	[SLO87]	[WAT81]
$O(N^{4/3})$	$O(N^2)$	$O(N^2)$	$O(N^{5/4})$	$O(N^{3/2})$

The divide-and-conquer algorithm by Lee and Schachter [LEE80] uses O(NlogN) for sort and merge operations. Lewis and Robinson [LEW78] conjecture the same figure for their scheme but this is disclaimed by Lee and Schachter who gives $O(N^2)$ to it. The fully surrounded edges by Suhara and Fukuda and others all require $O(N^2)$ arithmetic operations and scalar comparisons. Lee [LEE84] and Lo [LOH85] provide only a few empirical results but least squares fitting does not give any meaningful figures although Lee claims O(N) for his method.

In the class of algorithms creating nodes and elements simultaneously, Kleinstreuer [KLE80] gives $O(N^2)$ for his mesh refinement method mainly for the initial triangulation of the boundary points. This same process takes $O(N^2)$ and even $O(N^3)$ for multiply-connected regions by the generator in [DEF85].

From the above summary, it is clear that comparison between mesh generation algorithms of different categories is difficult and does not lead to any clear issue on the choice of which method to use as best. The decision is up to the implementor's development philosophy who takes what he thinks most suitable for his needs.

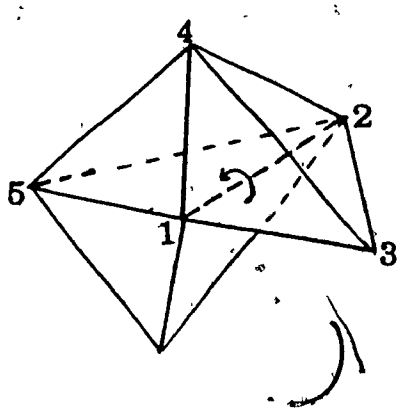
3. Three Dimensional Mesh Generation

While the designs for two-dimensional mesh generator are abundant in the research literature, the same topic in three dimensions reveals to be challenging because of the greatly increasing complexity; hence very few results as compared to 2D are reported. In fact 3D mesh generation is just becoming an active area, while development still goes on for 2D mesh generation. Our review of 3D mesh generation techniques will follow the same classification given in the first section. We shall report on the 3D methods available at hand which are the most representative methods, some being extended from the two-dimensional methods. Not all two-dimensional approaches have their counterpart in three dimensions, such as the mesh smoothing and conformal mapping methods; thus some subclasses will be exempted in the next presentation.

3.1 Node Connection Approach

Watson [WAT78] and Bowyer [BOW78] give similar iterative schemes for n-dimensional triangulation. In three dimensions, Watson's algorithm starts with a tetrahedron T_0 containing all points to be triangulated, and new internal tetrahedra are formed as the points are inserted one at a time. The circumsphere criterion generalized from the circle criterion on two dimensions is used to update the mesh at each insertion. A newly inserted point is tested to determine which circumballs of the existing tetrahedra contain the point. The associated tetrahedra are removed leaving a polyhedron containing the new point. Edges connecting the new point to all triangular faces of the polyhedron's surface are created, defining tetrahedra which fill the polyhedron. The result is a new Delaunay triangulation which includes the new point. Cavendish & al. [CAV85] observe several problems with Watson's approach and apply improvements in their implemented version.

The fully-surrounding method is extended to 3D by Nguyen [NGU82], using the condition that a line (edge) is fully surrounded by an angle of 360° in the space (see figure 2.19). The advancing front method by Lo [LOH85] is found in [LOH87] with the faces replacing the edges in 2D.



[1-2] edge to be surrounded

[1-2-3] base triangle

Figure 2.19 Fully surrounded edge in 3D

Node generation in 3D does not give any additional difficulty, hence is often discarded from discussion. Only the positioning for convenient triangulation is concerned and is resolved differently by implementors to their needs. For example, Cavendish & al [CAV85] generate nodes as in 2D on imaginary planes cutting the object.

3.2 Mapped-Element Approach

The same concept in two dimensions applies for three-dimensional problems. The extension have been studied and implemented by the authors who design the two-dimensional schemes. They replace the quadrilateral template by a cubic template and apply the mapping after subdividing the object to be meshed. Figure 2.20 illustrates Zienkiewicz and Philip's template for three dimensions [ZIE71].

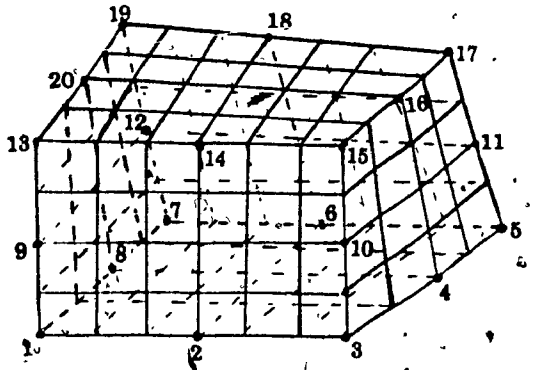


Figure 2.20 Cubic mapped element

Other types of mappings are possible, e.g. blended interpolation used by Cavendish & al [CAV77], discrete transfinite mapping by Haber & al [HAB82,PER82].

3.3 Grid-Based Approach

From all the two dimensional grid-based methods, only the quad-tree method is extended directly to three-dimensional with little changes in the procedure. The quad-tree representation is replaced by the octree equivalence in 3D with quadrants substituted by octants. Again the work focuses on boundary fitting for the object enclosed in the unit cube. However, the process becomes more complicated as the number of possible types of cut increases rapidly (see figure 2.21).

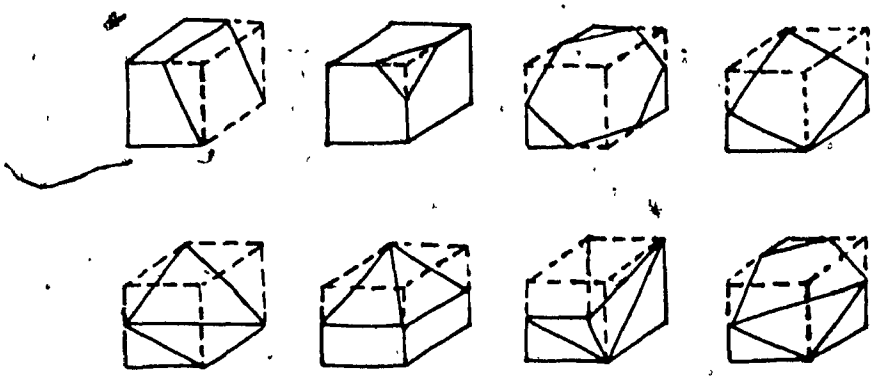


Figure 2.21 Boundary octant cuts

Yerry and Shephard [YER84,YER85,SHE87] allow only a fixed number of possible cutting positions on the sides of an octant which are at the corners and the mid-points of each side. Exact position on the boundary are merged to these cuts after the mesh is created. Kela & al [KEL87] devise similar scheme in their modified-octree mesh generator.

3.4 Topology decomposition

The three-dimensional mesh generator OMEGA [WOR84] using topology decomposition has two algorithmic components, the triangulation and refinement modules. The triangulation divides a 3D polyhedron into tetrahedra after all polygonal faces have been triangulated individually. This is equivalent to triangulation of

simple polygon with no interior nodes in two dimensions. The refinement increases the number of topological entities such as edges and nodes in each simplex, triangle or tetrahedron. The subdivision of a tetrahedron in the refinement may yield tetrahedra and/or polyhedra (see figure 2.22).

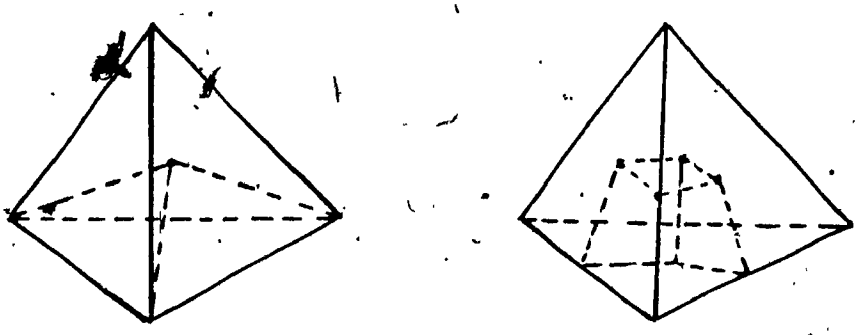


Figure 2.22 Subdivision of a tetrahedron

For the boundary triangulation, Woo and Thomas [WOO84] employ two operators to dig out a tetrahedron from a polyhedron with no holes (see figure 2.23). This is similar to geometry decomposition in two dimensions. The coarse mesh is then refined by subdivision of tetrahedra.

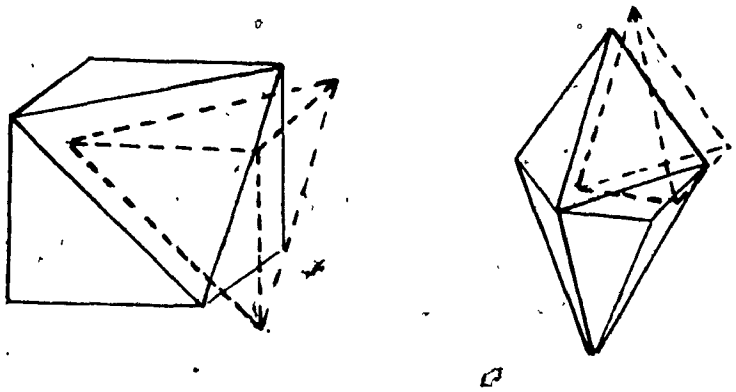


Figure 2.23 Polyhedron cutting

3.5 Geometry Decomposition

Three-dimensional methods in this approach do not operate around the object boundary as in two dimensions, but rather follow a fixed decomposition pattern by cutting the object into smaller objects of similar type, triangulating each of these smaller objects and re-establishing the connection. Imakufu & al. [IMA80] ask the user to decompose the object into blocks of simple geometry such as pentahedral, hexahedral, and the generator takes care of the local mesh generation and inter-block connection. A different scheme by Boubez & al. [BOU86] operates on the cross-sections of an object. Serial cross-sections through an object logically divide it into slices of finite thickness, with two consecutive sections taken to define the top and bottom surfaces of a slice. The slices are processed individually and put back together to form a complete

three-dimensional mesh.

3.6 Performance Evaluation

There is very little that can be said about the time and space complexities of three-dimensional mesh generation algorithms. In fact the main concern of researchers at this early development stage is on how to get a valid 3D mesh, rather than efficiency of the methods. However we can quote a few results available. Watson [WAT81] claims $O(N^{2/3})$ space and $O(N^{5/3})$ time for his algorithm. Boubez & al. report a figure of $0.0325N^2+0.3037N$ least squares fitting from experimental results.

Chapter Three

A TRIANGULAR MESH GENERATOR

INTRODUCTION

Chapter Two presents a complete review of techniques in two-dimensional and three-dimensional mesh generation methods following a logical classification. The four main classes of mesh generation methods are mesh topology first, node first, adapted mesh template, and simultaneous node and element creation. The common objective to all of these classes is to create finite element meshes with minimum user effort; that is, to fully automate the process. Generality shows to be an important factor; by this we mean the ability of treating simply-shaped domain as well as arbitrarily-shaped, simply or multiply connected input domains. The question of efficiency also needs attention of the authors when designing and implementing a mesh generator. Of the four classes, the only general approach is node connection in the class of node first algorithms. It is simple to understand and implement and possesses the following advantages that usually cannot be found in

other generation strategies.

a) No limitation to the domain boundaries so the orientation of the domain does not much affect the resulting mesh. Figure 3.1 shows an example of the effect of domain orientation on the mesh in the quad-tree method.

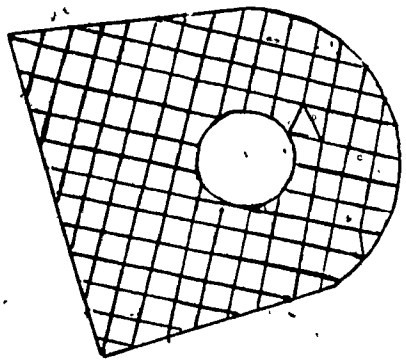


Figure 3.1 Effect of domain orientation on quad-tree representation

b) Openings within the domain can be tackled easily; subdivision of the domain into convex polygons is not required.

c) Ability to maintain the greatest possibility of mesh patterns, i.e. the number of elements around a node is not fixed and the relative positions of the nodes are not predetermined by mathematical formulae.

d) No need to search for any mathematical relation among boundaries, hence less effort and no computational errors are introduced.

The above advantages make node connection approach a general and promising scheme for mesh generation. Therefore we choose to adapt this approach to implement our two-dimensional mesh generator which we shall describe in this chapter. The next sections will present the following topics: description of the methods - original and enhanced versions, details of the implementation of the mesh generator, description of the interactive graphical interface, and finally, time and space efficiency.

1. The Advancing Front Technique for Mesh Generation

The original idea is due to S.H. Lo [LOH85] in his implementation of a two-phase mesh generator. The name "advancing-front" describes the triangulation algorithm's essence. The method has been found to be simple, efficient and have a promising possibility of extension to higher dimensions as well as other properties suitable for finite-element analysis [LOH87]. The main advantage of the method over other popular schemes such as Suhara-Fukuda method, fully-surrounded nodes is the reduction of validation tests of potential new nodes and elements. The triangulation algorithm has some flavour of Cavendish's scheme [CAV74] but node generation is completely different and more efficient.

1.1 Node Generation

Instead of using a superposed square grid, the method works on horizontal lines cutting the domain for which interior nodes are to be added. The horizontal lines are uniformly displaced with the distance determined from a density parameter. Nodes are then added on each line at the same interval spacing. This produces uniform distribution of points within the region bounded by the boundary. For varying density distributions, the domain must be subdivided into a number of regions, each of which having uniform distribution, and meshes are generated independently for each region. However the node distribution on the common boundary separating regions must be exactly the same so that the meshes for the regions when combined is the mesh for the input domain. The algorithm to generate uniformly distributed interior nodes is given below.

algorithm GENERATE—NODES

/* generate uniformly distributed nodes inside a domain given a set of boundary points coordinates (x,y) */

begin

1) Sort out the y_{min} and y_{max} of the domain;
2) Draw imaginary lines between y_{min} and y_{max} at regular interval equal to the average element size of the region;

3) For each of these horizontal lines do

3.1) Determine the intersection of the line with the region boundary and arrange the cut points in ascending magnitude of x-values. The number of cut points is always even;

3.2) The cuts are considered two by two with no repetition, each pair determining a line segment on which nodes are generated at regular distance. Each potential node position must be checked so that it is not too close to the boundary and any given fixed node;

end for;

end.

To determine the intersection of a horizontal line with the region's boundary in step 3.1, we examine the boundary segments for possible cut points. Consider a boundary segment PQ with $x_1 = x(P)$, $y_1 = y(P)$, $x_2 = x(Q)$, $y_2 = y(Q)$ and the horizontal line $y = C$. There is intersection if

(i) $(y_1-C)(y_2-C) < 0$, or

(ii) $(y_1-C)(y_2-C) = 0$ and $(C>y_1$ or $C>y_2)$.

The two cases can be easily derived from figure 3.2.

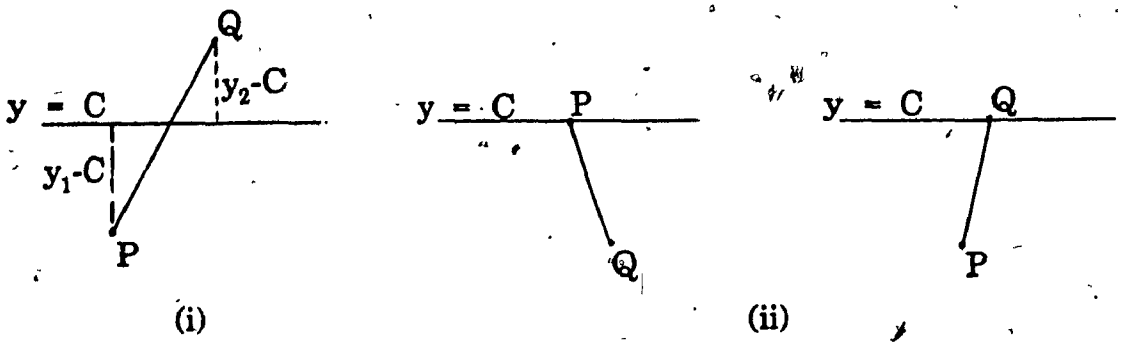


Figure 3.2 Intersection test: (i) intersection within PQ
(ii) intersection at P or Q

The third possibility $(y_1-C)(y_2-C) = 0$ and $(C<y_1$ or $C<y_2)$ needs not be tested in this context since the segments are consecutive; hence this test would capture twice the same intersection point.

The condition in step 3.2 is verified by simply calculating the distances between the potential point and the boundary nodes as well as with the fixed nodes, then comparing it with a constant determined from the density parameter of that region.

Figure 3.3 shows how the algorithm works on multiply-connected region.

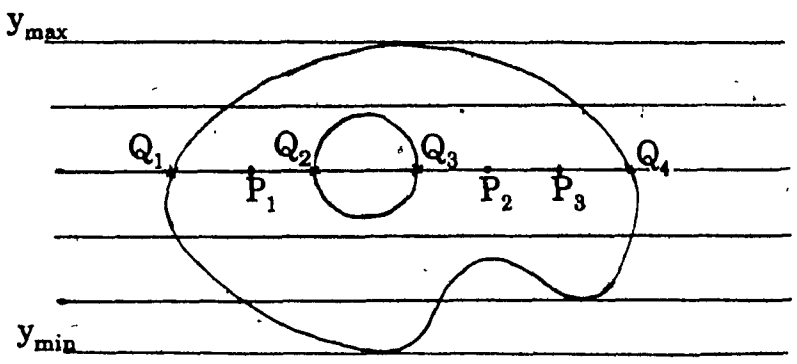


Figure 3.3 Node generation for multiply-connected region

For each region in the domain it is assumed that the spacing of the boundary nodes define the density for that region. Thus no additional points need be generated on the boundary. However it is now the user's responsibility to organize their input nodes so as to reflect the desired density. This task becomes tedious if a fine mesh is required for the analysis.

The following advantages can be extracted from the above node generation algorithm:

i) It is simpler than methods using superposed square grid [SUH72,CAV74] or data scanning method [FRE70];

ii) The node spacings need not be checked every time a node is generated; thus validity checking now reduces to only testing against boundary nodes.

However domain subdivision is still needed since the method only generate uniformly distributed nodes.

1.2 Triangulation

Assuming that the input consists of a set of boundary nodes defining an arbitrary domain and a set of nodes lying inside the domain, we wish to establish the interconnection between the nodes to form a triangular mesh of the given domain. The first concern of any triangulation algorithm is to ensure that no triangular elements are formed outside the specified domain. Thus some checking must be performed for this purpose. To simplify the task and make a clear distinction between the domain to be triangulated and its outside, we assume that a counterclockwise order is used for the nodes on the external boundary that encloses the whole domain and a clockwise order is used for all the nodes on the internal boundaries. By virtue of this ordering, the domain to be meshed has an interior area always situated to the left of the line segments connecting any two consecutive boundary nodes. Therefore it is easy to verify the position of element or node with respect to the domain boundary. The advancing-front method for triangulation then works as follows. The idea is to form triangles around the boundary and discard all interior edges that are shared by two recorded triangles and all boundary edges that belong to

any recorded triangle. This is exactly the fully-surrounded edge concept as described in chapter two. The triangulation usually works from the external boundary inwards, thus continuously reducing the region to be meshed (see figure 3.4). At any time during the process, the region to be meshed comprises a set of free nodes (nodes that have not been used in any recorded triangles) and those partially surrounded edges which form the boundary of the region. The triangulated region between the original boundary and the region to be meshed is ignored completely in subsequent examination. We call the boundary of the unmeshed region the generation front of the triangulation process.

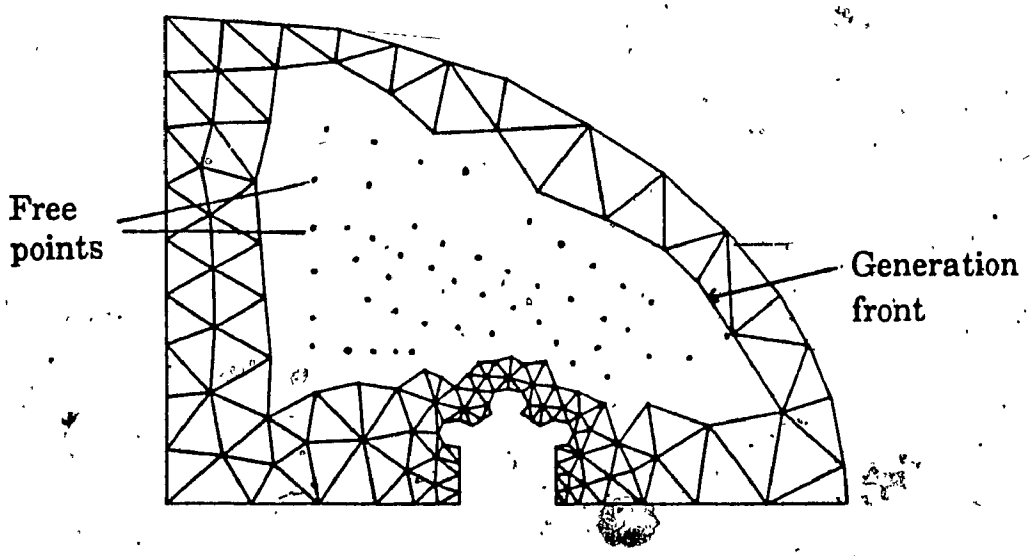


Figure 3.4 Advancing front triangulation

In brief, we define two dynamic sets for the method. The generation front is a set of interior edges belonging to only one triangle and free boundary edges not being part of any triangle.

The set of free nodes includes all interior nodes which have not been selected to form triangles. As the process goes on these two sets change every time a new element is recorded and eventually become empty when the mesh is completed. The generation front is initialized with the boundary segments and the set of free points includes all the interior points. A triangle base AB is selected from the segments on the front, usually the last one that was added. An apex node C is selected from the set of free points and the front nodes different from nodes A and B. The selection is made such that node C lies to the left of vector AB and triangle ABC is optimal as compared to all other possible choices of C. Once a triangle ABC is recorded, the base AB is removed from the front since it becomes fully surrounded and either AC or CB is chosen to be the new base if it was not already on the front. Note that any edge that belongs to two triangles is cleaned up from the unmeshed region at every such iteration. If the selected node C was a free node then it will be deleted from the set of available nodes and added to the generation front. This type of updating both the front and the set of free nodes guarantees that the termination condition will be met at some point in the triangulation; thus the algorithm must terminate.

The main difference and advantage of the advancing front method as compared to other fully-surrounded edge algorithm is the dynamic changes of the unmeshed region's boundary and the set of points involved in the element selection procedure. At the beginning of the triangulation, the generation front is exactly the domain boundaries, external and internal, and the set of free points comprises all the interior node points. While the input domain boundary remains unchanged throughout the process the generation front which defines the actual domain to be meshed changes dynamically and testing for domain crossing element need only be done against this new boundary. Also the checking of the node points lying in the meshed region so as to select a new apex node of a triangle is completely eliminated. These are two main reductions in the time complexity of the triangulation process. A complete algorithm interpreting the advancing-front technique is given below and an illustration can be found in figure 3.6.

algorithm ADVANCING—FRONT

/* Generate a triangular mesh using the advancing-front technique. The input, output and notation are as follows.

Input: i) A set of boundary nodes B, in counterclockwise order for external boundary and clockwise order for internal boundaries;

ii) A set of interior node points I.

Output: A triangular mesh.

Notation: F: generation front

P: set of free nodes

AB: directed line segment (vector)

*/

begin

1) Set F ← B;

P ← I;

2) While F and P are not empty do

2.1) Set AB ← last segment of F;

T ← F ∪ P;

2.2) Select a suitable point C ∈ T so that triangle ABC is optimal among other C_i in T;

2.3) If AC and/or CB intersect any segment of the generation front then

2.3.1) T ← T - { C };

2.3.2) Repeat from step 2.2

end if;

2.4) Record triangle ABC;

2.5) Update F and P

end while

end.

The criterion for selecting the apex node C in step 2.2 as described in [LOH85] relies on the even distribution of the interior nodes generated by the method given in section 1.1. It states that the consideration of the minimum value of the norm $[AB^2+BC^2]$ is sufficient to determine the point C, ensuring the best triangulation obtainable from the system of the interior nodes and the boundary nodes. However this criterion may not be sufficient to guarantee the best triangulation for irregular boundaries. The following test must be conducted after applying the minimum norm criterion. Select two nodes that are closest to the given base AB using the minimum norm criterion, say C_1 and C_2, such that the areas of triangles ABC_1 and ABC_2 are positive, i.e. C_1 and C_2 reside on the left hand side of vector AB. Then the following quantities, called the quality parameters are computed for each potential apex node C_i, i = 1, 2.

$$\alpha_1 = \frac{\text{area}(ABC_1)}{AB^2+C_1B^2+C_1A^2}$$

$$\beta_1 = \frac{\text{area}(C_1BC_2)}{C_1B^2+C_2B^2+C_1C_2^2}$$

$$\delta_1 = \frac{\text{area}(AC_1C_2)}{AC_1^2+C_1C_2^2+C_2A^2}$$

$$\lambda_1 = \max (\beta_1, \delta_1)$$

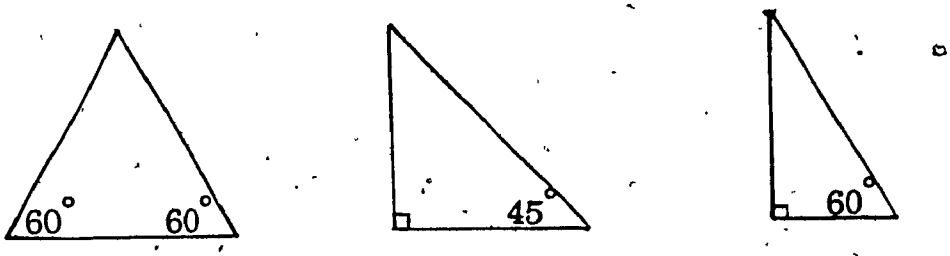
$$\alpha_2 = \frac{\text{area}(ABC_2)}{AB^2+C_2B^2+C_2A^2}$$

$$\beta_2 = -\beta_1$$

$$\delta_2 = -\delta_1$$

$$\lambda_2 = \max (\beta_2, \delta_2)$$

The selected node C is the node which has the larger value $\alpha\lambda$. The quantity α_i measures the quality of potential triangles ABC_i, i = 1, 2. The bigger the α value the better is the shape of the triangle (see figure 3.5). This value must also be positive to keep the counterclockwise ordering of triangle corners. The parameter λ is used to judge the quality of the triangles that could be formed as a consequence of the choice of C_1 or C_2.



α = 0.1443 α = 0.125 α = 0.1083

Figure 3.5 Triangle's quality

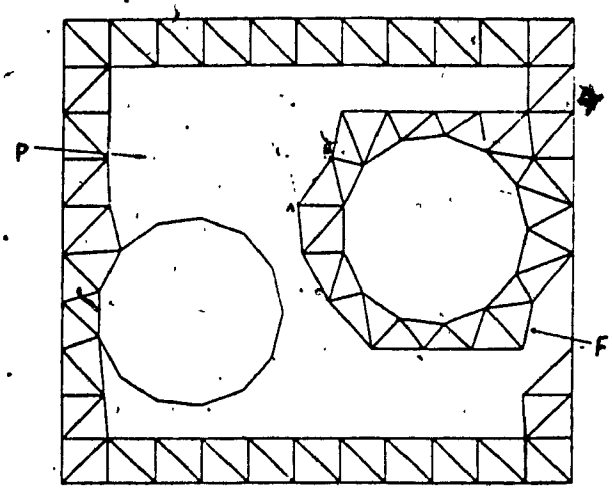


Figure 3.6 Intermediate stage of triangulation: node C is chosen to form $\triangle ABC$

Although the method already eliminates a considerable amount of time for element crossing verification, the time required to select a suitable point for every element is quite large, especially if the nodes have a dense distribution and we must compute the norm AC^2+BC^2 for each free node and front node. Can we determine a minimum number of nodes to compare based on their closeness to the generation front thus save processing time? The answer is "Yes". We now propose an improvement on the described method of mesh generation which gives an estimate time saving proportional to the node distribution inside the given domain.

2. Mesh Generation by Layers

Most of triangulation methods require some verifications before a potential element is accepted as a new element of the mesh. This test is necessary to ensure the validity of the mesh being constructed at any time during the process. In fact these checks are unavoidable for any triangulation method based on the node / edge surrounding scheme. In the advancing-front method, validation is carried out at the selection of new elements: each potential node must be examined using the minimum norm criterion then further comparison between the most suitable two nodes to take the best one. We propose an enhanced version of the above method from the following observations:

a) There are quite a few points that need not be checked at all if they are too "far" from the base vector, in other words, there may be one or more points falling inside the resulting triangles.

b) In fact the points to be tested are those which are closest to the generation front.

Suppose that the input domain is simply connected, i.e. it has no internal holes. The advancing front then works from the boundary inwards and our model looks like a continuously

shrinking unmeshed domain. Since the element's base vector is always taken from the boundary (front) the points of interest are those that are closest to the boundary. To identify these points we need an ordering scheme so that this can be done quickly otherwise we are just adding more processing time to the triangulation. This ordering of the generated points can be incorporated directly into the node generation phase as we will see in the next section. The new triangulation scheme that follows will be easy to catch.

2.1 Generation of Interior Node Points

Points are now generated by groups called layers. Layers are defined from the external boundary inwards; the first layer is exactly the external boundary. The next layer is obtained by scanning the previous layer to produce new segments at some distance from that layer. This distance between layers obviously must reflect the mesh density required. Since the next layer is always adapted from the previous one and the generation moves from the external boundary toward the domain center, the "plain" region for new points becomes smaller and smaller and eventually will be null, which marks the end of the generation process. In practice the generation can terminate as soon as the last layer has only one segment or it encloses a triangle which has area

smaller than some tolerance calculated from the mesh density parameters.

Using this method of calculating new nodes a discrete representation of the boundary would make it easier to implement the algorithm as well as inputting data. By virtue of the discrete representation each layer is equivalent to a set of line segments defining a closed curve. The segments do not have uniform length and more points are added between the two endpoints if the segment length is greater than twice the required density.

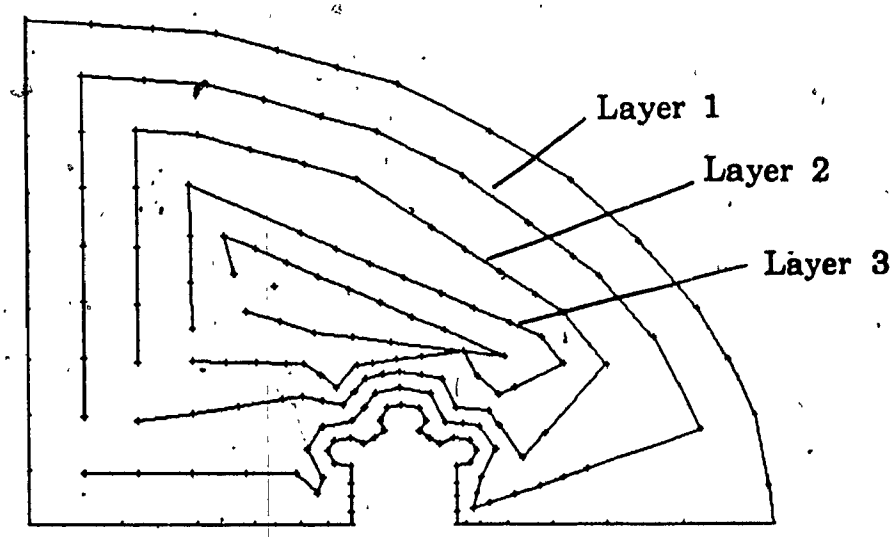


Figure 3.7 Generation of points by layers

The following algorithm expresses the above idea in a more systematic view. We assume a discretized boundary which is a set of consecutive line segments.

algorithm GENERATE—NODES—BY—LAYER

/* Generate nodes inside a domain given a set of boundary segments

Notation: #(L) ... number of segments in layer L

area(L) ... area enclosed by layer L */

begin

1) Current layer ← external boundary

New layer ← { }

2) For i ← 1 to #(Current layer) do

2.1) Determine the distance d between two layers

2.2) Compute the image segment i′ at a distance d from segment i

2.3) If segment i′ is invalid then next i

2.4) New layer ← New layer + {segment i′}

end for

3) Current ← New layer

4) If area(Current layer) ≥ Tolerance repeat from step 2

end

The distance between two layers may vary for each segment if the mesh density is not uniform throughout the domain. Hence this distance must be determined for each segment of the current layer. The image segment can be defined by calculating the two new endpoints such that they are at the computed distance from the current segment endpoints while respecting the shape of the

current layer. Computing the intersection of two lines is one way to find an image points.

The validity of a generated segment depends on the following factors:

a) The segment is completely or partially inside the region enclosed by the current layer. If it is partially inside the region, only the sub-segments which lie commpletely inside are taken for node generation.

b) The length of the segment cannot be smaller than the density associated with the region it happens to be in.

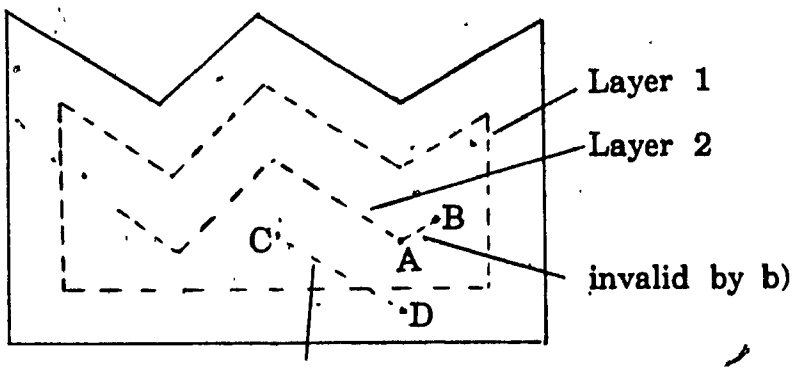


Figure 3.8 Segment Validation

To prove the correctness of the algorithm, we first observe that the algorithm is guaranteed to terminate because of the decrease in size of the region enclosed in the last generated layer. It obviously generates points by layer and all these points reside

inside the domain. Thus the algorithm is valid and perfectly suitable to generate node points inside a given domain.

2.2 Triangulation

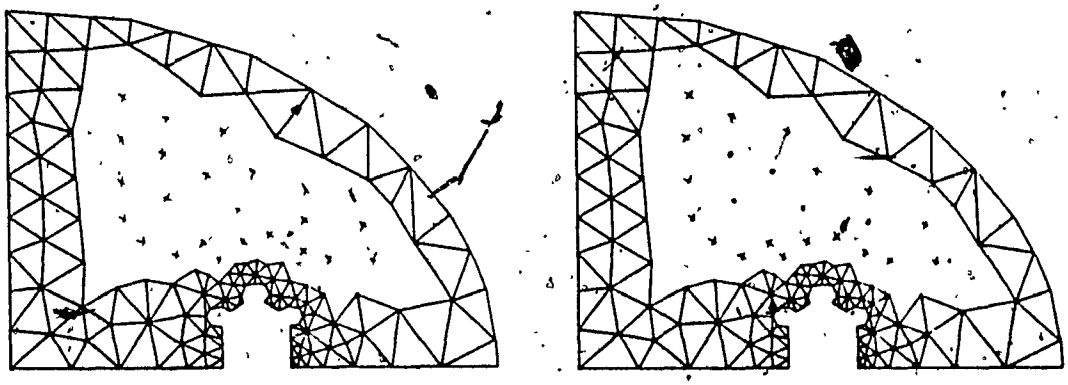
We use the advancing front technique with fully surrounded edges for our triangulation module. The essence of the advancing front is reflected from the definition of two dynamically changing sets for which we shall keep the same names: the generation front and the set of free nodes. To recapture briefly: the generation front comprises all edges which are adjacent to the unmeshed region (the whole input domain at the starting time of the process); the set of free points contains all points inside the unmeshed region, that is those points which have not been used in any recorded triangular elements. In the context of fully surrounded edges, any interior edge must be part of exactly two elements and any boundary edge can belong to exactly one element. It can be easily seen that, for our advancing front model, an edge is discarded if it satisfies the following two conditions:

i) It is on the generation front, and

ii) It is used as a based vector to form a new element.

Two consecutive discarded edges resume in the removal of the common endpoint from subsequent consideration. Thus when all points have been removed the triangulation is complete.

In our enhanced version we keep the same contents of the generation front; however the set of free nodes is narrowed to contain only potential nodes lying approximately closest to the generation front. If we classify the nodes by layers, then at any time in the process the set of free nodes covers only a few layers of points instead of the whole set of interior nodes. The verification time that is saved by using this definition becomes quite significant when there is a large number of interior nodes involved.



(a) (b)

Figure 3.9 The set of nodes to test

(a) Advancing front (b) Enhanced version

To keep the front movement uniform, that is, always from the external boundary inwards, we initialize the generation front by the external domain boundary and let internal boundary nodes be part of the interior nodes with some special consideration. The new algorithm is given below.

algorithm NEW—ADVANCING—FRONT

/* Generate a triangular mesh using the enhanced advancing-front technique. The input, output and notation are as follows.

Input: i) A set of boundary nodes $B = E \cup H$, in counterclockwise order for external boundary E and clockwise order for internal boundaries H;

ii) A set of interior node points grouped into layers L_i, i = 1, ... , n.

Output: A triangular mesh.

Notation: F: generation front

P: set of candidate free nodes

AB: directed line segment (vector)

l: current layer

*/

begin

1) Set F ← E;

P ← ∅;

l ← 0;

AB ← last segment on F;

2) While F and P are not empty do

2.1) If P is empty and l<n then $P \leftarrow L_{l \leftarrow l+1}$ end if;

2.2) T ← F ∪ P;

2.3) Select a suitable point C ∈ T so that
triangle ABC is optimal among other C_i in T;

2.4) If AC and/or CB intersect any segment of F then

2.4.1) T ← T - { C };

2.4.2) Repeat from step 2.3

end if;

2.5) If there is any point in L_{l+1} falling inside
triangle ABC then

2.5.1) P ← P ∪ $L_{l \leftarrow l+1}$;

2.5.2) T ← T ∪ L_l;

2.5.3) Repeat from step 2.3

end if;

2.6) Record triangle ABC;

2.7) Update F and P;

2.8) Set AB ← AC or CB or last segment on F

end while

end.

The algorithm is similar to the one given in previous section. The difference is in the definition of P and how the process continues after recording an element by not choosing any front edge, but one from the newly formed element to be the new base vector. Step 2.5 can be omitted for regions of simple shape, but highly re-entrant regions might cause some non-valid element to be

formed since the layers represent an approximate replicate of the domain boundary. However this probability is small and does not affect much of the processing time. Finally we also reduce the testing time by using a different but, of equivalent effect, criterion to select a suitable apex node of a triangle. We need only examine the apex angle $\widehat{ACB}$ and take point C which maximize this angle. This condition guarantees both the minimum norm and good element shape and yet simpler to perform.

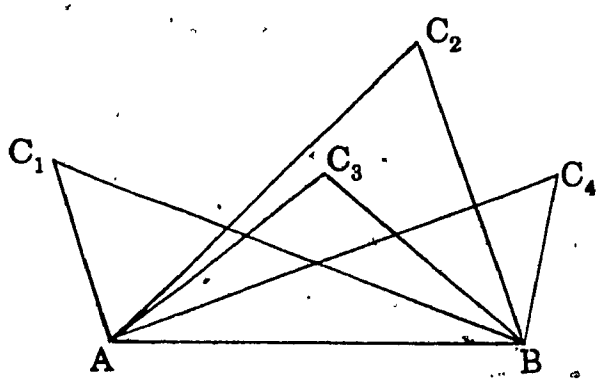


Figure 3.10 Maximum $\widehat{ACB}$ test

C_3 is most suitable

In the next sections we present the details of the implementation of a mesh generator using the described method. This includes description of the node generation module, the triangulator, the smoothing module, the interactive user interface by computer graphics, and performances.

3. Implementation of a Two-Dimensional Mesh Generator

The mesh generator described in this section is written in Fortran and runs on a VAX workstation. The design and implementation of the generator itself have passed through several selections and trials of various methods in [CAV74,NEL78,LOH85] before arriving at the final version which shows to be a net improvement over the implementation of the tested schemes. The program structure consists mainly of three modules each having independent tasks: node generation, triangulation, and smoothing. The execution of these modules follows the same logical order, with output from one module being fed back to the next module as input. Input data for the first module are user-defined and output from the last module is a triangular mesh of the input region satisfying certain analysis and density criteria. Thus our main program simply contains calls to the modules for each problem definition read in.

```
program MESH—GENERATOR
/* Generate triangular meshes */
begin
Repeat
 1) Read domain boundary B and density D for region R;
 2) Call NODE—GENERATOR using (B,D) giving interior nodes
```

I;

3) Call TRIANGULATOR using (B,I) giving mesh M;

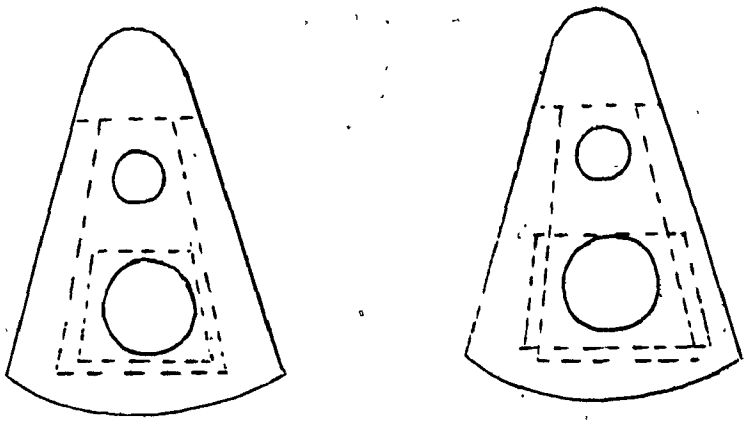
4) Call SMOOTH using M giving mesh M';

5) Output M'

until end-of-input

end.

The boundary data are the (x,y)-coordinates of the boundary points in counterclockwise order for external boundary and clockwise order for internal boundary. The density parameters are defined by partitionning the domain into sub-regions, each has a density defined by giving desired inter-nodal distance within the sub-region. These sub-regions are either disjoint or one may contain several others but partial inclusion, i.e. $R_i \cap R_j \neq \emptyset$ and $R_i \not\subset R_j$ and $R_j \not\subset R_i$, is not allowed, and the sub-region boundaries are input all in counterclockwise order of the node coordinates.



(a) (b)

Figure 3.11 Domain subdivision

(a) Valid definition (b) Invalid definition

The next sub-sections describe in order the node generator module, the triangulator nodule, the smoothing module, and the data structures of various entities used in the program.

3.1 Node Generation

Given a planar domain and its subdivsion's density, the node generator produces new nodal position on the boundary and inside the domain to satisfy the required density. In view of the new advancing front triangulation, nodes are generated by layers. The idea was described informally in previous section and we now present all details concerning the actual implementation and give more realistic algorithms, starting with the main one and then related problems whose solutions are not trivial.

algorithm NODE—GENERATOR

/* Generate new node points.

Input: i) A set of external boundary nodes B in counterclockwise order,

ii) A set of internal bounary nodes β in clockwise order,

iii) Definition of subregions R_1, R_2, ... , R_k and associated densities d_1, d_2, ... , d_k.

Output: Set of boundary and interior nodes satisfying the required density.

*/

begin

1) Add new boundary points to the external and internal boundary;

2) Eliminate collinear points in B and β giving working sets B′ and β';

3) Let $L = \{l_1,...,l_p\}$ be the current layer, and L′ be the layer being generated.

$L \leftarrow B'$;

4) While #(L)≥3 do

4.1) $L' \leftarrow \emptyset$;

4.2) For i←1 to #(L) do

4.2.1) Find the image l_i' of l_i;

4.2.2) If l_i' is valid then add it to L'

end for;

4.3) L ← L';

4.4) Generate points on layer L

end while

end.

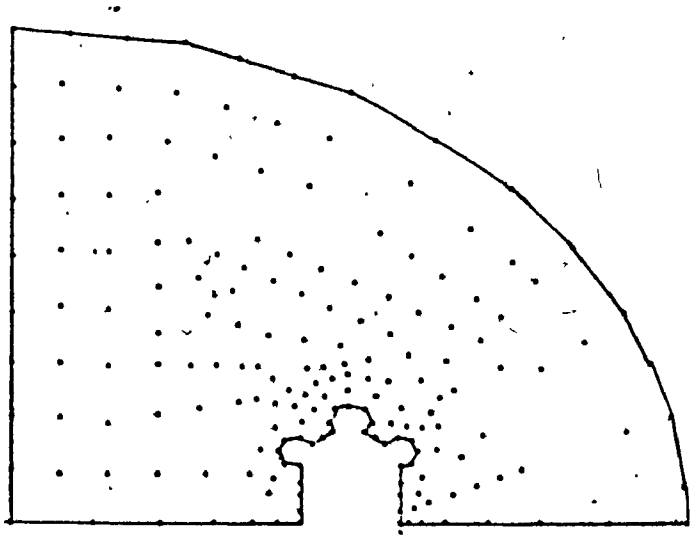


Figure 3.12 Points generated by NODE—GENERATOR

There are several steps in the above algorithm that need be elaborated on. We give them in the following list for easy capture.

i) Addition of new points on a polygonal boundary curve (step 1).

ii) Eliminate collinear points on a boundary curve (step 2). This extra pre-processing is not necessary, but since the boundary is used repeatedly for validity tests and the processing time is proprotional to the number of points, a pre-processing step to

reduce the number of points to minimum is strongly recommended.

iii) Find the image l_i' of a point l_i requires the knowledge of the region it is in so that the distance between the old and image points can be determined from the density parameter to compute l_i' (step 4.2.1).

iv) A point is valid (step 4.2.2) if it lies inside the domain and it is not too close to previously generated layer. Closeness can be verified by comparing the distances within an acceptable tolerance.

v) Generation of additional points on a layer given only the non-collinear endpoints of layer segments (step 4.4).

The problems (i) and (v) are similar in the sense that we compute new points on each segment of a polygonal boundary (layer). The main difference is how to deal with boundary cutting segment which might be present for layers within a multiply-connected domain. Problems (iii) and (iv) both require the In/Out test of a point in an arbitrary region. We describe these algorithms in the next sub-sections.

3.1.1 Locating a point inside an arbitrary multiply-connected region

This operation represents the most time consuming part in our node generation algorithm. Given a point located somewhere in the plane and a domain, determine the position of the point about the

domain, i.e. it is inside or outside or on the domain boundary. It is easy to see that if all the boundary segments are ordered counterclockwise, a point residing inside the domain must be to the left of all such directed segments of the bounadry. Thus the problem could be solved by simply checking the position of the point with respect to each boundary vector and an outer point will be to the right hand side of at least one boundary vector. The test would work perfectly if the domain is convex or comparison is made against the closest vector which contains the projection of the point on the line segment. This breaks the process into several cases to be able to deal with arbitrary regions. We use a simpler method in our node generator module which adapts to general situations.

Given a boundary curve and a point $P(P_x,P_y)$, draw an horizontal half-line $y = P_y$ starting at $x = P_x$ and growing to the positive direction, then count the number of cut points the half-line makes with the boundary. If this number is odd then P is inside or on the boundary curve, otherwise it is outside. For a multiply connected region P is in the domain if it is inside the external curve but outside of all internal curves.

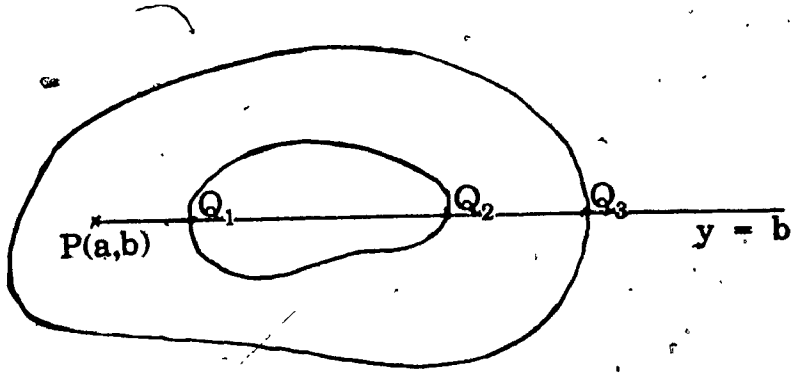


Figure 3.13 Locating a point in a region

The algorithm for simply connected regions is call SIMPLE—REGION. For multiply connected regions the algorithm MULTREGION should be applied.

algorithm SIMPLE—REGION (R, P)

/* Returns true if P is inside or on the boundary of region R, false otherwise. */

begin

1) Initialize n ← 0; 2) For all pairs of consecutive points of boundary curve of R do

2.1) Let (x_1,y_1) and (x_2,y_2) be the coordinates of the two points, compute

$$T = (y_1-P_y)(y_2-P_y);$$

2.2) If (T>0) or (T=0 and $P_y \leq y_1$ and $P_y \leq y_2$) then

there is no intersection

else if the intersection point is not duplicate

n←n+1

end if

end do;

3) If n is odd return true else return false

end.

algorithm MULTREGION (Ω, P)

/* Returns true if P is inside the region Ω, false otherwise.

Let $\partial\Omega$ represent the boundary curve(s) of the region,

$\partial\Omega = \bigcup_{i=1} \partial\Omega_i$ where $\partial\Omega_n$ is the external boundary.

*/

begin

1) If SIMPLE—REGION (Ω_n,P) = false return false;

2) Found ← true;

i ← 1;

3) While i<n and not Found do

3.1) Found ← SIMPLE—REGION($\partial\Omega_i$,P);

3.2) i ← i+1

end while;

4) return (Found)

end.

3.1.2 Determination of the smallest region containing a point

Since regions with different density parameters can overlap in the sense that one may contain several others, the smallest region in which a point P' lies must be found in order to have the associated density value. This value is used later to compute an image point of P by intersecting the two lines parallel to the lines intersecting at P.

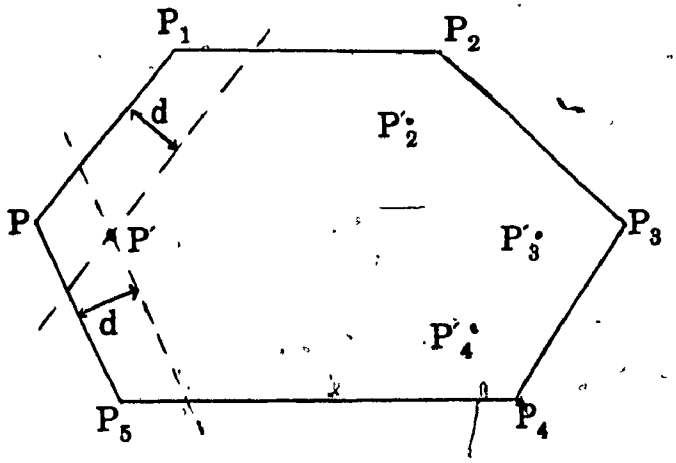


Figure 3.14 Finding image P' of a point P

To accomplish this task we first need the enclosure information of the sub-regions, i.e. each region contains which regions. This information can be supplied by the user along with the domain subdivision or can be automatically determined using the next algorithm.

algorithm ENCLOSE (R)

/* Determines the immediate enclosure ENC(i) for each region sub-region R_i, i=1,2,...,n of R. */

begin

```
/* Step 1: Determine enclosure ENC(i) for each R_i */
 1) For i←1 to n do
    1.1) ENC(i) ← ∅;
    1.2) For j←1 to n do
            If R_j is in R_i then
               ENC(i) ← ENC(i) ∪ {j}
            end if
    end for (j)
 end for (i);
/* Step 2: Determine immediate enclosure from ENC(i) */
 2) For i←1 to n do
    2.1) M ← #(ENC(i));
    2.2) While M>0 do
         2.2.1) j ← M^th element of ENC(i);
         2.2.2) if J is in the enclosure of some k∈ENC(i)
                then
                   ENC(i) ← ENC(i) - {j}
                end if;
         2.2.3) M ← M - 1
    end for;
 3) return (ENC)
end.
```

Step 1 determines all the sub-regions internal to each region. Step 2 refines the result by eliminating all multiple levels of enclosure, that is retaining only the sub-regions immediately contained in each region. Figure 3.15 clarifies this idea.

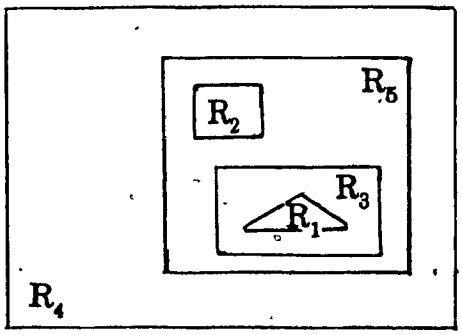


Figure 3.15 Region enclosures

The regions R_1, R_2, and R_3 all reside inside R_5 but only R_2 and R_3 are "immediately enclosed" in R_5, i.e. ENC(5) = {2,3}. The actual result can be viewed as a tree structure in which the children of each node are the immediately enclosed sub-regions and the root is the external region. The smallest sub-region containing a point can be determined by traversing this tree from the root node down to the appropriate leaf node.

3.1.3 Generating points on a line segment

Given two endpoints of a line segment, we wish to compute the position of n points, for some value n, lying on that segment. In the context of node generation the number of points n or

equivalently, the distance between two consecutive points is determined from the density parameters associated with the two endpoints. Let J_1 and J_2 be two segment endpoints with associated densities d_1 and d_2 respectively. Their coordinates are (x_1,y_1) and (x_2,y_2). We have two cases:

Case 1: $d_1 = d_2 = d$

Uniform distances between two consecutive points are used. The number of intervals and the x,y displacements are respectively,

$$n = \lfloor \frac{|J_1J_2|}{d} \rfloor + 1$$

$$d_x = \frac{x_2 - x_1}{n - 1}$$

$$d_y = \frac{y_2 - y_1}{n - 1}$$

Case 2: $d_1 \neq d_2$

Assuming without loss of generality that $d_1<d_2$, the points are placed at increasing distances from J_1 toward J_2. The number of intervals and the displacement of the i^{th} point with respect to J_1 are

$$n = \lfloor \frac{2|J_1J_2|}{d_1+d_2} \rfloor + 1$$

$$q_x^i = d_x + \frac{2(i-1)}{n-1}(\frac{x_2-x_1}{n} - d_x)$$

$$q_y^i = d_y + \frac{2(i-1)}{n-1}\left(\frac{y_2-y_1}{n}d_y\right)$$

where $d_x = d_1\cos\alpha$, $d_y = d_1\sin\alpha$, and α is the slope of the line passing through J_1 and J_2.

3.1.4 Generating points on a layer

We use the above calculations to add new points on each segment of a layer. A problem arises when we have holes inside the domain: the segments may cut these hole boundaries by a number of intersections (see figure 3.16a). In this case the points on the sub-segments outside the domain need not be computed at all. Our solution is to determine the cut points, and consequently the sub-segments, and then generate points on those sub-segments internal to the domain only (see figure 3.16b). If all the cut points are ordered, then these intervals are defined by pairs of consecutive points with no repetition, starting at one segment endpoint and ending at the other. This will eliminate the In/Out test for each point generated.

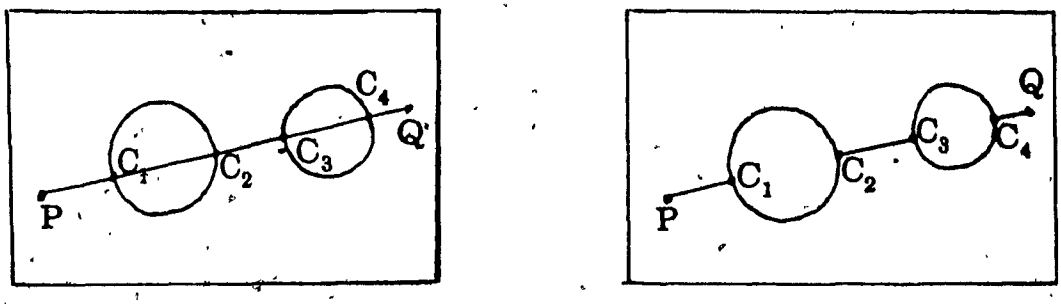


Figure 3.16 Generating points on a layer segment
(a) Partially inside segment (b) Valid sub-segments

We have given a complete description of our node generation module in the mesh generator. The next section will report on the triangulator which uses the points produced in the above fashion to form a triangular mesh for the input domain.

3.2 The Two-Dimensional Triangulator

To establish the connection between node points to form triangular elements, our triangulation module uses exactly the enhanced advancing front scheme described in section 2.2. The input is a set of boundary nodes and layers of interior nodes. The triangulation works from the external boundary inwards, with all generated elements residing behind the generation front. A base vector is chosen from the front at each iteration and a point is

selected from the front and the closest layer to give new triangle. The front and set of free nodes are then updated and the process is repeated until the front is null and there are no more free points.

At the implementation time we are faced with the question of internal boundary nodes: Do we include the internal boundary in the front at initialization time or leave them as interior free points? The consequences of each choice are as follows:

i) If the internal boundaries are part of the initial generation front, that is those edges that are considered as already used once, then we have a generation front composed of several closed curves and a base vector must be carefully chosen so that no illegal connection can occur. This is in fact a data structures design and manipulation problem.

ii) If the internal boundary points are taken as interior nodes, we need to remember that an edge formed by two such points cannot be re-used as a base vector in subsequent generation. This is an algorithm design problem.

We have chosen to employ the second approach in our triangulator to make it easier to implement and understand. In the algorithm NEW—ADVANCING—FRONT (section 2.2), each time a triangle ABC is recorded, the generation front and the set of free points are updated according to the following four cases: either

the point C is a free node or it is already on the front and there are three possible configurations.

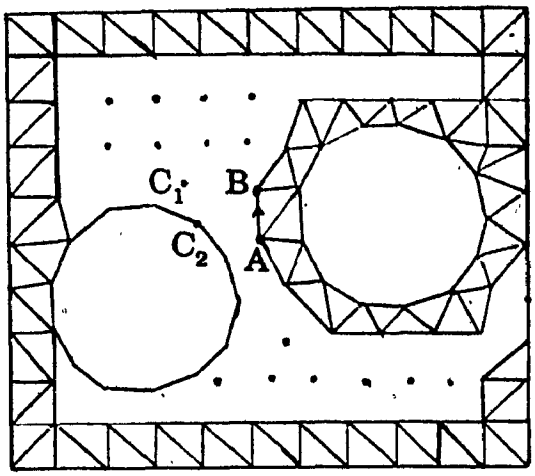


Figure 3.17 Apex node selection

C_1 is free point, C_2 is front point

By incorporating the fully-surrounding idea, we propose the treatments for each of these cases as follows. Assuming P is the set of free nodes to be tested, F is the generation front, AB is the base vector and C is the selected point.

Case 1: C belongs to P, remove C from P and add edges AC and CB to F, take either vector AC or CB as the new base.

Case 2: Edges AC and CB are already on the front, remove A, B, C from F, choose a new base from F if any.

Case 3: Edge AC (CB) belongs to F, remove A (B) from F, the

new base is CB (AC).

Case 4: Edges AC and CB are not in F but C is, introduce C once more in F resulting in a disconnected front, take AC or CB as the new base, and stack the other not chosen edge for later processing of the disconnected part.

The algortihm TRIANGULATOR presented below is a detailed sequential instructions for the above method. The choice of the second approach to treat internal boundary nodes as interior nodes requires the termination condition to be modified: either the front is null or it is precisely an internal boundary.

algorithm TRIANGULATOR

/* Generate a triangular mesh using the enhanced advancing-front technique. The input, output and notation are as follows.

Input: i) A set of boundary nodes $B = E \cup H$, in counterclockwise order for external boundary E and clockwise order for internal boundaries H;

ii) A set of interior node points grouped into layers L_i, $i = 1, \ldots, n$.

Output: A triangular mesh.

Variables: F: generation front

P: set of candidate free nodes

AB: directed line segment (vector)

l: current layer

*/

begin

1) Initialization: $F \leftarrow E$;

$P \leftarrow \emptyset$;

AB ← last edge in F;

$l \leftarrow 0$;

2) While $F \neq \emptyset$ do

2.1) If $P = \emptyset$ and $l < n$ then

2.1.1) $l \leftarrow l+1$;

2.1.2) $P \leftarrow P \cup L_l$

end if;

2.2) Select point $C \in P \cup H \cup F$ by computing the maximum angle ACB;

2.3) If there is a point $C' \in L_{l+1}$ falling inside or on a side of $\triangle ABC$ then

2.3.1) $C \leftarrow C'$;

2.3.2) $l \leftarrow l + 1$;

2.3.3) $P \leftarrow P \cup L_l$

end if;

2.4) Record $\triangle ABC$;

2.5) Case C of:

2.5.1) $C \in P \cup H$:

P∪H ← P∪H − {C};

F ← F∪{AC,CB};

AB ← AC;

2.5.2) AC∉F, CB∉F:

stack AC;

F ← F∪{AC,CB};

AB ← CB;

2.5.3) AC∈F, CB∉F:

F ← F∪{CB}−{AC,AB};

AB ← CB;

2.5.4) AC∉f, BC∈F:

F ← F∪{AC}−{CB,AB};

AB ← AC;

2.5.5) AC∈F, BC∈F:

F ← F−{CB,AB,AC};

AB ← ∅;

end case;

2.6) If (AB is on H or AB = ∅) and stack ≠ ∅ then

AB ← pop(stack)

else if (AB is on H or AB = ∅) and stack = ∅ then

2.6.1) Search F for a new AB;

2.6.2) If not found or F contains only internal nodes then

F ← ∅

end if;

end if;

end while;

end.

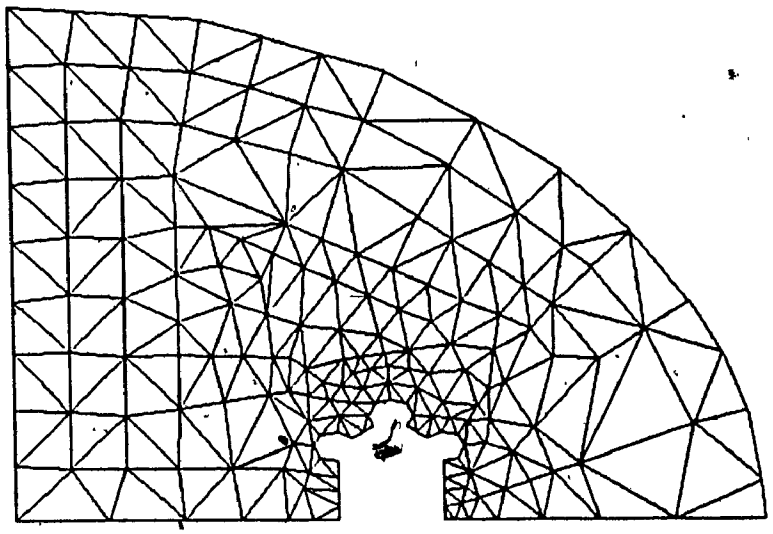


Figure 3.18 Mesh generated from figure 3.12

The automatically generated meshes by this method have an optimal construction of triangular elements with respect to the given input set of points. However the distribution of the nodes might cause some oddly shaped triangles to be formed. To complete the generation and produce good shapes of all elements, we apply the global optimization to the meshes by using the Laplacian smoothing procedure described previously and detailed in the next section.

3.3 Smoothing module

In this module we use the Laplacian scheme to improve the shapes of all elements in a given mesh. The Laplacian equation attempts to re-position the nodes so that each node is at the center of the polygon formed by the nodes surrounding it. The Laplacian operator is applied to each point P_i,

$$P_i = \frac{1}{n_i} \sum_{j=1}^{n_i} P_j, \quad i = 1,2,...,N$$

where n_i is the number of points connected to P_i and N is the total number of interior points since boundary points must not be affected by this re-positionning procedure. Assembling the equations for all the N points gives the system of linear equations,

$$A\underline{x} = \underline{b}_x$$

$$A\underline{y} = \underline{b}_y$$

where $\underline{x} = (x_1, x_2, ..., x_N)$, and

$\underline{y} = (y_1, y_2, ..., y_N)$.

The vectors $\underline{b}_x$ and $\underline{b}_y$ are obtained from the coordinates of the fixed boundary nodes.

To assemble the above system we need to know for each point the nodes that are connected to it. Output from the triangulator gives only a connectivity table of the elements but not for each individual point and we would have to search through this table to

determine the equation of a single point. In practice a sequential examination of the element connectivity table allows the construction of matrix A and vectors $\underline{b}_x$ and $\underline{b}_y$ with only one pass. We need to keep a list of boundary nodes connected to each interior node so that duplicates cannot be introduced into the right-hand side vectors. It is clear that

$$A = [A_1, A_2, \ldots, A_N]^T$$

where each row A_i has the form

$$A_{ij} = \begin{cases} n_i & \text{if } i=j \\ 0 & \text{if } P_j \text{ is not connected to } P_i \\ -1 & \text{if } P_j \text{ is connected to } P_i \end{cases}$$

Also,

$$\underline{b}_x = (b_1, b_2, \ldots, b_N)^T$$

where $b_i = \sum_{j=1}^{m_i} x_j$, m_i is the number of boundary points connected to P_i. The vector $\underline{b}_y$ is defined in a similar fashion.

The number m_i of boundary points connected to a given point P_i is usually small, only 0, 1, or 2. To keep memory usage to its lowest level using array-simulated list, we reserve a three-column matrix for this purpose. Let C denote this 3 × m matrix. The distribution for each node P_i, i = 1,2,...,N is

C(1-2, i) = node numbers, 0 if none

C(3, i) = 0 marks end of list

> 0 is the number of the third (and last) node

= k<0 points to the continuation location C(*,|k|)

The matrix A and vectors $\underline{b}_x$ and $\underline{b}_y$ of the above linear system are built by performing one pass through the connectivity table of elements in the mesh. The boundary connectivity of each node is also established during this pass. Let CY denote the element connectivity table and IJ a line segment (edge) with two endpoints I and J. The following algorithms perform the mesh smoothing process: algorithm ADD will insert values into matrix A and vectors $\underline{b}_x$, $\underline{b}_y$ for a given edge; algorithm SMOOTH calls ADD to build up the equation system and solve it to produce the final result (the new locations of interior nodes).

algorithm ADD (I, J, A, b_x, b_y, C)

/* Given an edge IJ in the mesh, update matrix A, vectors b_x, b_y, and the boundary connectivity matrix C. Denote C(i) the set of nodes connected to node i. */

begin

1) If I and J are both boundary nodes RETURN;

2) If either I or J is a boundary node then

/* Assume I is the interior node and J is the boundary node */

2.1) If J is not in C(I) then

2.1.1) C(i) ← C(I)+{J};

2.1.2) $b_x(I) \leftarrow b_x(I)+J_x$;

$b_y(I) \leftarrow b_y(I)+J_y$

end if;

else

2.2) If the connection I-J is not recorded then

2.2.1) A(I,I) ← A(I,I)+1;

A(J,J) ← A(J,J)+1;

2.2.2) A(I,J) ← -1;

A(J,I) ← -1

end if

end if

end.

algorithm SMOOTH (CY, X, Y)

/* Applies Laplacian smoothing scheme to the mesh given by the connectivity table CY and node point coordinates X, Y. */

begin

1) Initialize matrices A, C, vectors b_x, b_y to zero;

2) /* Construct A, C, b_x, b_y */

For e = 1 to number of elements do

2.1) Let triangle abc ← CY(e);

2.2) call ADD(a,b,A,b_x,b_y,C);

call ADD(b,c,A,b_x,b_y,C);

call ADD(c,a,A,b_x,b_y,C);

end for;

3) Solve AX = b_x;

Solve $AY = b_y$;

end.

After the linear system has been constructed any numerical method can be applied to obtain the solution. For this application the iterative method yields quite good result taking only a few iteration. The initial vectors X_0 and Y_0 are the original coordinates of the interior nodes. Figure 3.19 shows an improved mesh from previous example.

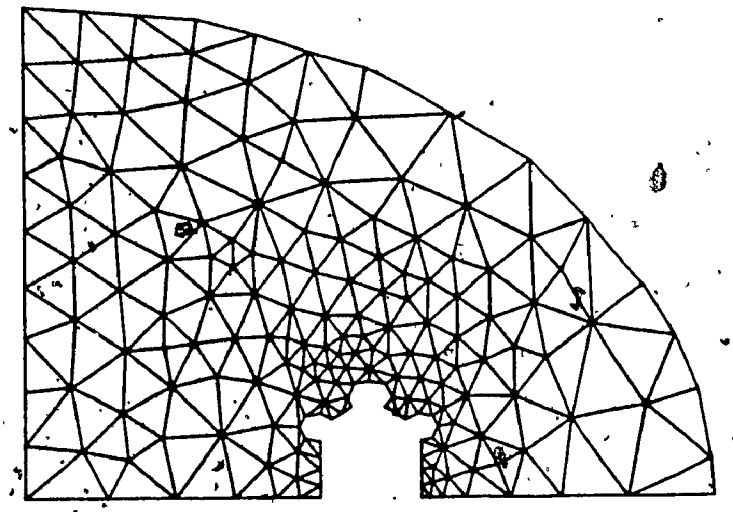


Figure 3.19 Smoothed mesh from figure 3.18

3.4 Data Structures

We now describe the data structures used in our implementation of the mesh generator. Since the Fortran language does not support linked list with pointers, we simulate the lists by using array and integer indexes as pointers.

The node points are stored by their two-dimensional coordinates (x,y) in different arrays using the same order in which point numbers are the indexes. Boundary nodes are stored consecutively as well as points within a single layer. Pointers now mark the starting and ending locations of nodes belonging to the same group. The following convention is used:

i) Boundary nodes occupy high positions, e.g. 1, 2, etc.

ii) External boundary has highest number in the ordering of boundary curves.

iii) Layers are counted from the external boundary inwards.

For the triangulation module the linked list structure is used to represent the generation front internally. Each point has pointers to its preceding and following nodes if they are on the front, or a negative value if there is no connection in the corresponding direction. The list of potential free nodes contains only node numbers and no other structures. Finally the element connectivity table is a 3×N array where each column contains the node numbers of the triangle's corners in counterclockwise order.

4. User Interface Using Interactive Computer Graphics

Automatic mesh generation requires some form of input from the user to define the domain for which a mesh is to be generated. The input usually is a discretization of the actual boundary curves and the work of preparing data might become tedious for complex boundary shapes. Interactive systems, as part of the CAD/CAM technology, makes good user-interface in the finite element analysis and other applications. An interactive mesh generator allows the analyst to enter data directly into the computer with a simple tool and to be able to physically view and modify his/her input on a graphics screen. Therefore interactive interface should always accompany an implementation of mesh generator.

In the next sections we present an interactive mesh generator implemented on a VAX/GPX Station II using GKS graphics. The program is easy to use and can be adapted to a different system without additional complexity. Our report comprises three parts: the user's view - how to use the mesh generator, the technical view - the program, and the portability of the program.

4.1 How to use the interactive mesh generator

The generator is mouse-driven. Movement of the mouse on a planar surface will change the cursor position on the graphics screen. Pressing a button on the mouse will enter user's request to

the program depending on the position of the cursor at that time. The screen is physically divided into three regions: heading, workspace, and command space. These regions are placed side by side as illustrated in figure 3.20.

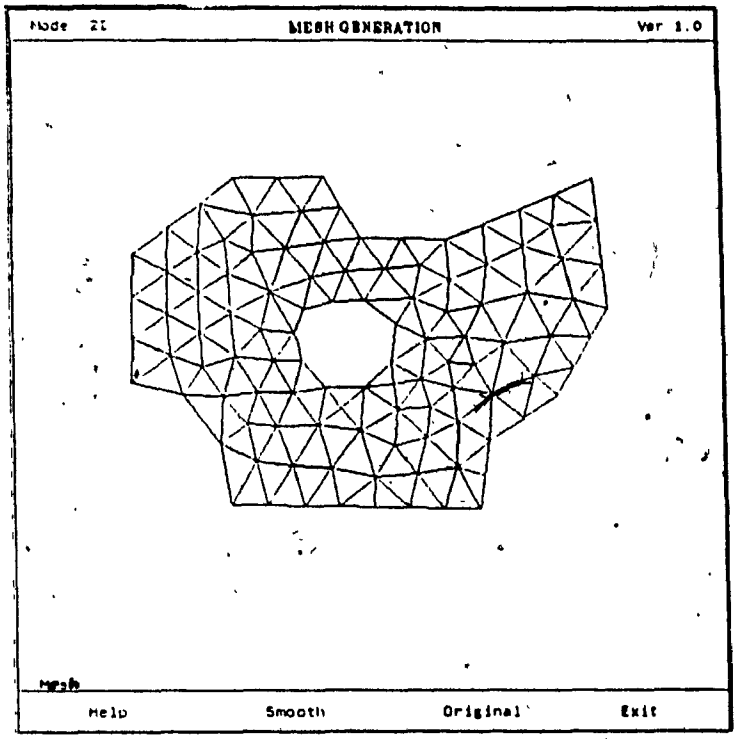


Figure 3.20 Screen layout

The heading displays the name and version of the software as well as the mode (2D or 3D) that is used for the session. The workspace is where the user can define a "problem" by entering data points. The commands are classified by levels as in a tree structure where each command might have a set of sub-commands. The command space shows all the commands available at the current operating level thus allows direct selection using the mouse. The keyboard is not used in general except when a file name

must be specified by the user for storage and retrieval operations.

The mouse-driven system makes it easy to enter data and commands. The leftmost button is used heavily throughout the interactive session. The button next to the leftmost one is used occasionally to signal no input or break on input for special purposes such as completion of a boundary curve. Since the cursor movement covers the whole screen, the user has freedom to switch operations at any time. It is important to remember that the action taken by the generator relies entirely on the position of the cursor when the leftmost button is pressed and the current level of operation. Any other buttons other than the two leftmost will not have effect on the program.

The commands available are organized in a tree-like fashion, that is, each command has its own sub-commands and so on. The sub-commands of any command (option) are displayed only when that option is selected. Exit points are provided at every level allowing the user to travel up and down in the tree. The software also offers HELP facility at all levels to assist the user in using the mesh generator.

We shall now describe in details the operations of the interactive mesh generator and how to define problems which are meaningful to the program.

4.1.1 Problem Definition: Domain and Subdivision

To the mesh generator, a domain is defined by a set of boundaries with one external boundary and zero or more internal boundaries representing holes inside the domain. Each boundary is a closed curve dicretized into a set of consecutive line segments. An advantage of using discretized boundaries is the simplification of input, since complex shapes might require several complicated functions to describe them. This form of boundary polygons is entered using a point by point basis with respect to the following rules:

i) Boundary points must be in counter-clockwise order;

ii) The first boundary to define must be the external boundary.

A problem is simply defined by entering the boundary points in the above order. Yet to allow more user control on the final mesh, it is possible to specify additional requirements for the mesh density as described in previous sections. Domain subdivision determines the different sub-regions of the domain which carries different density parameters. The definition of the sub-regions is very much similar to that of the domain with one additional line segment specifying the element size within the sub-region. By giving the length directly on screen the user can view the element size and have a pretty good idea of the distribution on the final

mesh. If the subdivision is not given, the generator will produce uniform element size depending on the smallest length of all boundary segments. It is worth mentioning that automatic introduction of new boundary points to match the required density is provided so that the users only have to enter the minimum number of points for the boundary curves.

4.1.2 Operating the mesh generator

At the first level of the command tree there are a number of main options which we shall refer to as operation modes. Within each mode the user can select the operation to be performed. Each selection of the currently displayed choices switches the operation mode and command menu. Two standard options at every level are HELP and EXIT. Selecting HELP mode causes the system to display information available on the current mode. The EXIT option will bring back the menu of the previous level and the EXIT at the top level terminates the interactive session and returns control to the operating system.

The available modes of operations are:

1) Define: to enter problem definition.

2) Mesh : to obtain the desired mesh.

3) View : to browse data or meshes.

4) Save : to save data or meshes in files.

5) Load : to retrieve saved data or meshes.

6) Reset : to clear the workspace.

We give detailed descriptions of each mode and its menu in the next sub-sections.

4.1.2.1 DEFINE mode

This mode allows the user to input data points. These points are entered by moving the cursor to the desired location and pressing the leftmost button. A line connecting the new point and previously entered point will appear on the screen. To terminate input for a polygonal boundary, press the button next to the leftmost after the last point has been entered. A line segment joining the first and last point will then be automatically drawn to complete the polygon. The definition of domain and sub-regions must always respect the convention given in the previous section.

With the standard HELP and EXIT the two other choices in this DEFINE mode are:

(1) DOMAIN: allows input of domain boundary points. The actual operations are:

DRAW : to enter boundary points.

ERASE: to remove previously entered points on any boundary.

(2) SUBDIVISION: allows input of sub-region boundaries and

density. A boundary definition must be followed by two points defining the element size for that sub-region. The operations are:

DRAW : to enter boundary points.

ERASE: to remove previously entered points.

MODIFY DENSITY: change the density of a defined sub-region.

4.1.2.2 MESH mode

After defining the problem the user has choices on the quality of the mesh he wants to see on the screen, either the intermediate un-smoothed mesh or the final mesh after passing through the smoothing procedure. The reason is that smoothed meshes generally have better element shapes while sometimes the original meshes might be preferred for some analytical conveniences. The two choices in this mode are ORIGINAL for original mesh displayed, and SMOOTH for smoothed mesh. To view the other mesh not displayed as result without repeating the generation, the user can go to the VIEW mode and select the mesh to be displayed.

4.1.2.3 VIEW mode

This option allows the users to quickly browse the data and meshes, if there is any. The two choices are DATA and MESH. The MESH option offers either ORIGINAL or SMOOTH as in the MESH mode.

4.1.2.4 SAVE mode

Any data or meshes created during an interactive session can be saved in files for later reference or modification. The program stores data in a special format which can only be interpreted by the generator when the data is loaded back into the session. The users have freedom to choose the name of the file onto which information will be saved. Under the VAX/VMS operating system these file names have a default extension .DAT. The DIRECTORY option allows the users to view the names of all such files on the graphics screen.

4.1.2.5 LOAD mode

Previously saved data and meshes can be loaded back into the workspace under this operation mode. The three choices on the menu are DATA - load a problem definition, MESH - load a mesh (original and smoothed), and DIRECTORY - display all file name with extension .DAT in the current directory. The user is requested to enter the file name of his choice and the system will verify

whether the file contains valid data before actually loading information. The retrieved information will then overwrite the corresponding data residing in the workspace.

4.1.2.6 RESET mode

There are two options under this mode. The user can choose to erase only the mesh and the subdivision associated with the active domain in the workspace, or to clean up the workspace for a new problem. Data which have been reset are lost and cannot be recovered unless they were previously saved in system files.

The above description has covered all the six operation modes of our interactive mesh generator. The HELP screens which are actually displayed by the HELP option can be found in appendix 1.

4.2 Inside the interactive mesh generator

This section is concerned mainly with the design and implementation of the mesh generator interface. The program is written in Fortran and uses the GKS graphics software on a VAX/GPX Station II. The basic program structure follows a tree-like pattern. The tree of commands is simulated using array of father/child nodes. There are two main variables:

1) TREE: array n×m where TREE(i,j) is the location of the j^{th} child of the i^{th} node in the tree when j>0. If j=0

TREE(i,j) denotes the location of the father node of the i^{th} node.

2) NUM-CHOICES: array(m) where NUM-CHOICES(i) is the number of children of the i^{th} node. Note that this array might be incorporated into TREE but we keep it apart for clarity and fast access.

Based on this structure the flow of control within the program is established by branching to the appropriate subroutine according to the choice entered by the user. The mouse-driven system is supported by the GKS functions which return the (x,y) coordinates of the cursor location on the screen when a button is pressed. The program then uses these values to determine which part of the screen the cursor is in and proceeds appropriate actions. The other GKS functions used in the interface include line drawing, text writing, marking, and filling. GKS offers segment type variable to store graphical data on the terminal screen, but redrawing the segments causes the screen to flash up, hence although it is convenient to use segment variables to switch displays, this interface does not employ this technique. However it is not difficult to adapt the program to use segment type variables on some other type of graphics terminals that support transparent segment re-drawing.

4.3 Portability of the mesh generator interface

The mesh generator and its interactive graphics interface are written in Fortran and use GKS graphics software for color monitor on a VAX/GPX Station II. All Fortran syntax and functions in the program are standard; hence no modification is necessary to run it on other systems. However many GKS subroutine calls might require minor or major changes when using different graphics software. An example is the difference between PLOT10 and GKS for line drawing. PLOT10 does not support multiple line drawing as does GKS; a single subroutine call in GKS must be replaced by a "for" loop to draw a sequence of line segments. Since the segment type variable, a special feature of GKS is not used, the program can be adapted easily to other systems. The last difference worth mentioning is the availability of different fonts on GKS which other graphics packages might not support. Using this feature just makes the screen appearance more attractive and does not have any effect on the behavior of the mesh generator.

5. Time and Space complexities

As discussed in chapter Two on the complexity of algorithms for mesh generation, it is difficult to give a net picture of the time and space efficiency of mesh generators. In this section we will estimate the space and time requirement of the implementation reported previously and make some empirical comparisons with some program available in the literature.

5.1 Time Complexity

In the following discussion the basic operation for the time complexity is often not a single operation such as comparison, but rather a comparison of the result from performing a set of arithmetic operations and comparisons; hence the given figure only represents an estimate of the complexity.

5.1.1 Node Generation

For the generation of node points on the boundary and inside the domain, we base our discussion on algorithm NODE—GENERATOR of section 3.1. Let n be the number of boundary nodes and N the total number of points generated. The number of segments per layer is bounded above by n since we are dealing with continuously shrunk boundary. Hence the time taken to generate layer segments is O(n). Finding the intersections of the segments with internal holes is also O(n). For L layers the total

time is O(nL) and since the number of points per layer is also bounded above by n this figure is equivalent to an O(N) time complexity for the node generator. This achievement is the same as other methods of node generation with simplicity gained.

5.1.2 Triangulation

In the triangulation algorithm (cf. sections 2.2 and 3.2), a number of tests must be performed each time a new element is recorded. We shall refer to this set of operations simply as a comparison in subsequent discussion on the time analysis. Let F_k and A_k denote the number of points on the generation front and in the set of potential free points at the k^{th} iteration. Also let there be n boundary points, p interior points, and a maximum of l points per layer. Then the following holds:

$$0 \leq F_k + A_k \leq n+l \leq 2n$$

The number of triangles N_e is known to satisfy the relation [MAU84]

$$N_e = 2p + n - 2$$

The total number of comparisons is then at most

$$\begin{aligned} \sum_{k=1}^{N_e} (F_k + A_k) &\leq \sum_{k=1}^{N_e} (n+l) \\ &\leq (2p+n-2)(n+l) \\ &\leq 4pn + 2n(n-2) \\ &\leq 2(p+n)^2 \end{aligned}$$

Since the total number of nodes is N = n + p we have an approximate $O(N^2)$ time for triangulation.

In the original advancing front method, the upper bound for F_k+A_k is n+p. If we compare the relative efficiency of the two algorithms asymptotically when p is large as compared to n, i.e. n<<p then

$$\lim_{p\to\infty} \frac{\Sigma(n+l)}{\Sigma(n+p)} = \lim_{p\to\infty} \frac{n}{p} = 0$$

because the number of points per layer is bounded above by the number of boundary points. Hence the advancing front with layers shows an increase in the efficiency when the number of interior points is large.

Most of empirical comparisons of performance of mesh generators are obtained from test domains of simple form such as square or rectangle [MOS81,HOL88]. We follow the same pattern and use an unit square and a circle as test domain. The only program available at hand is that of Nelson [NEL78] which improves on Cavendish's [CAV74]. Table 3.1 shows the time in CPU seconds for the mentioned program running on the same VAX work-station.

Table 3.1 CPU time for triangulation of a square (in secs)

N	Nelson	Adv.Front	Adv.Front w/Layers
12	0.10	0.04	0.03
25	0.43	0.11	0.10
101	9.70	1.97	1.23
225	53.72	10.24	4.64

Another program for triangulating a convex hull of a set of points from Sloan [SLO87] were also tested but the result is restricted to simple convex domains with no internal holes, hence cannot be compared to our general mesh generator. Testing on arbitrary domain reveals that Nelson's program does not work correctly as expected for some convex domain. In fact efficient treatment of generally-shaped domains always resumes in some trade-off between the time complexity and the geometry complexity. Our mesh generator is faster and produces correct results for both concave and convex domains with or without internal holes.

5.2 Space Complexity

In the node generation program permanent storage required is for the node coordinates and pointers of node location delimiters. We need at most 3N memory words for this process where N is the total number of nodes on the boundary and inside the domain.

In the triangulation process the linked list used for both F and P requires 3N words. The set of free nodes is just two pointers to denote the range of nodes in the list. We use the same memory space for node coordinates. The table of connectivity is a $3\times N_e$ array, where N_e is the number of elements, N_e = 2N-n-2, n is the number of boundary nodes. Therefore the total storage required is at most

$$S \leq 3N + 3N + 3(2N - n - 2)$$
$$\leq 12N$$

The result is satisfactory as compared to the storage required by other methods reported in chapter two.

Chapter Four

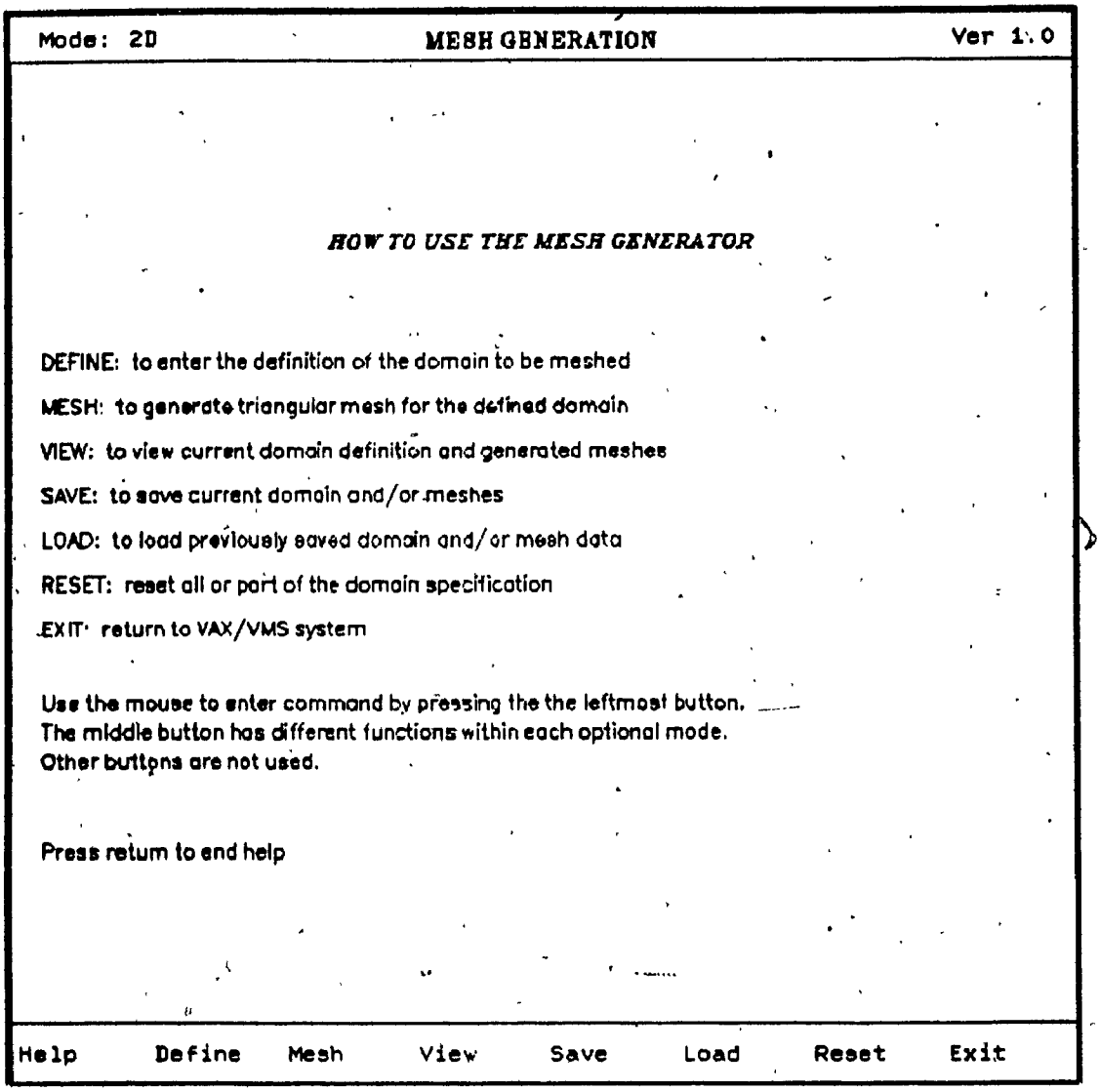
CONCLUSION

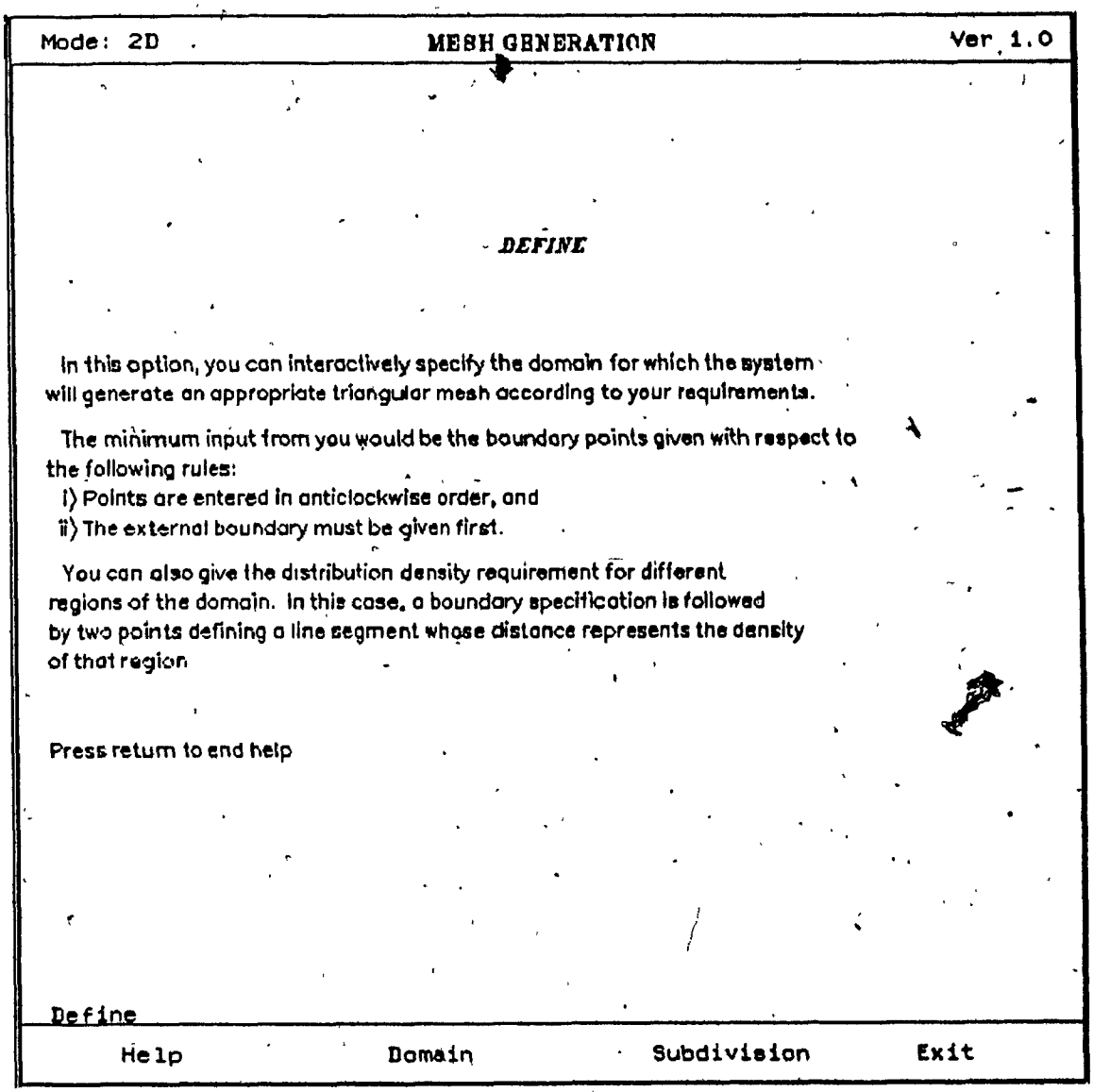
Mesh generation for finite element analysis is an active research area in the field of computer applications in engineering. It plays an important role in the success and popularity that the finite element methods have gained in every application and continue to expand to areas other than engineering. We have covered the concept and position of mesh generation within a finite element analysis process. Since the field has grown rapidly from its early days, the review in the second chapter gives an overview of the direction of the research and the valuable achievements of several authors. Investigation and implementation of a mesh generator represent a challenging task to put the computer to automatically produce meshes. A user interface using graphics facility makes the generator a truly useful tool of analysis for the engineers.

Our presentation has emphasized on the generation of a particular class of meshes in two-dimension, the triangular meshes, since this class is widely used in the real world. Other classes of meshes such as grids and mixed element type meshes are also investigated in the research literature. Although three-dimensional analysis is the ultimate and more practical aim of engineering applications, research in three-dimensional mesh generation has just become active recently in parallel with the two-dimensional mesh generation. Two-dimensional mesh generation remains a "hot" topics since researchers are looking at generalization of mesh generation methods by expansion from low to high dimension, rather than developing separate and dedicated algorithms. In fact the concept of bottom-up development is convenient and easy to follow, especially in the analysis of algorithm efficiency. It is exactly this feature of the advancing front method which exposes a possibility of expansion to higher dimensions as confirmed by Löhner in [LÖH87], that we have chosen to implement and enhance on. This work hence represents an important starting point for further investigation of mesh generation into higher dimensions.

APPENDIX

THE INTERACTIVE MESH GENERATOR HELP SCREENS




Mode: 2D
MESH GENERATION
Ver 1.0
DEFINE
In this option, you can interactively specify the domain for which the system will generate an appropriate triangular mesh according to your requirements.
The minimum input from you would be the boundary points given with respect to the following rules:
i) Points are entered in anticlockwise order, and
ii) The external boundary must be given first.
You can also give the distribution density requirement for different regions of the domain. In this case, a boundary specification is followed by two points defining a line segment whose distance represents the density of that region
Press return to end help
Define
Help
Domain
Subdivision
Exit

Mode: 2D | MESH GENERATION | Ver 1.0

DEFINING A DOMAIN

Use the mouse's left button to select an option and enter points.

DRAW: enter the boundary points in counterclockwise (CCW) order by pressing the leftmost button; end each boundary by pressing the middle button. Specify the external boundary first.

ERASE: remove a point on a boundary by moving the mouse cursor to that point and prese the leftmoet button. After erasing a point, if you want to insert point on that boundary, move the cursor to the opened end of the boundary (in CCW order), enter that point then continue entering new points.

EXIT: return to previous menu.

Press return to end help

Define /Domain

Help | Draw | Erase | Exit

Mode: 2D MESH GENERATION Ver 1.0

DEFINING A SUBDIVISION

Use the mouse's left button to select an option and enter points.

DRAW: enter the boundary points in counterclockwise (CCW) order by pressing the leftmost button; end each boundary by pressing the middle button. you are then required to enter two points defining the required density. Specify the external boundary first.

ERASE: remove a point on a boundary by moving the mouse cursor to that point and press the leftmost button.
After erasing a point, if you want to insert point on that boundary, move the cursor to the opened end of the boundary (in CCW order), enter that point then continue entering new points

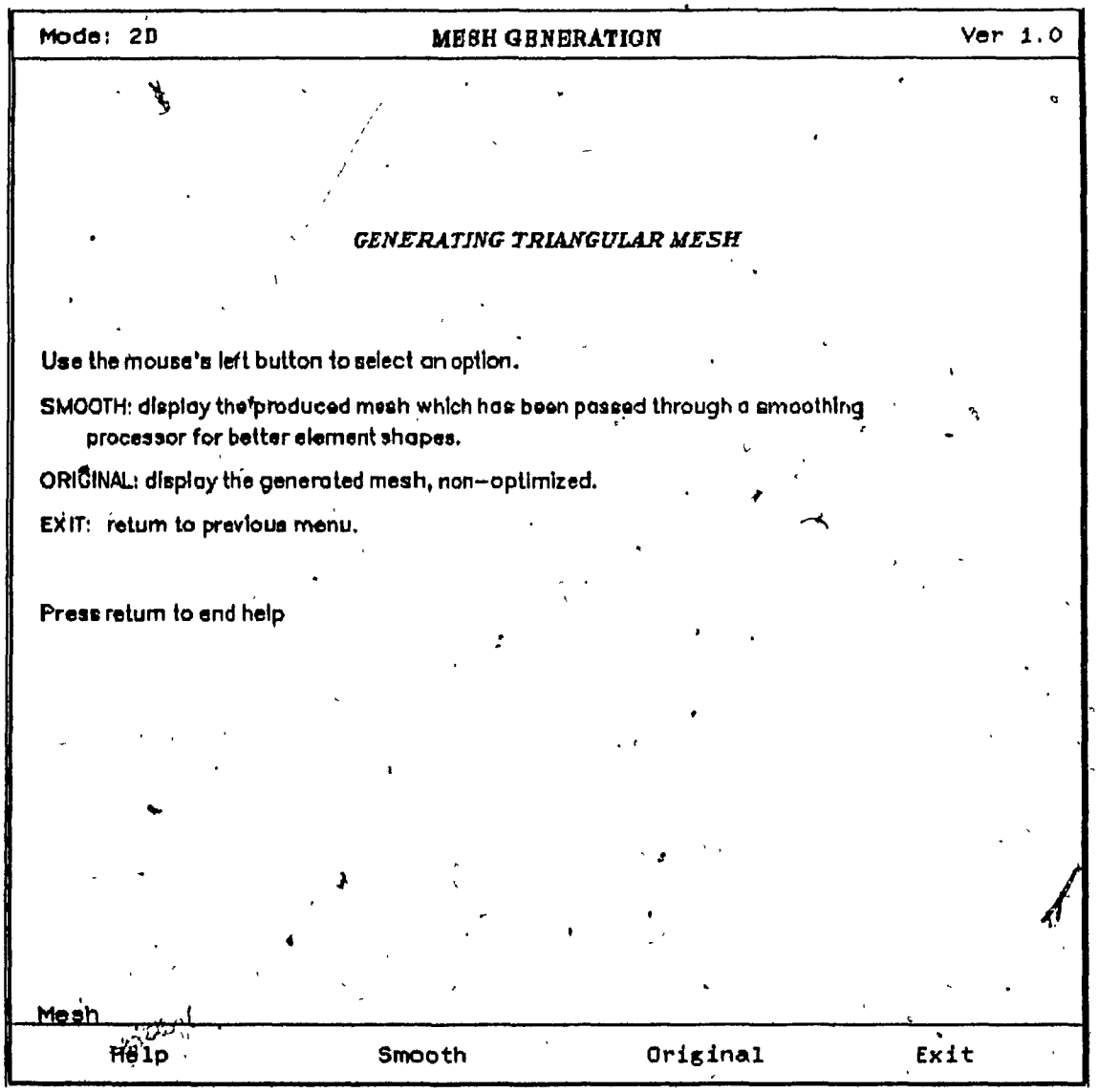
MOD.DENS.: to change only the density of a region, move the cursor to any point on the region's boundary, enter the point, then enter two points for the new density

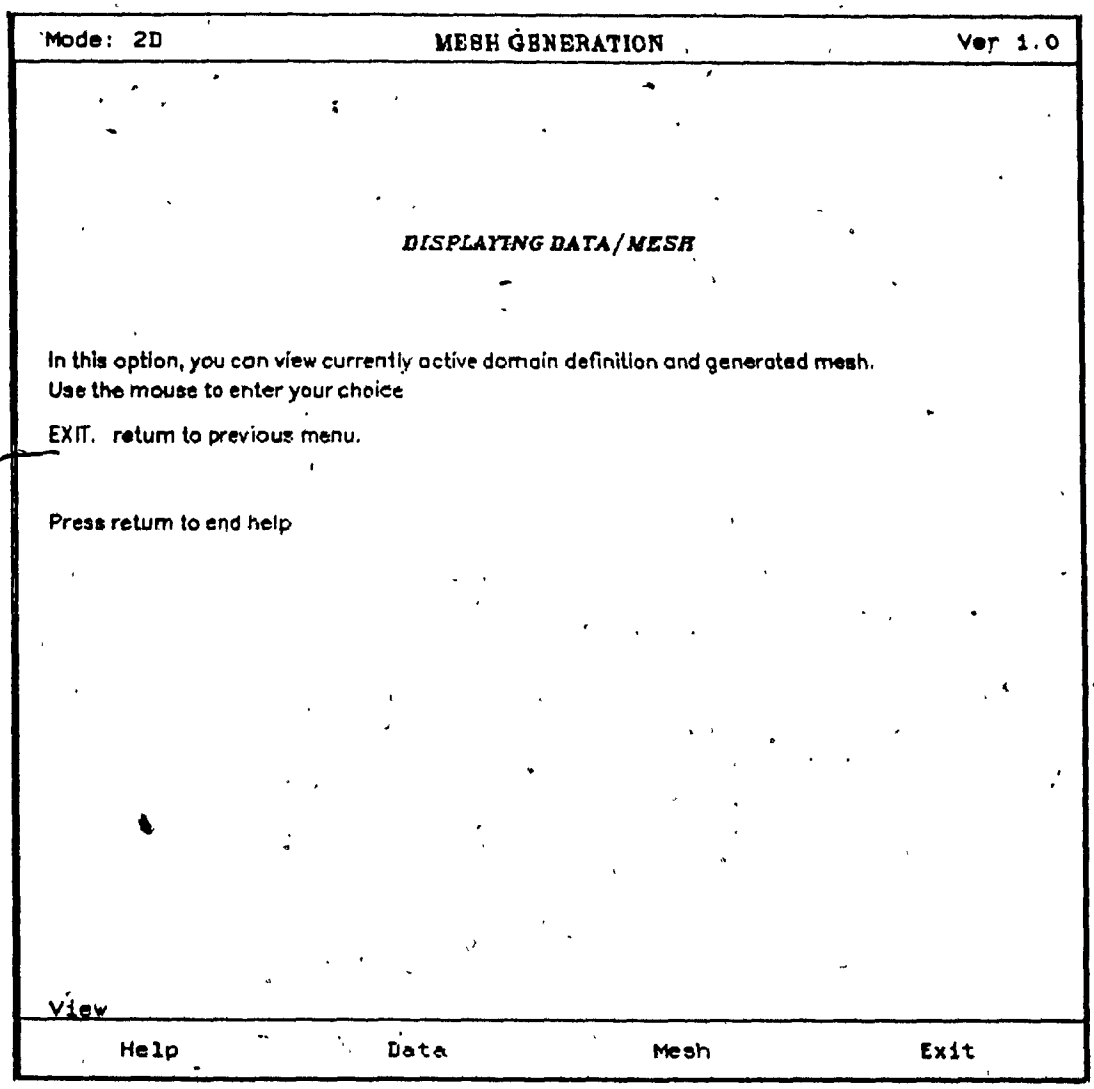
EXIT: return to previous menu.

Press return to end help

Define /Subdivision

Help Draw Erase Mod. Dens. Exit


Mode: 2D
MESH GENERATION
Ver 1.0
GENERATING TRIANGULAR MESH
Use the mouse's left button to select an option.
SMOOTH: display the produced mesh which has been passed through a smoothing processor for better element shapes.
ORIGINAL: display the generated mesh, non-optimized.
EXIT: return to previous menu.
Press return to end help
Mesh
Help
Smooth
Original
Exit


Mode: 2D
MESH GENERATION
Ver 1.0
DISPLAYING DATA/MESH
In this option, you can view currently active domain definition and generated mesh.
Use the mouse to enter your choice
EXIT. return to previous menu.
Press return to end help
View
Help
Data
Mesh
Exit

Mode: .2D | MESH GENERATION | Ver 1.0

VIEWING TRIANGULAR MESH

Use the mouse's left button to select an option.

SMOOTH: display the produced mesh which has been passed through a smoothing processor for better element shapes.

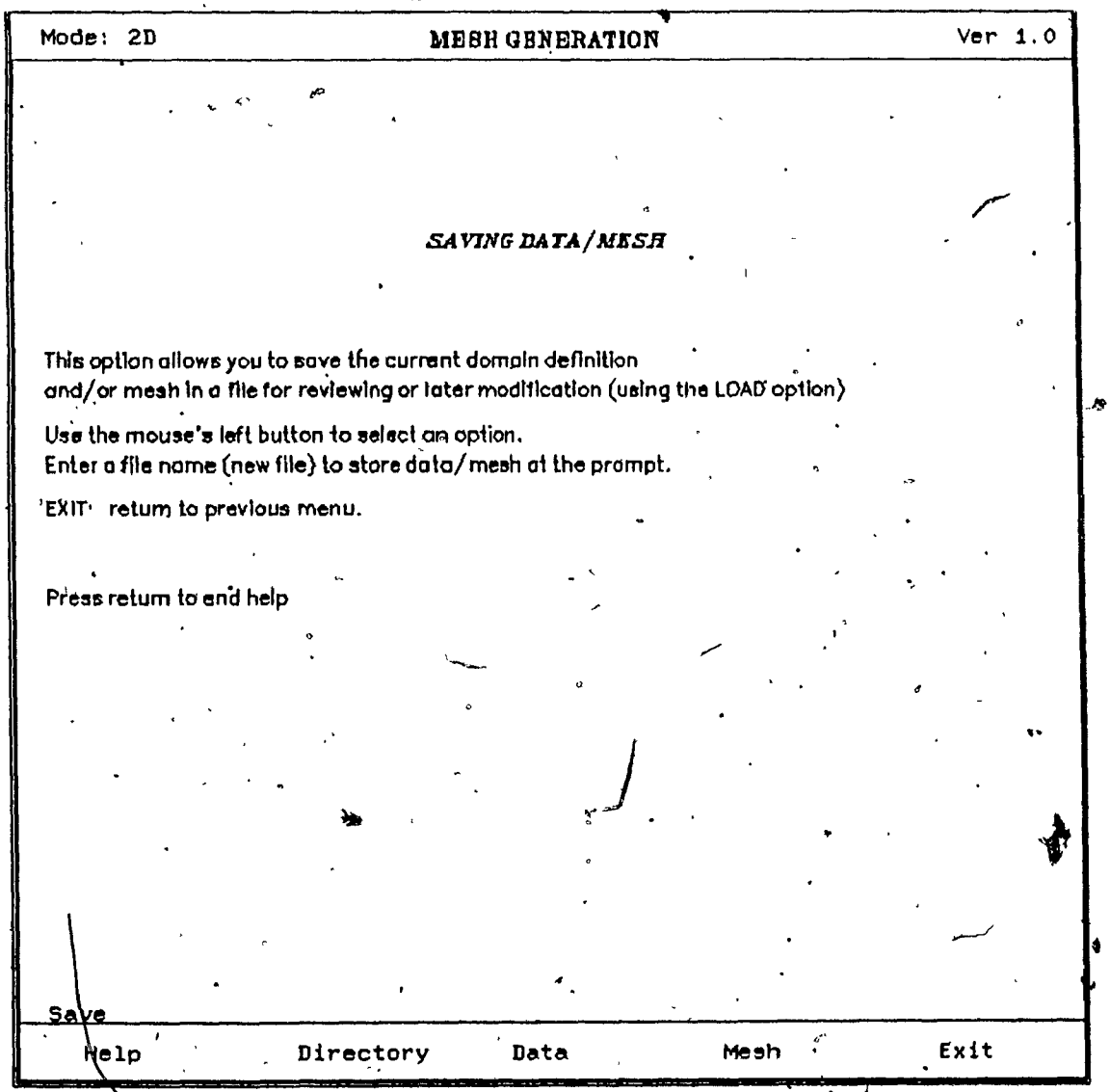
ORIGINAL: display the generated mesh, non-optimized.

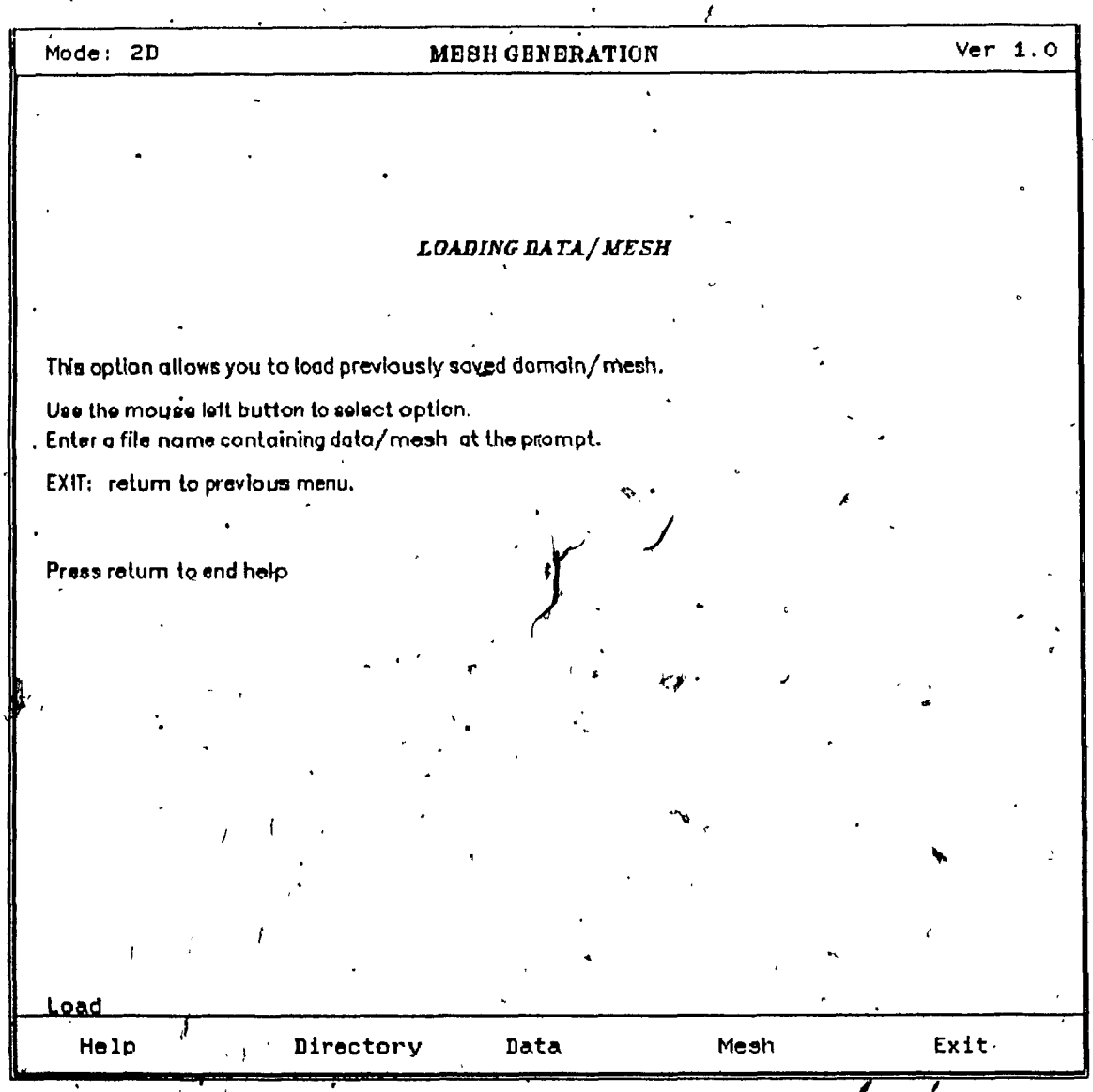
EXIT: return to previous menu.

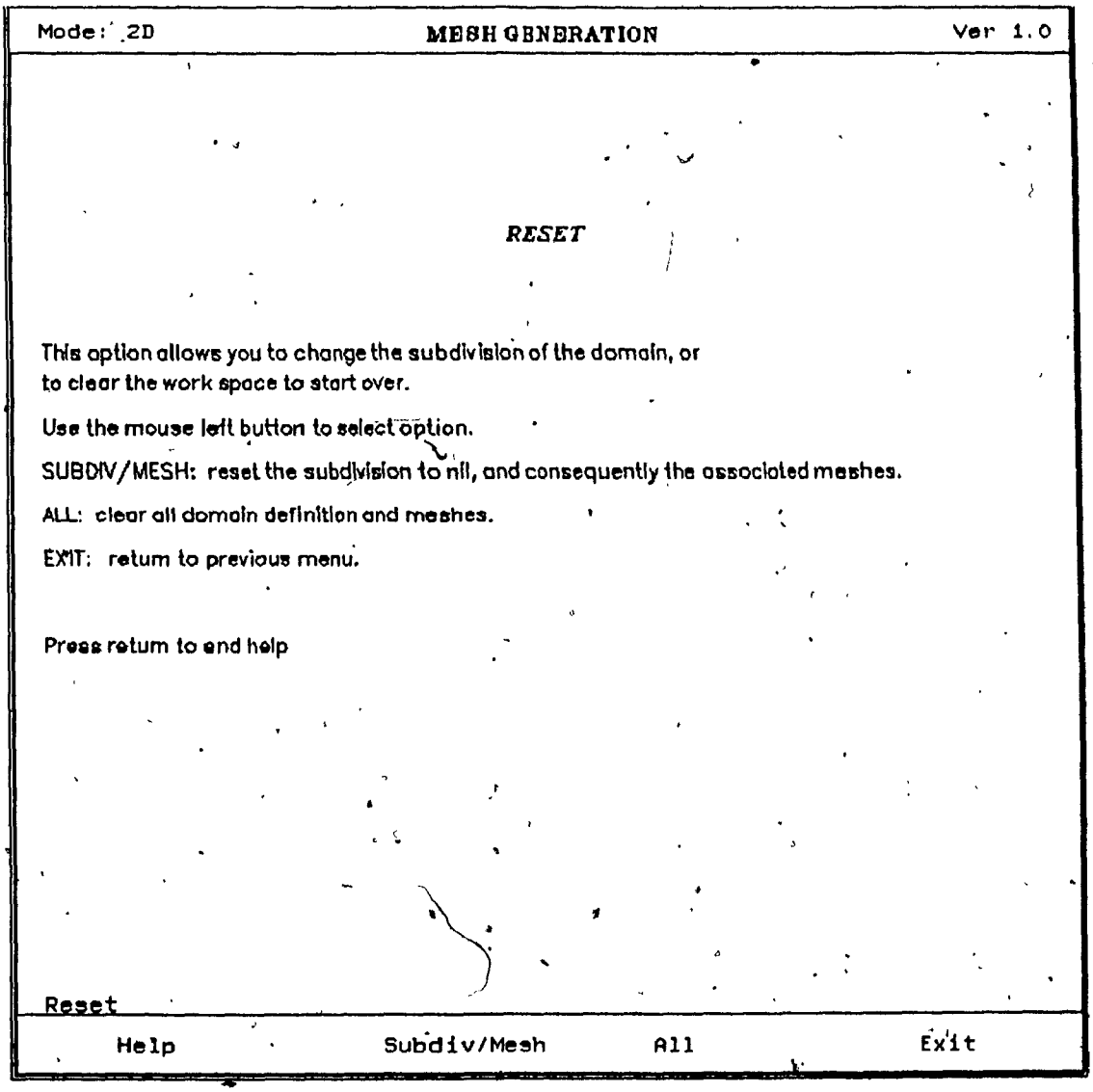
Press return to end help

View /Mesh

Help | Smooth | Original | Exit


Mode: 2D
MESH GENERATION
Ver 1.0
SAVING DATA/MESH
This option allows you to save the current domain definition
and/or mesh in a file for reviewing or later modification (using the LOAD option)
Use the mouse's left button to select an option.
Enter a file name (new file) to store data/mesh at the prompt.
'EXIT' return to previous menu.
Press return to end help
Save
Help
Directory
Data
Mesh
Exit


Mode: 2D
MESH GENERATION
Ver 1.0
LOADING DATA/MESH
This option allows you to load previously saved domain/mesh.
Use the mouse left button to select option.
Enter a file name containing data/mesh at the prompt.
EXIT: return to previous menu.
Press return to end help
Load
Help
Directory
Data
Mesh
Exit


Mode: 2D
MESH GENERATION
Ver 1.0
RESET
This option allows you to change the subdivision of the domain, or
to clear the work space to start over.
Use the mouse left button to select option.
SUBDIV/MESH: reset the subdivision to nil, and consequently the associated meshes.
ALL: clear all domain definition and meshes.
EXIT: return to previous menu.
Press return to end help
Reset
Help
Subdiv/Mesh
All
Exit

REFERENCES

[AKI82] J.E. Akin, Application and implementation of finite element methods, Academic Press 1982.

[BAE87] P.L. Baehmann, S.L. Wittchen, M.S. Shephard, K.R. Grice, M.A. Yerry, Robust, geometrically based, automatic two-dimensional mesh generation, Int. J. Num. Meth. Engng 24(1987), pp 1043-1078.

[BOW81] A. Bowyer, Computing Dirichlet tessellations, Comp. J. 24(1981), pp 162-166.

[BEC81] E.B. Becker, G.F. Carey, J.T. Oden, Finite elements, Volume 1: An introduction, Prentice-Hall 1981.

[BOU86] T.I. Boubez & al, Mesh generation for computational analysis, Comp. Aided Engng J. Oct 1986, pp 190-201.

[BRO82] P.R. Brown, D.R. Hayhurst, Using the Schwarz-Christoffel transformation in mesh generation for the solution of two-dimensional problems, Comp. Mech. Engng Aug 1982, pp 73-79.

[BUE73] W.R. Buell, B.A. Bush, Mesh generation - A survey, Trans. ASME, J. Engng Ind. Feb 1973, pp 332-338.

[BYK76] A. Bykat, Automatic generation of triangular grid: I - Subdivision of a general polygon into convex subregions, II - Triangulation of convex polygons, Int. J. Num. Meth. Engng 10(1976), pp 1329-1342.

[BYK83] A. Bykat, Design of a recursive, shape controlling mesh generator, Int. J. Num. Meth. Engng 19(1983), pp 1375-1390.

[CAR84] G.F. Carey, J.T. Oden, Finite elements, Volume 3: Computational aspects, Prentice-Hall 1984.

[CAV74] J.C. Cavendish, Automatic triangulation of arbitrary planar domains for the finite element method, Int. J. Num. Meth. Engng 8(1974), pp 679-696.

[CAV77] J.C. Cavendish, W.J. Gordon, Substructured macro elements based on locally blended interpolation, Int. J. Num. Meth. Engng 11(1977), pp 1405-1421.

[CAV85] J.C. Cavendish & al., An approach to automatic 3D finite element mesh generation, Int. J. Num. Meth. Engng 21(1985), pp 329-347.

[CLI84] A.K. Cline, R.L. Renka, A storage-efficient method for construction of a Thiessen triangulation, Rocky Mountain J. of Math. 14(1984), pp 119-139.

[COH80] H.D. Cohen, A method for the automatic generation of triangular elements on a surface, Int. J. Num. Meth. Engng 15(1980), pp 470-476.

[COO74] W.A. Cook, Body oriented (natural) coordinates for generating three dimensional meshes, Int. J. Num. Meth. Engng 8(1974), pp 27-43.

[COR87] Y. Correc, E. Chapuis, Fast computation of Delaunay triangulation, Adv. Engng Software '9(1987), pp 77-83.

[CRA87] R.H. Crawford, W.N. Waggenspack Jr, D.C. Anderson, Composite mappings for planar mesh generation, Int. J. Num. Meth. Engng 24(1987), pp 2241-2252.

[DEF85] L. De Floriani & al., Delaunay-based representation of surfaces defined over arbitrarily shaped domains, Comp. Vision, Graphics & Image Processing 32(1985), pp 127-140.

[DEL34] B. Delaunay, Sur la sphère vide, Bull. Acad. Sci. USSR(VII), Classe Sci. Mat. Nat. 1934, pp 739-800.

[DEN78] A. Denayer, Automatic generation of finite element meshes, Comp. Struct. 9(1978), pp 359-364.

[FEN75] R.T. Fenner, Finite element methods for engineers, Mcmillan Press Ltd, 1975.

[FRE70] C.O. Frederick & al., Two-dimensional automatic mesh generation for structural analysis, Int. J. Num. Meth. Engng 2(1970), pp 133-144.

[FRE87] W.H. Frey, Selective refinement: a new strategy for automatic node placement in graded triangular meshes, Int. J. Num. Meth. Engng 24(1987), pp 2183-2200.

[FUK72] J. Fukuda, J. Suhara, Automatic mesh generation for finite element analysis, Adv. in Computational Methods in Structural Mechanics and Designs, Eds J.T. Oden, R.W. Clough, Y. Yamamoto, UAH Press 1972.

[HAB81] R. Haber & al., A general two-dimensional, graphical finite element preprocessor utilizing discrete transfinite mappings, Int. J. Num. Meth. Engng 17(1981), pp 1015-1044.

[HAB82] R. Habel, J.F. Abel, Discrete transfinite mappings for the description and meshing of 3D surfaces using interactive computer graphics, Int. J. Num. Meth. Engng 18(1982), pp 41-66.

[HEI82] E.A. Heighway, C.S. Biddlecombe, Two dimensional automatic triangular mesh generation for the finite element electromagnetics package PE2D, IEEE Trans. Mag. MAG-18, No. 2, 1982, pp 594-598.

[HER76] L.R. Hermann, Laplacian-isoparametric grid generation scheme, J. Engng Mech. Div. ASCE 102(1976), pp 749-756.

[HER83] S. Hertel, K. Mehlorn, Fast triangulation of simple polygons, Proc. 1983 Int. FCT Conf., Lecture notes in C.Sc. 158, pp 207-218.

[HOL88] K. Ho-Le, Finite element mesh generation methods: a review and classification, CAD 20(1988), pp 27-38.

[IMA80] I. Imafuku, Y. Kodera, M. Sayawaki, A generalized

automatic mesh generation scheme for FEM, Int. J. Num. Meth. Engng 15(1980), pp 713-731.

[JOE86] B. Joe, Delaunay triangular meshes in convex polygons, SIAM J. Sci. Stat. Comput. 7(1986), pp 514-539.

[JOE86s] B. Joe, R.B. Simpson, Triangular meshes for regions of complicated shape, Int. J. Num. Meth. Engng 23(1986), pp 751-778.

[KEL86] A. Kela, R. Perucchio, H. Voelcker, Toward automatic finite element analysis, Comp. Mech. Engng Jul 1986, pp 57-71.

[KEL87] A. Kela, M. Saxena, R. Perucchio, A hierarchical structure for automatic meshing and adaptive FEM analysis, Eng. Comput. 4(1987), pp 104-112.

[KIR80] D.G. Kirkpatrick, A note on Delaunay and Optimal triangulation, Inf. Proc. Letters 10(1980), pp 127-128.

[KLE80] C. Kleinstreuer, J.T. Holdeman, A triangular finite element mesh generator for fluid dynamic systems of arbitrary geometry, Int. J. Num. Meth. Engng 15(1980), pp 1325-1334.

[KLI76] A. Klinger, C.R. Dyer, Experiments on picture representation using regular decomposition, Comp. Graphics & Image Processing 5(1976), pp 68-105.

[LAW77] C.L. Lawson, Software for C^1 surface interpolation, Mathematical Software III, Ed. J. Rice, New York Academic

1977, pp 161-193.

[LEE80] D.T. Lee, B.J. Schachter, Two algorithms for constructing a Delaunay triangulation, Int. J. Comp. Inf. Sc. 9(1980), pp 219-242.

[LEE84] Y.T. Lee, A. De Pennington, N.K. Shaw, Automatic finite element mesh generation from geometric models - A point-based approach, ACM Trans. Graph. 3(1984), pp 287-311.

[LEW78] B.A. Lewis, J.S. Robinson, Triangulation of planar regions with applications, Comp. J. 21(1978), pp 324-332.

[LI88] Z.C. Li & al, Finite element methods application to pattern recognition, to appear.

[LIN83] D.A. Lindholm, Automatic triangular mesh generation on surfaces of polyhedra, IEEE Trans. Mag. MAG-19, No. 6, 1983, pp 2539-2542.

[LOH85] S.H. Lo, A new mesh generation scheme for arbitrary planar domains, Int. J. Num. Meth. Engng 21(1985), pp 1403-1426.

[LOH87] R. Löhner, Three-dimensional grid generation by the advancing-front method, Fifth Int. Conf. Num. Meth. in Laminar and Turbulent Flow 1987, pp 1092-1105.

[MAU84] A. Maus, Delaunay triangulation and the convex hull of n points in expected linear time, BIT 24(1984), pp 151-163.

[MCG83c] M.B. McGirr, D.J.H. Corderoy, P.C. Easterbrook, A.K.

Hellier, Generation of finite element meshes from nodal arrays, Computational Techniques and Applications CTAC-83, Eds J. Noye, C. Fletcher, 1983, pp 223-228.

[MCG83k] M.B. McGirr, P. Krauklis, The automatic generation of arrays of nodes with varying density, CTAC-83, pp 229-235.

[MOS81] A.O. Moscardini, M. Cross, B.A. Lewis, Assessment of three automatic triangular mesh generators for planar regions, Adv. Engng Software 3(1981), pp 108-114.

[MOS83] A.O. Moscardini, B.A. Lewis, M. Cross, AGTHOM - Automatic generation of triangular and higher order meshes, Int. J. Num. Meth. Engng 19(1983), pp 1331-1353.

[NEL78] J.M. Nelson, A triangulation algorithm for arbitrary planar domains, Appl. Math. Modelling 2(1978), pp 151-159.

[NGU82] Nguyen-van-Phai, Automatic mesh generation with tetrahedron elements, Int. J. Num. Meth. Engng 18(1982), pp 273-281.

[PER82] R. Perucchio, A.R. Ingraffea, J.F. Abel, Interactive computer graphics preprocessing for 3D finite element analysis, Int. J. Num. Meth. Engng 18(1982), pp 909-926.

[PIS81] S. Pissanetzky, KUBIK: An automatic three-dimensional finite element mesh generator, Int. J. Num. Meth. Engng 17(1981), pp 255-269.

[PRE85] F.P. Preparata, M.I. Shamos, Computational geometry: an

introduction, Springer-Verlag, New York 1985.

[RIV84] M.C. Rivara, Algorithms for refining triangular grids suitable for adaptive and multigrid techniques, Int. J. Num. Meth. Engng 20(1984), pp 745-756.

[RIV87] M.C. Rivara, A grid generator based on 4-triangles conforming mesh refinement algorithms, Int. J. Num. Meth. Engng 24(1987), pp 1343-1354.

[SAD80] E.A. Sadek, A scheme for the automatic generation of triangular finite elements, Int. J. Num. Meth. Engng 15(1980), pp 1813-1822.

[SAW87] D.G. Sawkar, G.R. Shevare, S.P. Koruthu, Contour plotting for scattered data, Comput. & Graphics 11(1987), pp 101-104.

[SHA78] R.D. Shaw, R.G. Pitchen, Modifications to the Suhara-Fukuda method of network generation, Int. J. Num. Meth. Engng 12(1978), pp 93-99.

[SHA81] M. Shapiro, A note on Lee and Schachter's algorithms for Delaunay triangulation, Int. J. Comp. Inf. Sc. 10(1981), pp 413-418.

[SHE80] M.S. Shephard, R.H. Gallagher, J.F. Abel, The synthesis of near-optimum finite element meshes with interactive computer graphics, Int. J. Num. Meth. Engng 15(1980), pp 1021-1039.

[SHE83] M.S. Shephard, M.A. Yerry, Approaching the automatic generation of finite element meshes, Comp. Mech. Engng 1(Apr 1983), pp 49-56.

[SHE85] M.S. Shephard, Finite element modelling within an integrated geometric modelling environment, Engng with Computers 1(1985), pp 61-85.

[SHE87] M.S. Shephard, P.L. Baehmann, K.R. Grice, Automatic finite element modelling: geometry control for direct models, Eng. Comput. 4(1987), pp 119-125.

[SIM79] R.B. Simpson, Automatic local refinement for irregular rectangular meshes, Int. J. Num. Meth. Engng 14(1979), pp 1665-1678.

[SIM79s] R.B. Simpson, A survey of two-dimensional finite element generation, Proc. Ninth Manitoba Conf. on Num. Math. and Computing 1979, pp 49-124.

[SIM81] R.B. Simpson, A two-dimensional mesh verification algorithm, SIAM J. Sci. & Stat. Comput. 2(1981), pp 455-473.

[SLO83] S.W. Sloan, M.F. Randolph, Automatic element re-ordering for FE analysis with frontal solution, Int. J. Num. Meth. Engng 19(1983), pp 1153-1187.

[SLO87] S.W. Sloan, A fast algorithm for constructing Delaunay triangulation in the plane, Adv. Engng Software 9(1987), pp 34-55.

[TAN87] T. Taniguchi, Flexible mesh generation for triangular and quadrilateral areas, Adv. Engng Software 9(1987), pp 142-149.

[THA79] W.C. Thacker, An improved triangulation algorithm, Appl. Math. Modelling 3(1979), pp 471-472.

[THA80] W.C. Thacker, A brief review of techniques for generating irregular computational grids, Int. J. Num. Meth. Engng 15(1980), pp 1335-1341.

[THA80s] W.C. Thacker, A. Gonzalez, G.E. Putland, A method for automating the construction of irregular computational grids for storm surge forecast models, J. of Computational Physics 37(1980), pp 371-387.

[WAT81] D.F. Watson, Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes, Comp. J. 24(1981), 167-172.

[WAT84] D.F. Watson, G.M. Philip, Systematic triangulation, Comp. Vision, Graphics & Image Processing 26(1984), pp 217-223.

[WOO84] T.C. Woo, T. Thomasma, An algorithm for generating solid elements in objects with holes, Comp. Struct. 18(1984), pp 333-342.

[WOR84] B. Wordenweber, Finite element mesh generation, CAD 16(1986), pp 285-291.

[YER83] M.A. Yerry, M.S. Shephard, A modified quadtree approach to finite element mesh generation, IEEE Comp. Graph. Appl.

3(Jan 1983), pp 39-46.

[YER84] M.A. Yerry, M.S. Shephard, Automatic 3D mesh generation by the modified octree technique, Int. J. Num. Meth. Engng 20(1984), pp 1965-1990.

[YER85] M.A. Yerry, M.S. Shephard, Automatic mesh generation for 3D solids, Comp. Struct. 20(1985), pp 31-39.

[ZIE69] O.C. Zienkiewicz & al., Isoparametric and associated element families for two and three dimensional analysis, Eds I. Holland, K. Bell, Tapir Press, Trondheim, 1969, chapter 13.

[ZIE71] O.C. Zienkiewicz, D.V. Phillips, An automatic mesh generation scheme for plane and curved surfaces by isoparametric coordinates, Int. J. Num. Meth. Engng. 3(1971), pp 519-528.