1514454

Mesh Generation By Domain Bisection

Peter James Lawrence

A thesis submitted in partial fulfilment of the requirements of the University of Greenwich for the degree of Doctor of Philosophy.

March 1994

This research programme was funded by SERC

Centre for Numerical Modelling and Process Analysis
School of Mathematics Statistics and Computing
University of Greenwich
London U.K.

V 8350975

Abstract.

The research reported in this dissertation was undertaken to investigate efficient computational methods of automatically generating three dimensional unstructured tetrahedral meshes.

The work on two dimensional triangular unstructured grid generation by Lewis and Robinson [LeR76] is first examined, in which a recursive bisection technique of computational order *nlog(n)* was implemented. This technique is then extended to incorporate new methods of geometry input and the automatic handling of multi-connected regions. The method of two dimensional recursive mesh bisection is then further modified to incorporate an improved strategy for the selection of bisections. This enables an automatic nodal placement technique to be implemented in conjunction with the grid generator. The dissertation then investigates methods of generating triangular grids over parametric surfaces. This includes a new definition of surface Delaunay triangulation with the extension of grid improvement techniques to surfaces.

Based on the assumption that all surface grids of objects form polyhedral domains, a three dimensional mesh generation technique is derived. This technique is a hybrid of recursive domain bisection coupled with a min-max heuristic triangulation algorithm. This is done to achieve a computationlly efficient and reliable algorithm coupled with a fast nodal placement technique. The algorithm generates three dimensional unstructured tetrahedral grids over polyhedral domains with multi-connected regions in an average computational order of less than *nlog(n)*.

Acknowledgements

The author would like to express his gratitude to his supervisors, Professor M. Cross and Professor M.G. Everett, who provided him with guidance and the opportunity to work in a friendly environment.

Thanks also go out to the staff at the School of Mathematics, Statistics and Computing, and to the postgraduates at the Centre for Numerical Modelling and Process Analysis of the University of Greenwich, for their assistance and encouragement, and for providing a good working environment.

Finally, the financial support provided by the Science and Engineering Research Council is gratefully acknowledged.

Table of Contents

Part I

Introduction and Overview.

This dissertation is divided into three parts, in which this section will give an introduction to the aims and aspirations of the research. This section will also include an overview of current techniques with a discussion of related topics.

Chapter 1

Introduction.

1.1 Introduction.

Advances in engineering software, fuelled by hardware improvements, have led to an increased desire to model more complex geometries. This has resulted in a bottle neck of generating good quality three dimensional unstructured meshes for the analysis of these domains using methods based upon Control-Volume and Finite Element procedures.

Current 2D mesh generation methods tend to rely on interaction between the user and the mesh generating software to produce well structured meshes; this is much more difficult, and sometimes impossible with 3D mesh generation since there are still large visualization problems to overcome. In Chapter 2 the problem of visualization of three dimensional grids is discussed, together with how to evaluate grid quality before it is utilized for any further computational purposes.

There have been several methods applied to the problem of generating three dimensional meshes for complex geometries, such as the Advancing front [BoP91][Lo85], Octree [ScS90] and Delaunay [CFF85] triangulation techniques. These methods tend to be CPU intensive and often require large amounts of user interaction. A brief overview of these techniques together with examples, are covered in Chapter 3.

The aim of the thesis is to present a computationally efficient, reliable and good quality three dimensional mesh generation program using techniques that have an average computational order of *nlog(n)*. In Chapter 4 the method of *"recursive domain bisection"* mesh generation by Lewis and Robinson [LeR76] is outlined along with the modifications and extensions that have been applied.

Chapter 5 looks at the problem of generating grids over surfaces and outlines how 2D techniques can be extended. This chapter includes a new definition of parametric surface Delaunay triangulation and various *grid improvement* techniques for surface meshes.

Chapter 6, describes the initial attempt at recursive three dimensional mesh generation and how these ideas have been modified to form the current fully working technique. The following Chapter 7, describes the fundamental algorithms used in conjunction with the mesh generator, presented in Chapter 6. A new direct boundary constrained local min-max meshing algorithm, that is based on the Delaunay triangulation algorithm by Joe [Joe89], is also described.

In Chapter 8, the thesis then presents some example geometries and grids, with CPU times and various mesh quality measures. Overall conclusions and possible extensions to hexahedral element generation are presented in Chapter 9.

1.2 Problem reduction techniques.

The reduction of a model into simpler parts is fundamental to mesh generation. This is reflected in many methods, such as Octree [ScS90] and Medial axis [GüP91], which utilize a problem reduction technique to sub-divide the geometry into simpler regions, to enable the generation of the final mesh.

Problem reduction techniques, such as the Quicksort [Hoa61], which apply recursive methods to reduce the data space to sufficiently small segments so that a simple algorithm may be applied, have traditionally been more computationally efficient than alternative algorithms. The Quicksort is an order *nlog(n)* method [Hoa62], where the problem of sorting a vector is reduced to sorting shorter and shorter vectors, until vectors of length two are reached. These can then be sorted by one comparison and a conditional interchange. Lewis and Robinson [LeR76] extended this idea to two dimensional unstructured triangular grid generation, which resulted in a computationally efficient algorithm of order *nlog(n)*. This method of mesh generation, using their technique, forms the fundamental idea behind this research and is briefly outlined here. A fuller description is given in Chapter 3.

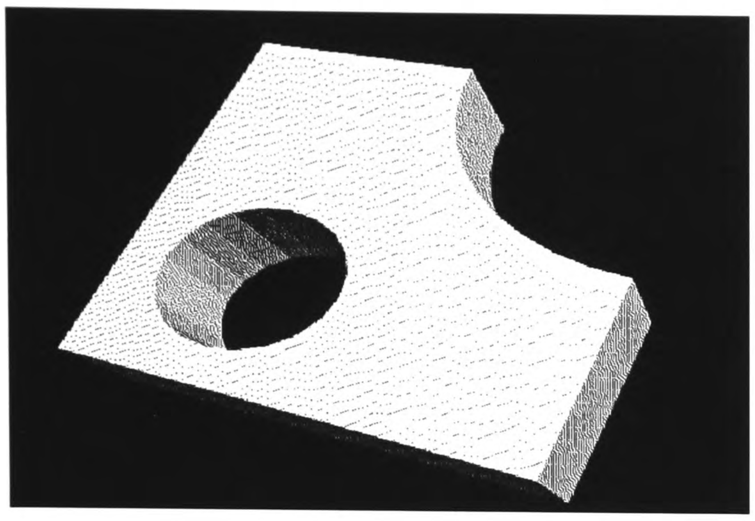
The method of Lewis and Robinson is a two dimensional technique, see Chapter 3 section 3.7. Since this thesis is primarily concerned with 3D geometry, the basic approach is depicted in Figure 1.2.1 with a three dimensional domain. The basic philosophy behind this technique for meshing a region R, see Figure 1.2.1a, is as follows:

(a) Splitting R into two sub-regions,R_1 and R_2, by choosing a plane of *best split.* A new boundary is generated across the interface of the regions to create two new closed independent domains, Figure 1.2.1b.
(Note this initial cut has a zigzag appearance as the splitting routine follows a path through the surface mesh closest to the cutting plane.)

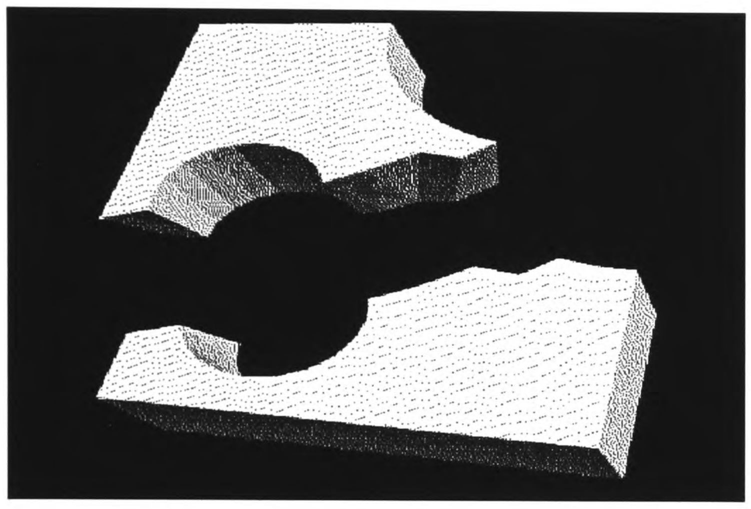
(b) Now solve the triangulation problem for R_1 and R_2

Step *a* and *b* are applied recursively until tetrahedral domains are formed, as in Figure 1.2.1c which contains no interior points, these being the elements of the mesh. Tetrahedral elements which contain internal points and sub-region for which no valid bisection exists, are dealt with by special algorithms. When all the sub-regions are resolved into tetrahedral elements the mesh is complete, Figure 1.2.1d.

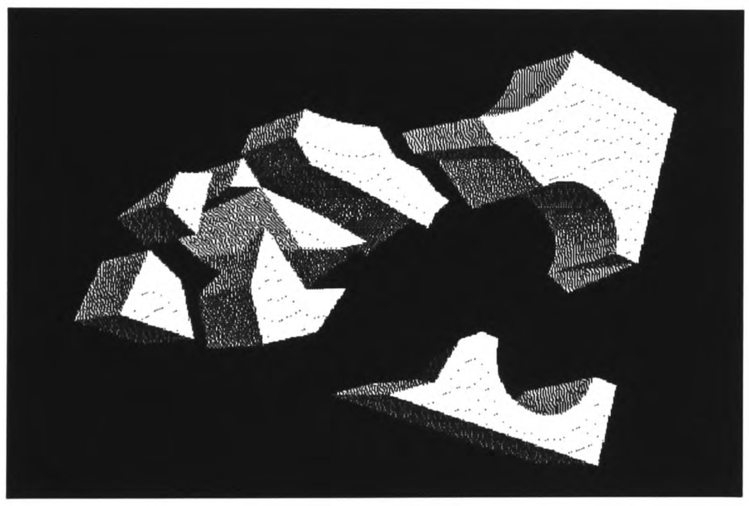
Figure 1.2.1: Example of mesh generation by recursive domain bi-section [Law91]



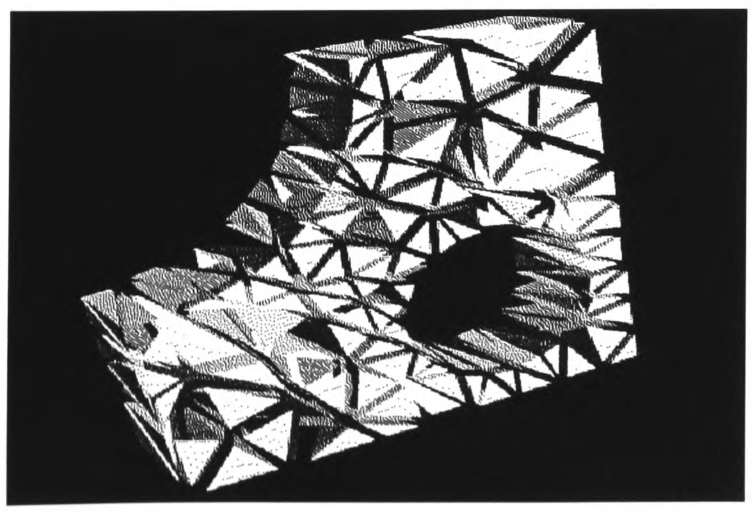
(a) Initial Domain



(b) Then apply first bi-section on domain.



(c) Then keep on applying bisections to domain.



(d) Until tetrahedra are formed. Hence the final tetrahedral unstructured mesh is generated.

1.3 Why Order of Execution is Important

Most of the algorithms in this thesis have an order of execution that fall into the following classes:

constant	: Order 1
log log	: O(lg lg n)
linear	: O(n)
n log n	: O(n lg n)
quadratic	: $O(n^2)$
cubic	: $O(n^3)$
exponential	: $O(2^n)$

The parameter *n* is a value that characterizes the size of the input to a given algorithm, and if we say the algorithm runs to completion in *O(f(n))* steps, we mean that the actual number of steps executed is no more than a constant times *f(n)*, for sufficiently large *n*. It is important to gain an intuitive feeling for these classes in order to have a comparative framework in which to understand performance properties of algorithms. Figure 1.3.1 shows the above functions plotted against CPU.

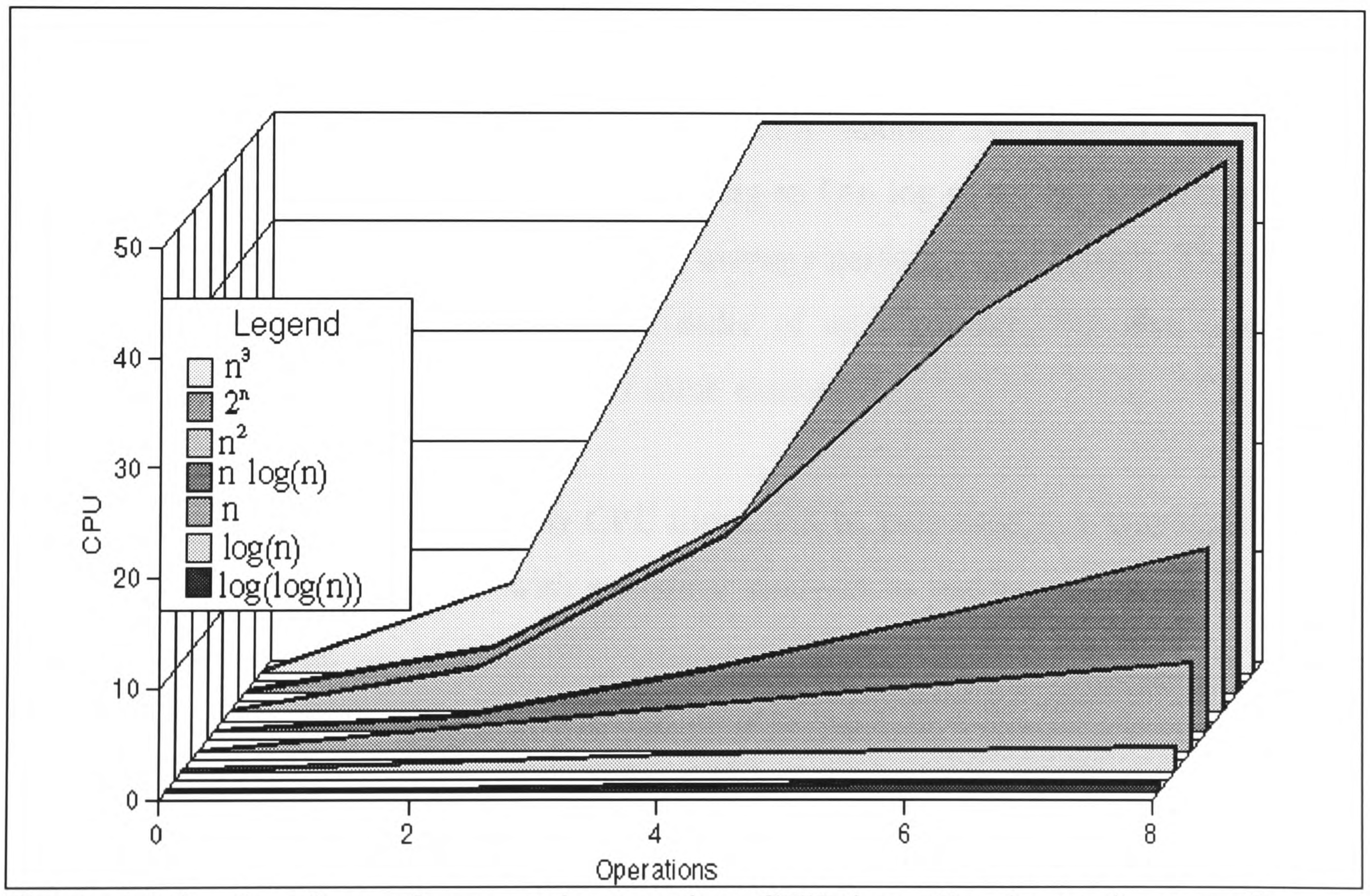


Figure 1.3.1: functions of n operations against CPU time. Graphs are in ascending order with O(lg lg n) at front and $O(n^3)$ at back as in legend.

As can be deduced from the Figure 1.3.1, for small problems the order of the algorithm is not important, but as the size of a problem increases the time difference between routines can become significant. If we had a problem that required over a million operations, a function of even O(n^2) would take over 7000 times longer than an O(n lg n) process. Therefore, if the running time of an algorithm is characterized by an exponential function, we cannot expect to solve practical problems of very large size. In 3D mesh generation even very modest problems are in the region of over a thousand nodes, so an algorithm that is anywhere near exponential is not practical.

A major problem is that most algorithms often do not fall precisely into anyone class. The order of most routines often depends to some degree on the form in which the data is presented to them or the complexity of the particular problem they are applied to. A common approach is to categorise an algorithm by its *worst* case and/or *average* situation.

If for example we compare two routines the Quicksort [Hoa62] and the Heapsort [Knu73], both these routines are reported to be of order n log(n) [Tho80]. However, the Quicksort is in fact only on average O(n log n) and is O(n^2) steps in the worst case where the initial distribution of the data is extremely random. The heapsort, on the other hand, is a routine that has the advantage of being an O(n log n) sorting algorithm, whose worst case performance is fairly close to its *average* performance [Tho80]. Therefore, it is often not just sufficient to quote the order of an algorithm, but also a standard deviation to address the above issues to some extent.

Throughout this thesis, many CPU times will be presented in a graphical format, and also may be accompanied with statistical analysis to address the above issues, at least, to some extent.

Chapter 2
3D Geometry, Visualization and Element Shape Measures in Mesh Generation.

2.1 Geometry for mesh generation.

The initial stage of an analysis of any model is the generation of the geometry. In three dimensional geometry solid modelling, there are many different ways of representing objects. The geometry representation of a model has a great effect on the types and form of geometry operations that can be applied, and therefore has an effect on the mesh generator. The mesh generator cannot be designed independent of the object definition and the topic is, therefore, discussed in this section.

Many geometry representation techniques have emerged due to the difficulties of perceiving a *real* physical object within the constraints of the virtual world of the computer. However, recently two main approaches have dominated, namely, constructive and surface representations.

2.1.1 Constructive models.

All constructive models consider solids as point sets of E^3. Their basic idea is to start from some sufficiently simple point sets that can be represented directly, and model other point sets in terms of very general combinations of the simple sets.

The main technique in this class is constructive solid geometry (CSG) where parameterized instances of *solid primitives* and boolean set operations are implemented.

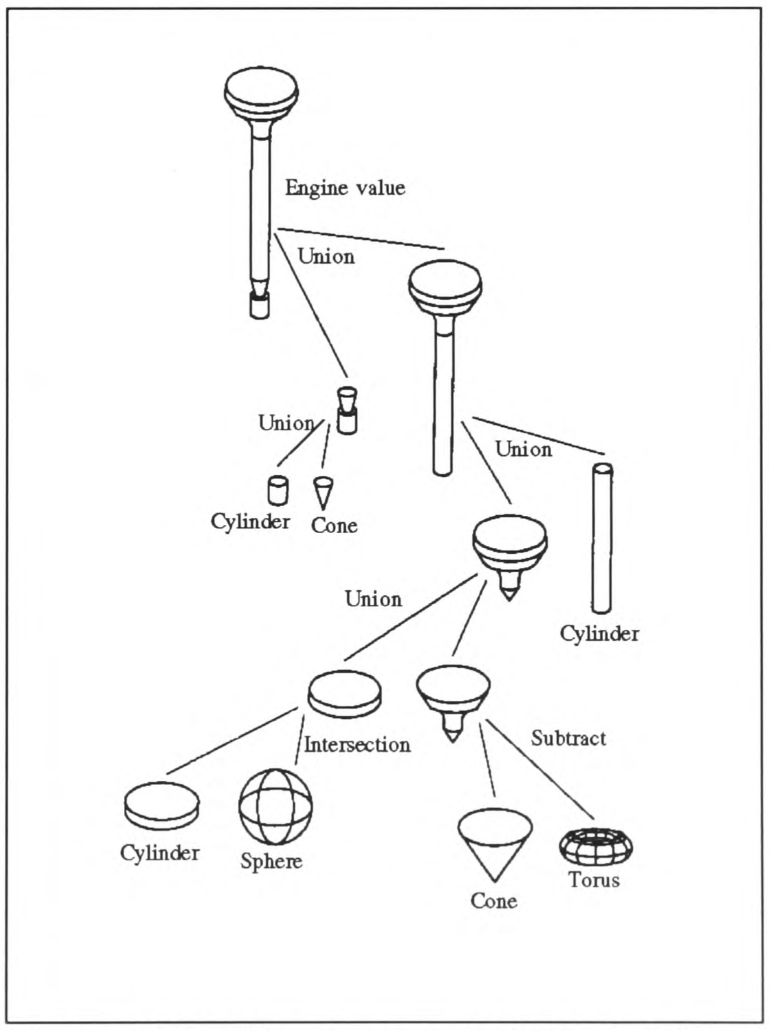


Figure 2.1.1 Binary tree of CSG model.

Figure 2.1.1 illustrates an engine valve generated using boolean operations applied to a set of primitives.

CSG modelling packages are often a useful and fast way of generating many machined parts. However, the user has no direct access to individual half-spaces (graphical primitives) and this can restrict the designer. An example is in aircraft design where curved surfaces on wings can be difficult to model.

2.1.2 Surface based models.

The surface based characterization of solids, looks at the boundary of a solid object. The boundary is considered to consist of a collection of *faces* that are *glued* together so that they form a complete, closing *skin* around an object. Figure 2.1.2A illustrates a *box object* represented by a collection of polygon faces, Figure 2.1.2B shows the same box with its faces separated.

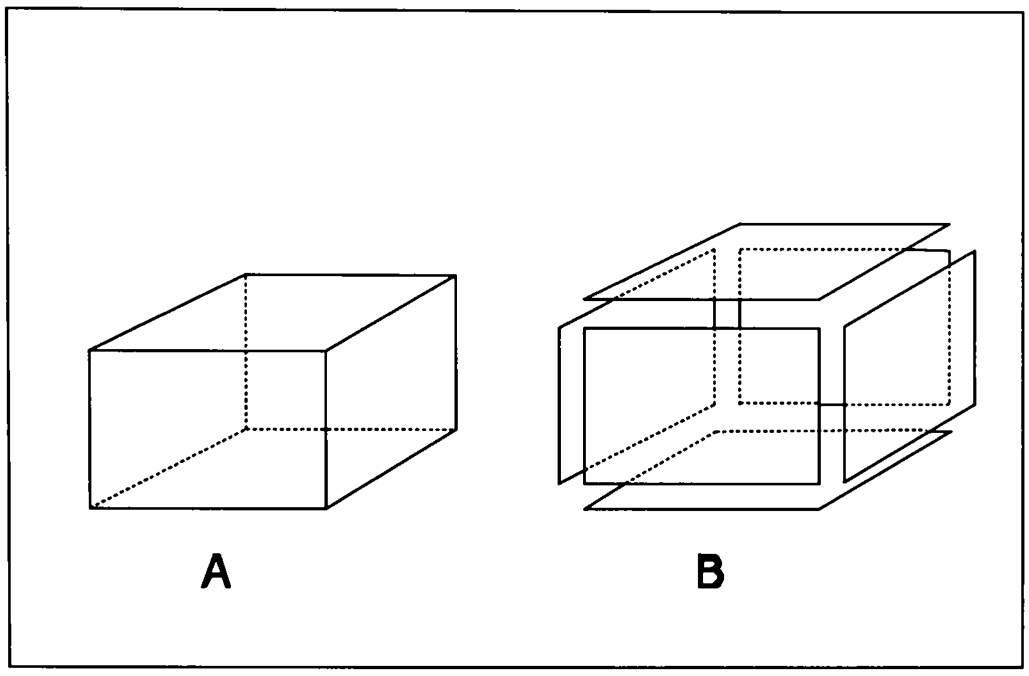


Figure 2.1.2 Boundary model of a box.

Many boundary modelling packages also encompass curved surfaces. These curved surfaces are often parametric patches that are manifolded together. Parametric patches include bilinear surfaces [Dew88], coons patches [Gas83], cubic patches [Dew88], Bezier surfaces [BaB83] etc, which can be defined using a number of control points. Recently NURBS [Pie91] (Non-Uniform Rational B-Spline) surfaces have made an impact in this area and are used widely in the aircraft and car industry.

A large number of objects can be represented using a boundary model technique, but these models are often difficult to generate. To assist in the generation of these models, research has been invested in new curved surface representations and a number of CAD packages have been developed. Since many objects can be quickly represented using CSG techniques, many modern boundary modelling packages incorporate some CSG features and provide predefined surface primitives such as sphere, torus etc. This has resulted in many hybrid modelling tools.

The type of model representation used affects the type and efficiency of operations carried out on the domain. This in turn affects the reliability, speed and type of mesh generation technique that can be applied to the region. A surface mesh is a boundary model of a domain. Therefore, boundary surface representations of models make a natural choice as the starting point of grid generation and many CSG models can generate output in this format.

2.2 Visualization of 3D geometry and meshes.

Visualization of geometry on the two dimensional device of a cathode ray tube provides its own problems. Complex models that are highly re-entrant with many cavities and sub-domains, such as those found in the casting industry, are difficult to perceive on the computer screen, often requiring many different viewing angles of the model to be displayed simultaneously. Frequently a number of slices through the domain are required to show any hidden features and cavities. This problem is particularly acute in the generation of geometry, in which the model has to be manipulated into a particular angle and location before a new facet can be generated manually by the designer. Many other fields, such as contouring [Sab85], have suffered from the problem of visualization.

A three dimensional mesh, especially an unstructured mesh, is a complex geometry with many features hidden below the surface skin of the domain. A number of techniques have been applied to try and display the hidden detail of a mesh. Such techniques include domain slicing [Bur90] and element shrinking [Law91][TaA91]. In domain slicing a number of planes are passed through the domain to try and expose some of the internal mesh features. However, this can present a false picture, depending on how individual elements are bisected by the cutting plane, giving an impression of regions of the mesh being of finer or coarser density than they really are. The method of element shrinking reduces all the elements' size by a given amount ς but keeping their centroids fixed. This results in small gaps being created between the elements. Both methods do little more than prove the existence of a grid, they provide no information on element quality and whether elements intersect.

A number of highly complex CAD and visualization packages have to be used in the course of grid generation. Visualization of complex models has proven to be such a difficulty that a new generation of packages have been developed to try and address some of the above problems. The next two pages depict illustrations from apE [Bro92] and AVS [Bro92], which are advanced visualization packages used throughout this thesis for the generation of many of the illustrations. They are pipe line systems in which a user builds up a network of operations that are required for a particular visualization task.

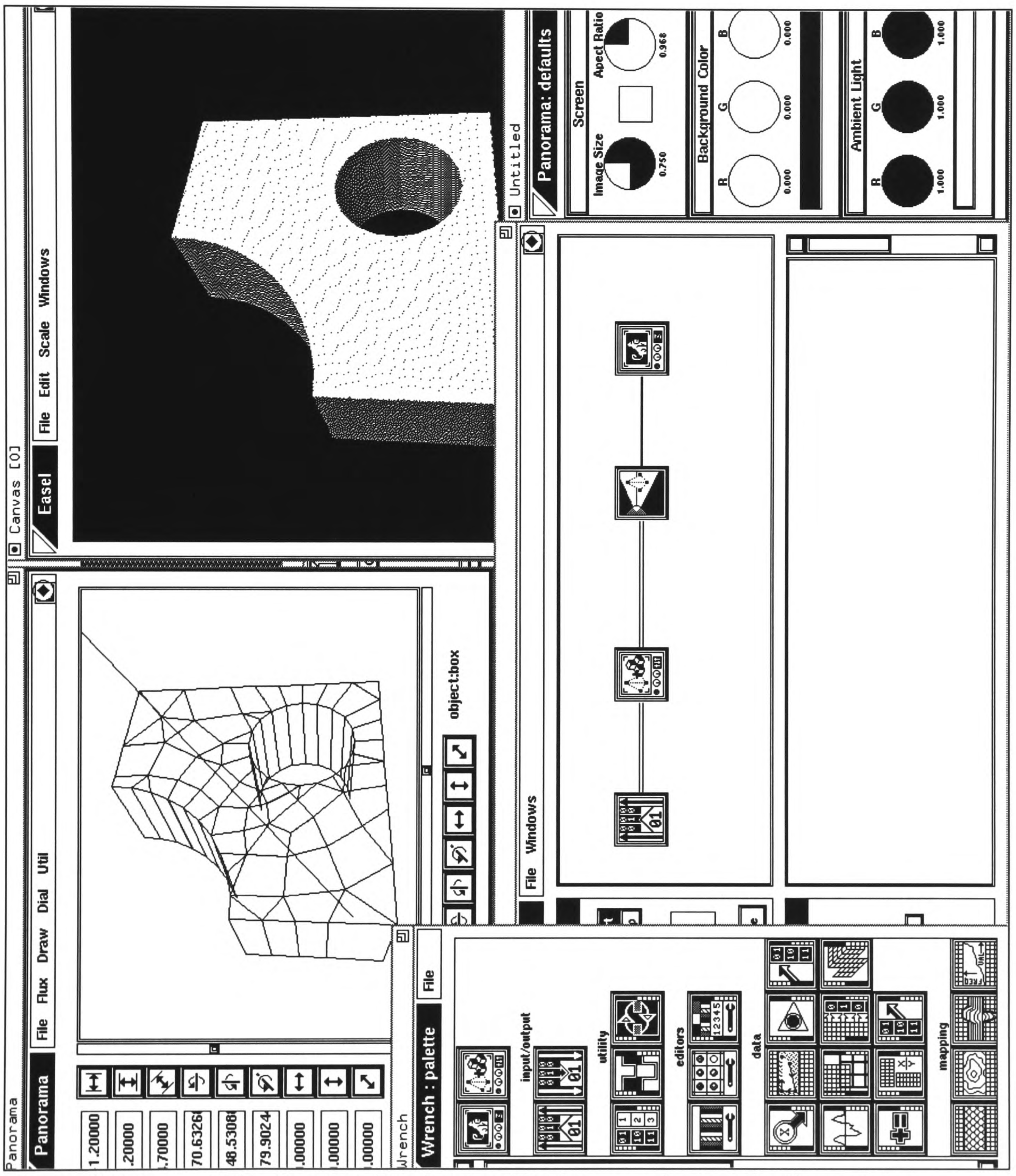


Figure 2.2.1: apE (Animation Production Environment) visualization package.

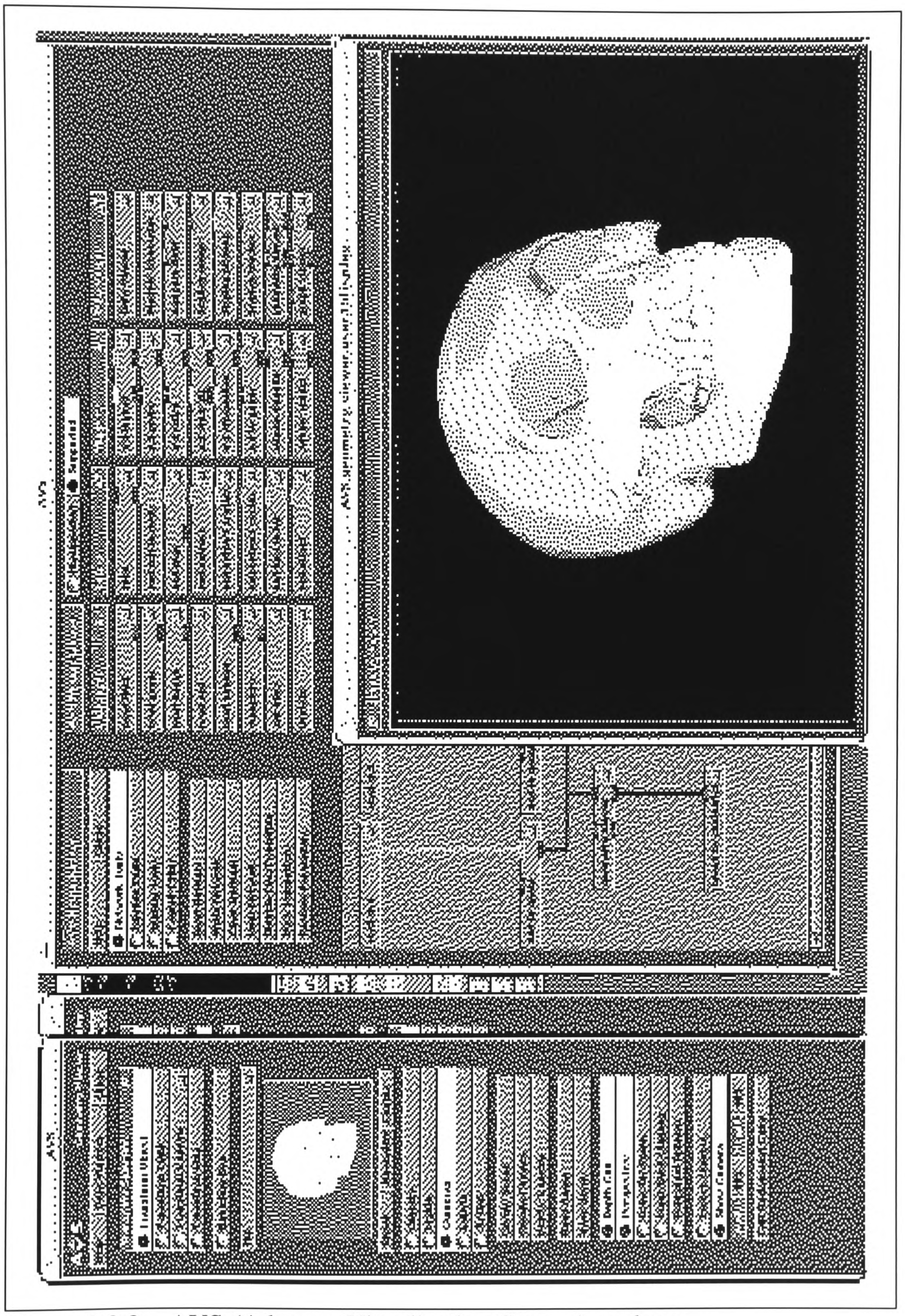


Figure 2.2.2: AVS (Advance Visualization System) package.

2.3 Constructive Solid Geometry (CSG).

The geometry input format for the new bisection mesh generator is polyhedral domains; the reason for this is discussed in Chapter 6. CSG Modelling packages are often a useful and fast way of generating many machined parts, and they provide a convenient method of output in the form of polyhedral surfaces. The drawback of using these polyhedral domains generated in this fashion is that the polyhedral faces are often degenerate and elongated. Sometimes the polyhedral faces can be of a magnitude that is smaller than the element size required for the mesh. Even the order in which primitives are combined have an effect on the form of polyhedral domains generated. Below are three identical examples of a pipe like component generated by different combinations of CSG operations and the resulting polyhedral domains generated.

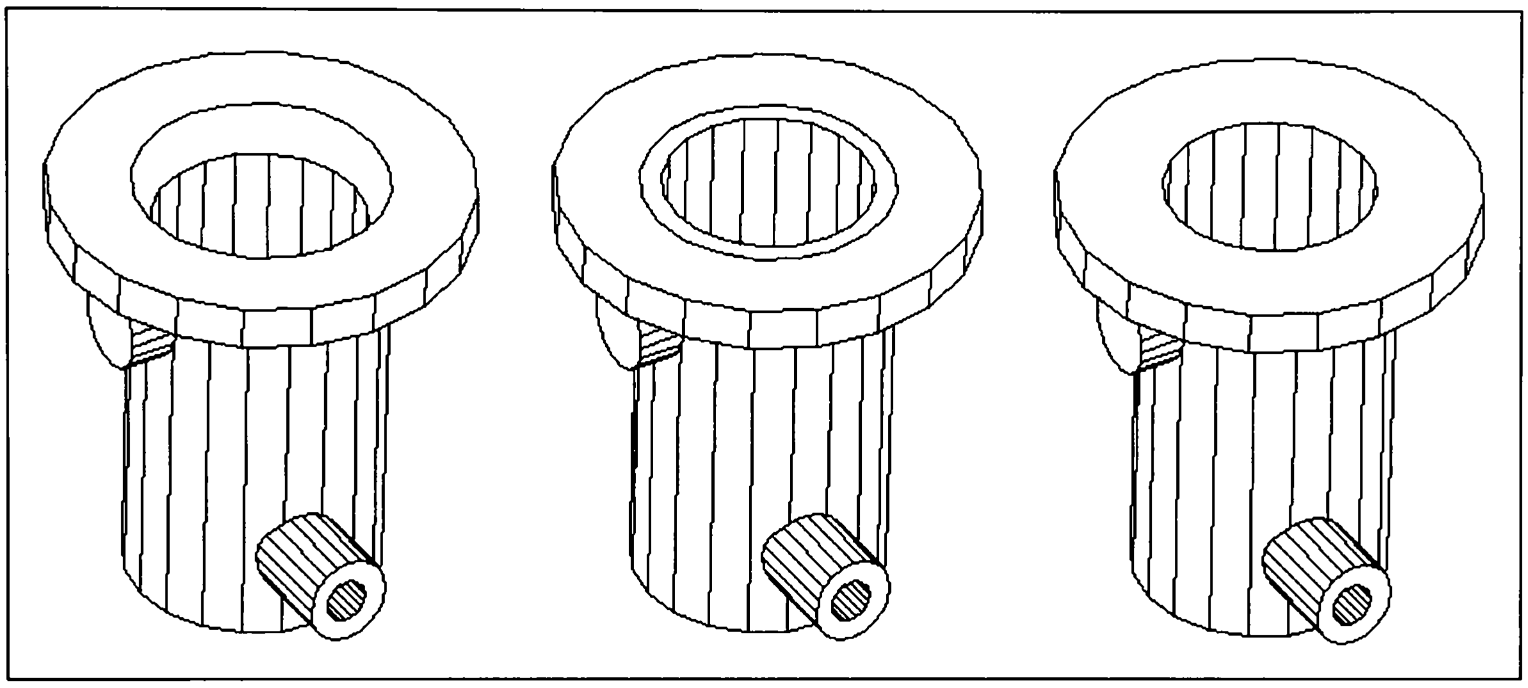


Figure 2.3.1 : Three identical pipes with different polyhedral definitions; this is especially prominent around top flange of pipes.

The figures generated in the above diagram were displayed without internal lines, these are extra edges added to the domain by the CSG model to ensure that all faces are valid planar polygon surfaces. In this particular modeller the polygon elements had to be convex, since this speeds up most ray tracing and hidden line removal algorithms. Since this simplification of the surfaces is for applications where the quality of the elements is not essential, this often results in very poor surface elements (Figure 2.3.2).

This problem has often been encountered during this research. As a consequence, several algorithms have been derived, which take a polyhedral domain and by joining faces and swapping vertices improve the initial surface elements. This has worked to some degree, but it is often almost impossible to remove all poorly defined elements.

The problem with most CSG modellers is that the polyhedral domain sub-division is done for speed, rather than for the quality of the bisected surfaces. The algorithm used within these CSG packages, from the experiences gained during this research, for polyhedral convex subdivision are very similar to the algorithms used within the mesh generation code. However, the grid generation code is more selective about which bisection edge is used to divide the domain. Therefore, for most CSG packages, only a small modification is necessary to generate reasonable surface elements.

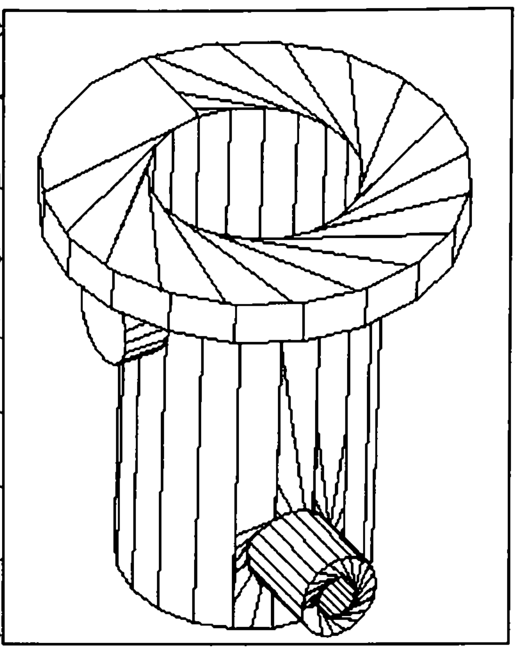


Figure 2.3.2:Typical polyhedral domain.

CSG software tools are often geared towards object visualization, therefore they often incorporate utilities to aid in this task, such as tools to guide the resolution of curved surfaces. The resolution parameter, for example, on a cylinder would increase/ decrease the number of polygons used to represent the outer perimeter, just as in the case of a circle, the more straight lines used to represent it, the better the definition. This resolution factor can, in effect, help to guide the meshing algorithm nodal placement. Hence, if the designer had requested a higher definition on a surface they would probably require a denser mesh over that region, and vice-versa for a coarser resolution factor.

The conclusion which can be drawn, from CSG geometry modellers is that they tend to provide the necessary information for generating a three dimensional grid, but the quality of the output often leaves a lot to be desired and generally requires some manipulation. However, these problems could be overcome by a small modification to the CAD package, to gear it more towards grid generation rather than just visualization.

CSG modellers have intrigued Software Engineers to such an extent, that there is currently work being undertaken which integrates CSG directly with meshing routines [Cox93]. This method which is called Domain Composition builds the mesh simultaneously as the model is being created. Each primitive object has a predefined 3D grid. For example, Figure 2.3.3, if we have a region *D*, which was formed by a Boolean operation on the domain *A* and *B*. The mesh over the region *D*, is formed by taking the original grids of *A* and *B*, and then applying the same Boolean operation with the use of grid interpolations, where necessary. However, in Lee's thesis [Lee81], he argues that this technique is not a practical method for the generation of three dimensional meshes.

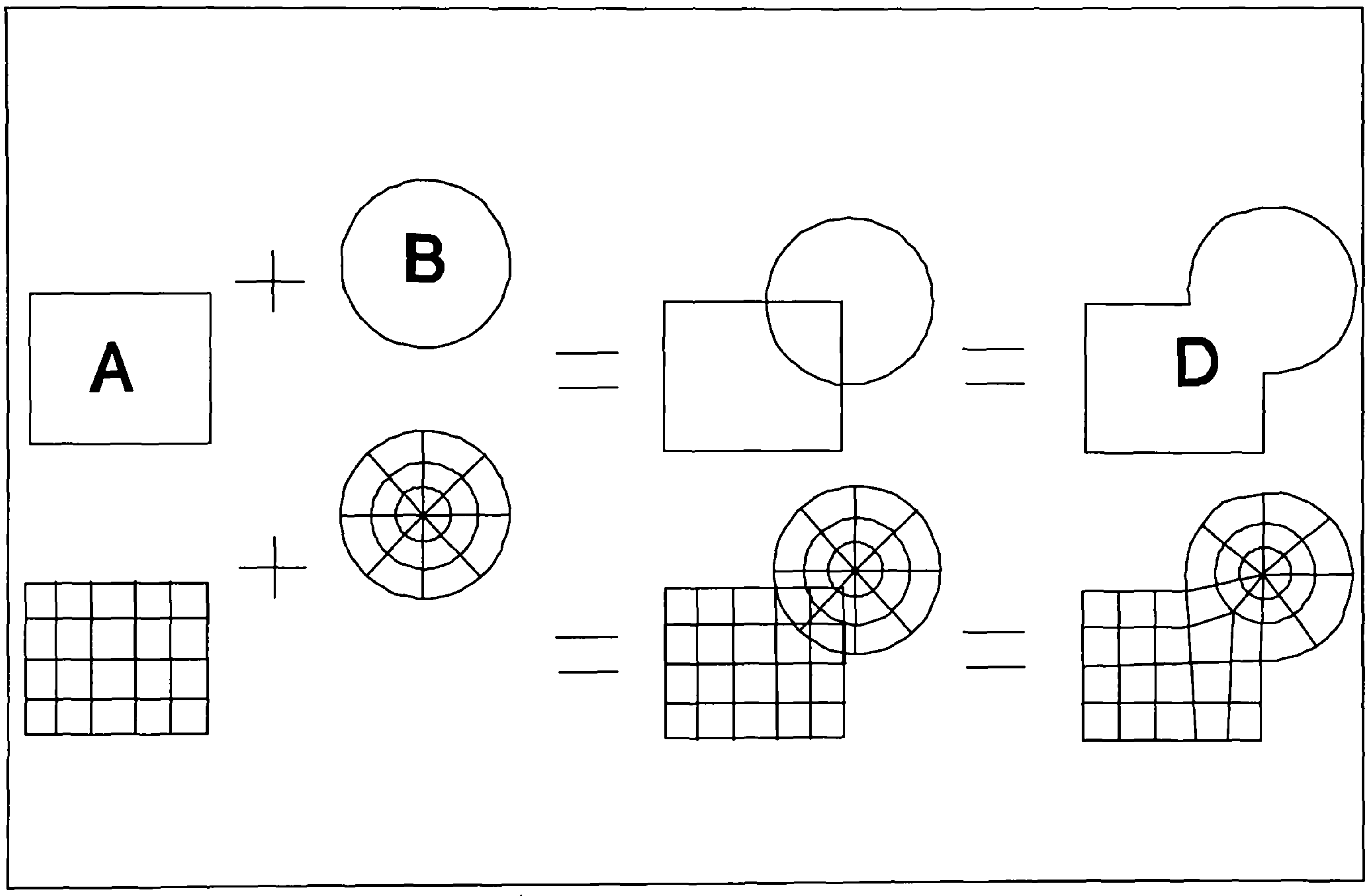


Figure 2.3.3 : Domain Composition

2.4 Tetrahedral shape measures.

One of the main problems in tetrahedral mesh generation is how to measure the quality of a mesh, since poorly shaped tetrahedra may cause numerical difficulties in the under lying numerical technique, e.g finite element analysis. Papers on tetrahedral mesh generation have used various quantities for measuring the shape or quality of tetrahedra. In this section two approaches will be described.

2.4.1 Solid angle.

In 2D triangulation mesh generation, the minimum interior angle of a triangulation is a commonly used triangle shape measure. A natural extension of the minimum interior angle to three dimensions is the minimum solid angle θ_{min}.

Unlike a triangle a tetrahedron has many different angle measurements :

12 planar angles (three in each of the 4 faces),

6 dihedral angles (one at each of the 6 edges),

4 solid or dihedral angles at the vertices.

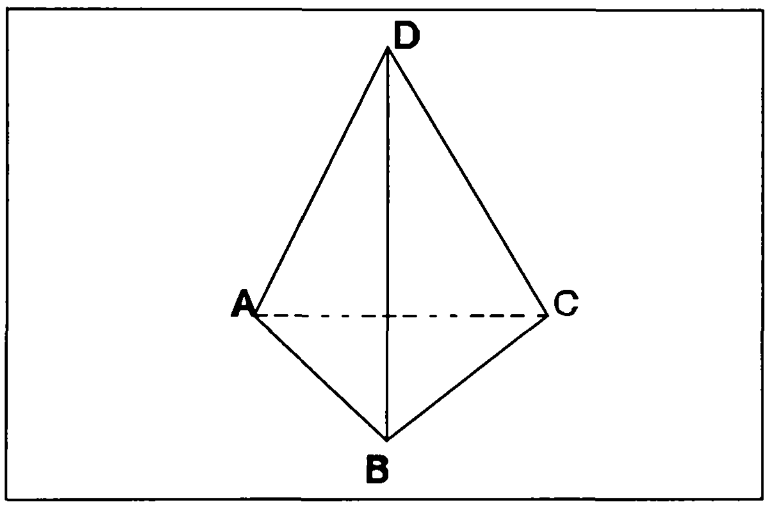


Figure 2.4.1 : tetrahedron

The solid angle θ_i at v_i is the surface area formed by projecting each point on the face not containing the vertex v_i onto the surface of the unit sphere with v_i at its centre. However, for a tetrahedron the solid angle at D, Figure 2.4.1 can be computed as $\alpha+\beta+\gamma-\pi$ [Bey81], where α,β and γ are the dihedral angles at edges AD, DB and CD respectively.

It can be shown [Gad52] that $0 \leq \sum_{i=0}^{3} \theta_i \leq 2\pi$. Therefore a very large solid angle, near 2π, for a tetrahedron implies that there also exists a small solid angle, and this is the reason why we only consider the minimum solid angle. Also if the tetrahedron is regular, all face angles are $\pi/3$ and all solid angles are the same.

2.4.2 Tetrahedral goodness function.

An alternative way of measuring mesh quality is to use a tetrahedral *goodness function* or *Gamma value* [ShL91]:

$$\lambda_i = \beta \frac{\Gamma_i^2}{\delta\Gamma_i^3}$$

Where :

λ_i	is the element's normalized shape parameter for tetrahedron i.
Γ	is the volume of tetrahedron i.
$\delta\Gamma_i$	is the surface area of tetrahedron i.
β	is a normalization factor[Shl91] (374.123) which yields λ_i =1 for an equilateral tetrahedron.

The above equation returns a value of 1 if the tetrahedron is equilateral. As the tetrahedron deviates from the ideal shape so does the value of λ_i, the larger the deviation of λ_i from 1 the poorer the element quality. A λ_i value above 0.8 is considered to represent an extremely good tetrahedral element [ShL91].

Both the tetrahedral *"solid angle"* and *"goodness function"* offer practical measurements for measuring mesh quality. These measures are only a guide, and the only true mesh quality test is to use the grid for analysis of the domain. However, they do offer a quick quality measure and a means of comparing different grids over the same geometry model. Throughout this thesis the results from the grid generator will be presented using both the above tetrahedral shape measures.

2.5 Summary

The problem of representing complex three dimensional models has given rise to a number of alternative techniques for the representation of geometry. The technique of mesh generation must be considered in conjunction with various geometry representations. A number of software tools have been developed for the generation and representation of three dimensional geometries, however they are often not designed for providing suitable geometry models for computational analysis.

Many problems exist in measuring the quality of three dimensional unstructured grids, and visual techniques cannot practically be applied. Therefore, several computational methods of measuring mesh quality do exist, of which two are described in this Chapter. Opinion is still divided over which measure gives the best indication of mesh quality, and research is being undertaken [LiJ93] to establish which technique is best. However, these techniques can only provide an indication to the true mesh quality and a means of comparing different grids over identical geometry.

Chapter 3
Current Major Mesh Generation Techniques.

3.1 Introduction

Many techniques have already been applied to the problem of generating unstructured grids over three dimensional geometry. This chapter will give a brief overview of some of the major methods that have been examined during this research. This chapter is not intended to be a complete overview of all current mesh generation, but rather a subset of the techniques that have, with some degree of success, been applied to 3D mesh generation and to some extent influenced the research described here. This section will attempt to give the reader an idea of the philosophy behind these methods, how they have been applied, together with their advantages and disadvantages.

Two techniques are covered in more depth, *Delaunay [ScS90][ScS88][Joe86] [CFF85][Law72]* and *Binary mesh operators [ShL91]*, since these methods have been implemented in conjunction with the new bisection method, see Chapter 6. Delaunay is of particular interest, as it is the technique that offers the best computational order of the current mesh generation algorithms and forms part of many hybrid mesh generation codes.

The chapter is completed with a description of mesh generation by Recursive domain bisection [LeR76]. It is then concluded with a discussion of the problems of the these techniques and discusses why mesh generation by recursive bisection offers a practical solution.

Key Words: Advancing Front [BoP91][PPF85][Lo85], Delaunay triangulation *[ScS90] [ScS88][Joe86][CFF85][Law72]*, Binary mesh operators [Shl91], Paving [BsC91], Medial Axis [TPA93][TaA91][GũP91], Recursive domain bisection [LeR76].

3.2 Advancing front technique

The advancing front method has been extensively developed by workers such as Lo [Lo85] and Peraire [PPF88].

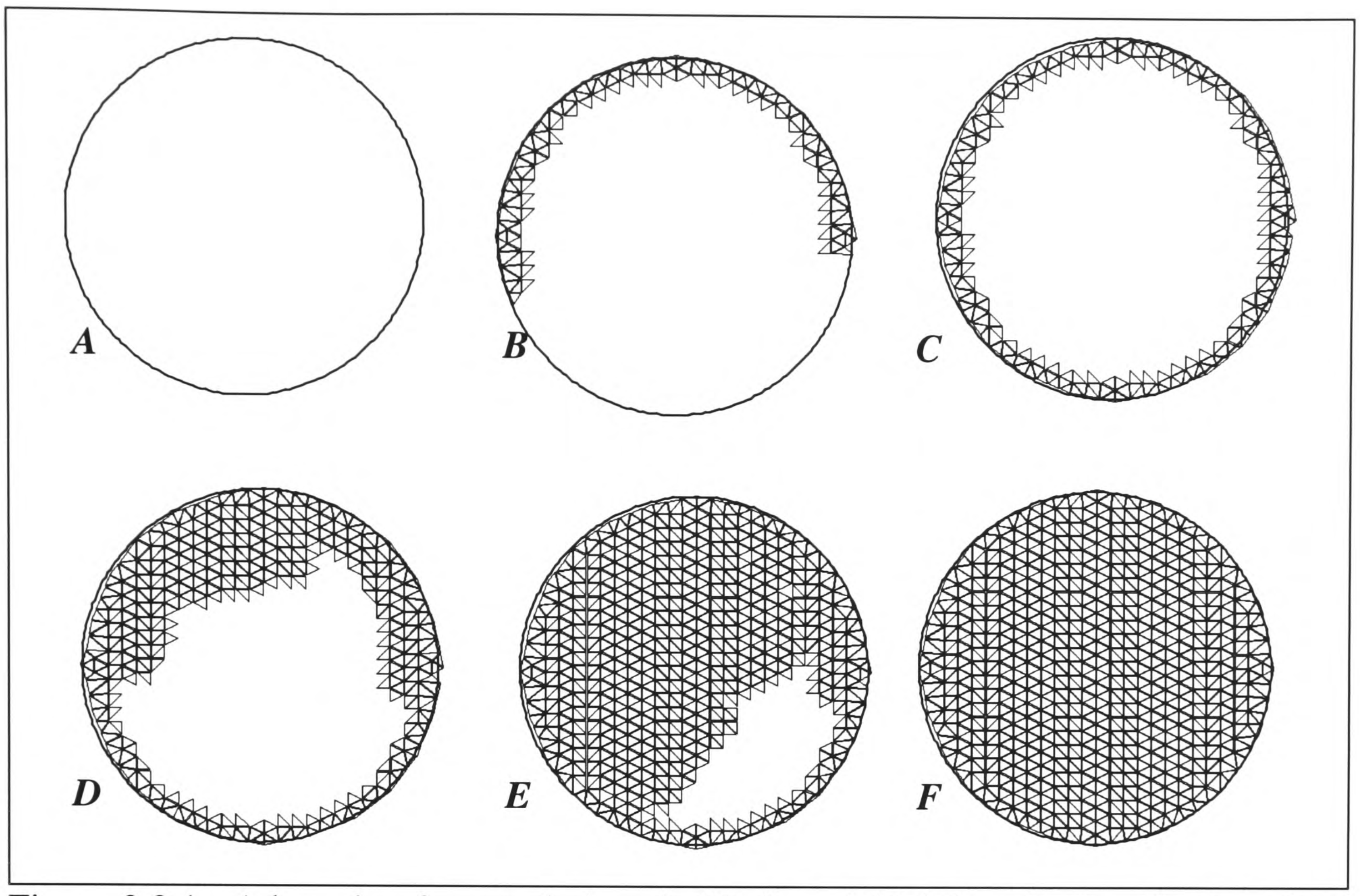


Figure 3.2.1 : Advancing front technique in 2D from initial domain *A* to final mesh *F*.

The basic underlying concept of the advancing front [Lo85][LPG88] method is illustrated in Figure 3.2.1 for the generation of a uniform size triangular mesh over a two dimensional domain. The boundary of the domain to be meshed is first discretized. Points are placed on the boundary, and contiguous points are joined by straight line segments and assembled to form the initial generation front. At this stage the triangulation loop begins. A side from the front is chosen and a triangle is generated that will have this selected side as one edge. In generating this new triangle an interior node may be created or an existing node in the front may be chosen. At this stage it is necessary to ensure that the element generated does not intersect with any existing side in the front. After generating the new element the front is conveniently updated in such a way that it always contains the sides that are available to form a new triangle. The generation is completed when no sides are left in the front.

This method has progressed over the years from a very high order method, above n^2, to around order nlogn, but still remains one of the most CPU expensive methods because of the large number of surface intersections that have to be tested for.

3.3 Binary Mesh Operations.

Mesh generation by binary operations [ShL91][Wei88][Lo88][Wör83], is the implementation of a limited set of geometry operations that are sufficient to generate a complete grid in 2D or 3D. In 3D there are three basic operations that can be used to generate a coarse grid: face removal, edge removal and vertex removal.

Face removal: Carves a tetrahedron from the object being triangulated by the introduction of a new vertex in the interior of the domain.

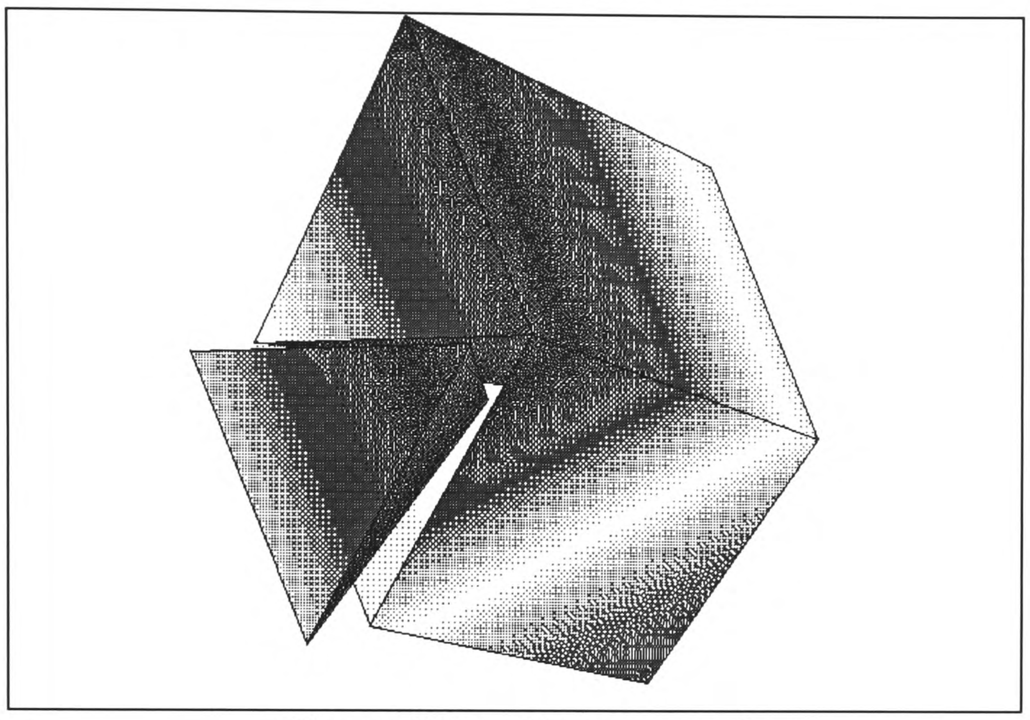


Figure 3.3.1: Face removal

Edge removal: Carves a tetrahedron from the domain by selecting two adjacent non-planar triangular faces and generates a new edge inside the domain.

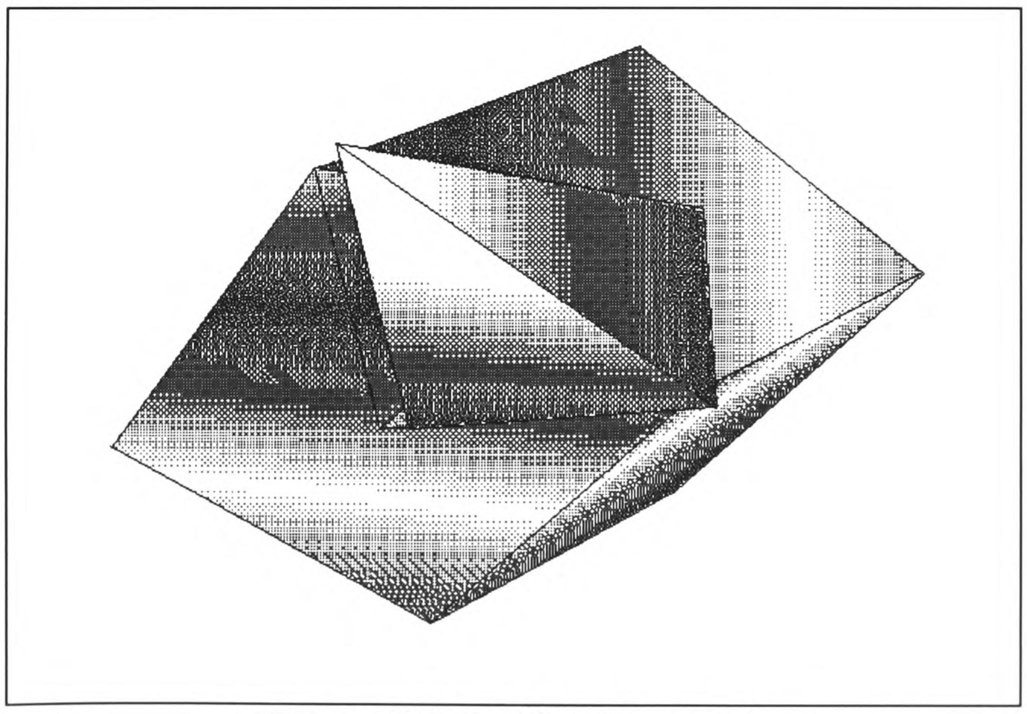


Figure 3.3.2: Edge Removal.

Vertex removal: Carves a tetrahedron from the domain by removing one complex vertex and all its associated edges. (Removes three adjacent faces from the domain)

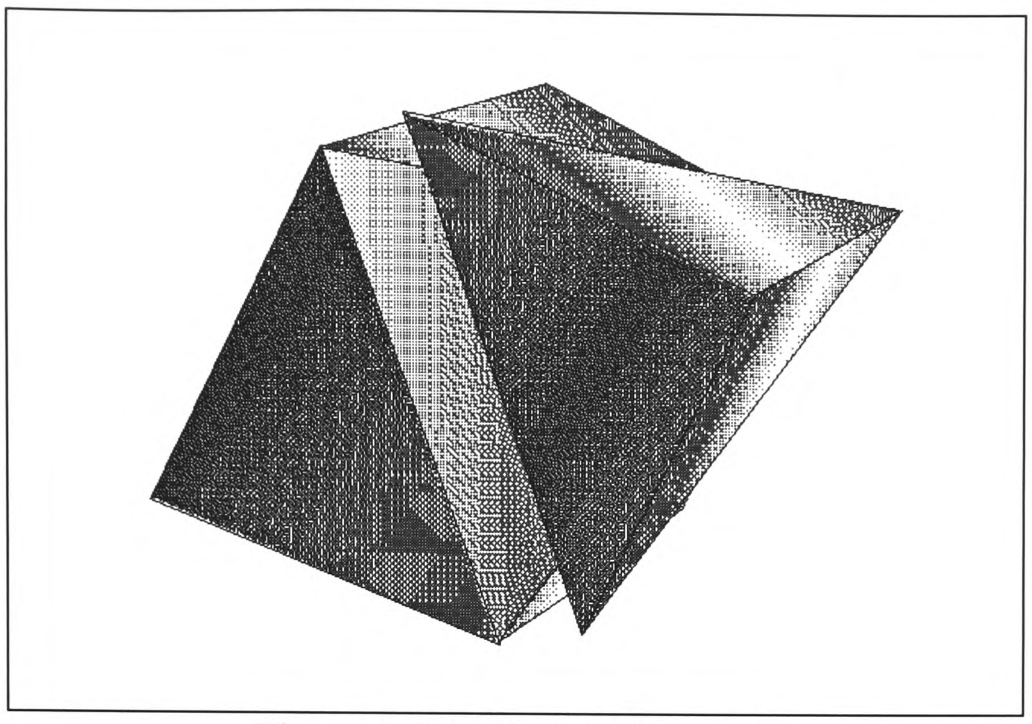


Figure 3.3.3: Vertex Removal.

The method by *Shephard and Lo* [ShL91] applies these operations to generate coarse grids that can be refined later. The algorithm gives each operator a priority based on its ability to reduce the geometric complexity of the domain. The measure of the geometric complexity is the number of topological entities in the geometric model and their adjacencies. Therefore, the routine attempts to use *vertex removal* first on the current geometry. However, if this cannot be applied, it then tries edge removal. Subsequently if an *edge removal* fails, *face removal* is used, which is the only binary operation that can be applied to any geometry. These set of binary operations are coupled with an *element shape* control function in a bid to improve the quality of the final mesh.

Mesh generation by binary operations is strongly related to the advancing front technique with similar draw backs in computational order. It could be argued that these methods are identical except in the priority of applying the mesh operations, i.e advancing front applies face removal to a domain first and if this does not generate any acceptable elements the other binary operations are attempted.

Below is a list of the types of procedures involved in each binary operation.

Face removal:

Choose a polyhedral face.

Generate a point inside the domain.

Point is inward and normal to face.

Test to see if the line from the centre of face to the point, does not intersect any other faces in the domain (may adjust position of point).

Test to see if lines from the nodes of face can be joined up to the new point.

Check newly formed surfaces are not too close to other surfaces in domain.

Check that the new domain does not contain any other domain points.

Check *Gamma value** to see if a good tetrahedon was formed.

Check/correct direction of face normals, of the new face elements.

Edge removal:

Find two adjacent polyhedral faces.

Check that the edge joining non-common nodes is inside domains.

Check that the edge joining non-common nodes does not intersect other polyhedral faces in domains.

Check that the new surfaces do not intersect any other surfaces.

Check that the new surfaces are not too close to other surfaces.

Correct the direction of polyhedral normal.

Check on *Gamma value* of the tetrahedron formed.

Vertex removal:

Find three polyhedral faces that are adjacent to each other.

Check to see that the domain does not contain any other nodes.

Check that the tetrahedron formed is inside domains.

Test to see if the new face is not too close to other faces in the domain.

Correct direction of the new face.

*Gamma value : Tetrahedral shape measure, see section 2.4.2.

3.4 Paving.

The paving method, which has been primarily developed by Blacker and Stephenson [BSC91] is depicted below.

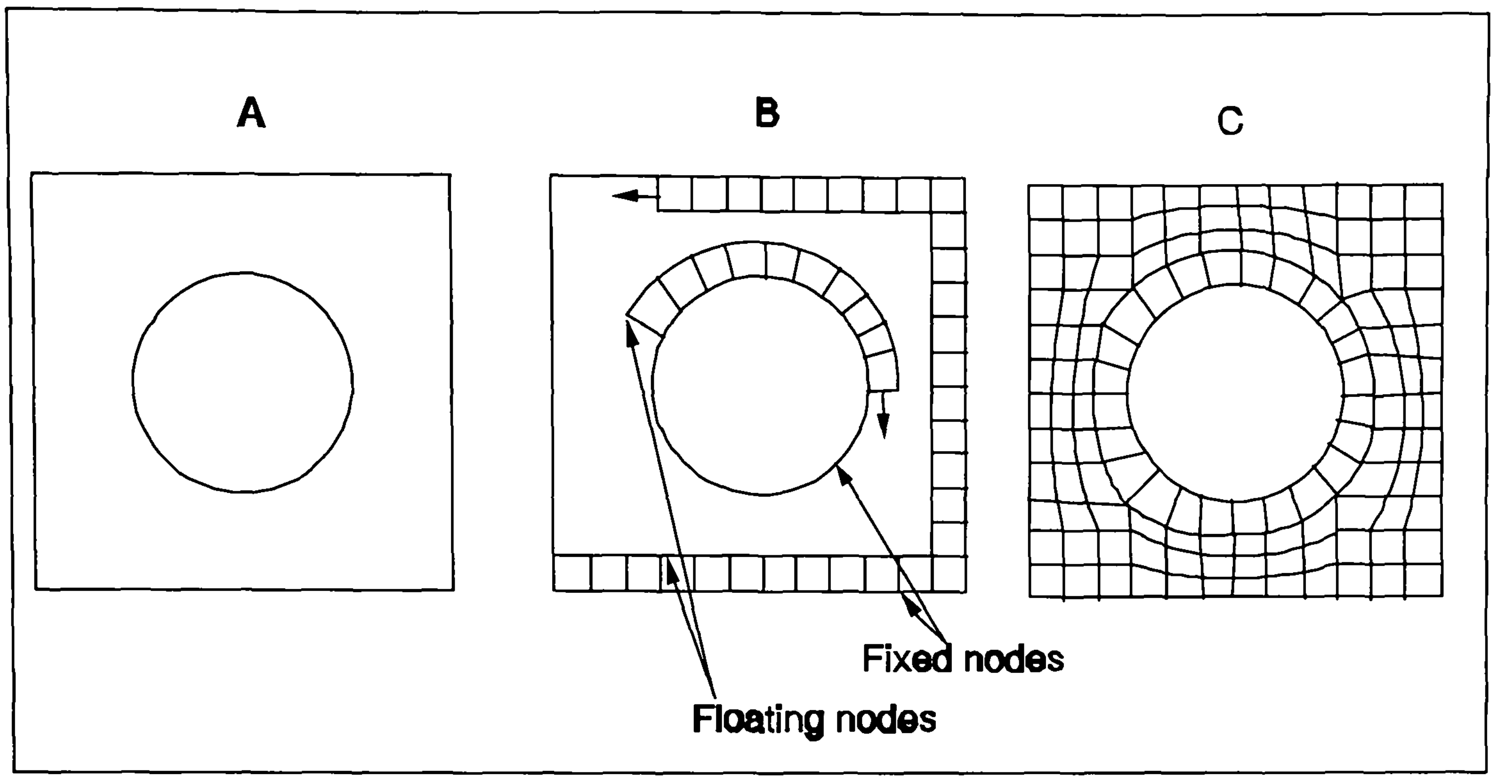


Figure 3.4.1 Example of paving from geometry (A) to mesh (C) [BSC91].

Paving begins with the input of one or more ordered, closed loops of connected nodes, Figure 3.4.1A. These loops form the boundary of the mesh and contain the *fixed nodes*. During the mesh generation process, the paving technique always operates on the boundaries of connected nodes referred to as *paving boundaries*. The paving boundaries are transient in nature and progress as the mesh is generated, Figure 3.4.1B. A point is selected on each paving boundary to start the element paving. The method then walks around the domain, keeping the boundary to its right, generating elements. In Figure 3.4.1B the arrows on the elements' faces indicate the direction of element generation. Each complete loop of elements is called a *row*. Rows are generated from a number of portions. Once a row portion of elements is generated they are smoothed [BSC91], by adjusting nodal positions to improve elements' shapes. If any of the newly generated elements intersect with other rows of elements these are *seamed* or closed by connecting opposing cells. After the completion of each row, it is adjusted to correct for small or large elements, and again checked for intersection.

The paving method has a paving boundary that advances into the domain in a similar way to the advancing front. Therefore, it inherits some of the computational and intersection problems of the foresaid method. Unlike the previous techniques the paving algorithm has the benefit that it generates quadrilaterals and in 3D hexahedral elements.

3.5 Delaunay Triangulation

The Delaunay triangulation in 2D is a well researched method [Wat81],[ScS88] and has been successful in that it has been shown to produce well structured meshes that satisfy the min-max angle criterion (optimal triangles).

The definition of Delaunay triangulation is that the circumcircle of any triangle *i* in the mesh, does not contain any exterior vertices of the element *i*.

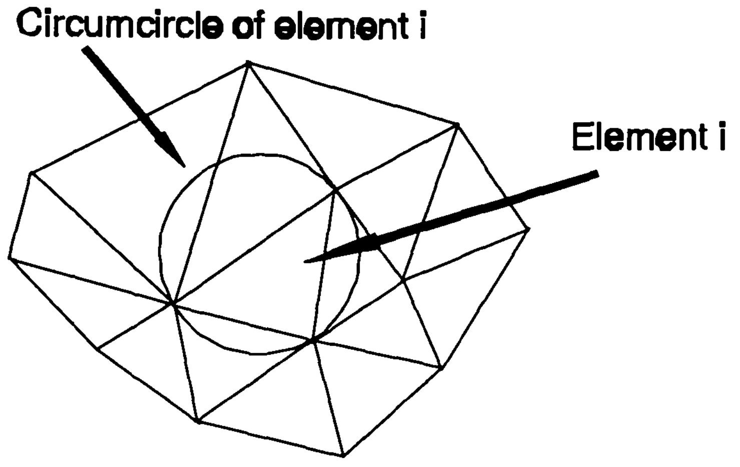


Figure 3.5.1. Illustration of circumcircle of Delaunay triangle.

3D Delaunay triangulation consists of several tetrahedra in an array of points. The four vertices of each tetrahedron lie on the surface of a sphere and no other vertex of the array lies within that sphere. Delaunay triangulation in 3D does not in general satisfy the min-max solid angle criterion and does not seem to satisfy any optimal angle condition. In fact Cavendish [CFF85] reports the creation of slivers (tetrahedron with a small volume, which is almost flat).

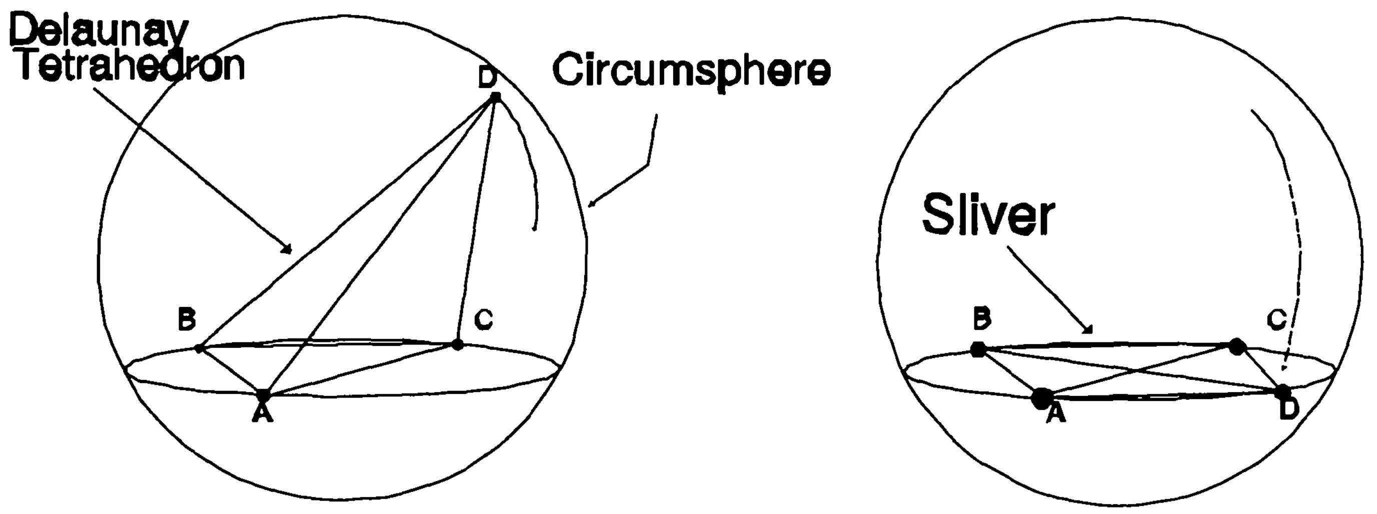


Figure 3.5.2. Illustration of circumsphere of tetrahedral and a sliver element.

3.5.1 Watson's Algorithm [SLH84].

The Delaunay triangulation has several degenerate cases and like all grid generation methods, is subject to computer accumulated rounding errors. In 2D these problems have been minimized by special ordering of nodes in the generation of the grid and the use of a combination of both Watson's [Wat81] and Lawson's [Law72] procedures to make the method more robust computationally. Watson's algorithm is illustrated below:

Each node is taken in turn and inserted into the mesh. A search for all the elements whose circumcircle contains this node (Figure 3.5.3) is made.

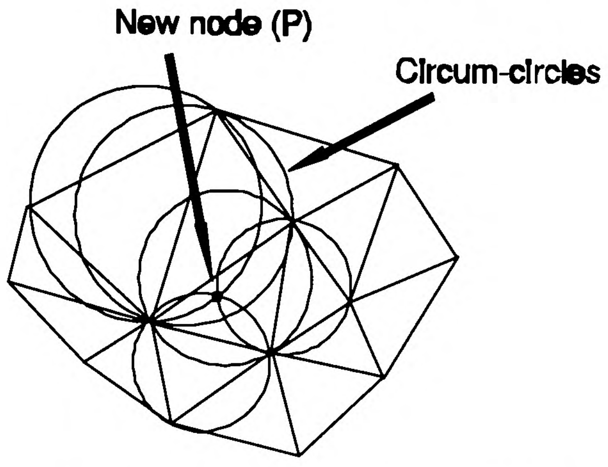


Figure 3.5.3. Insertion of Node.

The method then removes these elements, Figure 3.5.4, and the external boundaries of the set of elements form a polygon.

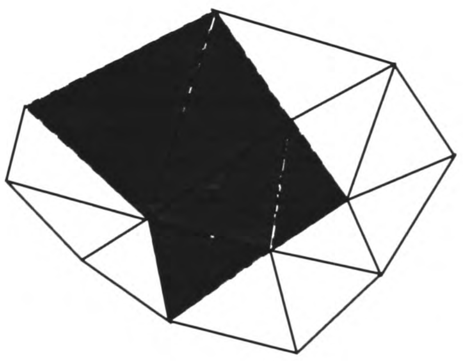


Figure 3.5.4. shaded elements are removed.

The vertices of this polygon are then joined to the newly inserted node. Which then forms a new Delaunay triangulation that includes the inserted node.

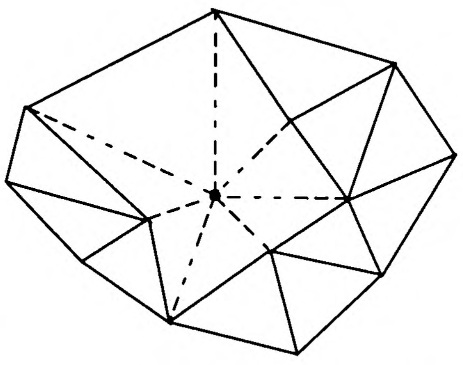


Figure 3.5.5 Vertices of the polygon are joined up to the new node.

In 3D Lawson's swapping algorithm cannot be used, but recent developments in 3D Delaunay triangulation by Joe [Joe89] using local transformation of tetrahedra (see section 6.10 3D vertex swapping) have resulted in a very robust and fast method of generating Delaunay meshes. The 3D method has a worst case computational order of n^2, however, on most practical cases it is of order *nlog(n)*. Despite its computational efficiency Delaunay triangulation in 3D does not generate well shaped elements [CFF85].

3.5.2 Topological incompatibility in Delaunay grids.

Delaunay triangulation is based solely on the location of the points of the domain and higher order topological information does not affect the resulting computational mesh. Therefore, the Delaunay triangulation of certain geometric models with particular distributions of points will produce a mesh that is incompatible with the model's topology.

To correct this problem, we have to search the geometry of the model for intersection with the elements formed by the triangulation. Where elements intersect the surface of the model, we introduce extra 'stitching points', to make the triangulation conform to the geometry. This is illustrated in Figures 3.5.6 to 3.5.8.

Meshing the geometric model below

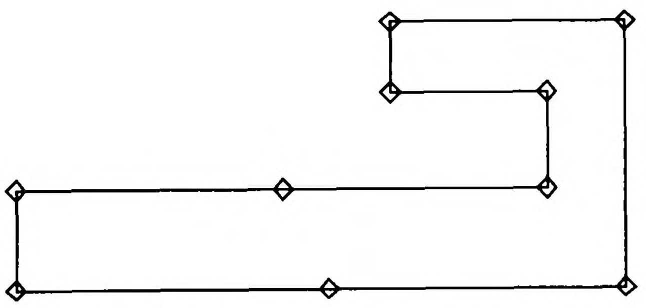


Figure 3.5.6 : Initial geometry

This results in a topologically incompatible mesh

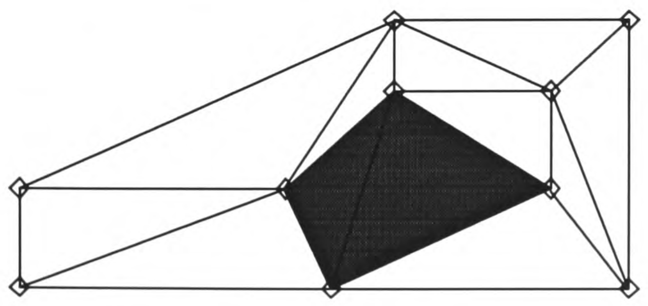


Figure 3.5.7 : Initial Delaunay triangulation.

Resolved by introducing a stitching point

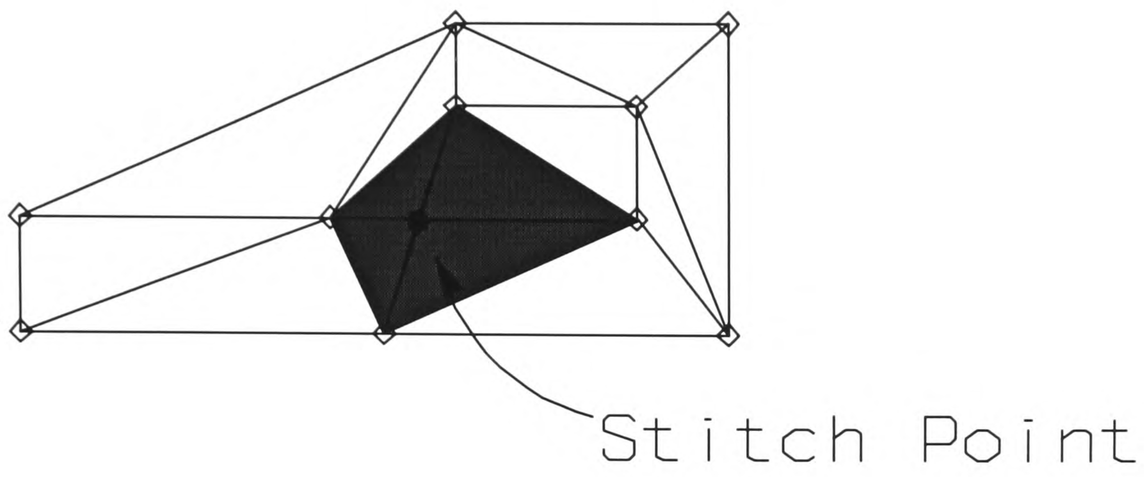


Figure 3.5.8 : Insertion of a stitch point

An alternative method is to force the Delaunay algorithm to generate only geometry compatible meshes. This is achieved by ensuring that the nodes on the boundary of the model form *Delaunay edges* [Joe86]. A *Delaunay edge* is defined as two adjacent vertices on the boundary and the circum-circle through these two points does not contain any other boundary vertices. Figure 3.5.9 illustrates the definition Delaunay edge and shows how it can be used to spot areas of incompatibility.

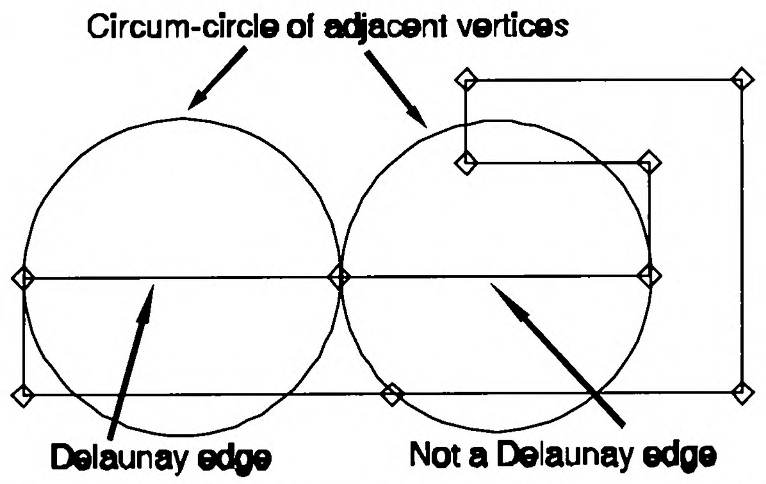


Figure 3.5.9 Illustration of Delaunay and non-Delaunay edges.

3.6 Mesh Generation by Medial axis Subdivision

The medial axis subdivision is a relatively new and novel technique for generating various types of grids using triangular and quadrilateral elements. Grids that have been generated this way tend to be well structured and of high quality {Tam and Armstrong 1991 [TaA91]}.

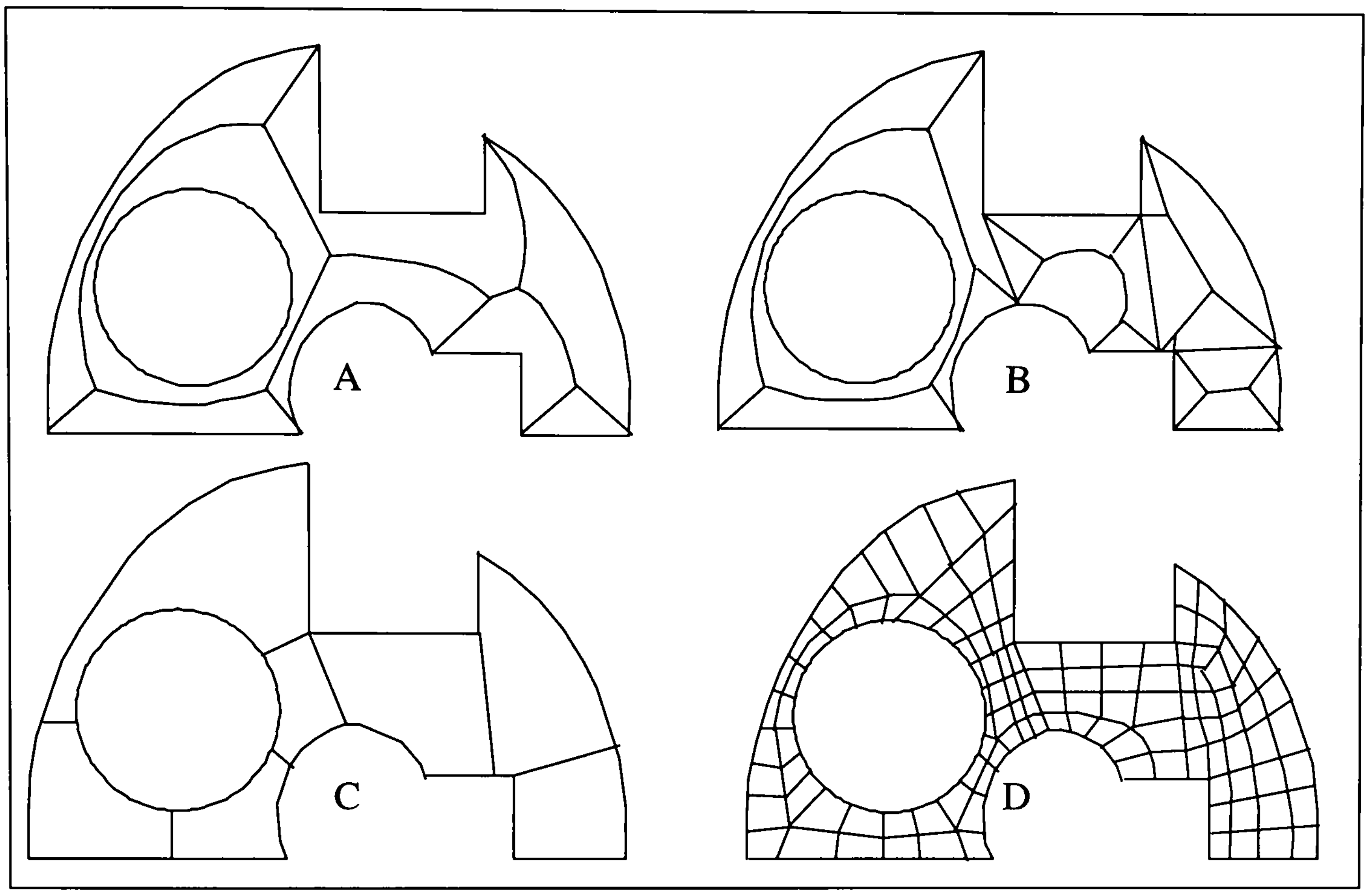


Figure 3.6.1 : Example of stages in mesh generation by medial axis subdivision.

The main concept behind this method, as the title suggests, is the generation of the medial axis or Voronoi diagram of the domain that is shown in Figure 3.6.1A. The motivation behind the generation of this diagram is the belief that elements should flow round the object in the general direction specified by the medial axis. The Medial axes diagram is often generated by first triangulating the domain using Delaunay triangulation and from this triangulation the Voronoi diagram is derived. Once the medial axis is derived this is then processed first to remove concavities, Figure 3.6.1B and then chain splitting , Figure 3.6.1C, to generate the sub-domains that can then be meshed with any suitable mesh type and pattern to generate the final mesh Figure 3.6.4D.

3.7 Recursive Domain Bisection.

Recursive domain bisection, is a method first implemented by Lewis and Robinson [LeR76], which applies a 'problem-reduction' technique to triangulate domains. This technique consists of dividing the original data space into disjointed segments, and then solving the problem for each of the smaller segments. This technique is applied recursively on each domain and its sub-domains until each data space is sufficiently small for a very simple algorithm to be applied. This method is similar to the Quicksort algorithm [Tho80], where the problem of sorting vectors is reduced to sorting shorter and shorter vectors, until vectors of length two are generated. These can then be sorted by one comparison and a conditional interchange.

Therefore, the triangulation of region R (Figure 3.7.1(a)) can be achieved by:

(a) Splitting R into two sub-regions, R_1 and R_2, by creating a new boundary across the region.

(b) Solving the triangulation problem for R_1 and R_2 separately (Figure 3.7.1(b)).

The new boundary has a zigzag appearance as it consists of the join of points lying near a line that passes through two 'opposite' boundary points (Figure 4.7.1(c)).

Sub-domains are divided until triangles with no interior points are formed, these being the elements of the triangulation; triangles containing interior points are split by two lines joining an interior point to two vertices.

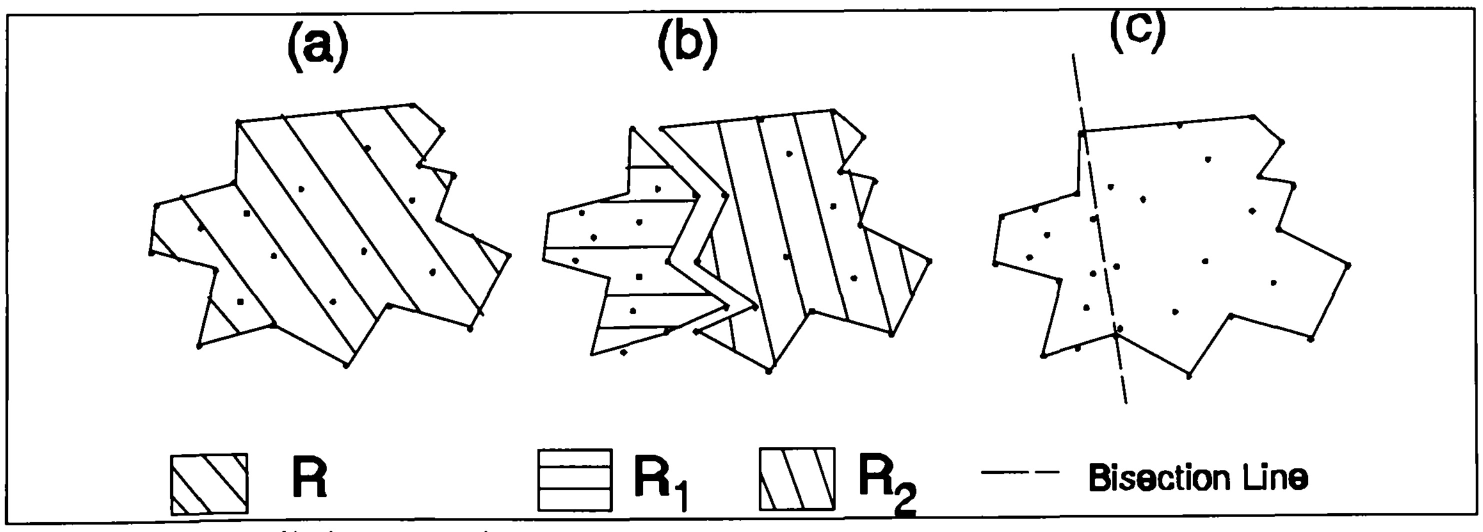


Figure 3.7.1 Splitting a region

There are usually numerous possibilities for the selection of a bisection line to divide any arbitrary domain. Lewis and Robinson outline a method that attempts to divide the domain into two 'circular equally sized halves', this is described in more detail in Chapter 4.

3.7.1 Advantages and disadvantages.

The computation order of the above algorithm was shown by Lewis to be n log(n) and in the worst case n^2. The worst case scenario is based on the assumption that most proposed splits of the region are invalid, so finding valid splits is the dominant part of the algorithm. The worst case occurs on difficult geometries where the vertex removal routine has to be used in the majority of bisections. However, it was shown that the ratio between the main bisection routine and the simple vertex removal method is 1:10. Therefore, we have a routine whose performance has a good average computational order.

However, this routine does not include a nodal placement method, all nodes have to be provided prior to the grid generation. It was required that the user provided all nodal position prior, either generated by hand or using a rudimentary nodal placement algorithm [MLC83].

Multiply connected regions are dealt with the manual addition of a cut line (Figure 3.7.2), this decomposes the region into simple polygons. However, this could be overcome by the introduction of an automatic method of decomposing multiply connected regions into simple polygons, as described by Joe and Simpson [JoS86].

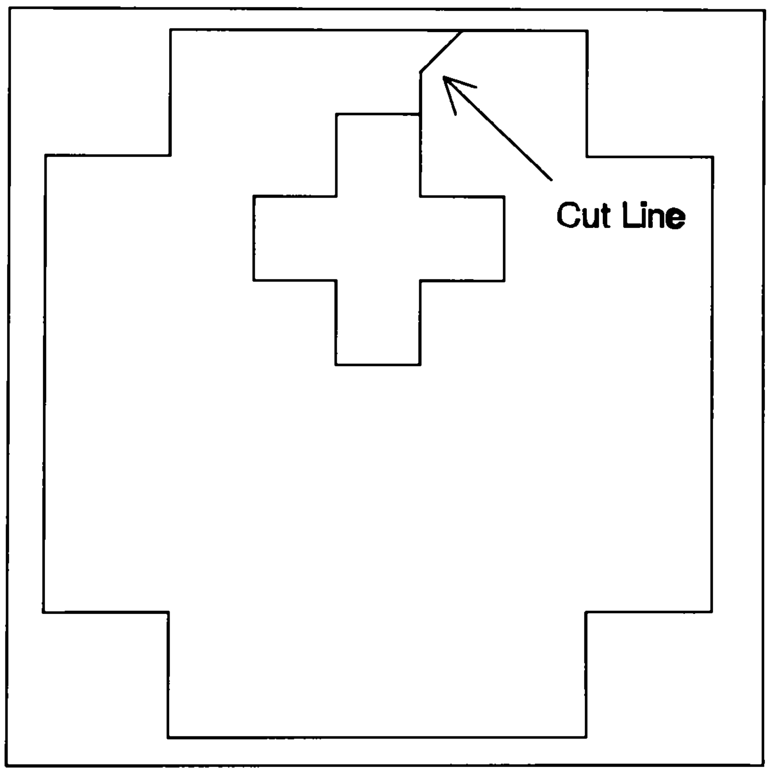


Figure 3.7.2 A multiply-connected region.

3.8 Triquamesh.

Triquamesh [SBS79][SlH82] is a mesh generator, developed in the early 80's, which generates triangular and quadrilateral elements in 2D. The technique used in Triquamesh is a recursive bisection method, and is similar to the technique used by Lewis et al [LeR76], see Section 3.7. However, Schoofs et al [SBS79] used a different heuristic, in Triquamesh, to guide the selection of bisections. Schoofs et al's technique was to introduce a bisection which divides the largest "edge angle" in the domain. This is repeated recursively on the resulting sub-domains, until sub-domains form triangular regions, which are the elements of the mesh.

Triquamesh incorporates an automatic nodal placement technique, which generates nodes automatically along each newly generated bisection edge. It is similar to the method described in Chapter 4, section 4.5.2. It was not implemented in the new 2D bisection technique described in the thesis, as it tends to needlessly over refine certain regions within the domain, see Chapter 4 section 4.6.2.

Quadrilateral element generation, in Triquamesh, is achieved by converting each triangular element into three quadrilaterals, see Chapter 4 section 4.7.1. This technique was also dropped from the new bisection technique, described in this thesis, as it tends to produce quadrilateral elements with poor aspect ratios, See Chapter 4 section 4.7.1.

In the paper by Sluiter[SlH82] Triquamesh was extended to 3D tetrahedral mesh generation. However, the 3D domains it could handle were limited, since it could not handle multi-connected regions. The tetrahedral meshes it generated were of poor quality, since it had no tetrahedral optimization technique. 3D Triquamesh also generates hexahedral elements, in a similar way to the 2D technique, by converting each tetrahedron element to 4 hexahedral elements.

The 3D bisection mesh generator, described in this thesis, has overcome many of the problems which were associated with the 3D Triquamesh, see Chapter 6. The 3D mesh generation method, described in this thesis, can handle multi-connected regions and has element optimization routines which improve the quality of the final tetrahedral mesh (e.g local 3D min-max vertex transformations, see Chapter 7 section 7.5). The new mesh generator, presented in this thesis, has an advanced nodal placement technique (Chapter 7 section 7.4) which avoids unnecessary over refinement of certain regions of the mesh, unlike the technique implemented in Triquamesh.

3.9 Summary and conclusions.

The following methods, Advancing front, Binary mesh operations, Recursive bisection and Paving methods require a large number of face, plane and line intersection tests. Three dimensional plane and line intersection testing is notorious for problems with computer arithmetic errors [For87], and forms a major area of research [Sar83][Dew88][BoW83]. Therefore, we can conclude, just by probability, that the more intersection tests carried out, the greater the chances of an incorrect geometry interrogation. For example, if a comparison is made between an order *nlogn* method (2D Recursive mesh bisection) and an order n^2 method (2D Advancing front) using similar algorithms for line, plane and surface intersections. The order n^2 method would have a larger probability of generating an invalid mesh than the order *nlogn* technique, since the *nlogn* method requires fewer geometry tests for a similar sized problem.

Delaunay triangulation has the advantage of being a computationally efficient algorithm, however the technique does not generate well shaped tetrahedral elements. In fact, Delaunay triangulation in 3D is the method that is most likely to generate an invalid grid. Delaunay triangulation suffers not just from computational rounding errors for sphere point in-out tests, but also the algorithm does not consider any geometry information or satisfies any min-max angle criterion. Mesh generation by Medial axis often requires a Delaunay triangulation of the domain to enable the sub-division of the geometry. Therefore, the Medial axis technique inherits its major problems from the Delaunay algorithm.

Mesh generation by Recursive domain bisection is the only method that offers geometry compatibility, together with computational efficiency. The computational reliability of this algorithm is linked to its computational efficiency, requiring on average less complex geometry tests than its counter part methods, such as Advancing front and Paving algorithms.

The reader is referred to Chapter 8 section 8.2, for a further description of some additional three dimensional meshing techniques.

Part II
Two dimensional and surface mesh generation.

The next two chapters will cover the initial developments of the bisection algorithm in the 2D plane. This is then followed with a discussion of extending certain mesh generation techniques to surfaces.

Chapter 4

2D Mesh Generation.

4.1 Introduction.

This chapter is not intended to be a detailed description of all the research carried out in two dimensions. It is intended to introduce to the reader some of the ideas that will be later extended to three dimensional grid generation.

The first section will outline the objectives initially set, and outline the basic requirements of a 2D geometry data structure. It will then go onto describe the fundamental algorithms of the recursive domain bisection technique, which were first outlined by Lewis et el [LeR76]. This will be followed by a description of some of the techniques used on 2D grids to improve their quality.

The second section will outline some of the fundamental extensions, which have focused upon 2D recursive mesh bisection. This includes the data input format and types of geometry that can be handled, together with some of the initial success and results. The chapter will then go on to explain some of the further extensions applied to improve the initial mesh quality, by improved nodal placement and bisection algorithms.

This chapter will then be concluded with further extensions, i.e. triangle to quadrilateral conversion routines, nodal refinement algorithms and polygon elements.

4.2 Objectives.

The objectives of the meshing tool, are to provide a method of generating two dimensional grids over a planar region. The true objective of the 2D grid generator was to provide a platform to launch the 3D version. Therefore, it was necessary that the ideas used were readily extendible to 3D.

The grid generator's requirements were to generate meshes that could be used for initial computational purposes with limited user control over nodal placement. Optimization of the mesh was to be left to other adaptive methods such as P, R or H refinement techniques, see [Thm85], [EOD93], [LoS91], [Ran87] amongst others.

The Geometry input requirements are to model multi-connect domains, with holes, interfaces and sub-domains, as illustrated in Figure 4.2.1.

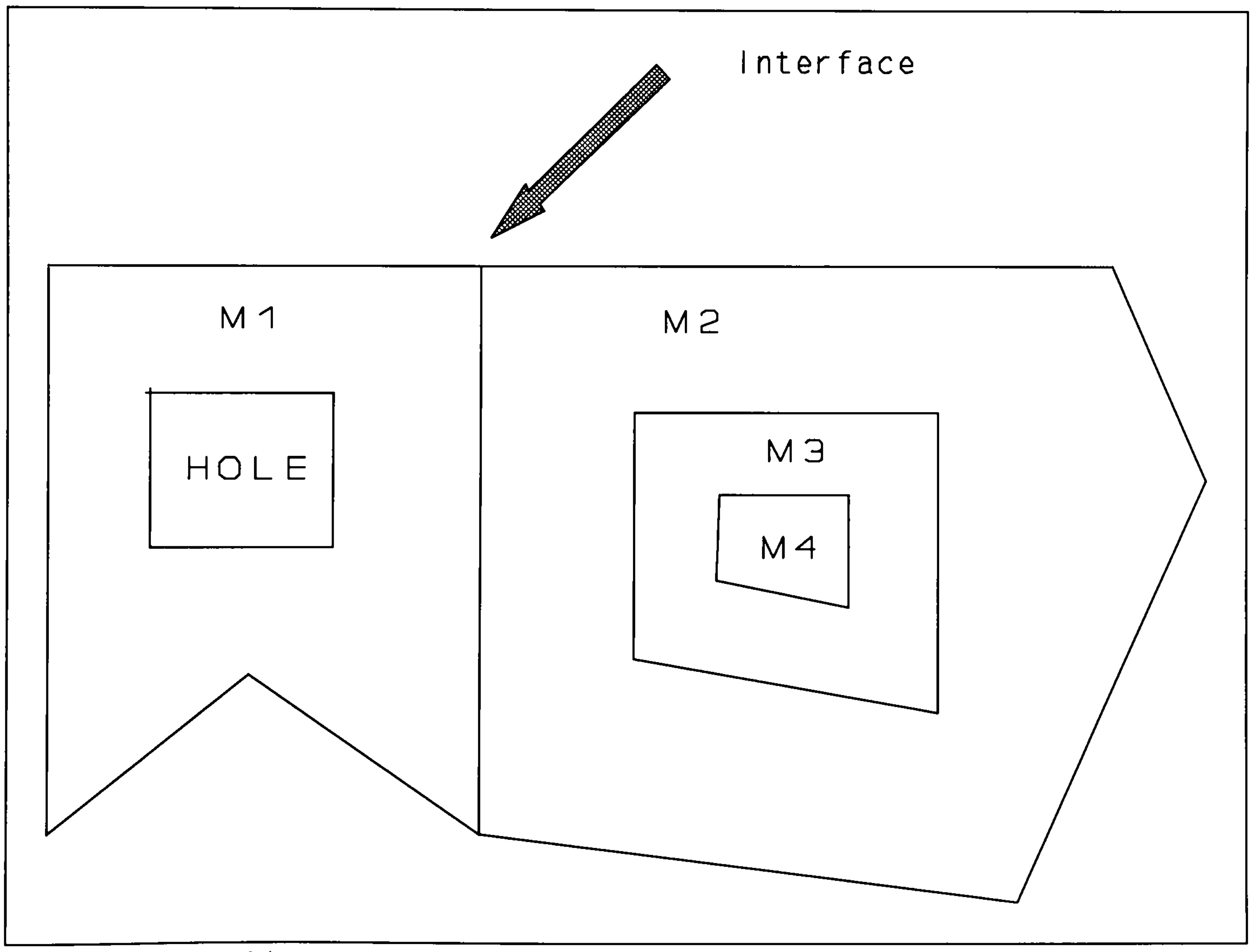


Figure 4.2.1 Multi connected region. M1,M2,M3 and M4 are different materials.

4.3 Recursive Domain Bisection.

This section will outline the fundamental algorithms behind the method of Recursive bisection, Lewis and Robinson [LeR76], which was initially described in Chapter 3.

4.3.1 Choosing the bisection line.

There are usually numerous possibilities for the selections of a bisection line to divide any arbitrary domain. Lewis and Robinson outline a method that attempts to divide the domain into two 'circular equally sized halves'.

Not all possible splits are examined in this algorithm, for computational speed, therefore the search is terminated when a number of solutions are found. The search is organized so that splits between 'opposite' boundary points are tested first. Each possible bisection line is given a weight depending on the function ΠE^b. Π is the product of the distance of the boundary points to the split line, Figure 4.3.1, b is the number of boundary points and E is the minimum of:

(a) half of the average distance between the boundary points, and

(b) the distance from the split line of the nearest interior points contained within any rectangle having the split line as a side, see Figure 4.3.2.

Figure 4.3.1 illustrates a domain with a possible bisecting line that divides the region into T_1 and T_2.
S_i is the distance of boundary points in T_2 from the bisecting line.
d_i is the distance of boundary points in T_1 from the bisecting line.
Here $\Pi_1 = S_1S_2S_3$ and $\Pi_2 = d_1d_2$, hence $\Pi = \Pi_1\Pi_2$.

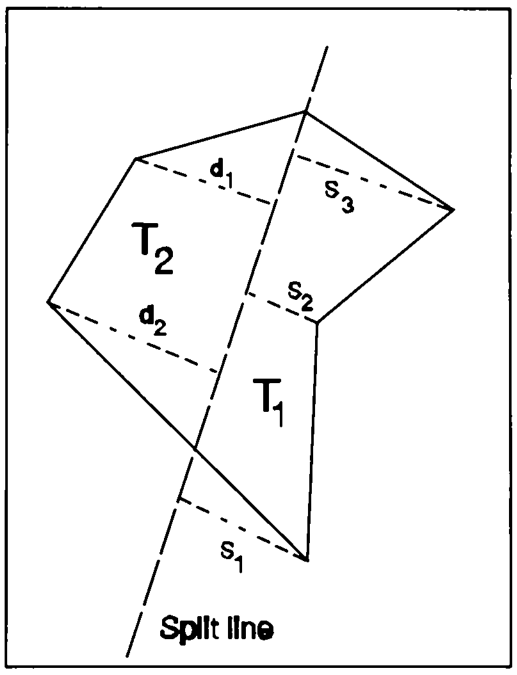


Figure 4.3.1 Calculating weighting function.

4.3.2 The actual split used.

Once a particular bisection of a region is selected, points on the interior, that lie 'close' to the proposed split line are included as part of the new boundaries. Selection of points is done in such a manner to avoid long thin elements. The method used to sort points into their respective halves tends to reveal, which points are close to the split line. Points are chosen to be a member of the interface edge if:

(a) they lie within a rectangle with the split line as a side and,

(b) their distance from the split line is less than E defined above.

Figure 4.3.2 demonstrates which points to include as part of the new boundary. Points P_1 and P_3 would not become part of the new boundary whilst P_2 would. L_1 and L_2 form the outer edges of the rectangle.

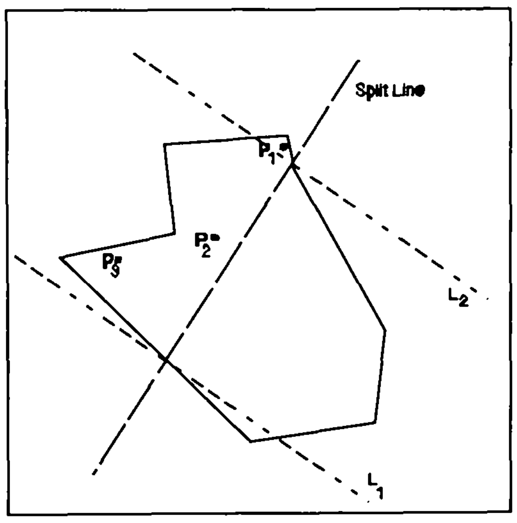


Figure 4.3.2 choosing new boundary points.

4.3.3 Domains of peculiarity.

If the current region has no possible bisection a cruder approach is adopted. The binary mesh operation of vertex removal is applied, see Chapter 3 section 3.3, and the split line is the join of a boundary point to its next but one neighbour. The actual split made is such that the triangle cut off has its smallest angle maximised. Once the split is chosen the code then proceeds as in section 4.3.2.

If the region to be split is simply a triangle containing interior points, then the split is performed by joining two vertices of the triangle to one of the interior points. The only extra boundary points are those that lie on the split.

4.4 Grid quality improvements

There are numerous methods for taking a 2D grid and improving the quality of elements for computational proposes. These methods include Laplace smoothing [KaE70], Vertex swapping [Law77], local refinement and de-refinement to name but a few. Lawson [Law77] showed that planar grids could be transformed to another by a finite set of operations, this technique is used in most planar Delaunay triangulations. Lewis and Robinson used a technique of vertex swapping [Law77] to improve the quality of their grids. The following sections will outline two of the techniques used to improve the grids generated. The reader should note that the following methods are of order *nlogn*, and have been modified to optimize their execution rates.

4.4.1 Vertex Swapping

Vertex swapping [Law77] of elements' faces is a well known technique used in 2D mesh generation to achieve local min-max or max-min angle criterion. This method is based on the observation that there are two possible triangulations of a convex quadrilateral. The better triangulation is the one that makes the resulting triangles most equi-angular, as measured by the size of the smallest angle. For example, Figure 4.4.1 shows two adjacent triangles I and J that have been generated by some initial mesh generator.

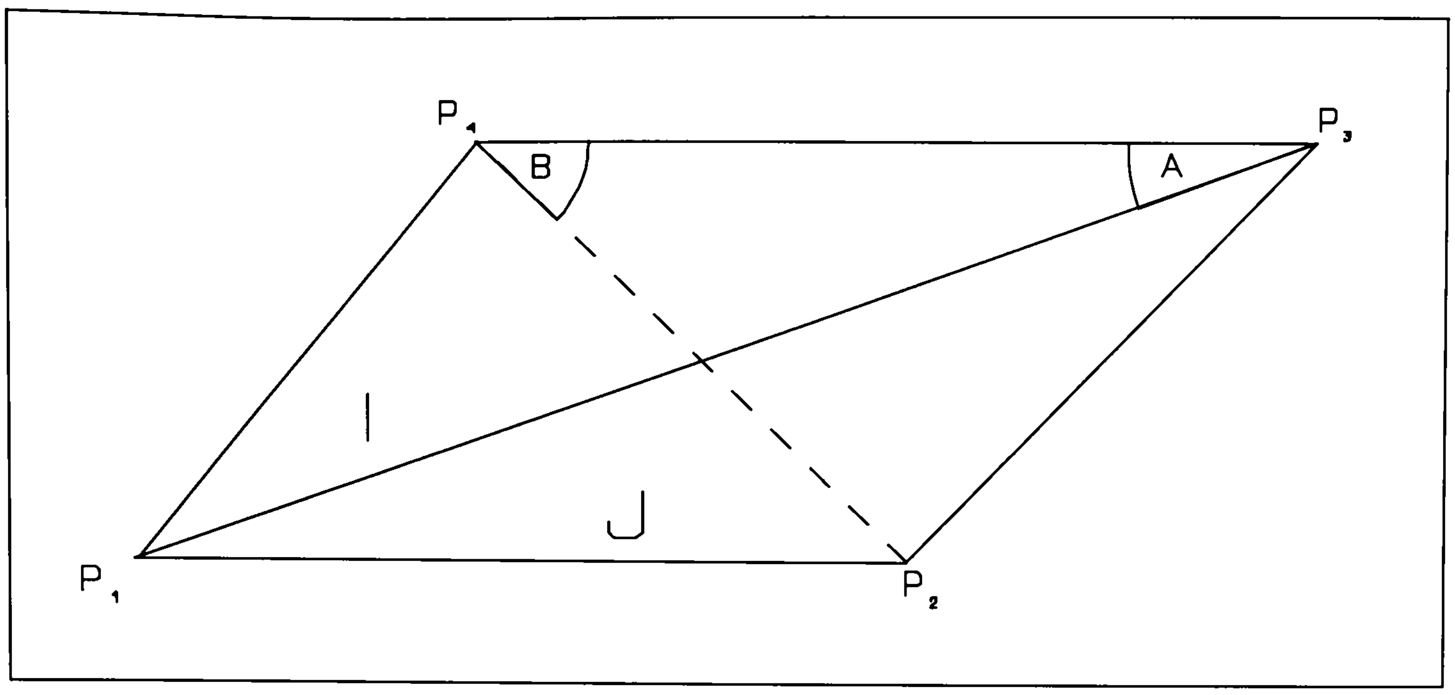
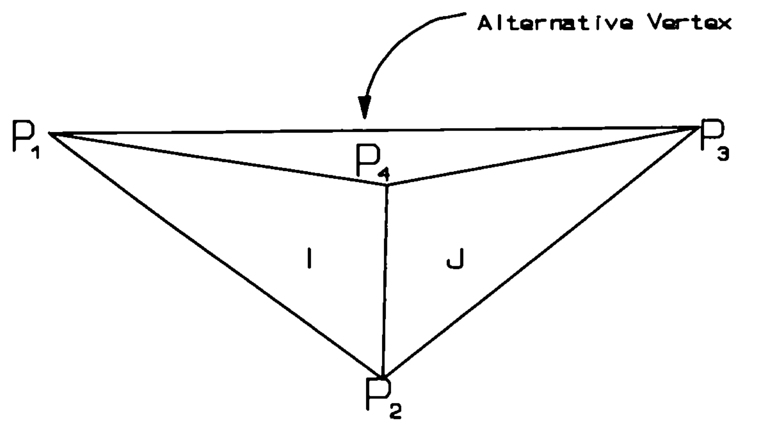


Figure 4.4.1 Two triangles with alternative vertex shown.

In Figure 4.4.1 the line P_2P_4 lies within the polygon formed by triangles I and J, a new split of the quadrilateral $P_1P_2P_3P_4$ is possible, i.e. triangles $P_1P_2P_4$ and $P_2P_3P_4$ may be formed. The smallest angle A of the original triangles and B the smallest angle of the new triangles, may be calculated. No change is made if $A > B$, but if $A < B$ the new triangles replace I and J.

In the above method it has to be established which two triangles form a convex quadrilateral. This section will describe a method that uses the fact that most meshes have their elements' nodes stored in a fix order (counter clock-wise). This has been found more reliable than other techniques that are based on testing which side of a line points lay, like ray testing [Rog85] algorithms or special methods base on the geometry uniqueness of a triangle [Sar83][Bow83]. For example, in Figure 4.4.2, the alternative vertex P_3P_1 lies outside the quadrilateral which forms the triangles shown in Figure 4.4.3 i.e. triangle I is contained in J.



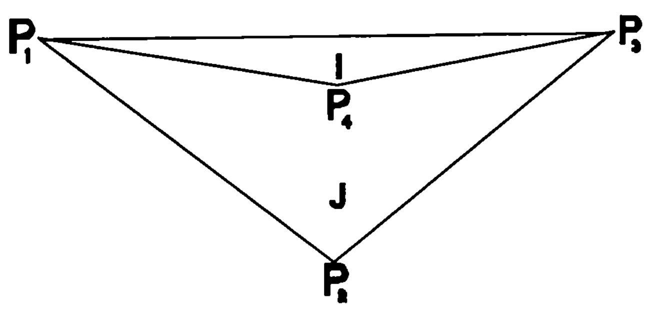


Figure 4.4.2

Figure 4.4.3

The areas of the triangles $P_1P_2P_4$ and $P_2P_3P_4$, Figure 4.4.2 are both positive, since both triangles' nodes are in counter clockwise order. However, the areas of triangles $P_1P_3P_4$ and $P_1P_2P_3$, Figure 4.4.3, have different signs. Triangle 'I', Figure 4.4.3 has a negative area because it is contained inside triangle 'J', therefore its nodes are in clockwise order, i.e. the quadrilateral $P_1P_2P_3P_4$ is not convex.

Therefore, from the above information we can derive a method of applying a vertex swapping algorithm to an initial mesh as follows:

```
i)      Repeat
ii)         For each triangle,I, in the mesh do
iii)            For each edge, IEDGE, do
iv)                 Find neighbouring triangle J, on the edge IEDGE.
v)                  If triangle I and J form a convert quadrilateral then
vi)                   Find minimal angle of triangles I and J (MIN1)

vii)                  Find minimum angle of the alternative triangles
                      that can be formed with I and J (MIN2).
viii)                 If MIN2>MIN1 then swap vertex of triangles
viiii)               endif
x)              end for each edge...
xi)         end for each triangle...
xii)    until (No more swaps performed or maximum number of passes reached)
```

The above algorithm is a simplification, the full method includes a stack that stops neighbouring pairs of triangles being tested more than once. It also stores which triangles were affected by transformations on each pass of the algorithm, therefore on each iteration it only examines elements that were swapped previously. Also it was found that the above algorithm can further be improved by taking each triangle in turn and looking at all its neighbouring elements first. If the triangle needs to have a vertex swapped we choose the neighbouring element that forms the set of triangles with the maximum minimum angle. Figure 4.4.4 shows a triangle with its neighbouring elements and possible vertices swaps. This dramatically reduces the number of iterations required.

Because of finite precision of the machine, oscillation of edges between each pass can occur. Therefore, it was required to store the minimum angle of the grid on each pass. If there is only a small change in this value between consecutive iterations the routine terminates.

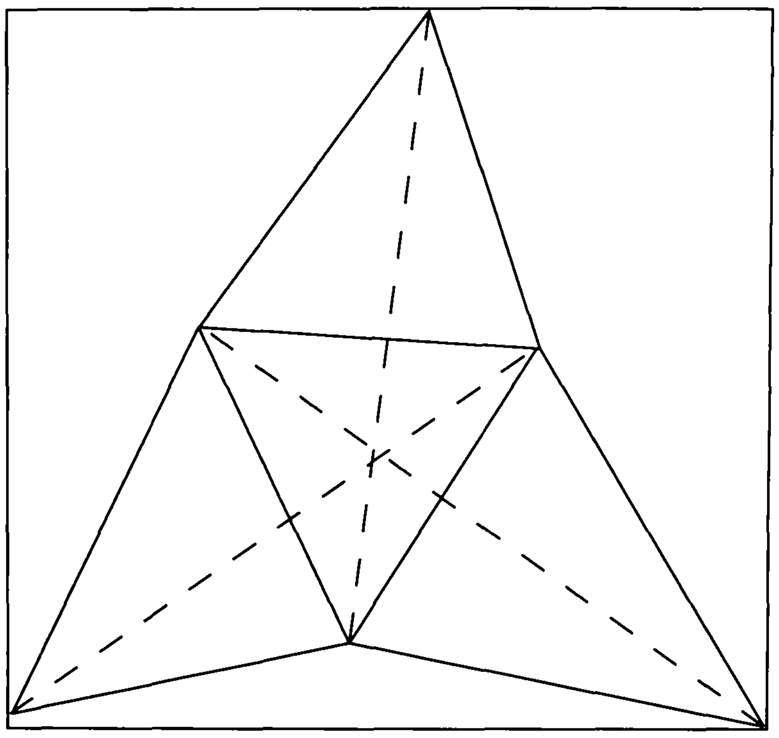


Figure 4.4.4 Alternative vertex searching.

4.4.2 Laplace's smoothing.

Laplace's smoothing [KaE70][Rec73][MeP77][Her77] is a simple but effective method used in 2D and 3D mesh generation to improve the general shape of elements. This is achived by removing some of the skewness of elements locally [Wör81]. In Laplace smoothing each node is taken in turn and moved to a new location that is the average of all the adjacent vertices positions.

Hence node's *i* location becomes :-

$$P_i = \sum_{j=1}^{n} R_j/n$$

'R' is the set of size 'n' of all nodes directly connected to Node i

'R_j' is a positional vector of node j in 'R'

'P_i' is a positional vector of node i

Laplace smoothing is a highly efficient algorithm that is applied iteratively until there is only a small change in the nodal positions. However, two passes were found to be sufficient for the majority of the meshes generated by the 2D mesh generator presented in this Chapter.

It was found that the above two algorithms 4.4.1 and 4.4.2 are often enough to convert most grids with badly shaped elements to reasonable quality. They have both been shown to be of order n [LeR76] and add a very small overhead to any meshing routine.

4.5 Preliminary extensions.

The initial aspirations were as follows:

(a) Use a superior bisection algorithm.

(b) To remove the requirement of adding a cut line to multiply connected regions.

(c) To enable the automatic generation of grid points.

(d) To generate a code that is so robust that could operate in single precision.

The above requirements were to enable the extension of the procedure to three dimensions, and if the code could work in single precision in 2D, then the 3D version would have a greater chance of working robustly on complex geometries.

4.5.1 Improved bisection algorithm.

The improved bisection algorithm is illustrated in Figure 4.5.1

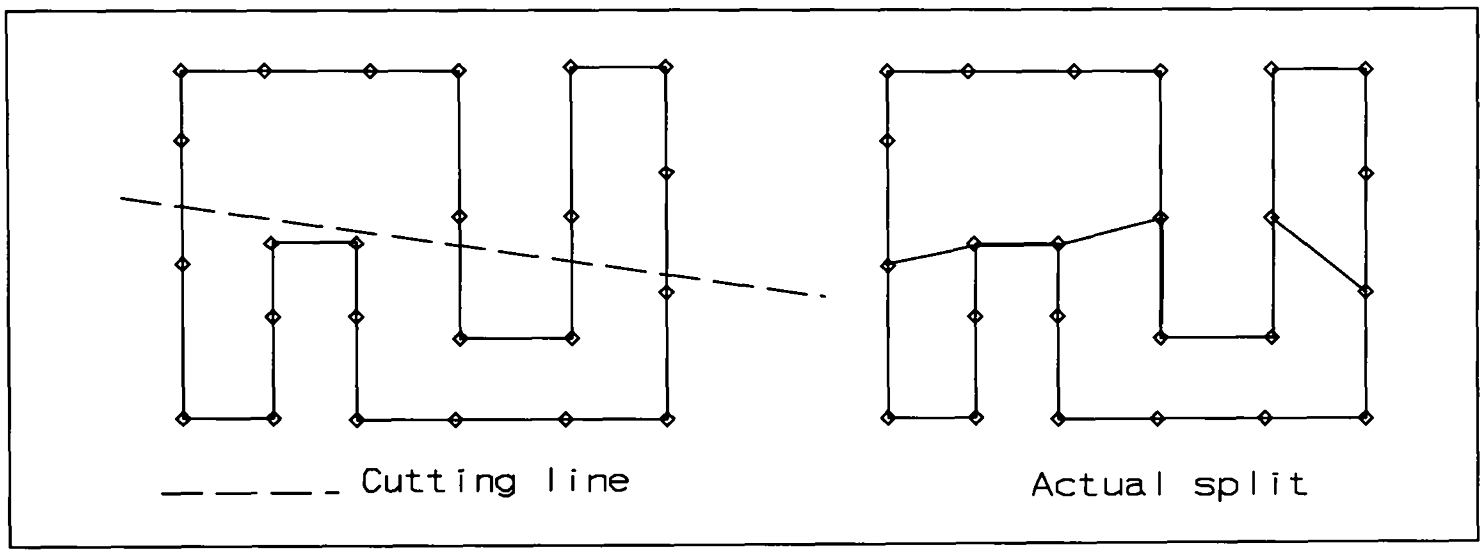


Figure 4.5.1 Advance bisection routine.

The method of choosing the bisection line is the same as described in section 4.3.2. Once a cut line is selected the boundary segments are sorted to their respective sides, Figure 4.5.2a, segments are the edges contained between two vertices. These edges form two sets of boundary points, any boundary point that is contained within both regions is a boundary interface node. The list of boundary interface nodes, are then sorted into sequence along the interface, see Figure 4.5.2b. If the number of boundary interface nodes is a multiple of two, the nodes can be joined in the following order to generate the new edges, 1 to 2, 3 to 4 etc. However, if the number of boundary interface nodes is odd, the interface is complex and this bisection line is rejected. The generation of a new boundary is illustrated in Figures 4.5.2b and 4.5.2c.

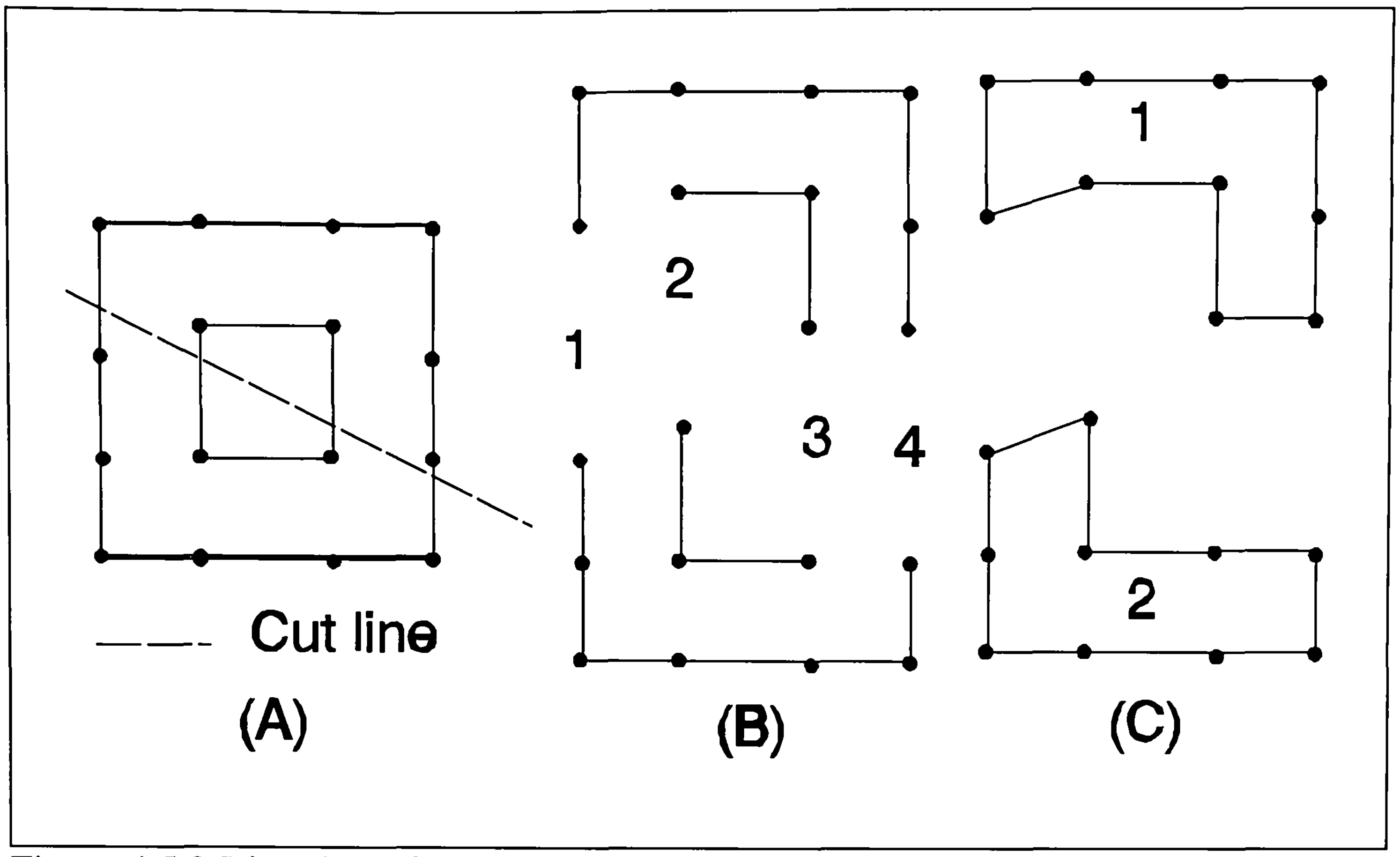


Figure 4.5.2 Bisection of a multiply connected region.

The above technique has removed the requirement for the addition of a cut line for multiply connected regions, described in section 4.4.3, see Figure 4.5.2. However, this method does not consider any internal nodes, because of the difficulty of sorting nodes into their respective regions. Nevertheless the algorithm is more reliable and fails less often than the original method.

The requirement for the binary mesh operation of edge removal was also found necessary. The region in Figure 4.5.3a was found to fail on both the mesh bisection and vertex removal [section 4.3.4] algorithms. Edge removal is the selection of one edge and a point, which may be internal to the domain or on an opposite boundary.
Figure 4.5.3b shows an element ϕ generated by edge removal.

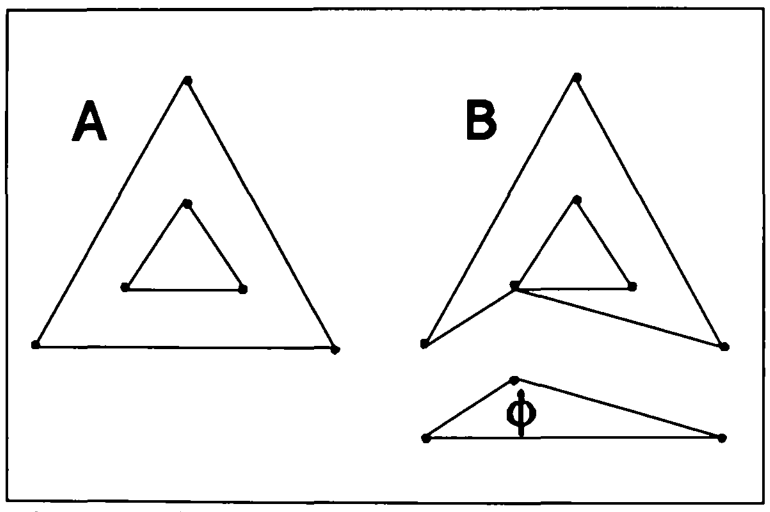


Figure 4.5.3 Edge removal.

The above two algorithms were implemented on a Sun Sparc 4 using single precision arithmetic in addition to other minor changes to the code. A simplistic data format was used, which required the input of boundary nodes and connectivity.

4.5.2 Preliminary nodal insertion routine.

The new bisection algorithm could not handle internal nodes. Therefore the first solution was to generate the boundary constraint mesh, which is a grid generated from just the boundary nodes. Then each internal node is taken in turn and inserted into the mesh, using techniques derived from algorithms developed for planar Delaunay triangulation [ScS86][SlH84][CeS85]. This algorithm is very simple and described as follows:

```
(1)  Take an internal node i.
(2)          Search the mesh for triangle J which contains the node i.
(3)          Join this node up to the three vertices of element J to form three new
             triangles.
```

The above steps are simplified. In step (2) a method *of element walking* [SlH84] is utilized to find the triangle *J* which contains the node *i*, which is an order *nlog(n)* technique. It is also possible for the node *i*, in step (2), to co-inside with an edge or node of an element, and this is also taken account of in the full algorithm.

A method of generating nodes simultaneously was then implemented, based on the technique described by Connor [Con89]. The user provides the boundary nodes and these guide the mesh generator's nodal placement algorithm. Therefore, if there is a fine concentration of nodes around an area of the boundary, the internal mesh would reflect this. Figure 4.5.4 illustrates the rudiments of the nodal placement algorithm. Figure 4.5.4A shows a bisection line and Figure 4.5.4B shows newly generated nodes along the interface.

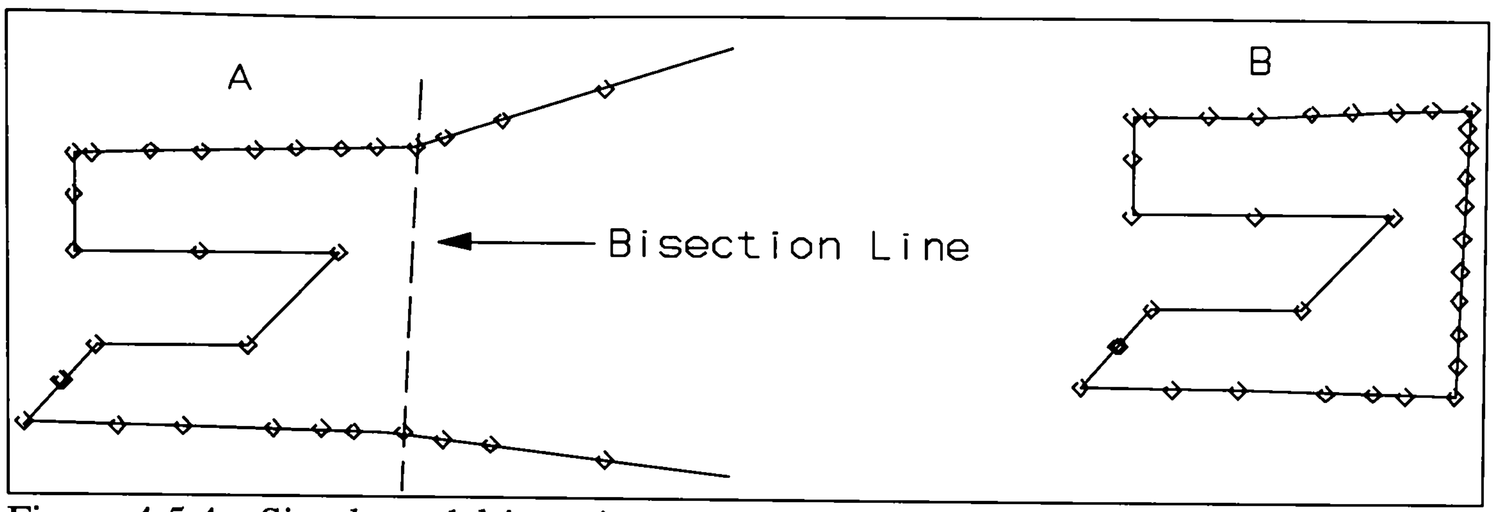


Figure 4.5.4 : Simple nodal insertion routine.

When an interface is generated the boundary interface nodes are given a nodal spacing. This spacing is calculated from the average distance of adjacent nodes. Nodes are then generated along an interface element, the spacing of these nodes are interpolated from the two nodal spacing values assigned to the end nodes. For example, if ϕ_i and ϕ_j are the nodal spacing at two adjacent interface nodes and let *t* be the parameter location between nodes i,j where $t>0$ and $t<1$. The nodal spacing at position *t* is then given by $\phi_i+t(\phi_j-\phi_i)$. However, before a new node is inserted into a grid, an additional check is carried out to ensure that this point is not too close to other nodes in its subregion. This occurs when the region is highly re-entrant, see Figure 4.5.4.

To illustrate the robustness of the initial code and its ability to cope with multiply connected domains, the geometry in Figure 4.5.5 was used. The completed mesh is illustrated in Figure 4.5.6; note that no internal points have been added.

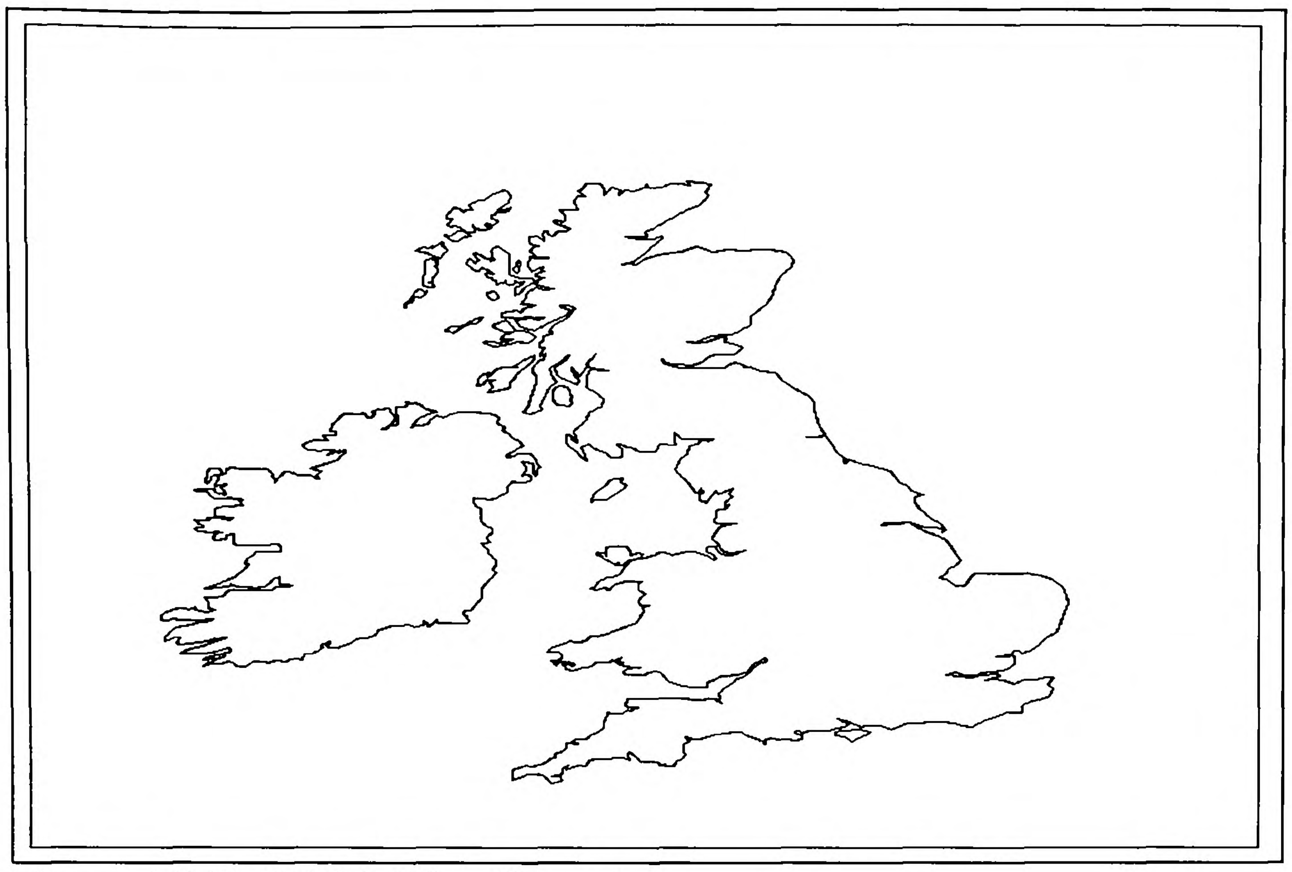


Figure 4.5.5 : British Isles Geometry.

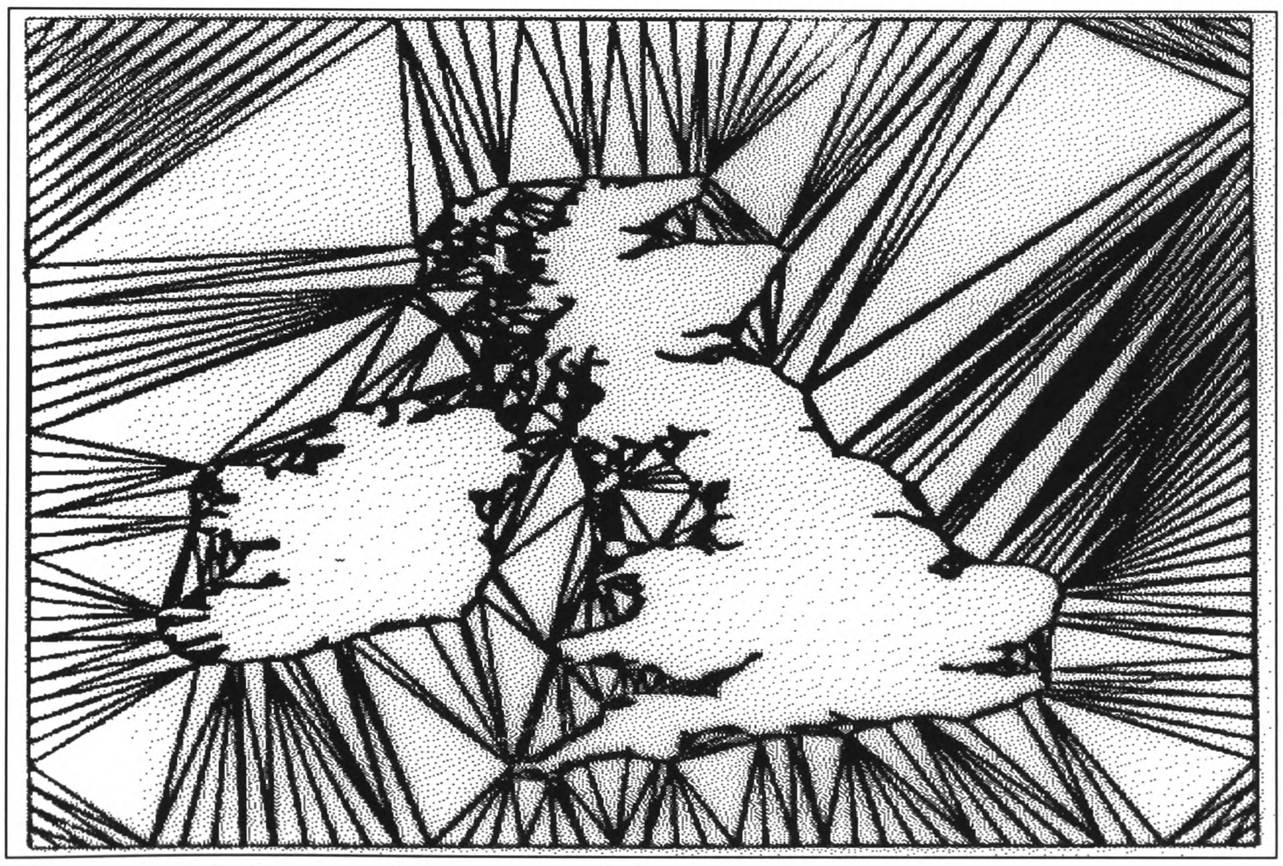


Figure 4.5.6 : British Isles with minimal mesh.

Figure 4.5.7 shows the British Isles grid generated using the nodal placement algorithm, section 4.5.2, and Figure 4.5.8 shows the mesh after optimization.

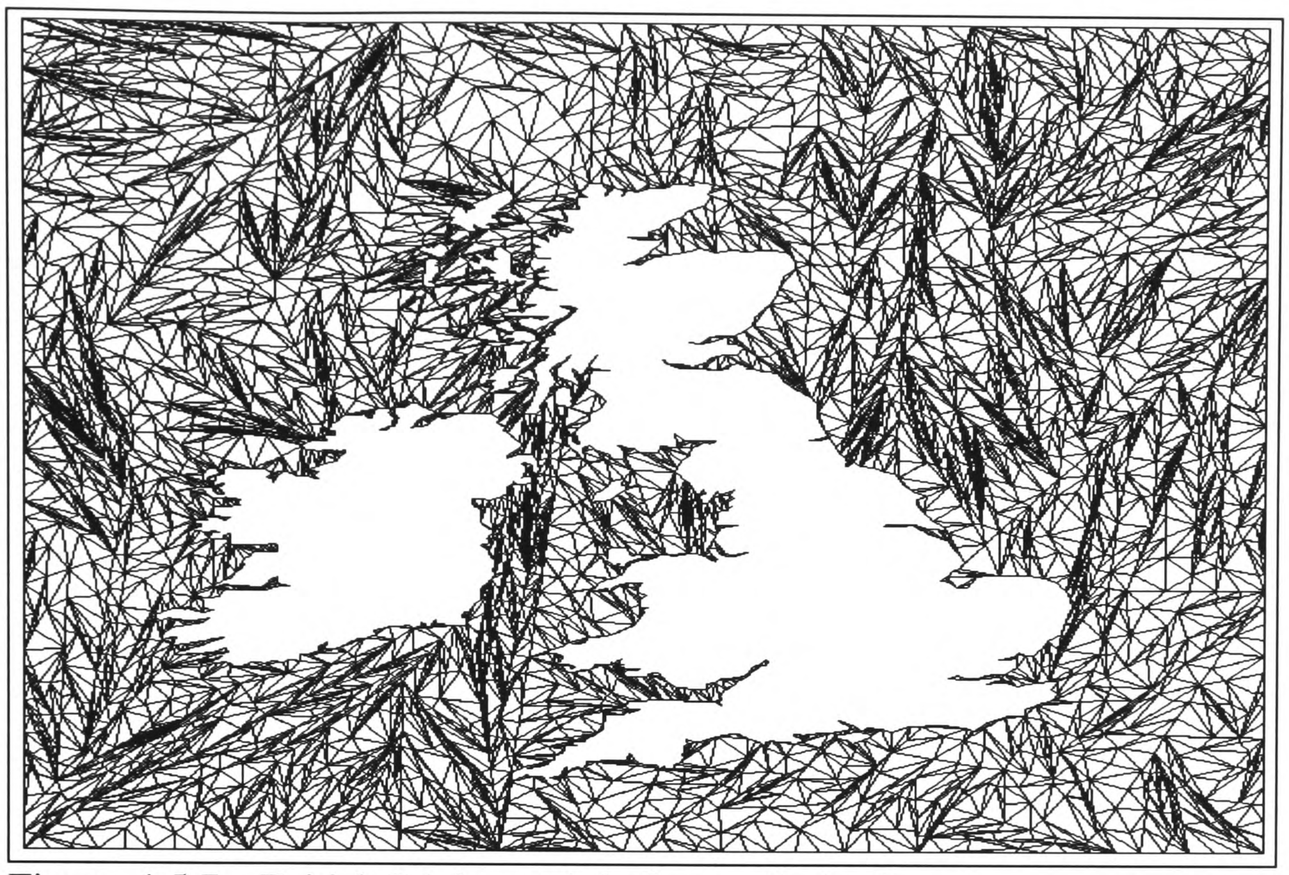


Figure 4.5.7 : British Isle's mesh before optimization.

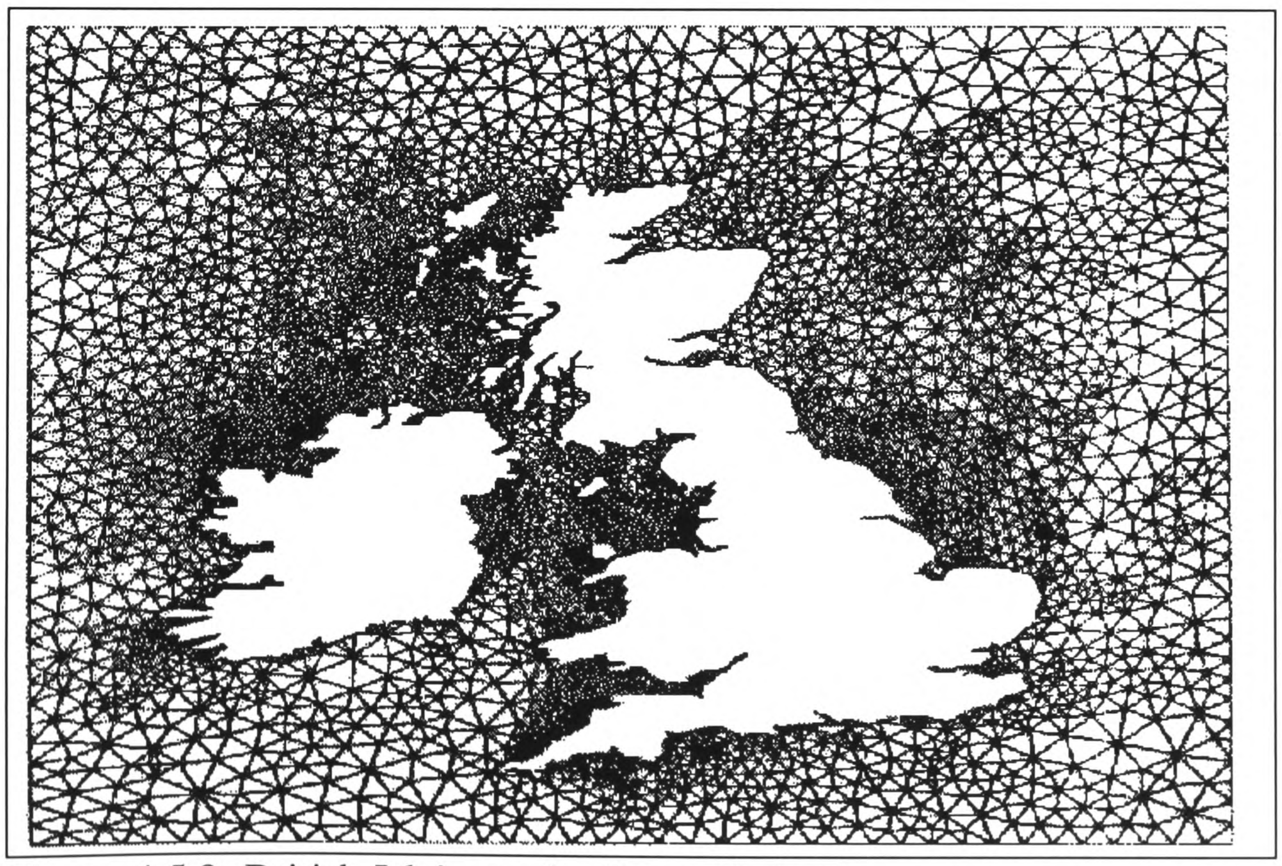


Figure 4.5.8: British Isle's mesh after optimization.

4.6 Subsequent extensions.

From the initial work carried out, it was soon established that a far more sophisticated nodal placement algorithm was required, with improved geometry input specifications.

4.6.1 Data input requirements.

It was found, for bench mark application, that a specific number of elements was required rather than a nodal spacing. There is a need to cope with multi-materials, and the following geometry input requirements were identified.

(a) The number of elements the mesh generator should generate for this problem.

(b) A list of boundary nodes of the domain/domains.

(c) The number of polygon regions in the model.

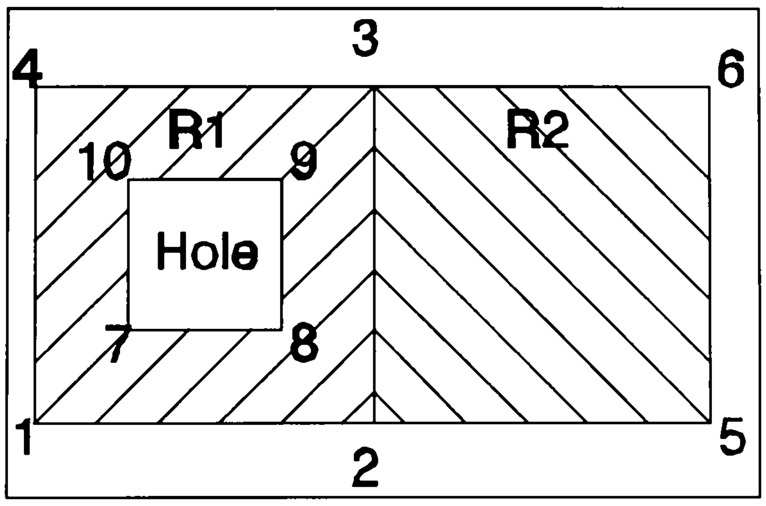
(d) Number of boundary nodes in each polygon domain.

(e) List of boundary nodes which form these regions

Two simple examples of typical data input follow:-

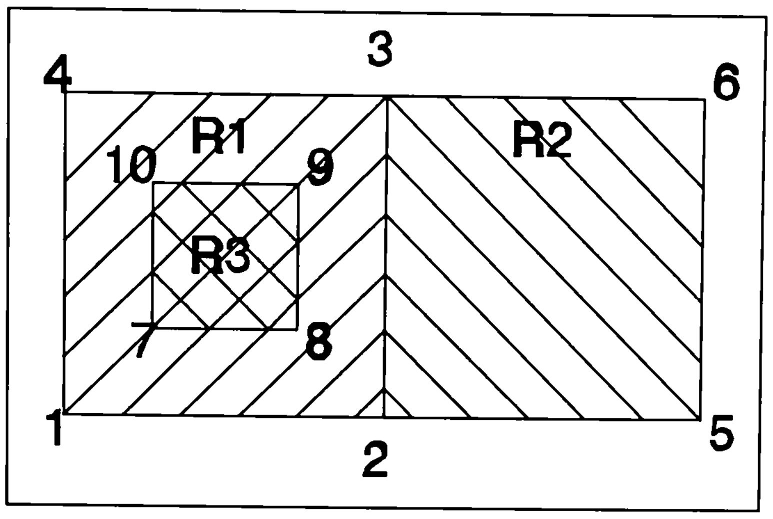
Example 1 square with hole, adjacent to another square

```
SQUARE WITH HOLE ADJACENT TO ANOTHER SQUARE
50          -- Number of elements required.
10 3        -- Number of nodes, number of polygons.
4 4 -4      -- Number of nodes in each domain, negative if hole polygon.
0.00 0.00   -- list of 10 nodes
1.0  0.0
1.0  1.0
0.0  1.0
2.0  0.0
2.0  1.0
0.25 0.25
0.75 0.25
0.75 0.75
0.25 0.75
1 2 3 4     -- Polygon outer R1.
2 5 6 3     -- Polygon node list outer R2.
7 8 9 10    -- Inner hole polygon of R1.
```



Example 2 : square with sub-domain inside, adjacent to another square.

```
SQUARE WITH SUB-DOMAIN ADJACENT TO ANOTHER SQUARE
50            -- Number of elements required.
10 3          -- Number of nodes, number of polygons.
4 4 4         -- Number of nodes in each domain.
0.00  0.00
1.0   0.0
1.0   1.0     -- list of 10 nodes
0.0   1.0
2.0   0.0
2.0   1.0
0.25 0.25
0.75 0.25
0.75 0.75
0.25 0.75
1 2 3 4       -- Polygon outer R1.
2 5 6 3       -- Polygon node list outer R2.
7 8 9 10      -- Inner polygon outer R3.
```



The above data format only handles linear elements, curved lines have to be broken down into several line segments. However, the above format handles most cases which have been provided by other co-workers [Cho93][Fry94] at the Centre for Numerical Modelling and Process Analysis, University of Greenwich.

The new mesh generator identifies which polygons are internal and their associated external counterparts. It also reorders the polygon list into anti-clockwise order so the domain is always to the left as you travel round the boundary. The identification of interface elements is also found so nodes generated on these elements' faces coincide with both domains. However the boundary for sub-region R3, in example 2, is stored as two lists one in anti-clockwise order and the other, clockwise with all nodes marked as interface points.

4.6.2 Nodal density calculation.

The previous nodal placement algorithm tended to needlessly over refine certain regions, also a method of generating grids where a certain number of elements is specified was required. It was found that if a domain was broken down into several simpler convex regions, these could be used to calculate the total area [Mid87] of the domain. Once the total area is calculated a measure of the nodal spacing can be estimated as follows:

Area of element = area of domain divided by number of required elements

Hence:

Nodal spacing = square root of four times area of element squared divided by root three.

This equation calculates the length of an equilateral triangle's side.

4.6.3 Decomposition of regions into convex polygons.

Before the generation of nodes the region is first divided up into convex polygons. The dividing of regions into convex parts is a well researched area with a large number of papers published. The method which was selected is by Chazelle [Cha84] whose algorithm has a linear computational order, see also [FeP75],[Sch78],[JoS86],[Lyu63] [GiA81],[BaD92].

It is vital that the selection of separators does not generate small angles and narrow regions. It is also required that any newly generated nodes do not lie too close to adjacent points. Let R be a simply connected region with vertices in counterclockwise order. We then select a vertex v_0 such that its interior angle is larger than 180 degrees. An inner cone is defined as in Figure 4.6.1 which defines a section ∂R of R. From ∂R it is found the subset VS visible section. In Figure 4.6.1A VS $=N_0,N_1,V_5,V_6,N_2$ where N_0,N_1 and N_2 are used to define end points of visible polygons. Therefore, a point on VS connected to v_0 is a separator which resolves the reflex angle at v_0.

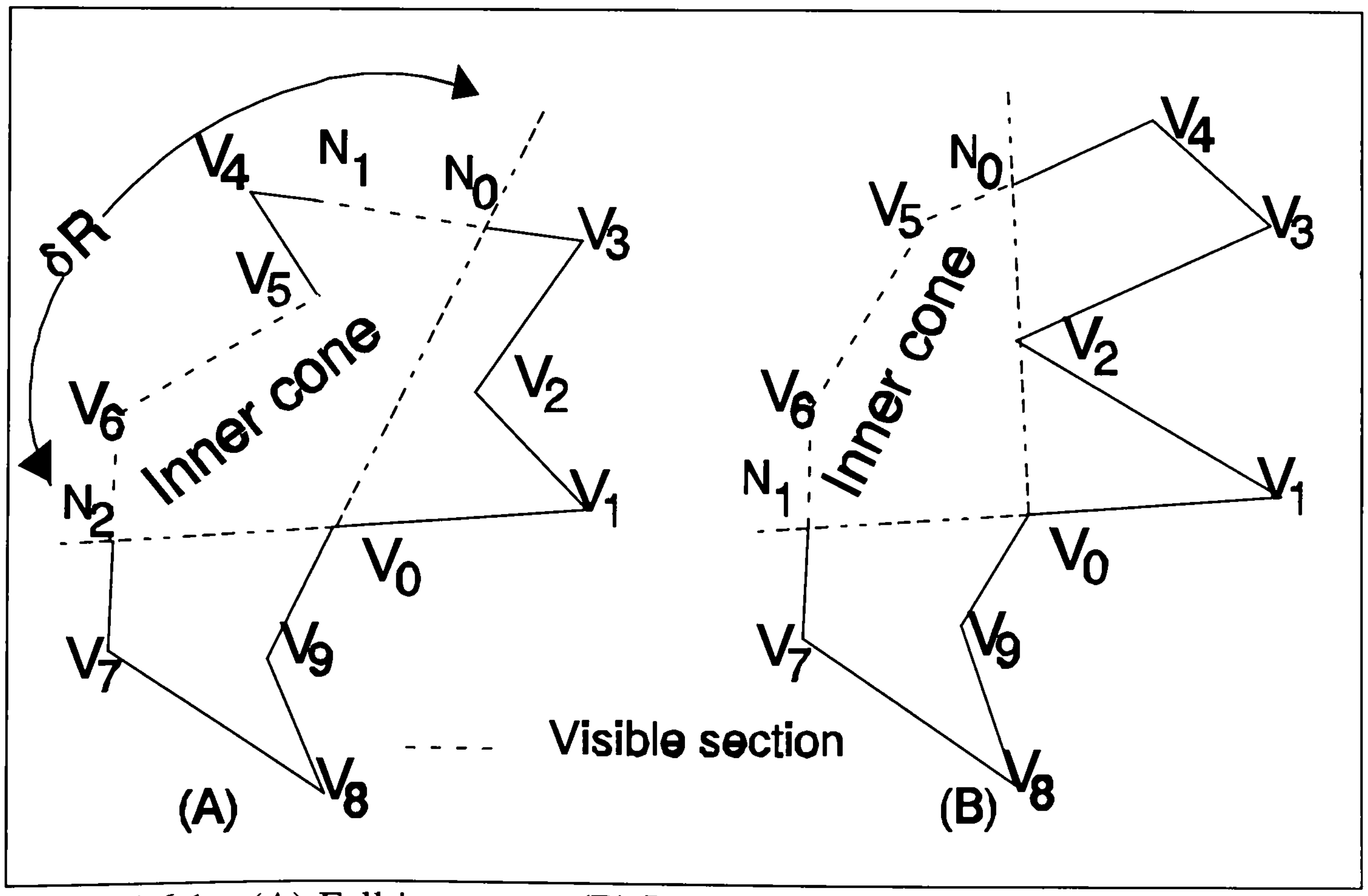


Figure 4.6.1 : (A) Full inner cone,(B) Inner cone restricted by vertex V_2

From VS we generate a set of separators, these do not include N_i points as these generate inferior regions. However a set of additional ϖ points are generated by dividing the inner cone angle, Figure 4.6.2. The subtended angle σ is between 20 to 30 degrees depending on inner cone. Therefore the candidates for separators are selected from the set of V_i, which lie in VS plus the ϖ_i nodes. The selection criterion is dependent on the angles the separator make with the region's sides together with how close it is to adjacent vertices.

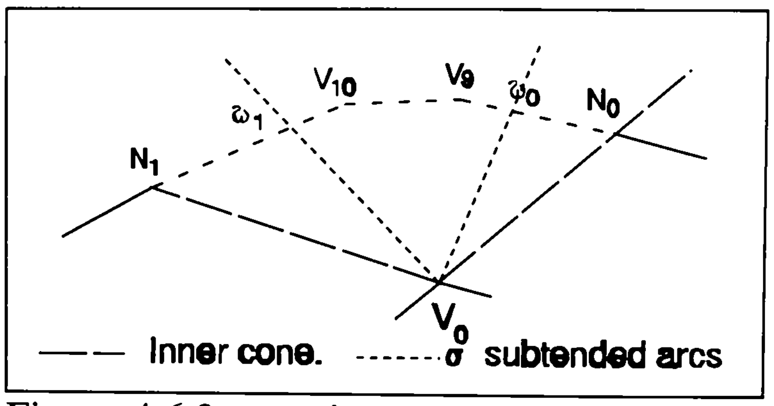


Figure 4.6.2: ϖ points.

4.6.4 Decomposition of multiply connected regions.

Multiply connected regions have to be decomposed into simple polygons. The following method outlined is based on a paper by Joe and Simpson [JoS86]. Given polygon H, which is contained in outer polygon P, search H for V_0 that is a vertex on H closest to P. Then proceed to find a separator that joins V_0 to P, which forms the cut interface segment. The search for the separator is similar to the method outline in the above section, Figure 4.6.3A illustrates inner cone.

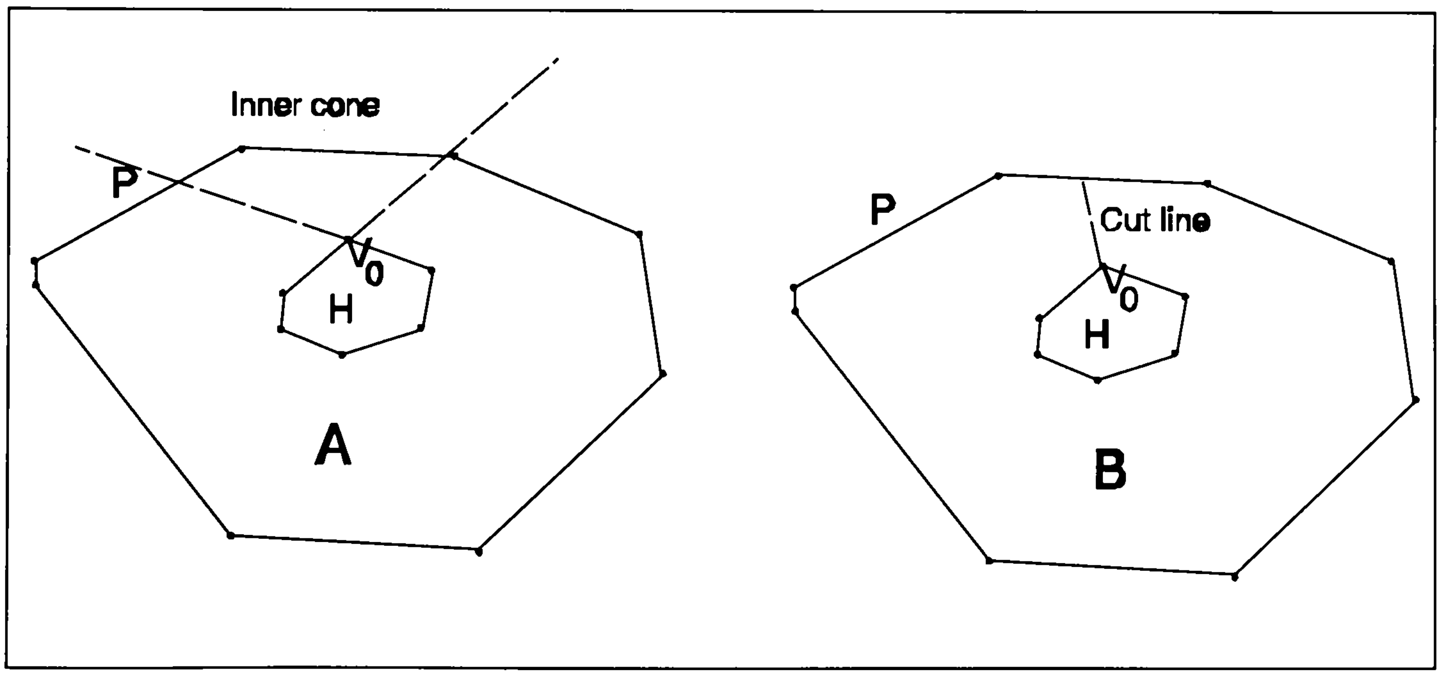


Figure 4.6.3 Cut edge generation for multiply connected region.

If no separator can be found for V_0, the algorithm proceeds by selecting nodes in a counterclockwise fashion starting at V_1 until a valid cut can be made.

4.6.5 Nodal placement

The nodal placement algorithm is based on the properties of shrinking convex polygon. Figure 4.6.4 illustrates the basic steps in this method. The first step is to generate concentric polygons, that are shrunk by a factor R, Figure 4.6.4A. We then generate nodes on the boundary of these polygons with a nodal spacing R, Figure 4.6.4B. The shrunken polygons are then discarded to leave Figure 4.6.4C.

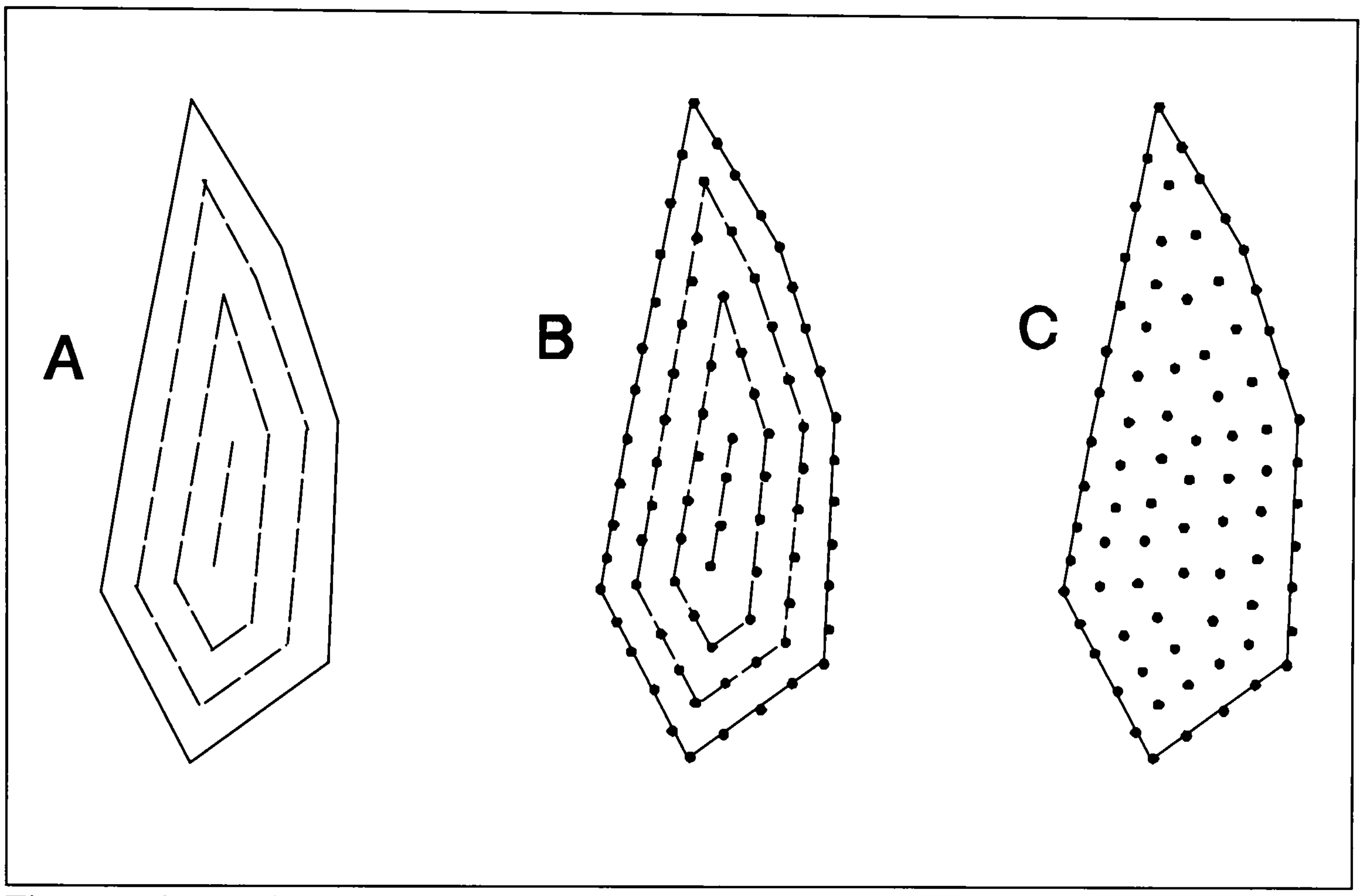


Figure 4.6.4 Node generation on a convex region.

The nodal placement technique implemented, is based on a method by Johnston [Joh92], of generating nodal points using the method of normal offsetting. Johnston's technique operates on arbitrary regions and has a computational order of approximately $O(n^2)$. However, by restricting the technique to convex regions and using a *order n* method, by Joe [Joe86] of shrinking a convex polygon, has greatly improved the computational efficiency of this technique.

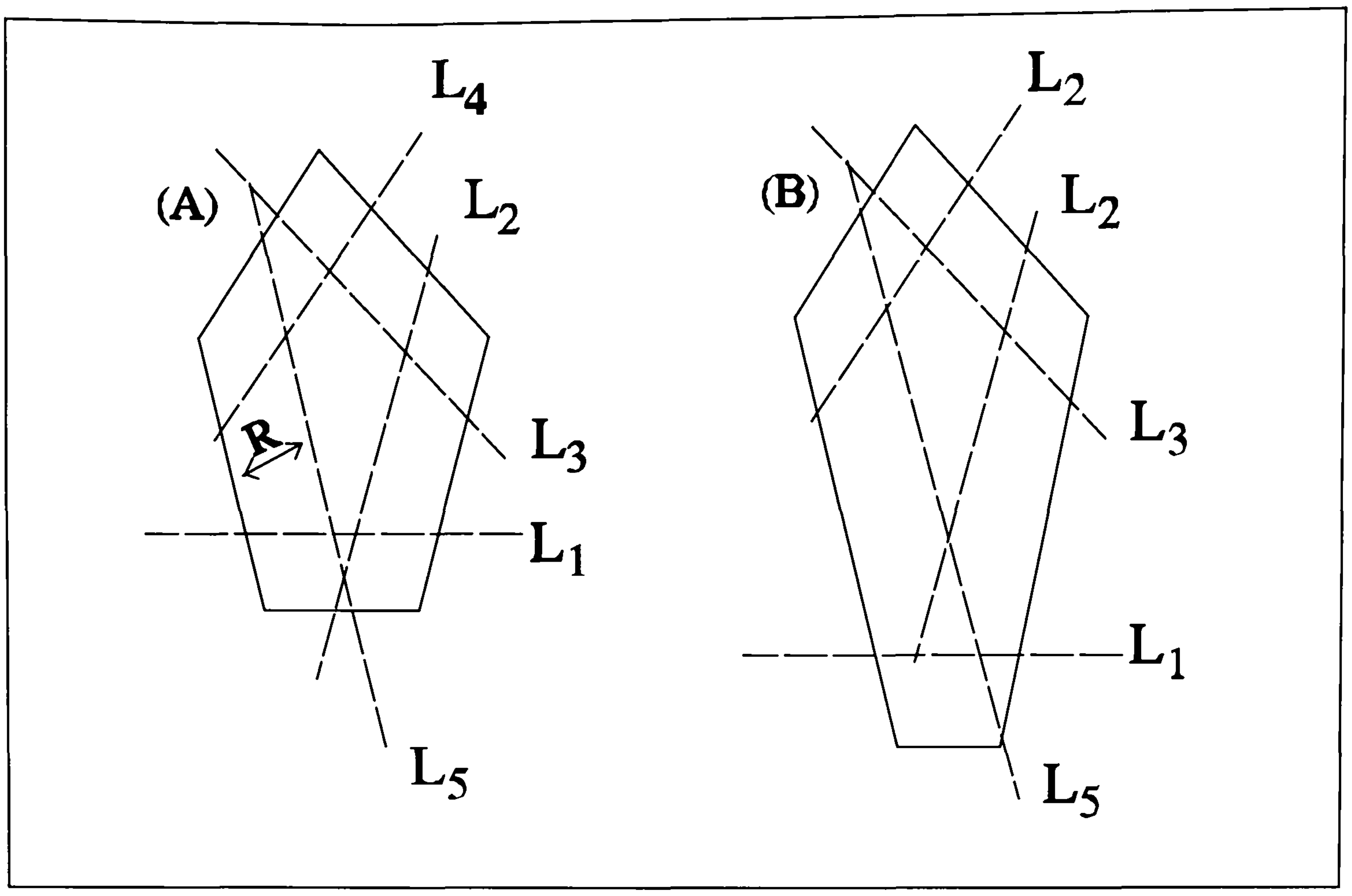


Figure 4.6.5 : Shrinking convex polygons.

Let $p_0,p_1...p_m$ be vertices of the convex region P in counterclockwise order. Now let L_i be the direction vector parallel and of a distance R to the left of the line $p_i\ p_{i+1}$. The half-plane to the left of and including L_i is H_i. If the shrink factor R is sufficiently small then the intersections of the half-planes define a convex polygon. If the intersection of the vectors L_{i-1} and L_i lie on the edge of this region these define a point q_i. Figure 4.6.5 shows two examples of shrinking convex polygons, Figure 4.6.5B show an example where an edge L_1 is not include in the sub-polygon.

It should be noted that if R is too large the shrunken polygon can be degenerate, as in Figure 4.6.6, or a line segment or even a single point. The algorithm for the generation of a shrunken convex polygon is based on paper by Joe [Joe86] and runs in computational time of order n.

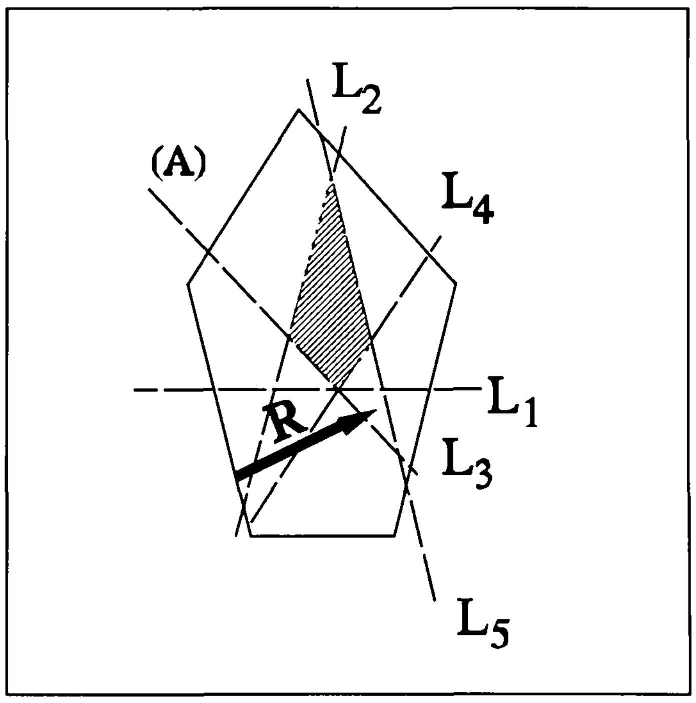


Figure 4.6.6 Degenerate polygon.

Once the sub-polygon is generated it is then processed to ensure an even nodal spacing. This is achieved by searching for narrowness or short edges. Narrowness in the convex region is identified by sharp angles. Therefore, adjacent edges that have an angle ϕ, less than a minimal value of 30 degrees [JoS92], are removed, Figure 4.6.7A. This is accomplished by generating a line L_c, Figure 4.6.7B, offset into the domain by a distance α from the node d_n, where d_n is the common node of the adjacent edges. The value of α is set proportional to the angle Φ and the nodal spacing H, and is calculated by $H/2tan(\Phi/2)$. The line L_c is then used to remove the sharp angle, Figure 4.6.7C. Short edges, vertices whose distance apart is less than δ, are then searched for, and any vertices found badly placed are merged as in Figure 4.6.8.

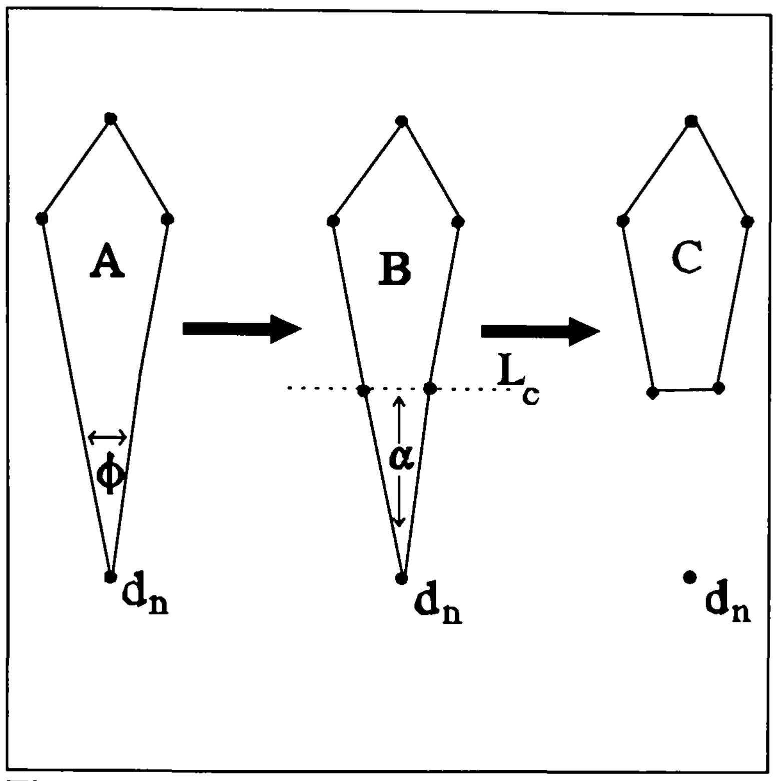


Figure 4.6.7 : Removing narrowness in the domain.

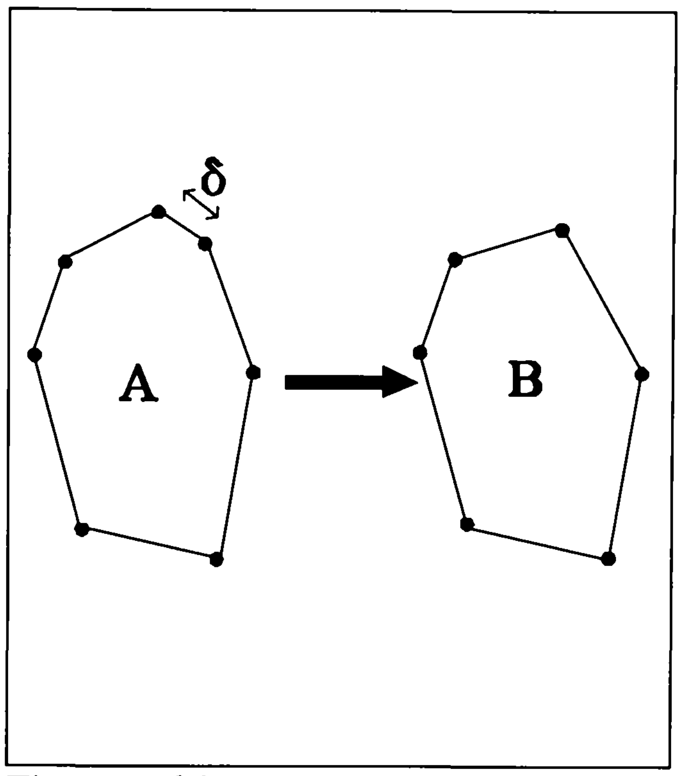


Figure 4.6.8 : Removing degenerate vertices.

The shrink factor R is the nodal spacing required for the region, however this can vary over the whole domain. In the mesh generator each vertex is assigned a nodal spacing parameter, so the mesh can be graded around areas of complexity. For example around a curve where the boundary nodes are closer together, possibly smaller than the global nodal spacing, then the local nodal spacing would reflect this.

For domains where no sub-regions can be generated, not even a point sub-polygon, the area of the region is found [Mid87]. If the area of the region is larger than 1.5(n-2) times the required element area, where n is number of nodes on the boundary, a node is placed at the centroid to ensure an even nodal density.

4.6.5 The grid generator.

Once the nodes are generated in a convex region, the method by Lewis and Robinson is applied to generate the grid. When all the sub-regions are meshed the domain is then smoothed using the methods in section 4.4.

4.6.5 Example problem.

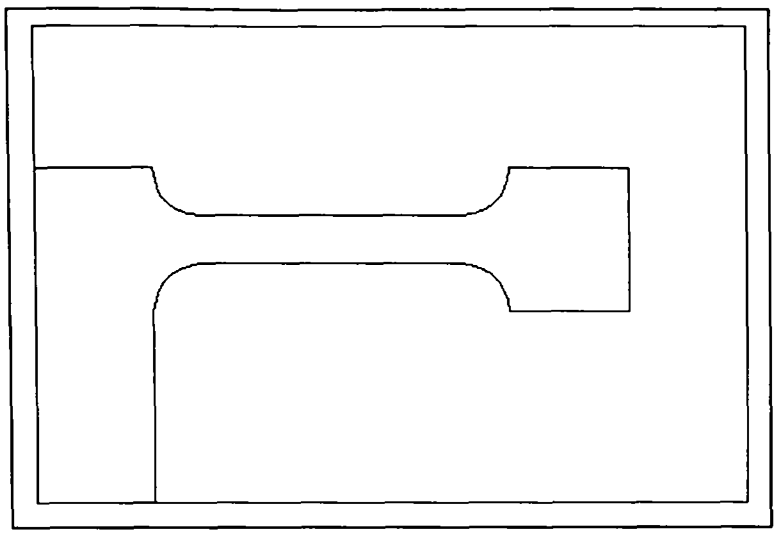


Figure 4.6.9 : Initial geometry.

Here is an example of a two material casting problem.

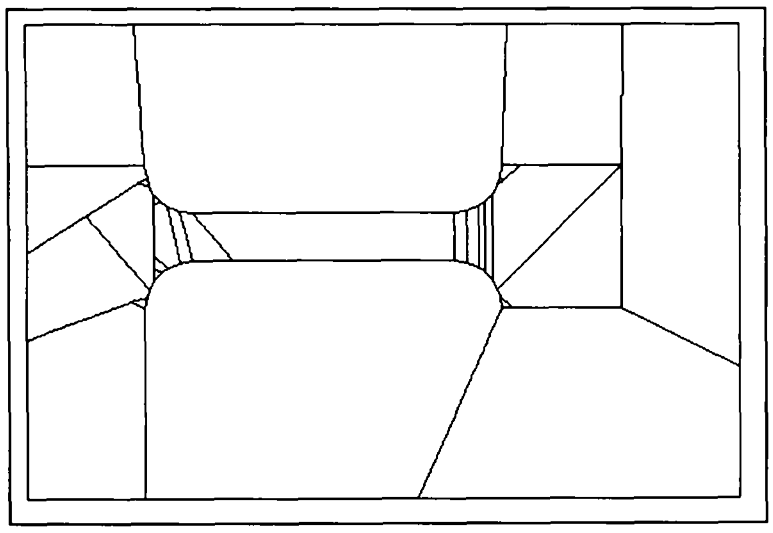


Figure 4.6.10 Sub-division into convex parts.

The division into sub-regions took 0.033 CPU seconds on a Sun sparc ELC with no compiler optimization. The region was subdivided into 28 convex polygons.

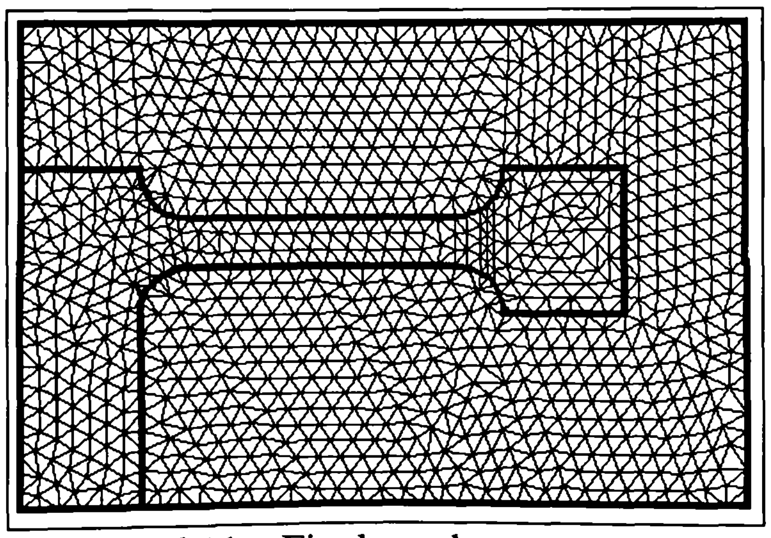


Figure 4.6.11 : Final mesh.

The complete mesh of 1099 vertices and 2068 elements took 0.1 CPU seconds after sub-division.

Minimal element angle is 15 degrees.

4.7 Higher polygon order.

The need to generate higher order elements (i.e. not just triangular) is a requirement of most applications. In this section, three different methods for the generation of other element types are outlined.

4.7.1 Conversion of triangular element to quadrilaterals.

Various methods [JSK91][MLC83] have been used to convert triangular meshes to quadrilaterals. One method is where the subdivision of triangular elements into three quadrilaterals [JSK91] is used, Figure 4.7.1.

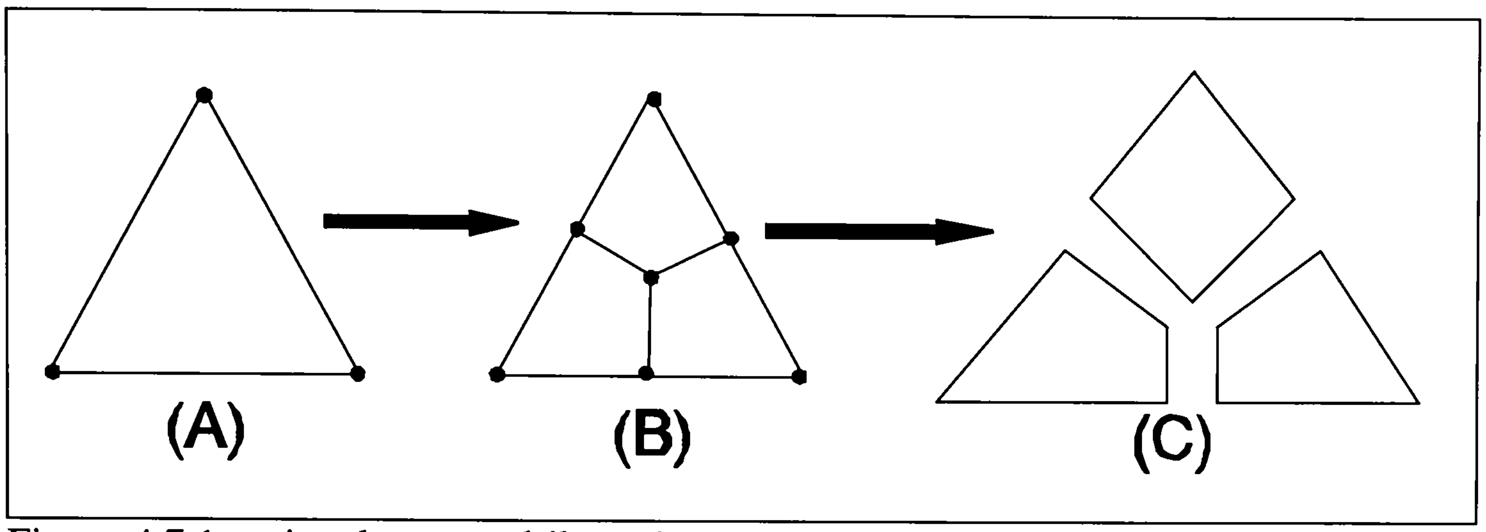


Figure 4.7.1 : triangle to quadrilaterals conversion.

This is done by generating three nodes at the mid-point of each edge with an additional node at its centroid. The nodes are then connected as in Figure 4.7.1b to form three new quadrilateral elements. The method has the advantage that all the elements are converted, however the initial mesh should be generated with less than half the required elements. This method tends to generate poorly shaped elements that are often not suitable for further computational proposes.

4.7.2 Walking method of generating quadrilaterals.

A method that generates meshes of mixed elements is illustrate in Figure 4.7.2.

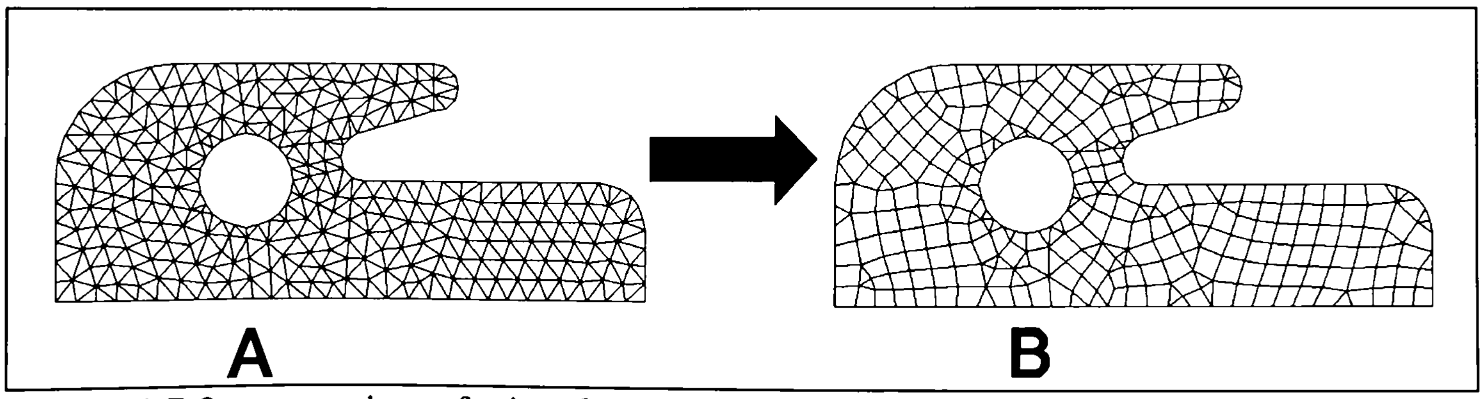


Figure 4.7.2 conversion of triangles to quadrilaterals.

The algorithm is as follows.

```
(a)  Find two adjacent elements that form a reasonable quadrilateral
(b)  Push quadrilateral onto stack
(c)  Pull quadrilateral 'Iquad' off stack
(d)  Loop over sides 'Jsides' of Iquad
(e)          if Jside has an adjacent triangles Itri
(f)                  see if we can form a new quad 'Jquad' with adjacent triangle of
                     Itri
(g)                  if new quadrilateral formed push 'Jquad' onto stack
(h)          end if
(i)  endloop
(j)  If stack not empty goto (c)
(k)  Search remaining triangle elements to see if we can form a quadrilateral
(l)  if new quadrilateral formed goto (b)
(m)  Terminate.
```

To speed up the search for adjacent elements the above routine stores a list of non-converted elements. The routine always selects the optimal adjacent element when forming quadrilaterals. However no quadrilateral is formed if any angle is less than 20 degrees. After the generation of quadrilaterals the mesh is smoothed to optimize angles. Similar techniques have been implemented by Johston et al [JSK91] and Moscardini et al [MCL81].

4.7.3 Generation of high order polyhedral cells.

A region shown in Figure 4.7.3 can be decomposed into convex parts, Figure 4.7.4.

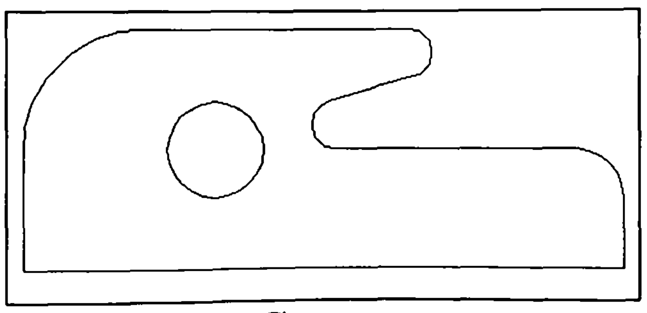


Figure 4.7.3 : Geometry

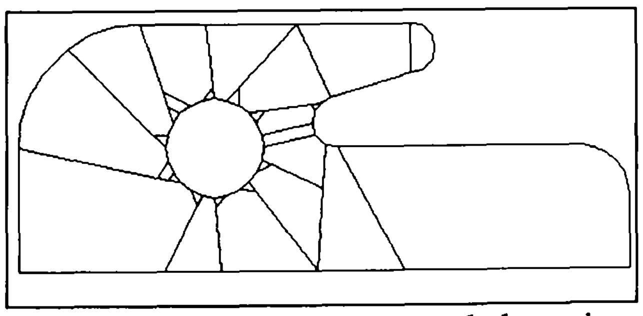


Figure 4.7.4 : Decomposed domain.

Figure 4.7.4 could be classified as a "mesh" and from this initial decomposition we can proceed to generate grids over these sub-domains using several techniques.

As an example, these convex sub-domain can be further subdivided to generate arbitrary polygon cells, Figure 4.7.5.

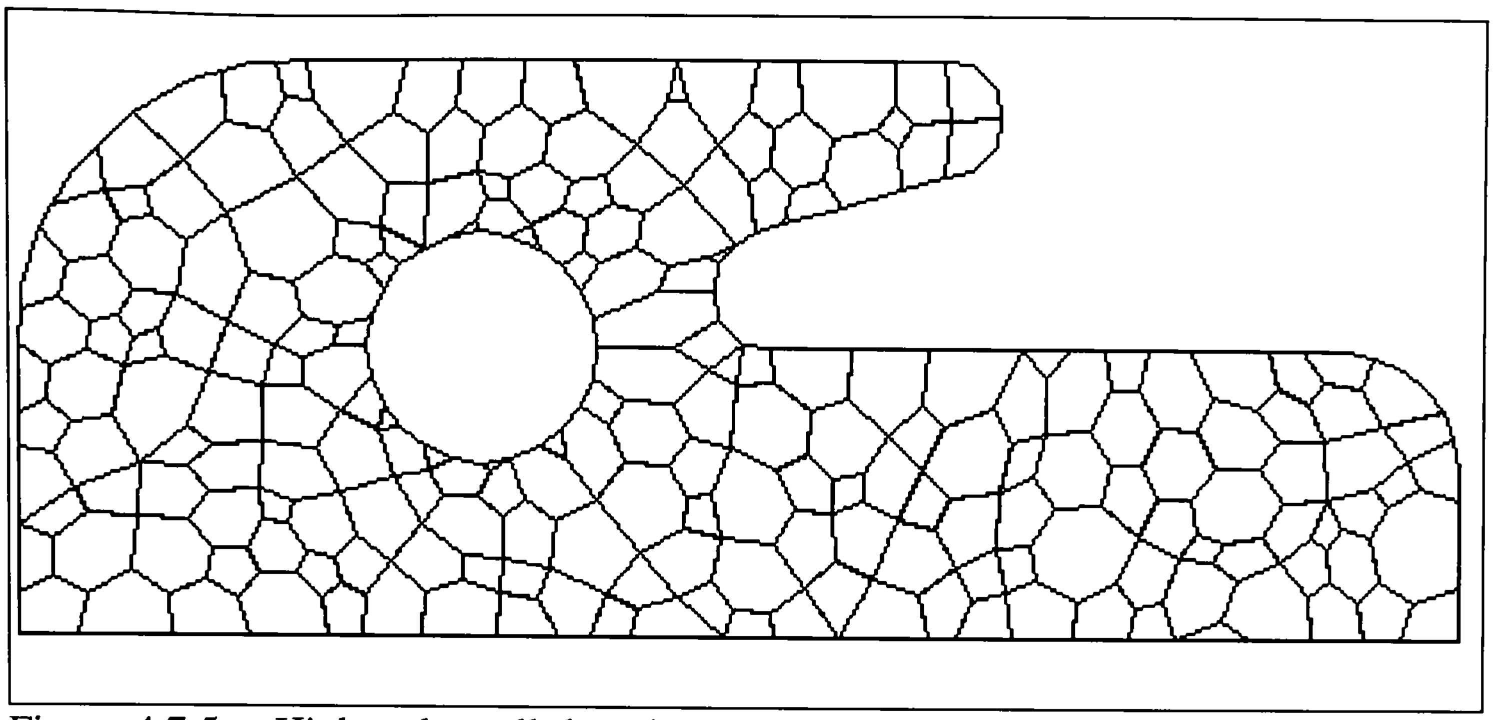


Figure 4.7.5 : High order cell domain.

The above domain was generated using a technique where each cell was divided until its area was within a special tolerance. This tolerance was calculated from taking the total area of the domain and dividing by the total number of required elements. If we increase the number of elements we end up with the mesh shown in Figure 4.7.6.

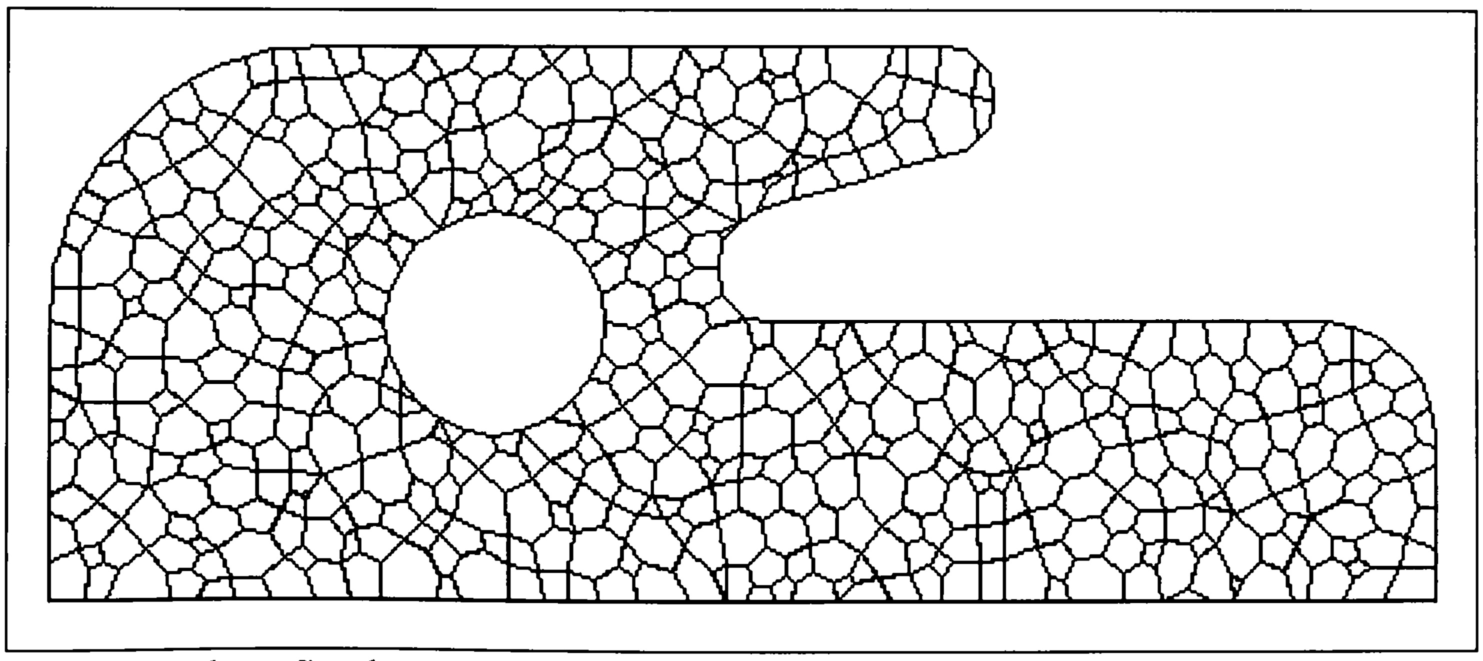


Figure 4.7.6 : refined mesh.

Figures 4.7.5 to 4.7.6 have been smoothed. Before optimization these meshes would have resembled Figure 4.7.7.

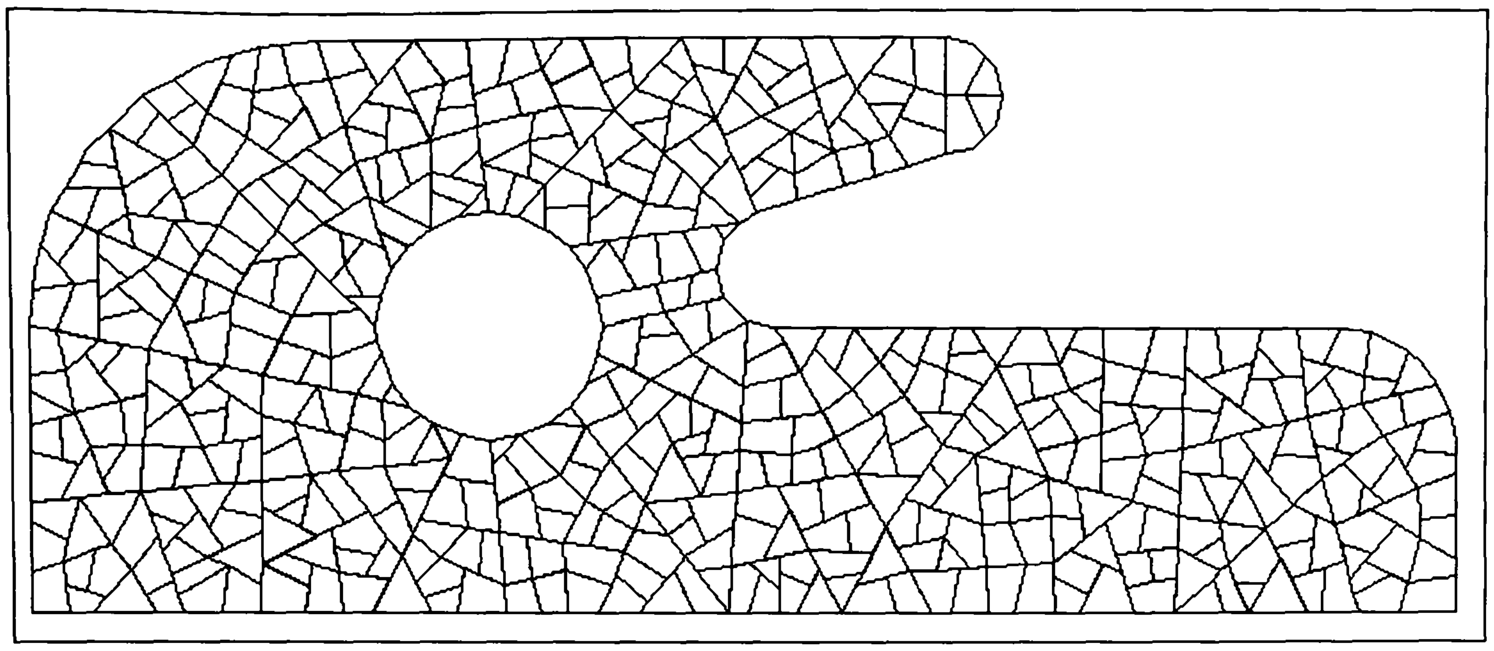


Figure 4.7.7 : Crazy pavement mesh.

Figures 4.7.5 to 4.7.6 are similar to the types of grids generated by methods such as the Voronoi Diagram [CFM91].

4.8 Conclusions.

All the algorithms presented in this chapter are of computational order of at worst n^2 with the majority of order n. Therefore, the method of recursive domain bisection potentially offers a fast and efficient way of generating grids. The method can cope with complex geometries with relative ease and can generate grids of usable quality. Therefore, the method presented in this chapter compares favourably with other efficient algorithms such as Delaunay. However, the new bisection method does not suffer from the drawbacks of Delaunay triangulation. For example, Delaunay triangulation requires expensive algorithms after the generation of the grid, to ensure the mesh represents the model (See Chapter 3).

Since the new bisection technique is a practical and efficient method, with some algorithms already extended to 3D by other researches in the field of solid modelling, it seems realistic to extend this technique to three dimensions.

4.8.1 Further extensions.

The new recursive domain bisection method is a very flexible technique, because once the domain is divided up, there are many possibilities. Many CAD packages, PATRAN [Pat89], FEMGEM [Fem91] etc , offer simple mesh generating tools that require the sub-division of a complex region into a number of simpler parts. Since the initial stage in the code generates simple convex domains, many of these tools can be applied directly to the resulting geometry. One example is where we have taken the geometry of a 2D car in a wind tunnel. We then applied the domain bisection algorithm to subdivide the domain into simpler parts. These subregions were imported into FEMGEN that generated the final grid. Therefore, once a region is divided up, a method of generating quadrilaterals in these simpler domains could be applied, in a similar way to the medial axis method [TaA91], see Chapter 9.

Chapter 5

Generation of Grids Over Surfaces.

5.1 Introduction.

The generation of grids over the surface of complex geometry is a major area of three dimensional mesh generation. The ability to generate surface grids is an integral part of the overall tetrahedral mesh generator.

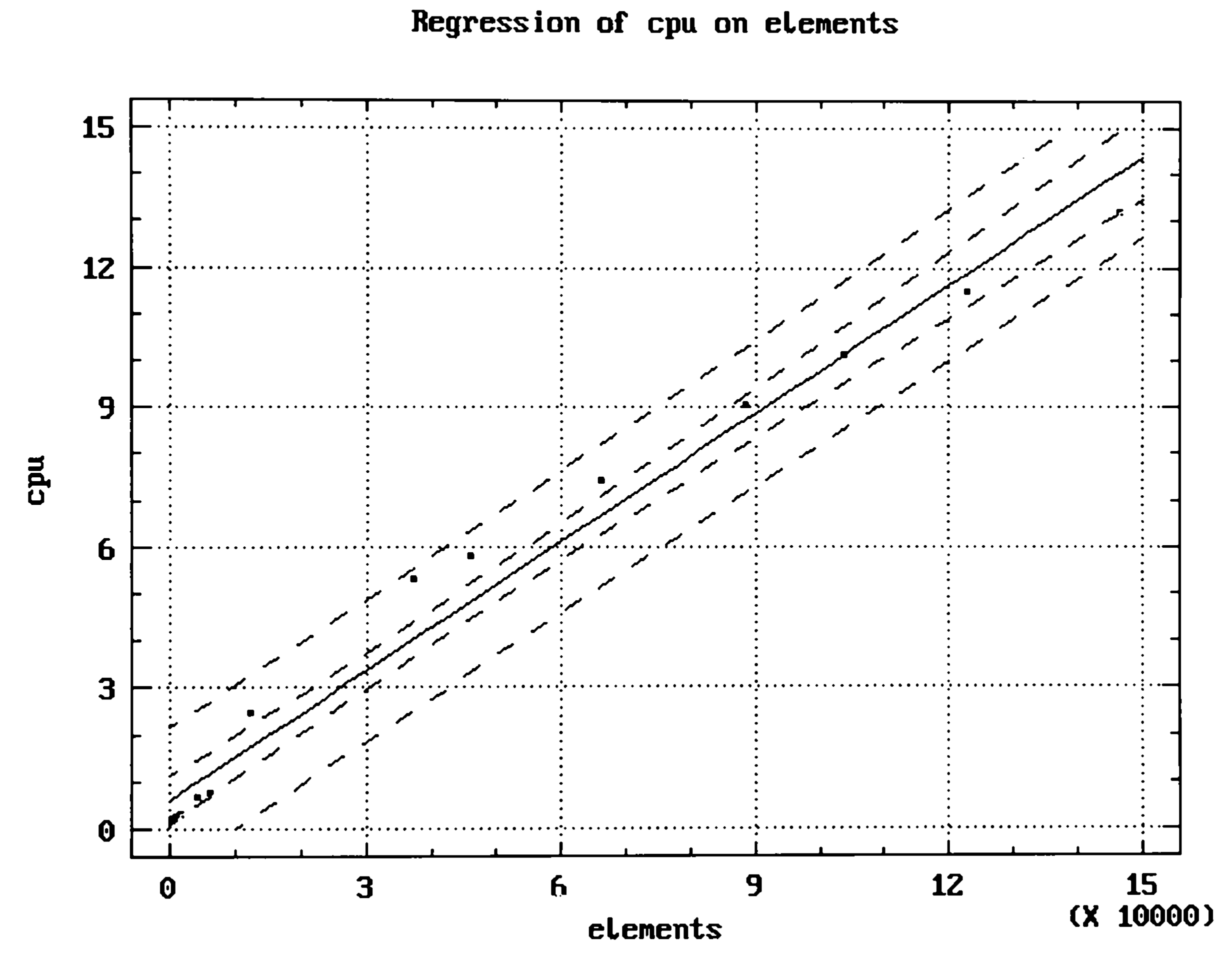
The first section outlines a method of generating surface grids over planar polyhedral structures, with examples that show an almost linear computational order. Then grid generation over parametric surfaces will be presented, which will include a new definition of surface Delaunay triangulation. The chapter will then be concluded with how the methods used in 2D mesh optimization have been modified to cope with parametric forms of surfaces.

The Delaunay triangulation is presented in this chapter, as this work was the initial development of extending grid generation to surfaces. The recursive bisection technique is not included in this chapter, as at the time this work was undertaken, it was still under development. However, it is intended that the Recursive bisection method will later be extended to parametric surfaces.

5.2 Surface mesh generation on polyhedral domains.

Generating grids over the surface of polyhedral domains is probably the simplest form of surface mesh generation. A polyhedral domain is a collection of closed nonintersecting planar polygon faces. Each polygon face can either be simple or multi-connected.

A polyhedral domain can be treated as a collection of planar regions with interface boundaries along all edges. The reason for this is to ensure that elements generated on adjacent polyhedra are consistent. Therefore, each face can be taken in turn, rotated onto the XY plane and meshed using any reasonable planar grid generation algorithm. As can be expected, see regression analysis below, the surface polyhedral mesh generator is equivalent in computational order to the 2D version.



The above graph shows a linear regression analysis of the data given in table 5.2.1 The regression analysis of the fitted line gave a value of R^2 of 0.97 that means a linear representation of the data is good. Therefore, we can conclude that for the lug shape model in Figure 5.2.1 we have an order N algorithm.

The table below shows CPU time in seconds to generate a surface mesh of 'n' elements using Delaunay triangulation on the Polyhedral domain in Figure 5.2.1.

Table 5.2.1

No. Elements	CPU Time
272	0.20
308	0.21
436	0.25
922	0.28
1508	0.30
4136	0.70
6160	0.77
12374	2.48
37224	5.30
45960	5.83
65978	7.43
88462	9.07
103582	10.12
122968	11.5
148814	13.18

The above times were calculated on a Sun SLC with compile options -g (debug).

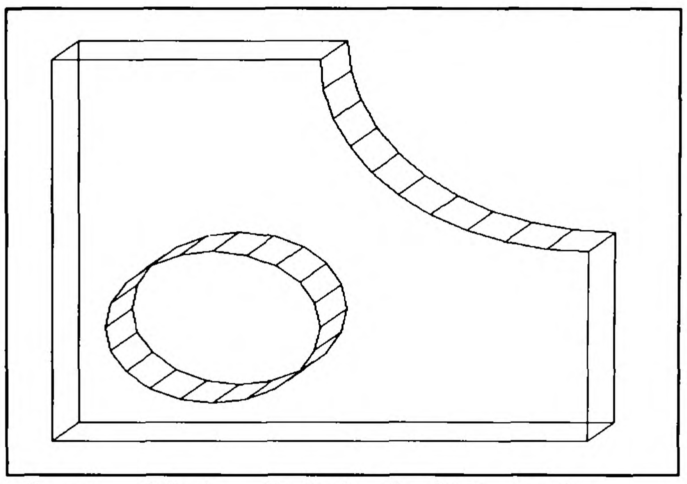


Figure 5.2.1: Polyhedral Domain.

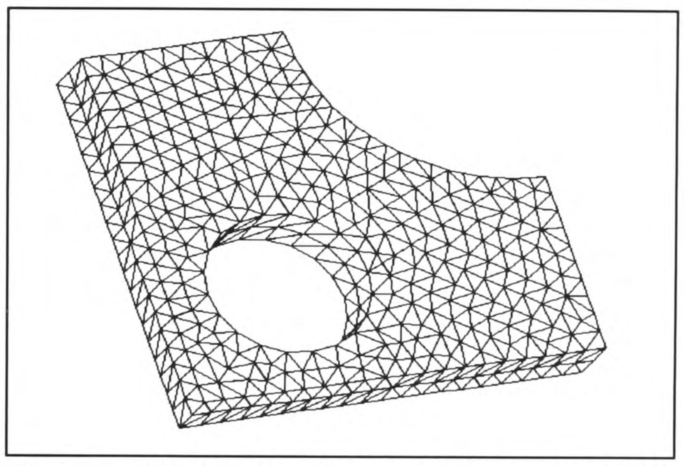


Figure 5.2.2: Surface mesh with 1508 elements.

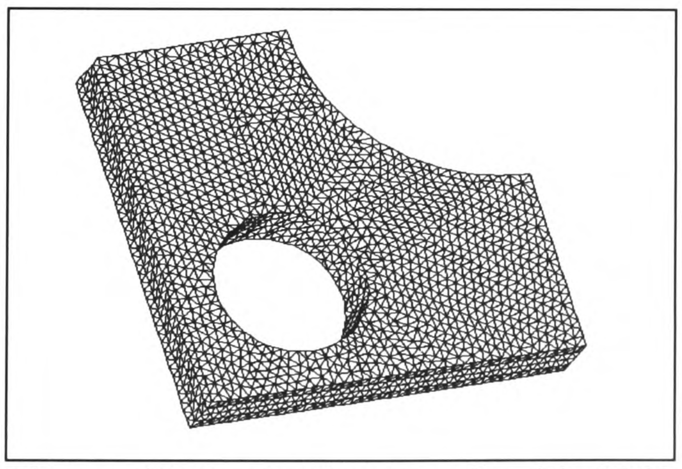


Figure 5.2.3: Surface mesh with 6160 elements.

5.3 Meshing Surfaces using Delaunay Triangulation.

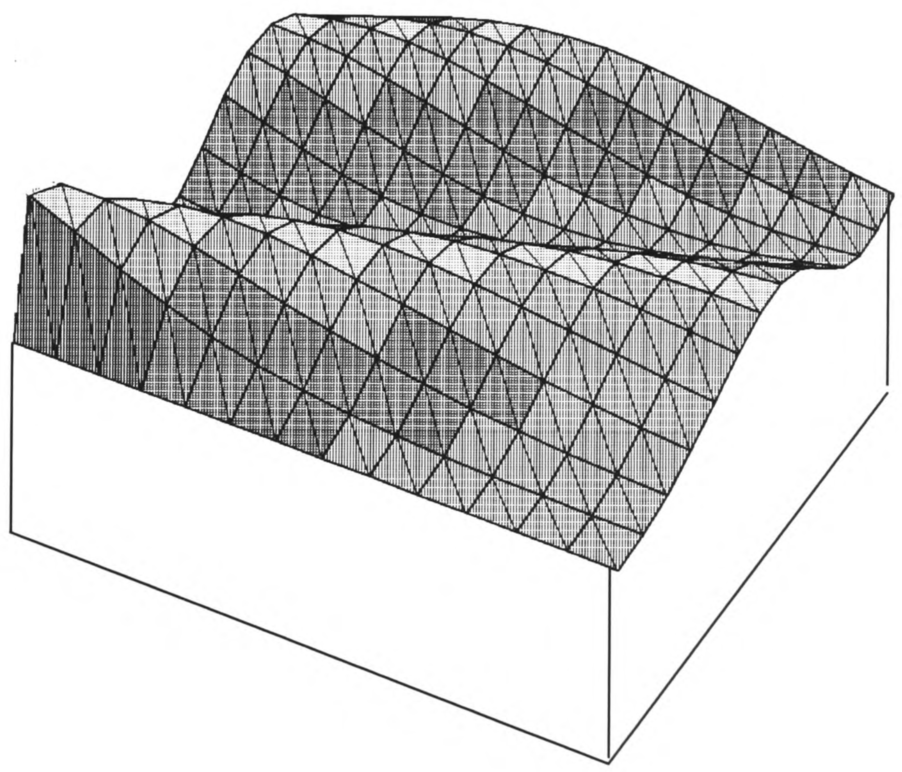
The triangulation of surfaces up to now has mainly been done by transforming the surface onto a 2D planar region and then applying some well established 2D mesh generator to this domain [CFM91b][HaA82][ZiP71]. Once the 2D planar region mesh is completed it is then converted back to the surface. Often this is done with no consideration of how well structured the mesh will be, once it is mapped onto the surface. So although you may have a well structured 2D planar region, the resulting surface mesh can be worthless for any further analysis purposes.

A method of generating well shaped elements on any surface that has an associated parametric definition, will now be considered, by means of Delaunay triangulation. An initial method will then be demonstrated, and how this can be expanded to arrive at a definition of surface Delaunay triangulation.

One way of generating well structured triangles over a surface is to use 3D Delaunay triangulation.

This is done by:-

(i) Taking a surface and generating a "Bounding base Box" (Figure 5.3.1)



Surface With Bounding Base Box,
Figure 5.3.1.

(ii) Generate nodes over this surface (using some nodal placement algorithm)

Note, that it is also beneficial to generate nodes on the base of the bounding box, as this improves the mesh generator's robustness to rounding errors i.e. cuts down the number of tetrahedral elements sharing the same node, which reduces the number of near parallel lines.

(iii) Then use a 3D Delaunay triangulation program to generate the tetrahedral mesh.

(iv) Apply some stitching method on the triangulation to make the Delaunay mesh conform to the domain. (See Figure 5.3.2)

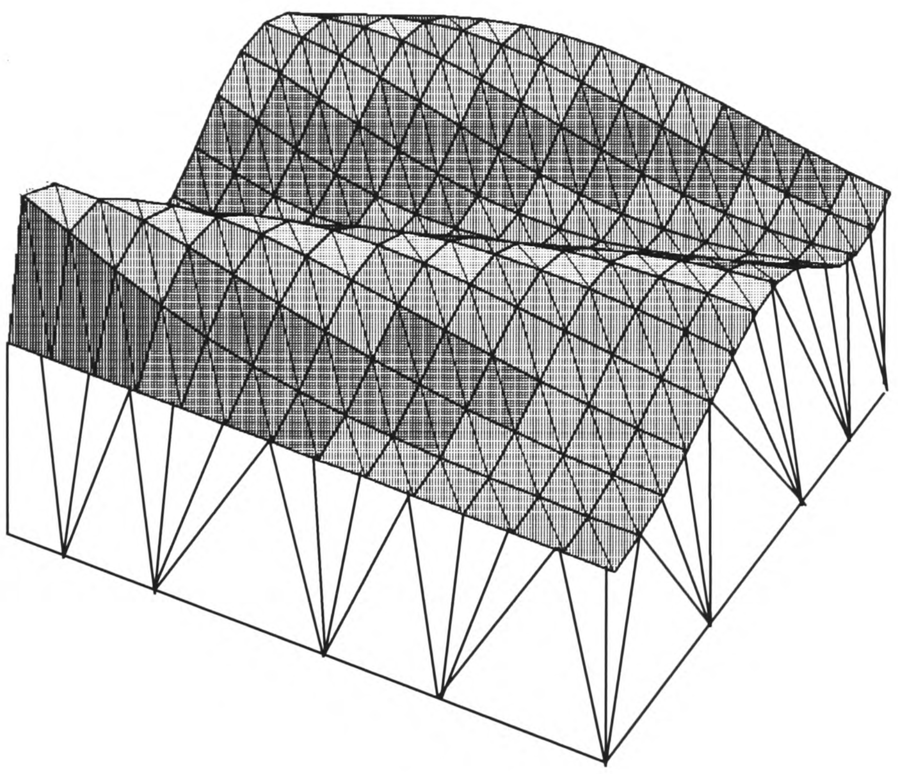


Figure 5.3.2

(v) The surface Delaunay mesh is then defined by the tetrahedral faces that are adjacent to the surface.

Note that before removing the tetrahedra, it is possible to apply a nodal refinement algorithm.

The above method does produce satisfactory surface meshes, but the surface mesh produced is dependent on the distance between the base of the bounding box and the surface. So by moving the bounding box base in or out, it is possible to change the resulting solution; this situation is not ill-conditioned as the nodal position on the surface has a greater effect on the outcome. Actually, to have any effect on the solution, a significant increase/decrease in the distance between the base and the surface must occur.

It is possible to take the above concept further and instead of imagining a surface with a half bounding box, we can think of two twin (identical) coinciding surfaces with a gap δ between them, e.g two parallel planes or parametric cubic surfaces, as illustrated in Figure 5.3.3.

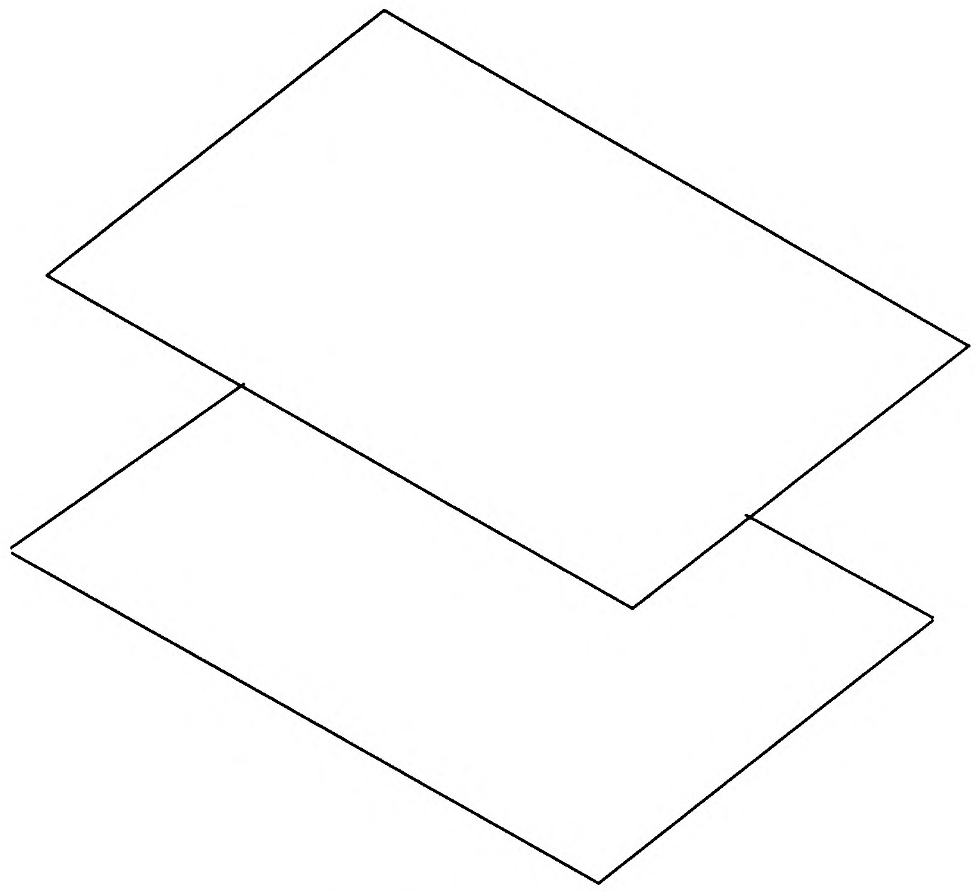


Figure 5.3.3: Parallel Planes

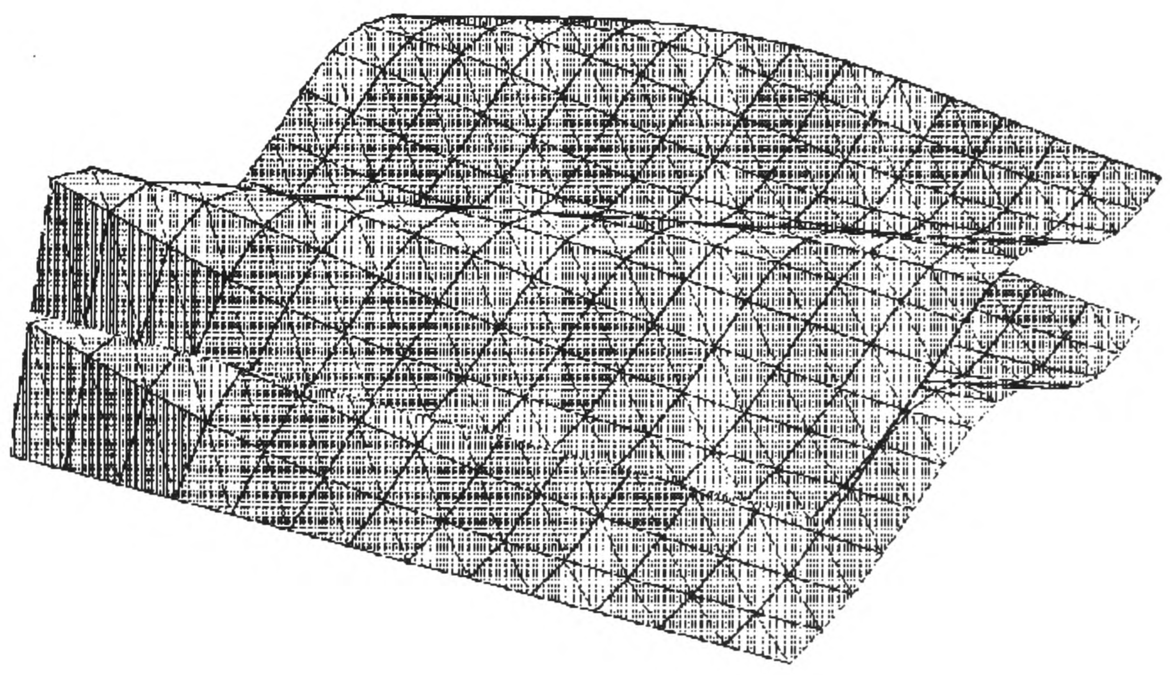


Figure 5.3.4: Coinciding Surfaces

Now generate nodes on both domains, such that the distribution and location of points are identical with respect to the surfaces' origins, and join the surfaces up to form a closed domain, see Figure 5.3.4. We can now generate a Delaunay mesh over the points in 3D space, stitch the domain and remove external tetrahedra.

If we now take a closer look at the surface of the parallel planes (Figure 5.3.3), it can establish not just the fact that both planes have identical meshes, but also the fact that this surface conforms to the definition of planar surface Delaunay triangulation. Take the parallel plane example, and let the distance between the planes tend to zero. As the distance between these surfaces decreases, the meshes on the planes do not change and the circum-spheres of the tetrahedra also decrease. The tetrahedra tend to triangles and the circum-spheres tend to minimal circum-spheres (Figure 5.3.5) of triangles as the points on the two planes start to coincide. Therefore, we are left with a valid Delaunay triangulation, but these triangles do not have circum-circles like the planar Delaunay triangulation but have minimal circum-spheres.

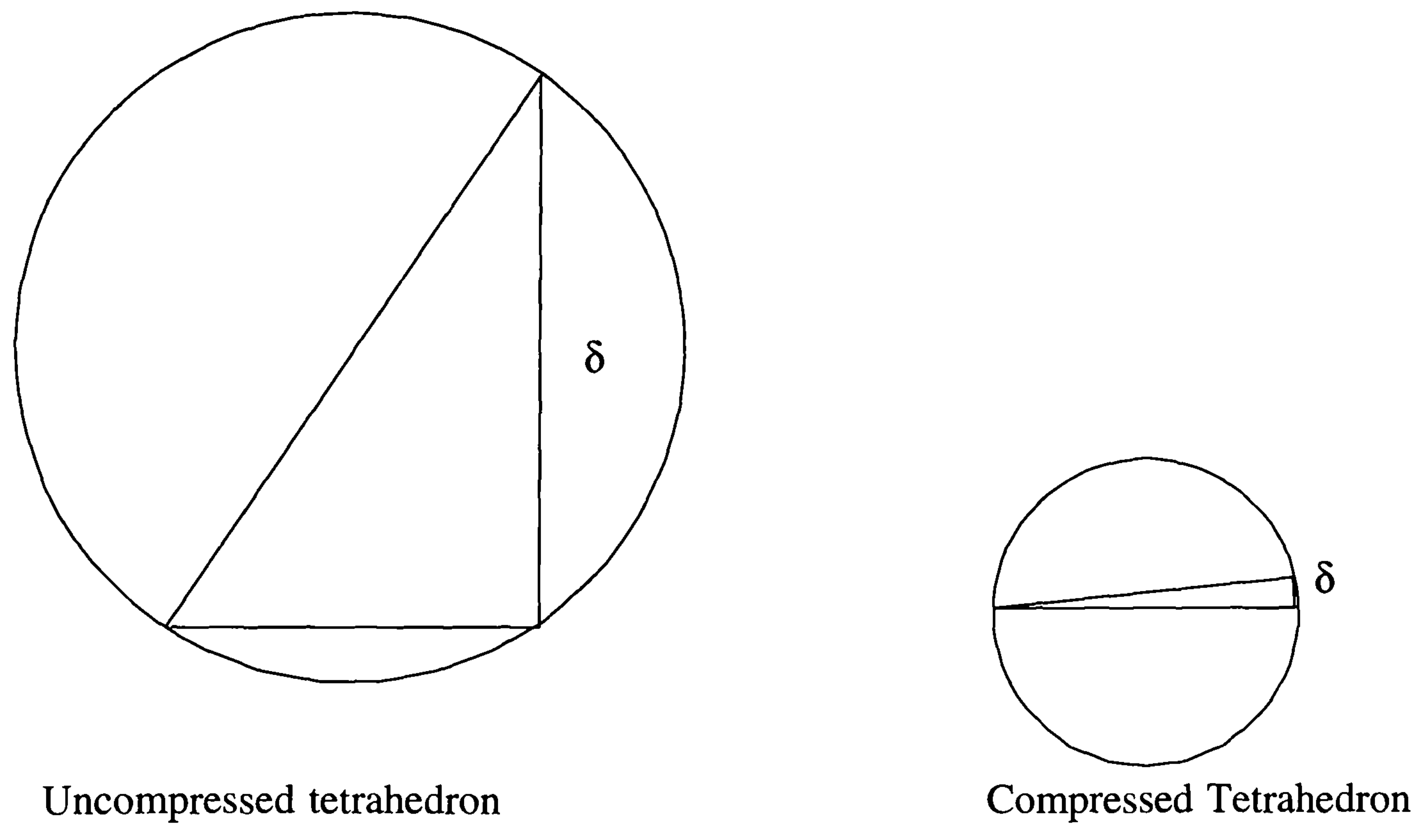
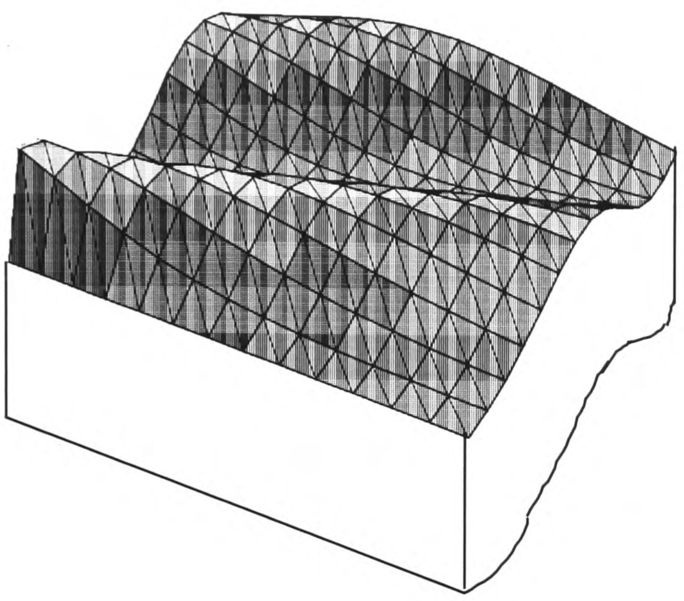
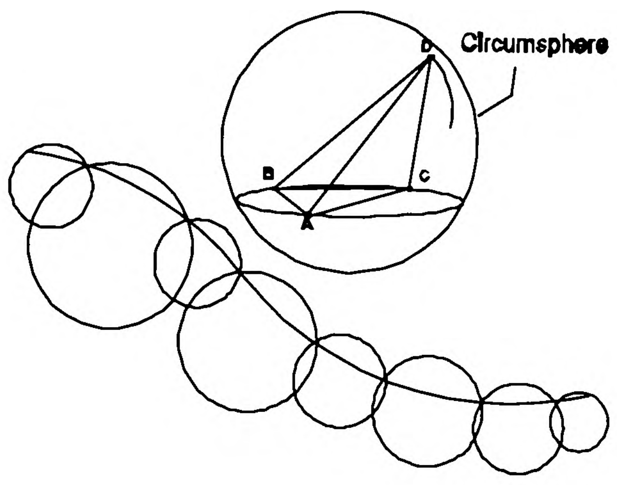


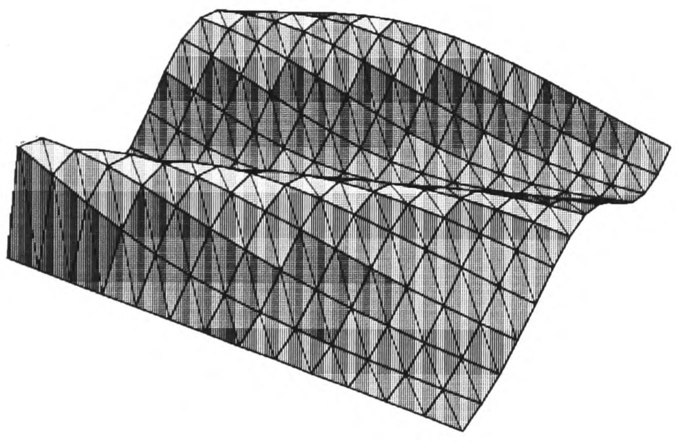
Figure 5.3.5:- Shows a cross section of circum-spheres of a tetrahedral as height δ tends to zero.

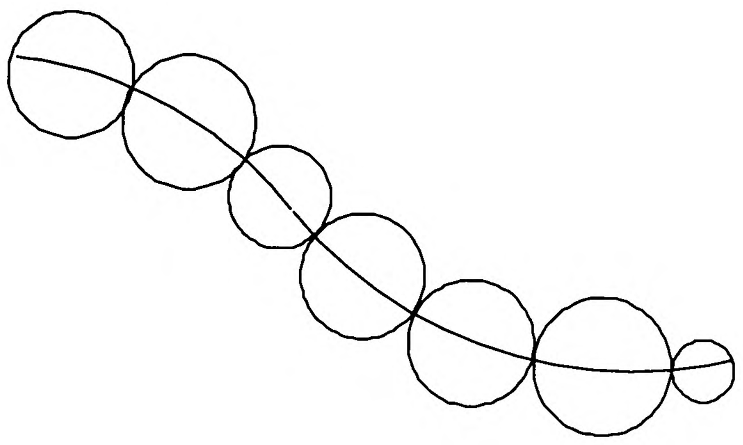
If we do the same for the twin surfaces (Figure 5.3.4) we have a surface triangulation in which no triangle's minimal circum-sphere contains any other triangle's vertices (Figure 5.3.6).





Cross section of top surface circum-spheres when $\delta > 0$





Cross section of circum-spheres when $\delta=0$
Figure 5.3.6

Therefore, the definition of surface Delaunay triangulation is : The minimal circum-sphere of each triangle and the intersection of this sphere and the surface defines a region, in which no vertices of any other triangle in that surface can be contained.

Two possible methods of producing a surface Delaunay triangulation, will now be presented.

Method 1:

(i) First sort the points in parameter space, such that they are in lexicographically increasing order.

(ii) Join the first three points to form the initial triangle.

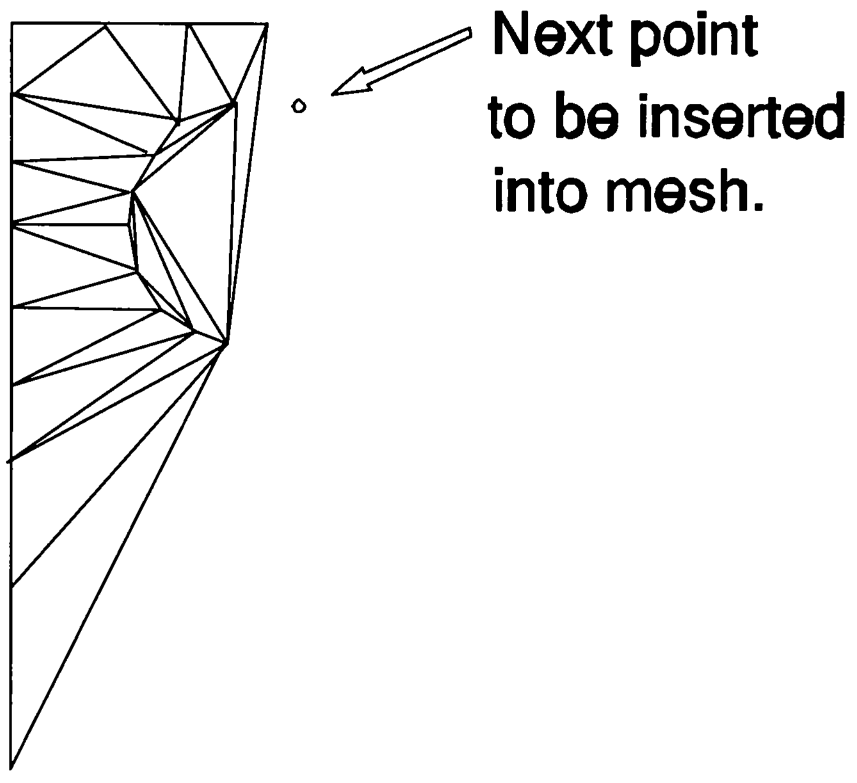
(iii) Then insert the points one at a time from outside the convex hull, see Figures 5.3.7a and 5.3.7d.

(iv) Find all edge faces of convex hull visible from last inserted point.

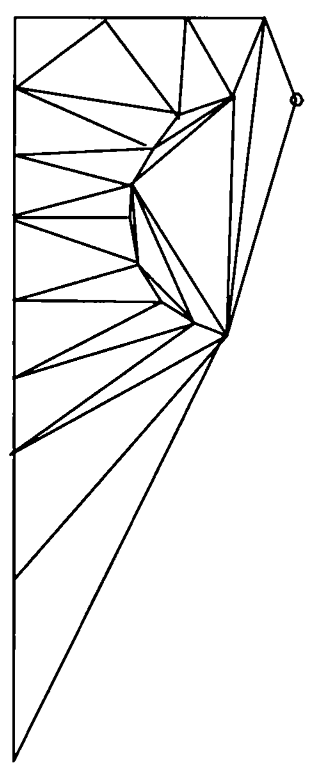
(v) Join this point up to the vertices of these faces to form new triangles, see Figure 5.3.7b

(vi) Then apply Lawson's swapping procedure to make newly generated triangles which conforms to surface Delaunay triangulation, see Figure 5.3.7c. i.e check each triangle minimal circum-sphere with neighbouring triangles' vertices. If neighbour's edge is contained in this region swap its adjacent vertex with current triangle.

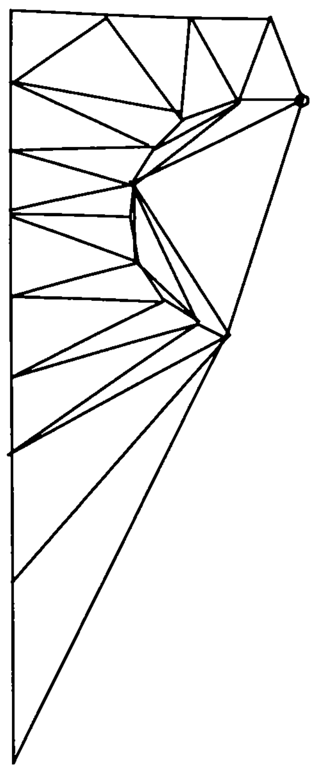
(vii) Repeat steps iv to vi until all points are inserted.



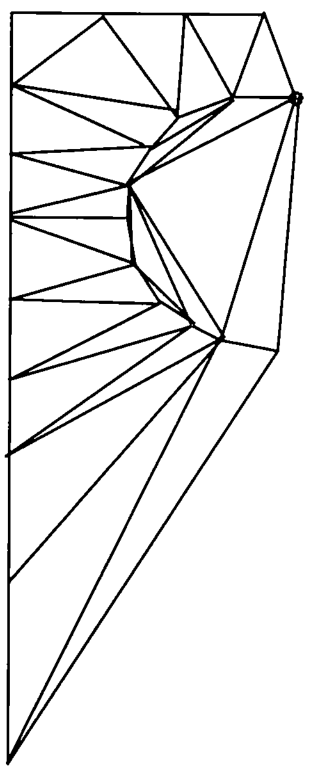
(a) Convex hull



(B) Mesh just after point is inserted, point has been joined up to visible triangle's faces



(c) After Lawson's swapping procedure.



(D) After next point is added to domain.

Figure 5.3.7

Note that a bounding or super triangle [SlH84], is not used in the above method as it is impossible to find. Also Lawson's swapping algorithm is used not just to overcome some of the problems associated with numerical rounding error, but also to address the situation when the circum-sphere of a triangle intersects the surface more than once (Figure 5.3.8). This is overcome by the fact that we are only checking neighbouring triangles' circum-spheres.

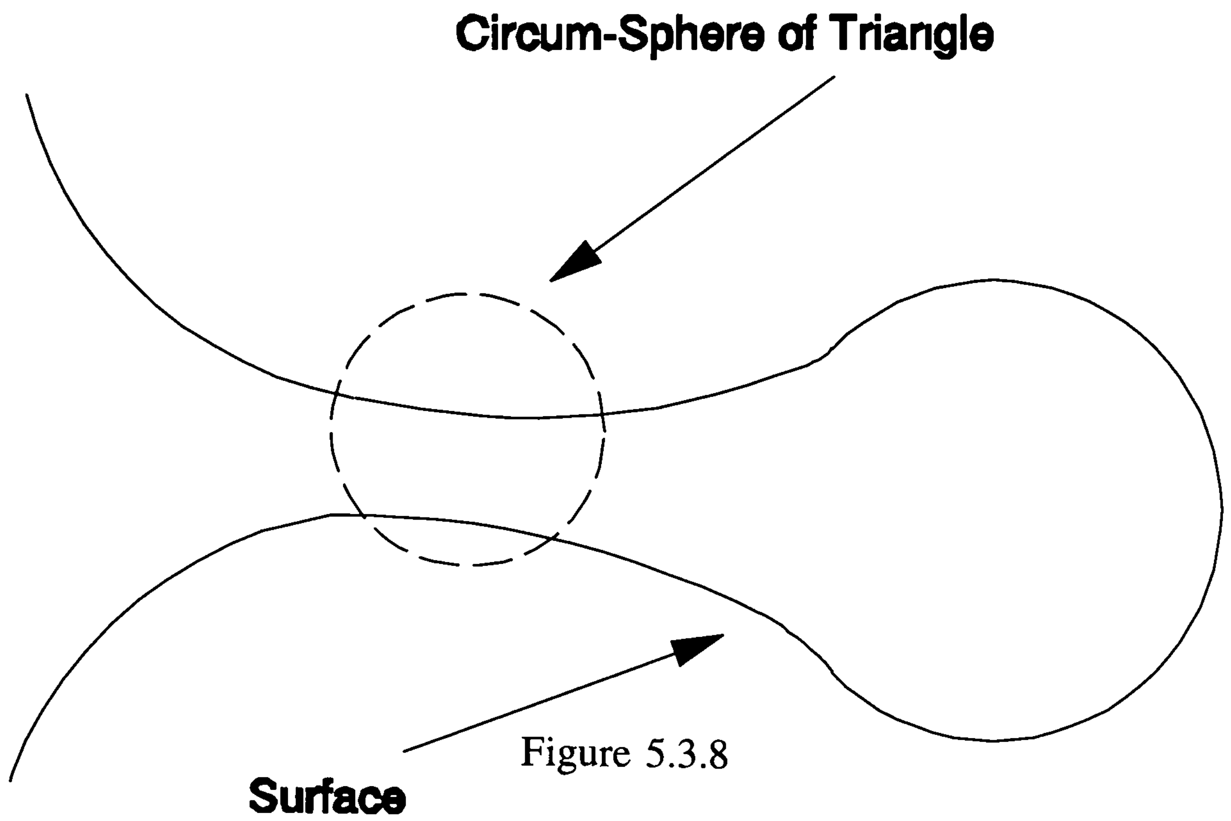


Figure 5.3.8

The Figure 5.3.8 shows a minimal circum-sphere of a triangle intersecting the surface more than once. This situation presents no problem to the technique.

Method 2 :

i) Use 2D planar Delaunay triangulation in parameter space, to give an initial approximation to Surface Delaunay Triangulation.

ii) Once the initial approximate mesh is formed, apply Lawson's swapping algorithm to generate the Surface Delaunay Triangulation.

Of the two methods described above, it was found that the second technique was more robust. This method avoids the need to establish which edges of triangles are visible from a given point, which was found to be the part of the code most influenced by rounding errors in Method 1. Figures 5.3.9Aand 3.3.9B shows two examples of meshes generated over a parametric cubic patch. The first example shows 2D Delaunay triangulation of the surface meshed only in parameter space, the second is meshed by the second method described above and forms a valid surface Delaunay triangulation. The major difference between the two figures is most noticeable around the leading lower edge of the two surfaces.

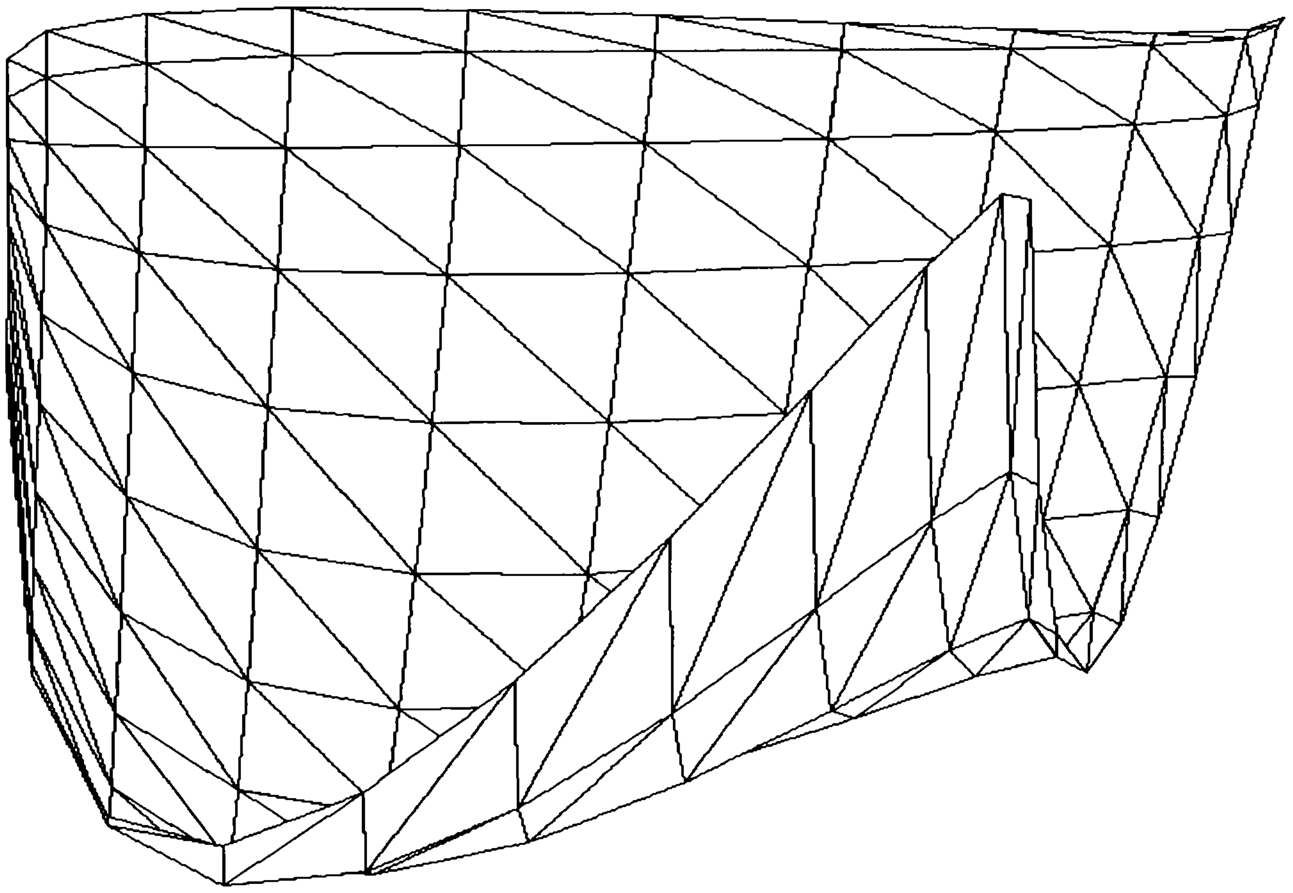


Figure 5.3.9A, Parameter 2D Delaunay Triangulation

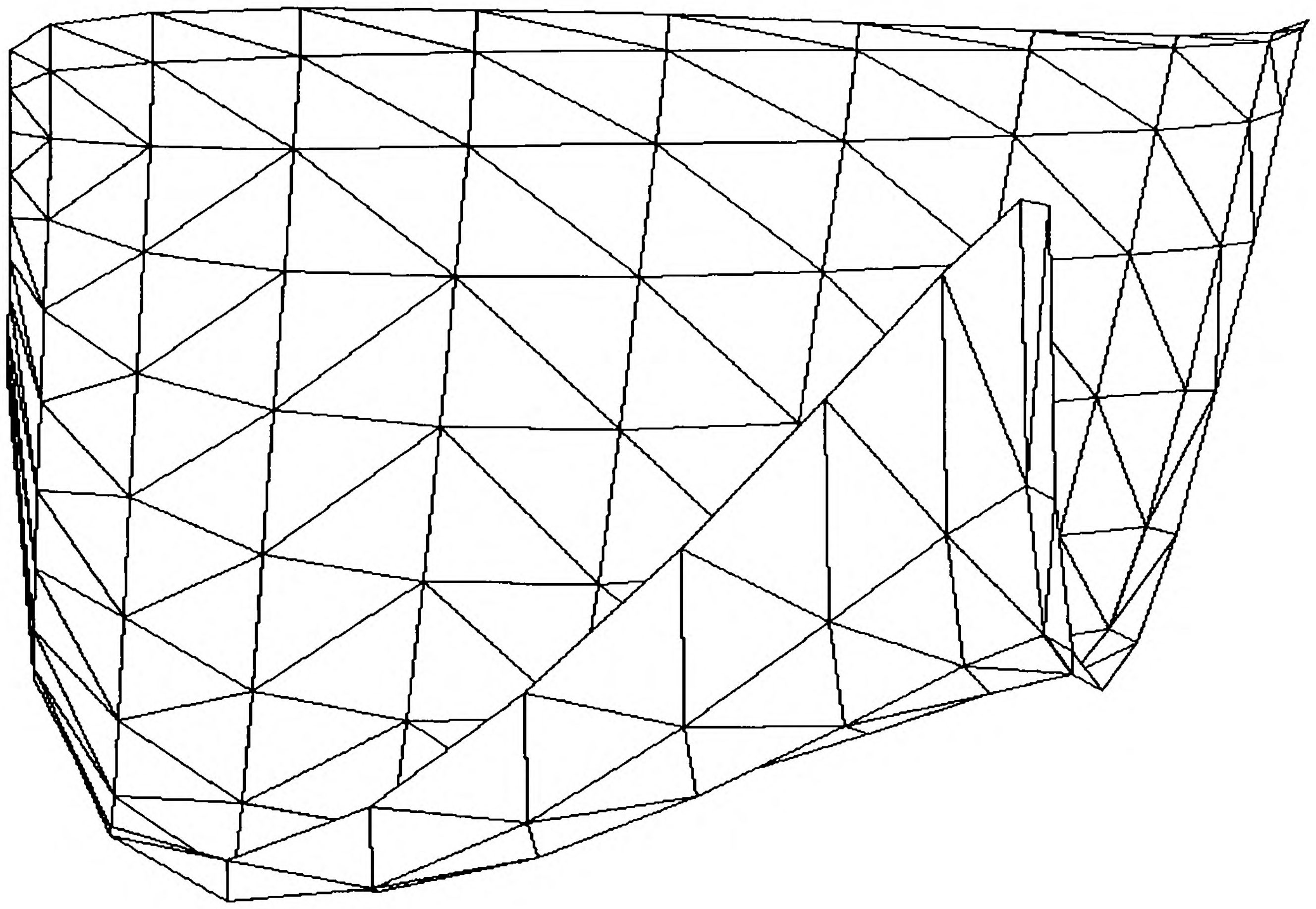


Figure 5.3.9B, Surface Delaunay Triangulation

5.3.1 Surface Delaunay results.

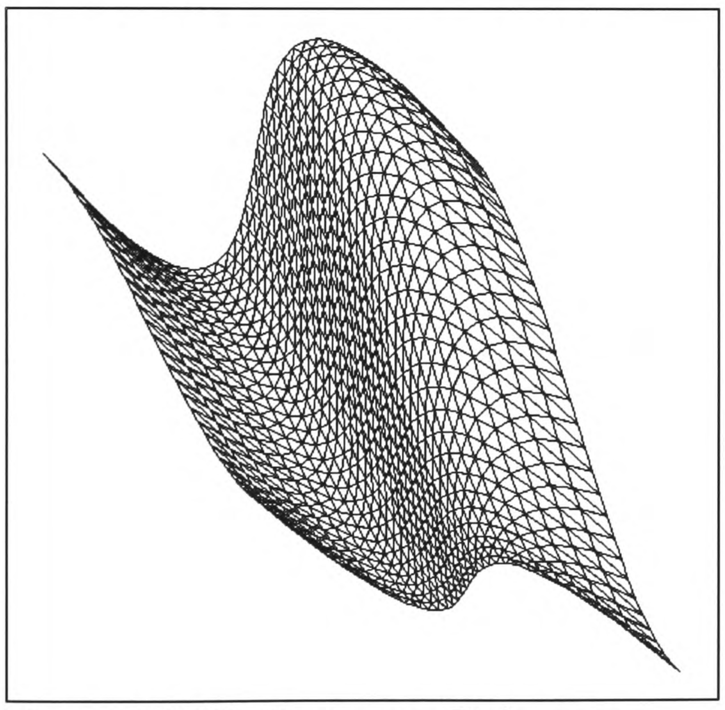


Figure 5.3.10: Parameter Delaunay triangulation.

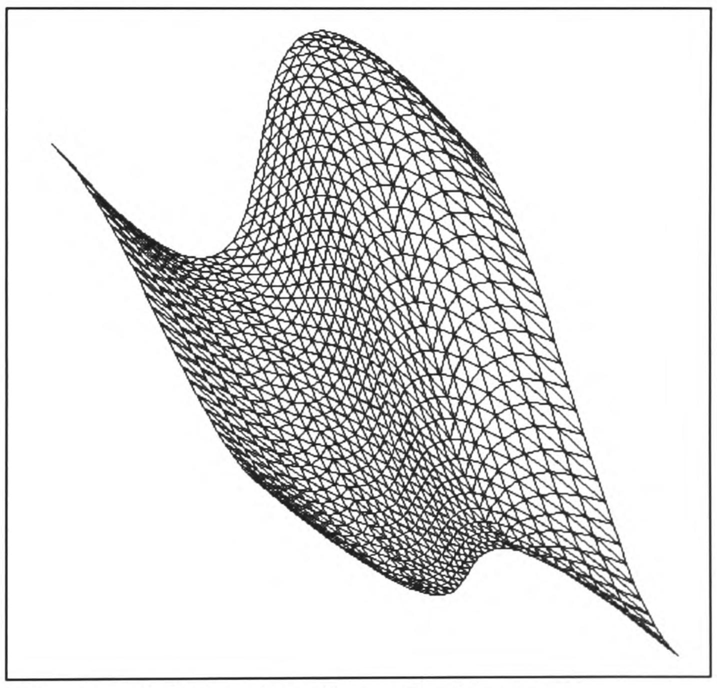


Figure 5.3.11: Surface Delaunay triangulation.

The above two surfaces were used in the experimental results generated in table 5.3.1.

The following table 5.3.1, lists a comparison between standard parametric Delaunay and surface Delaunay, giving the maximum and minimum angle of each mesh with average surface error at the mid node of each triangular element side.

Table 5.3.1

	Parametric Delaunay			Surface Delaunay		
Tri	Max ang	Min ang	Sur Err	Max Ang	Min ang	Sur Err
8	1.5708	0.67474	0.19267	1.5708	0.67474	0.19267
32	2.49978	0.25173	0.04913	1.8677	0.54352	0.04849
50	2.62658	0.21078	0.03085	1.89385	0.47648	0.03066
98	2.64741	0.20684	0.01597	1.93021	0.41602	0.01586
162	2.65236	0.20632	0.00972	1.95157	0.36733	0.00964
200	2.64807	0.20453	0.00793	1.96856	0.35228	0.00786
242	2.65338	0.20661	0.00653	1.96375	0.34101	0.00647
288	2.6543	0.20399	0.00551	1.96527	0.33231	0.00547
392	2.65678	0.20413	0.00405	1.96439	0.31984	0.00402
450	2.6543	0.20761	0.00352	1.97563	0.31524	0.00349
512	2.65762	0.20452	0.0031	1.9799	0.31139	0.00308
578	2.65608	0.2074	0.00274	1.99139	0.30812	0.00272
648	2.65774	0.20499	0.00245	1.99912	0.30533	0.00243
722	2.65673	0.20654	0.0022	2.00387	0.3029	0.00218
800	2.65947	0.20547	0.00199	2.00775	0.30078	0.00197
882	2.6568	0.20612	0.0018	2.01541	0.29892	0.00178
1058	2.65656	0.20596	0.0015	2.02436	0.29578	0.00149
1250	2.65739	0.20596	0.00127	2.03116	0.29326	0.00126
1458	2.65842	0.20604	0.00109	2.03802	0.29118	0.00108
1682	2.659000	0.20619	0.00094	2.0423	0.28944	0.00094

Figure 5.3.12, plots the minimal and maximum angles of each of the two types of triangulations, here it can be clearly seen that the surface Delaunay gives for all cases an improved triangulation.

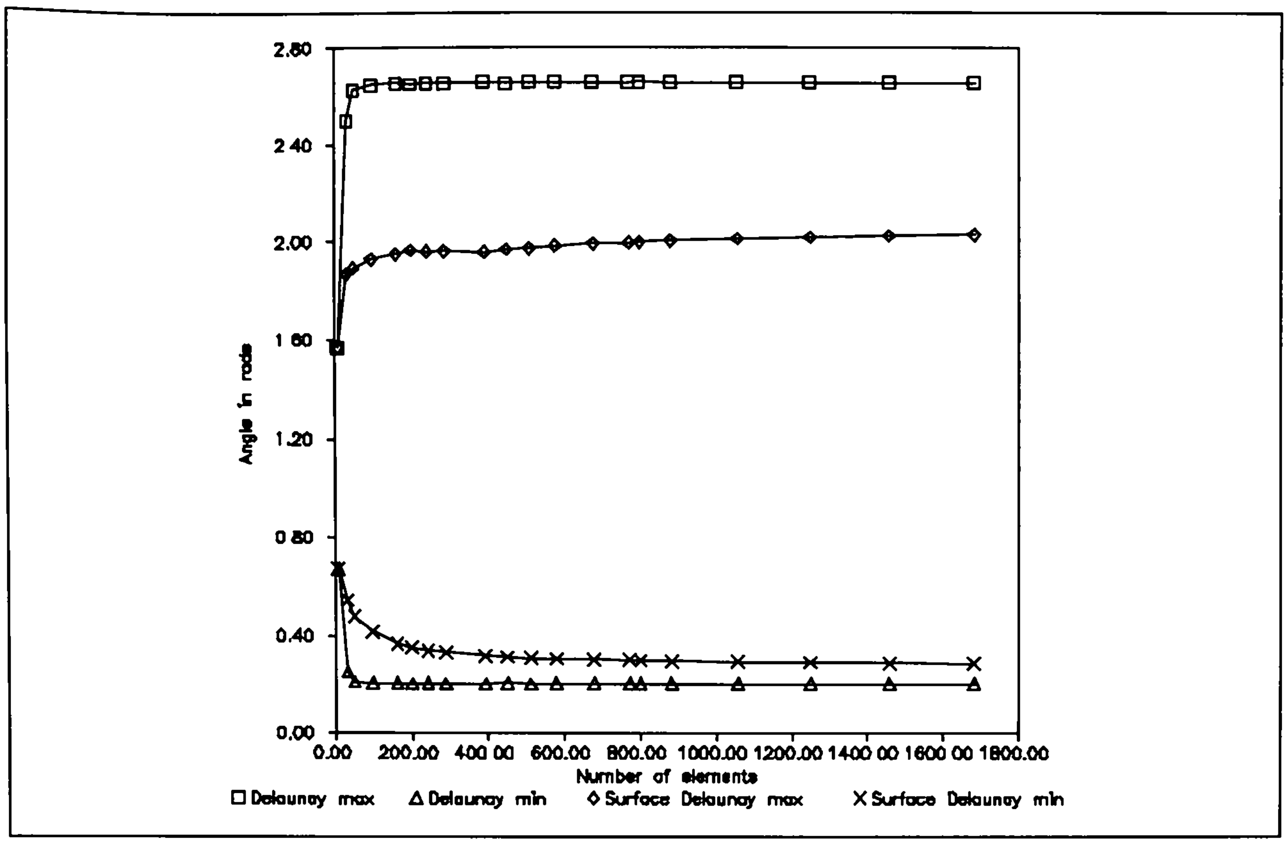


Figure 5.3.12: Comparisons between angles in grids.

The Figure below, plots the difference between Parametric Delaunay error against surface Delaunay triangulation error, a negative value indicates Surface Delaunay returns a better approximation. Surface error is the average of the distance of mid point of each element's edge from the parametric surface.

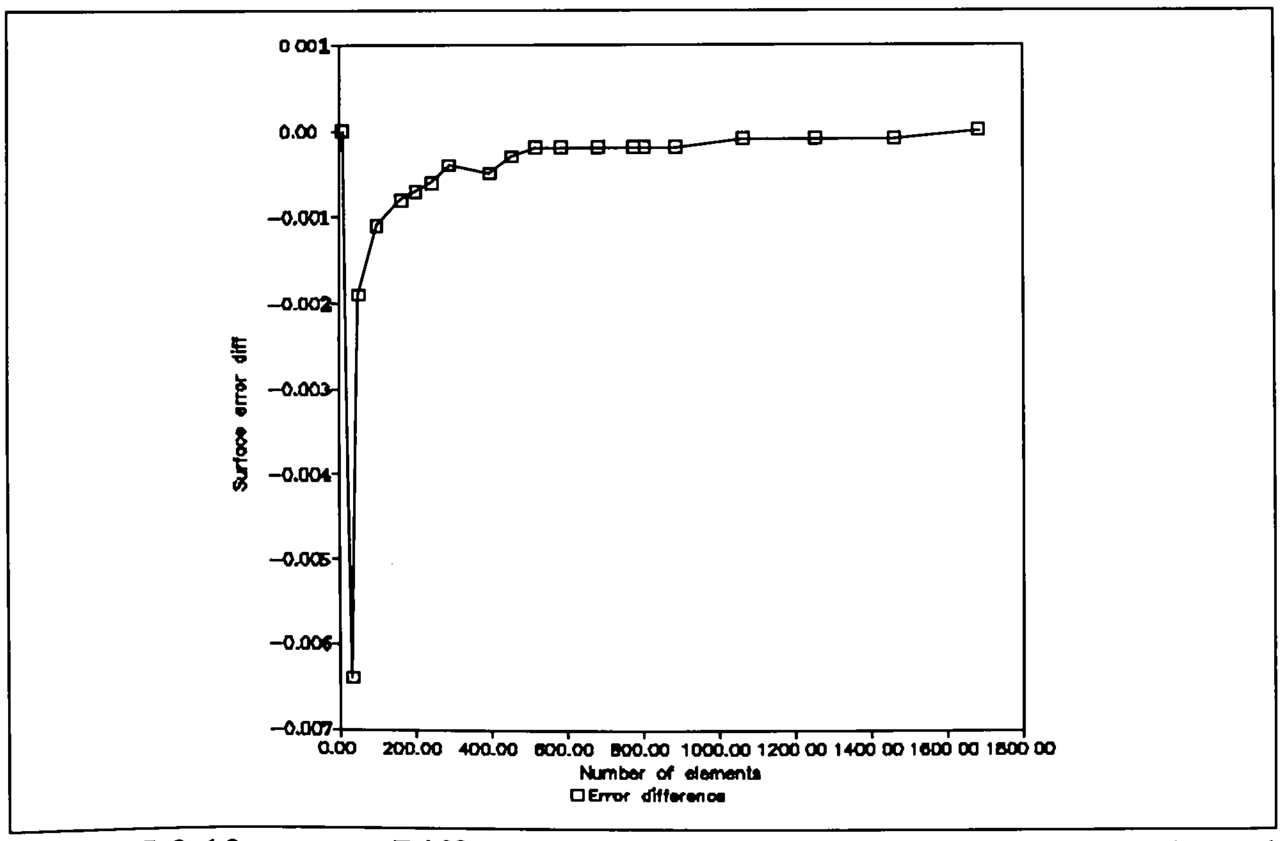


Figure 5.3.13 : Difference in surface error between parametric and surface Delaunay.

5.3.1 Surface Delaunay triangulation example.

The table below shows CPU time in seconds to generate a surface Delaunay triangulation of 'n' elements over a parametric surface shown in Figure 5.3.14.

Table 5.3.2

No. Of triangles	CPU time seconds
8	0.067
32	0.267
50	0.450
98	0.600
162	0.850
200	0.867
242	1.267
288	1.483
392	2.050
512	2.650
648	3.397
722	3.95
882	4.7
1058	5.55
1250	8.733
1458	6.767
1682	7.950

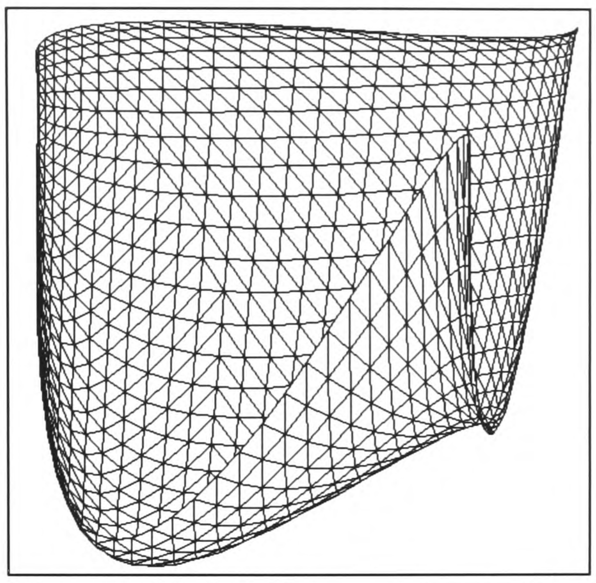


Figure 5.3.14: Parametric surface with 1682 surface triangles.

The above CPU times were calculated on a Sun ELC; the code was compiled without optimization.

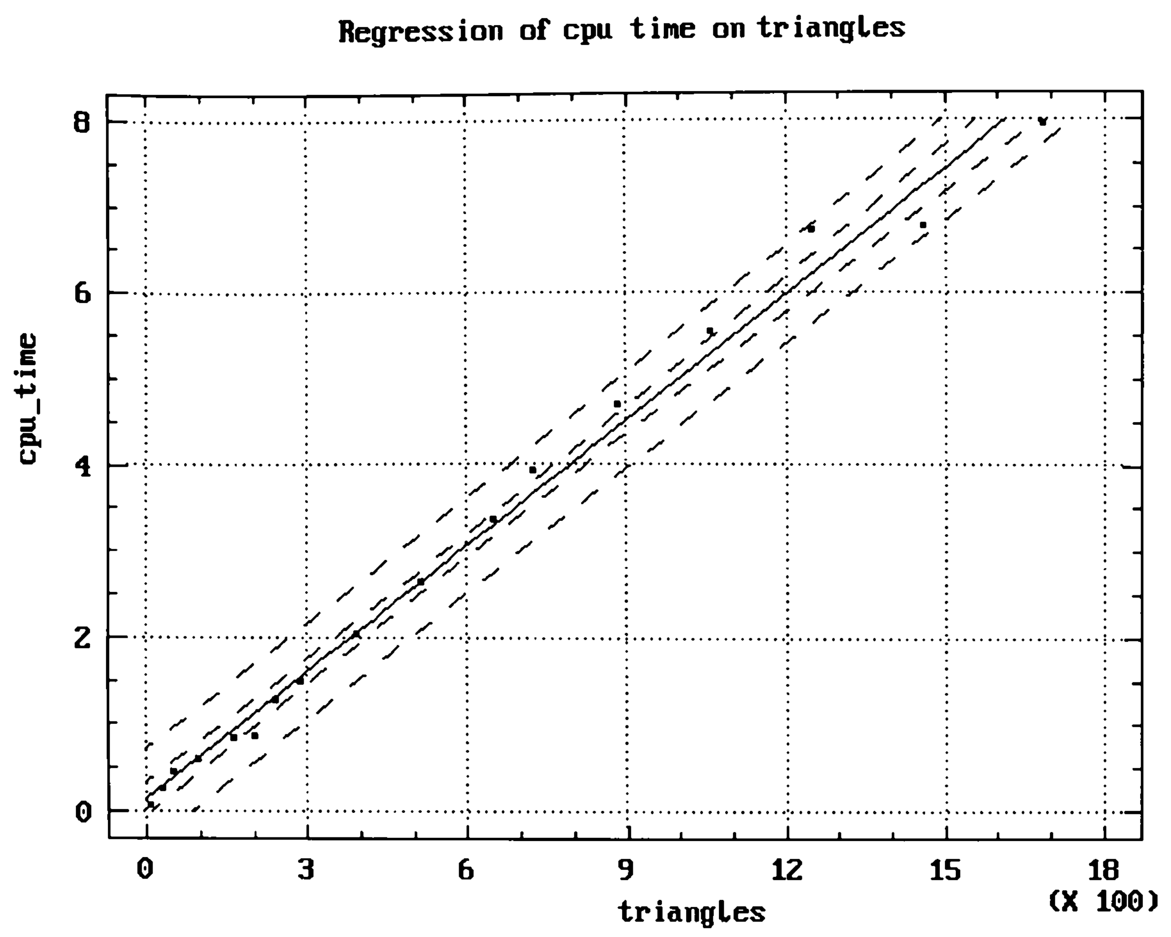


Figure 5.3.15 : CPU time correlation.

The above graph shows a linear regression analysis of the data given in Table 5.3.2. The regression analysis of the fitted line gave a value of R^2 of 0.99, which indicates that a linear representation of the data is an appropriate model. Therefore, we can conclude that for the above parametric model in Figure 5.3.15 we have an order N algorithm.

5.4 Surface Laplace Smoothing

Laplace Smoothing [KaE70] is a simple but effective method used in 2D and 3D mesh generation to improve the general shape of elements. In Laplace smoothing each point is taken in turn and moved to a new location, which is the average of the nodal positions of all the vertices that are directly connected, see Chapter 4 section 4.4.3.

The problem with Laplace smoothing is it cannot be directly applied to surfaces, therefore, there is a need to derive a method which weights points in parametric space. This should enable it to become possible to apply a smoothing algorithm in the parameter space that improves the shape quality of the elements on the 3D surface.

To achieve Laplace smoothing to a point in parametric space in such a way as to optimise the element in 3D space the following information is needed:

L - Length of line connecting the points *i* and *j* on the surface
(for simplicity this will be the absolute distance between the two points in E^3)

PL - Length of the line connecting the points i and j in parametric space.

SR - Sum of the ratios of L/PL of all points connected to point i in the mesh.

n - Number of vertices connected to the point i

Hence the new position for point 'i' becomes:

$$P_i = \frac{1}{SR} \sum_{j=1}^{n} P_j (L_j / PL_j)$$

A simple check for this is to see if it is consistent when we let PLj tend to Lj i.e no difference between the parametric definition and the true surface. The above equation tends to :-

$$P_i = \sum_{j=1}^{n} Pj/n$$

which is the definition of Laplace smoothing, see Chapter 4 section 4.4.2.

The above methods, in general need a few extra iterations to generate reasonable results than the standard Laplace smoothing. However, the standard method cannot be applied to these surfaces. This method tends to improve the shape of the elements with all the nodes remaining on the surface. However, it cannot guarantee to improve the surface approximation, which is approximated by the planar elements.

5.4.1 Surface Laplace Smoothing Results

Below two identical surfaces are depicted; Figure 5.4.1 and 5.4.2 show the initial surface and Figures 5.4.3 and 5.4.4 are the resultant mesh after smoothing was applied.

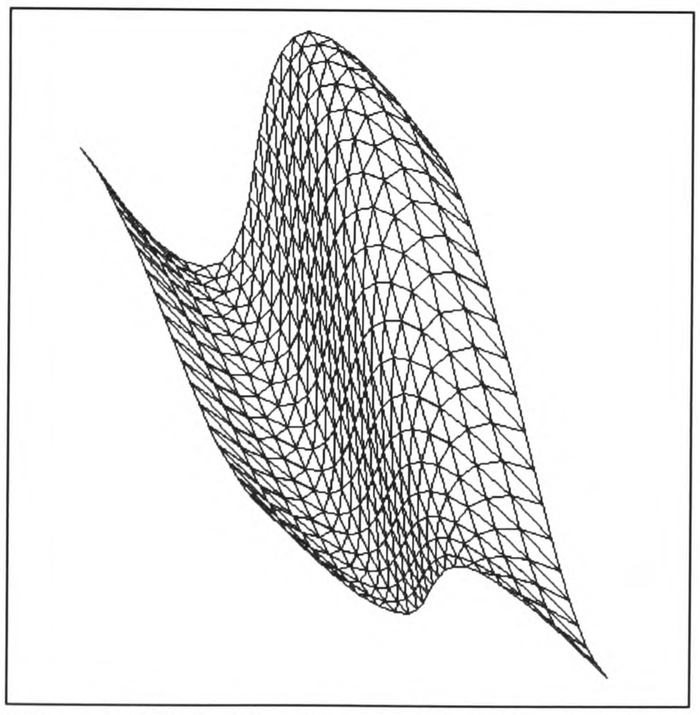


Figure 5.4.1: Surface before Smoothing.

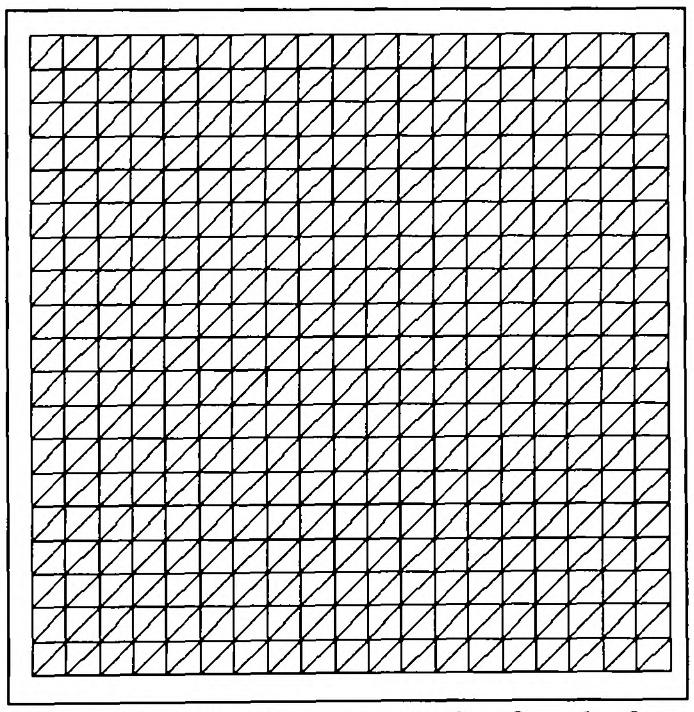


Figure 5.4.2: Parametric Surface before Smoothing.

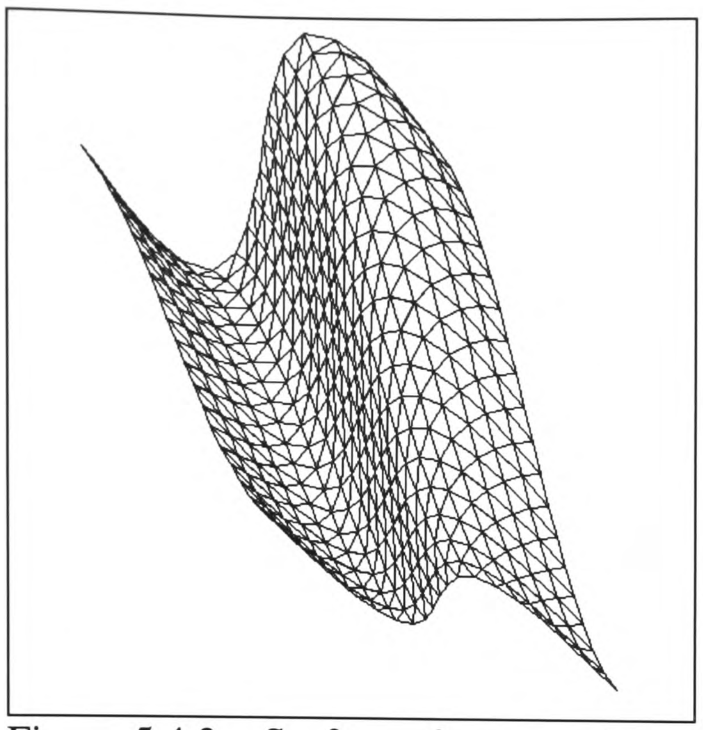


Figure 5.4.3: Surface after smoothing.

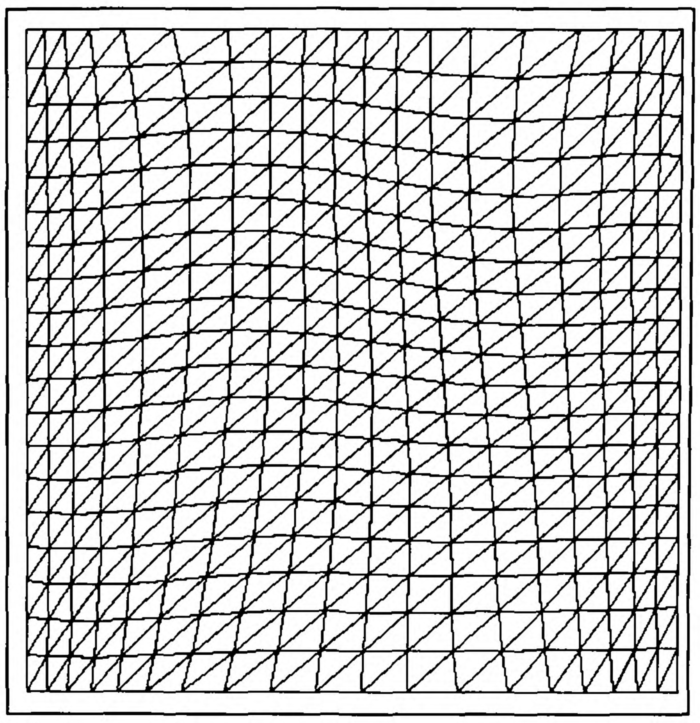


Figure 5.4.4: Parametric surface after smoothing.

Figures 5.4.2 and 5.4.4 depict the surface in parameter space, where u and v are two parameters which sweep out the surface and take values in the range 0 to 1.

The graph below shows a comparison between surface minimal and maximum angles of elements before and after smoothing. The smoothed surface angles were taken for 5 passes of the Laplace smoother. It can be clearly seen that the Laplace angle smoother improves the quality of the meshed surface. However, the improvement is not that distinct, because of the limitations of the geometry e.g the minimal and/or maximal angle can be restricted by the geometry.

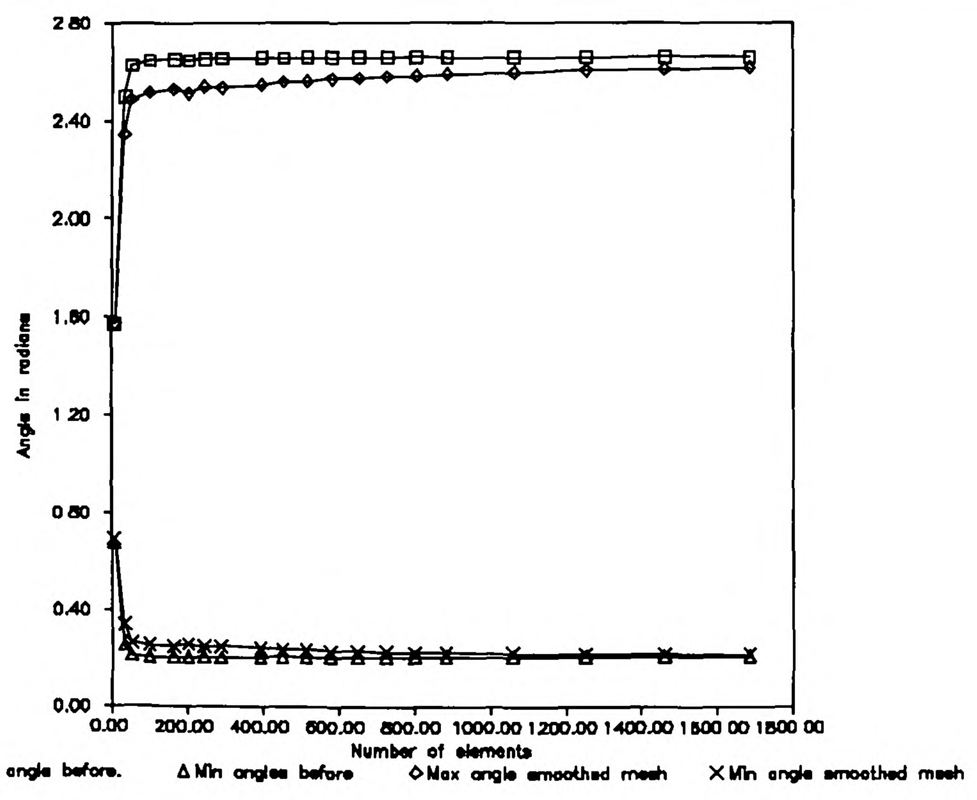


Figure 5.4.5 : Plot of minimum and maximum angles of mesh before and after smoothing for various number of elements.

The graph below shows how the angles of the mesh improve, for each pass of the Laplace smoothing routine.

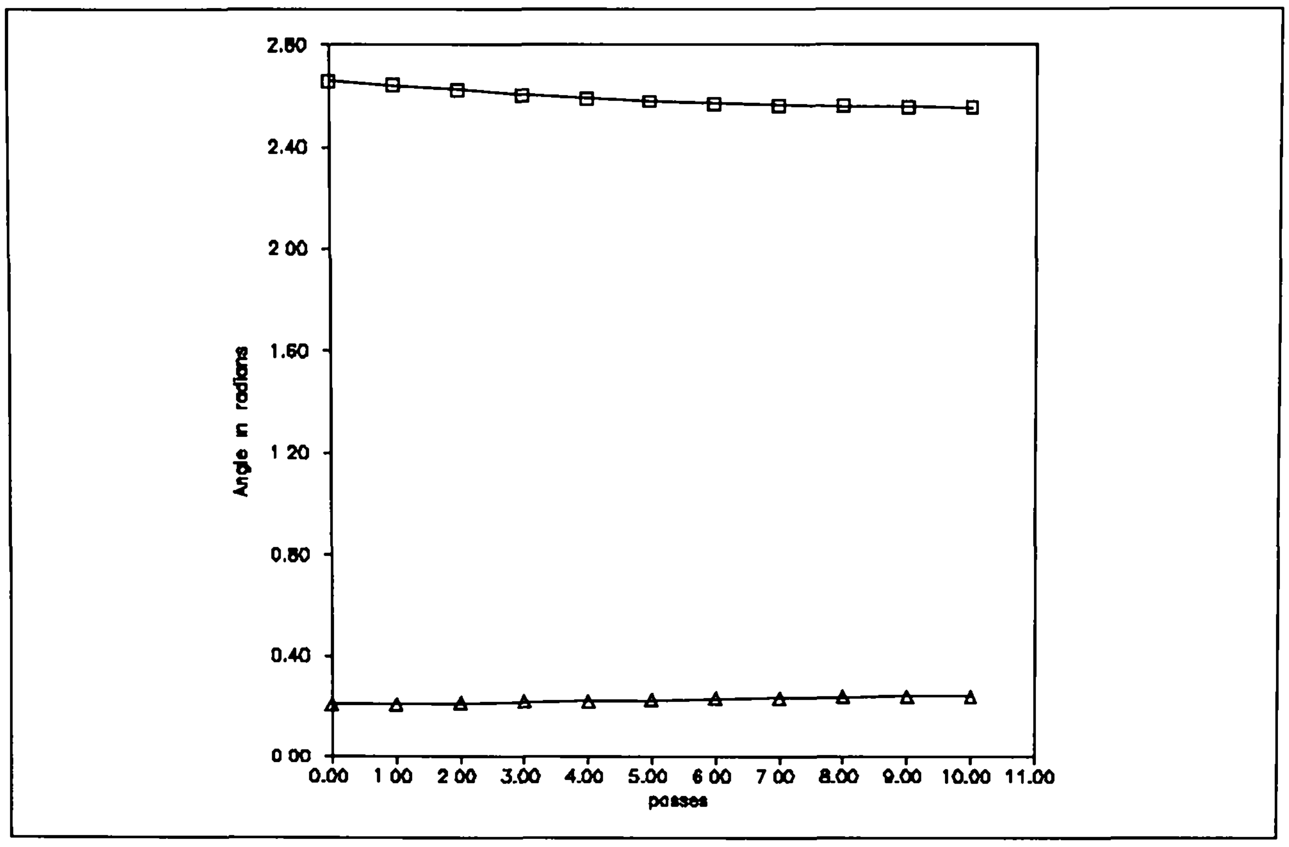


Figure 5.4.6 : Angle quality after each pass of the smoothing algorithm.

The order of time complexity for Laplace smoothing is almost order n, which is depicted in the next graph that plots the number of elements against CPU time for 5 passes of the algorithm.

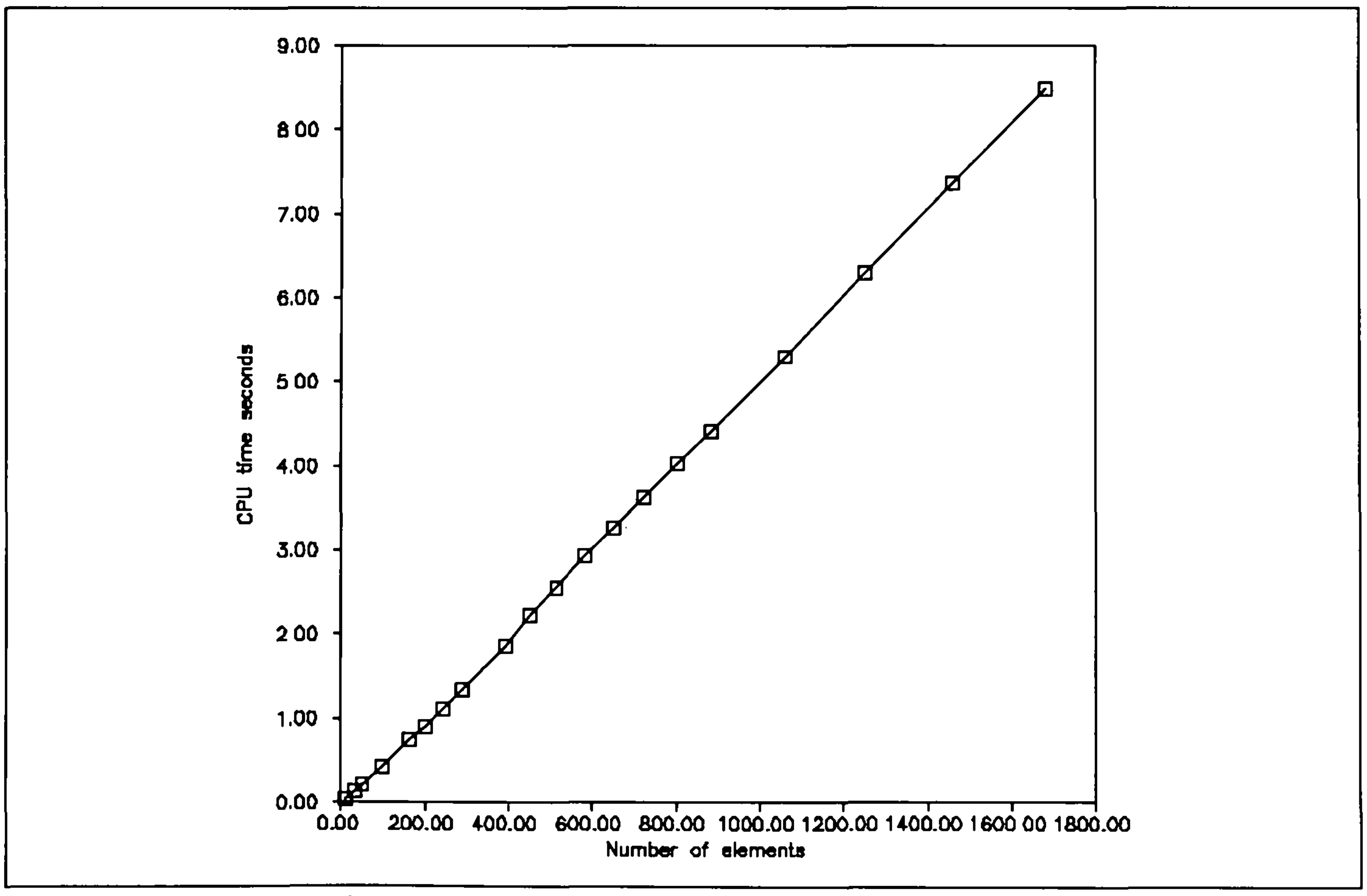


Figure 5.4.7 : CPU time to smooth *n* elements.

5.5 Vertex Swapping on parametric surfaces

A surface mesh only forms an approximation to the true surface. When a vertex swapping algorithm [Law77] is applied, it does not just effect the element's shape, but also the error between the true surface and the surface formed by the mesh.

Therefore a vertex swapping algorithm has two objectives:-

(i) To minimize the maximum angle of the mesh

(ii) To minimize the surface mesh error.

It is almost impossible to satisfy both these goals, but we can assign a weight to how important each one is for our purposes. For example, if $\xi 1$ and $\xi 2$ are the errors for the surface approximation for the two possible triangulations of a given convex quadrilateral. The values min1 and min2 are the minimum angles of the two possible triangulations and W_1,W_2 the weighting functions, the following relations can be defined:

(1) $f_1 = \min1 + W_1 * \xi 2/\xi 1$

(2) $f_2 = \min2 + W_2 * \xi 1/\xi 2$

For the special case when $\xi 1 \approx 0$, for relation 1, the second term is set to W_1, since this occurs when $\xi 2 \approx 0$, hence it is assumed zero divided by zero is 1. Similarly for equation 2 when $\xi 2 \approx 0$, the second term is set to W_2.

From relation 1 it can be seen that the second term $\xi 2/\xi 1$ increases as the error $\xi 1$ of the initial triangulation decreases; consequently relation 1 decreases as $\xi 2$ increases. This also holds for equation 2, but vice-versa. Hence the vertices of the triangles are swapped if $f_2 > f_1$.

The main problem with the above relations how to choose the weighting functions. W_1 and W_2 could both be constants, so if there is a large difference between the two errors $\xi 1$ and $\xi 2$, we choose the one with the smallest error, but if $\xi 1 \approx \xi 2$ the size of the angles has the greater effect on the choice, therefore we could then use

$W_1=W_2=1$. Another choice is to let W_1=Min2 and W_2=Min1, which relates the error to the angles of the alternate triangulation.

It is also possible to use different functions like :-

(3) f_1= min1*ξ2/ξ1

(4) f_2= min2*ξ1/ξ2

This has the advantage that as ξ1,ξ2 tend to zero, the method tends directly to the planar swapping algorithm. However, it is difficult to adjust the relationship between angle and surface error.

5.5.1 How To Calculate Surface Error (Local Surface Error).

To calculate the true surface error can be complex and CPU intensive. However, it was found that the true surface error is not really necessary, as a rough estimate is often sufficient, and the reasons for this are discussed below.

One method is to take two adjacent triangles and estimate the error over these elements by calculating the distance of their centroids from the surface. However, a quicker method is to calculate the distance of the centroid of the adjacent edge from the surface, since the algorithm is only based on swapping elements faces. It is then possible to apply the vertex swapping algorithm by calculating the minimum angles of the two sets of triangles plus the two distances of the centroids of the two possible adjacent edges.

5.5.2 What is the effect of the above method?

There is a complete contradiction between an element's optimal shape for meshing and for surface approximation. For example, when the above method is used on surfaces that have a constant gradient in one direction and variable in the other, this tends to produce elements that are compressed in the variable direction, i.e elongated in the direction of constant gradient. Since minimize the length of an element in the direction of the largest gradient of the surface, reduces the surface error.

Therefore, a better method is to swap elements' vertices based solely on their angles. We only then test for surface error when the two sets of angles are similar.

5.5.3 Surface Vertex swapping Results

Below two identical surfaces are depicted; Figure 5.5.1 and 5.5.2 show the initial surface and Figures 5.5.3 and 5.5.4 are the resultant mesh after applying the surface vertex swapping method.

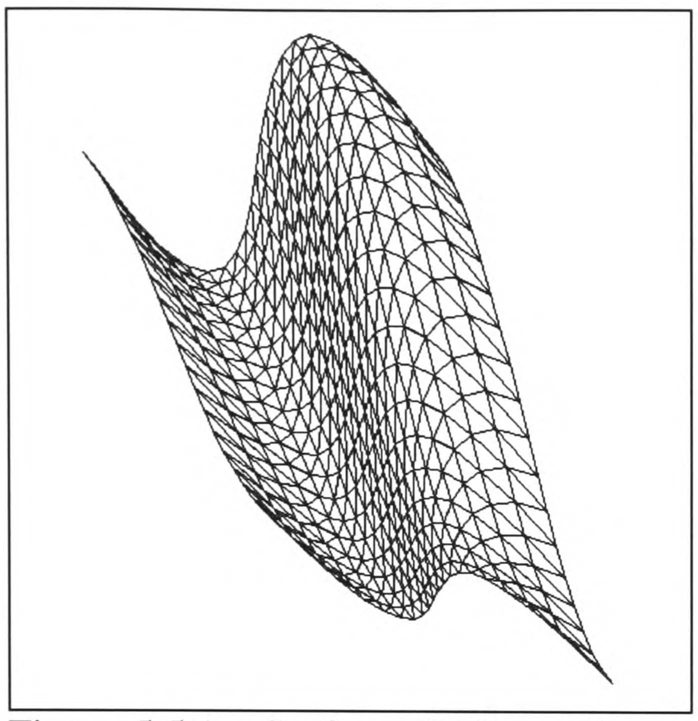


Figure 5.5.1 : Surface before Vertex swapping was applied.

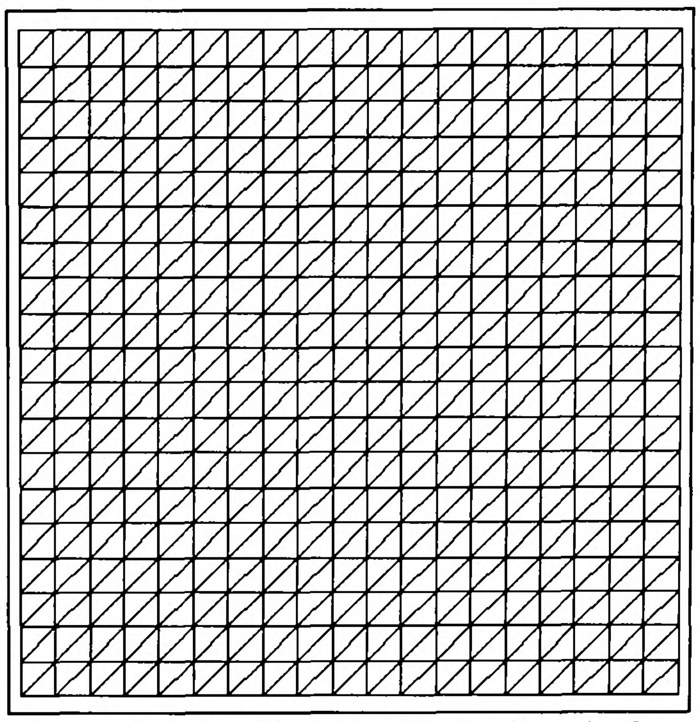


Figure 5.5.2 : Parametric Surface before Vertex swapping is applied.

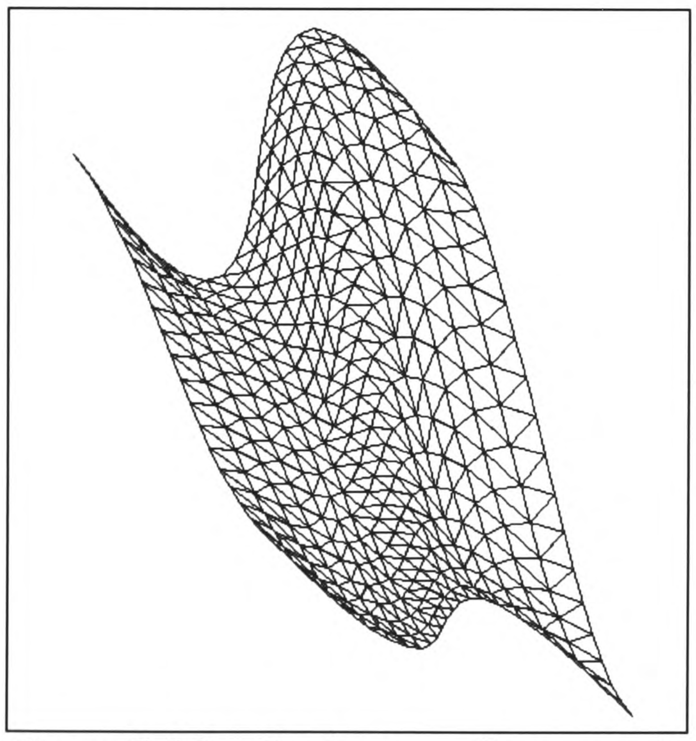


Figure 5.5.3 : Surface after angle optimization using vertex swapping.

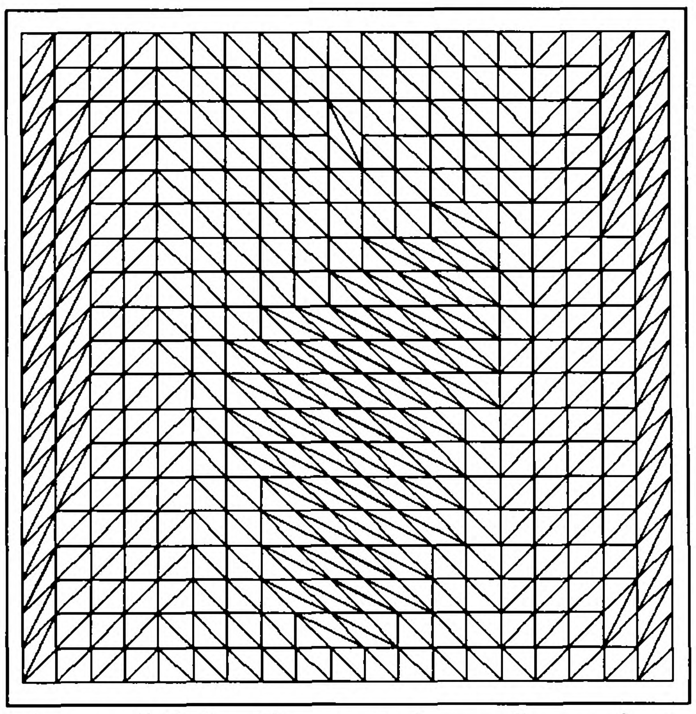


Figure 5.5.4 : Parametric surface after optimization.

Figures 5.5.2 and 5.5.4 depicts the surface in parameter space, where u and v are the two parameters which sweep out the surface, and are in the range 0 to 1 to form a unit square.

The graph below shows a comparison between surface minimal and maximum angles of elements for the surface before and after angle optimization. It can clearly be seen that vertex swapping improves the quality of the meshed surface.

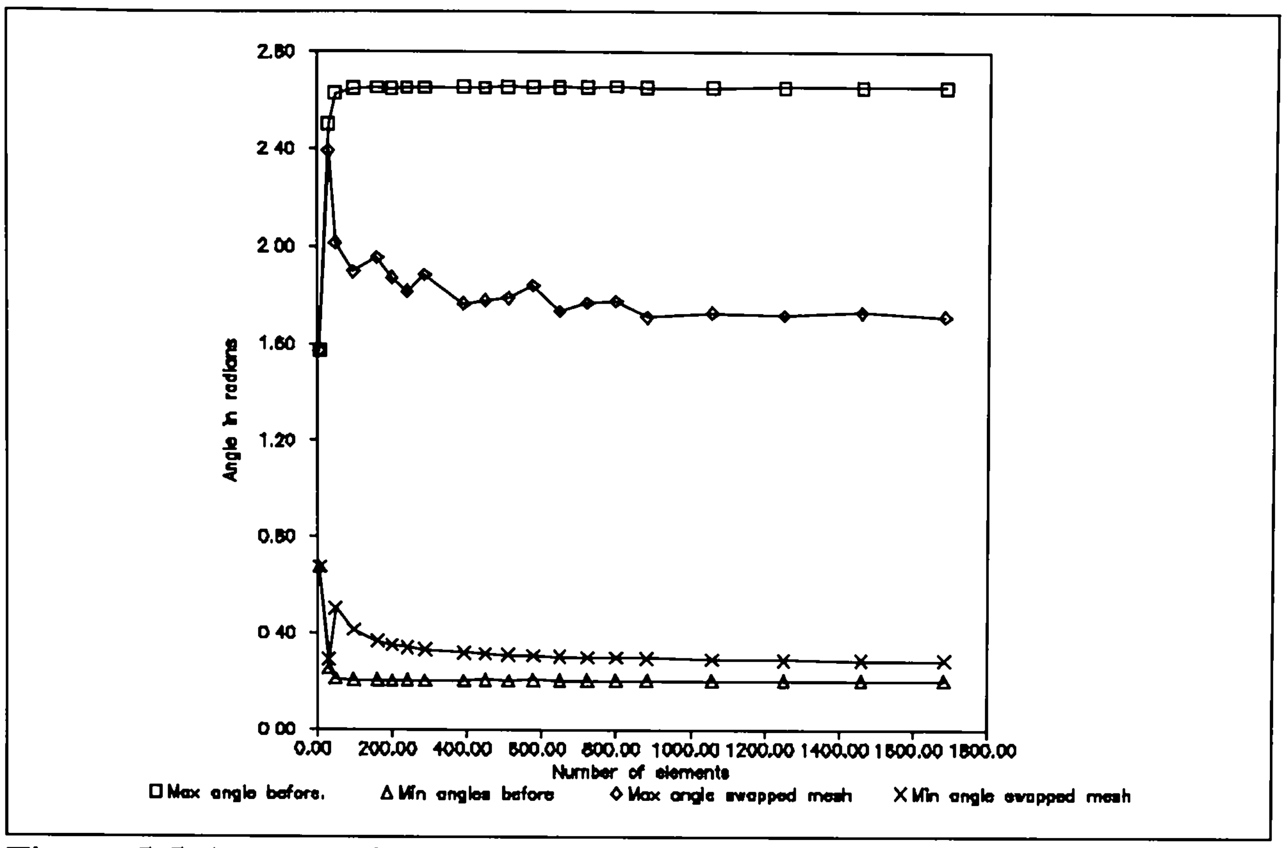


Figure 5.5.5 : Plot of minimum and maximum angles of mesh before and after optimization for various number of elements.

The order of time complexity for vertex angle optimization is almost order n, which is depicted in the next graph, which plots number of elements against cpu time.

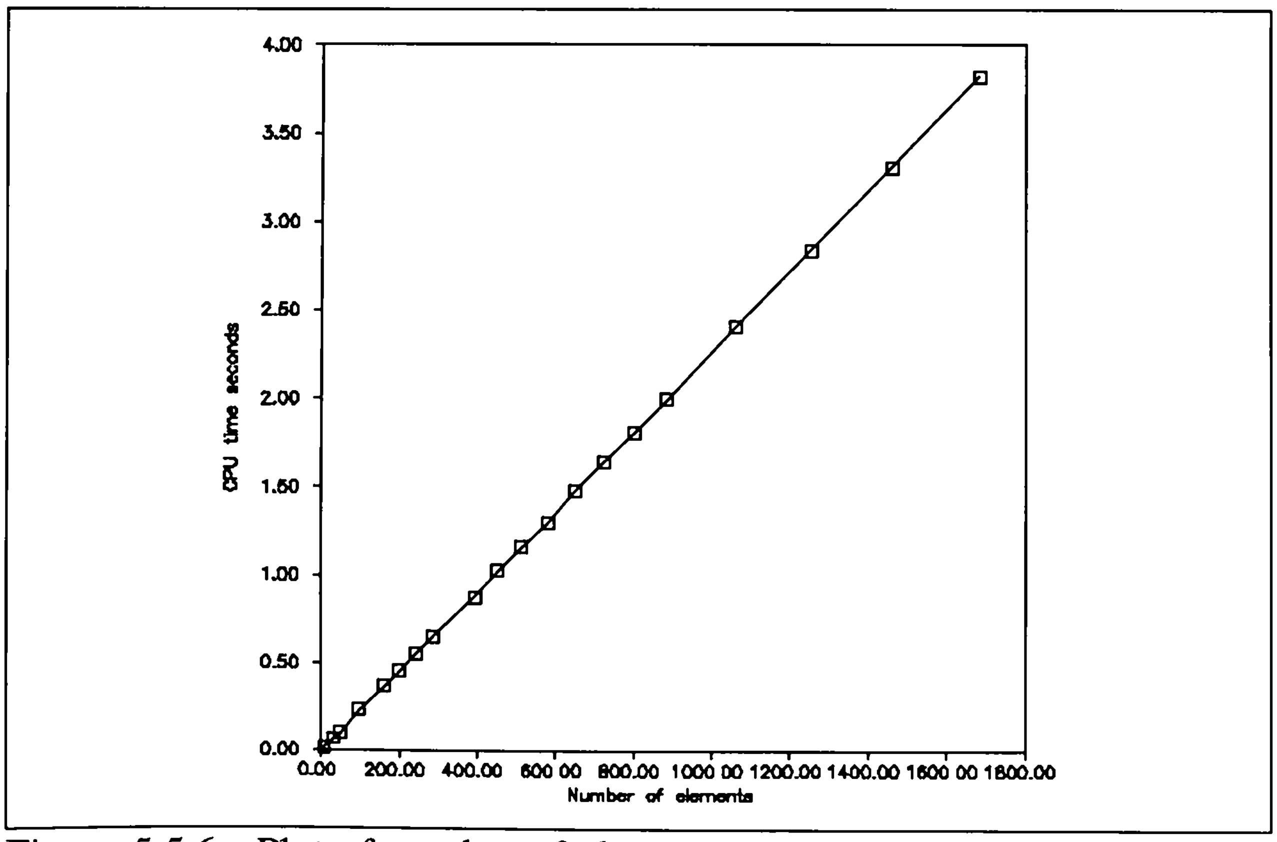


Figure 5.5.6 : Plot of number of elements against CPU time in seconds.

5.6 Summary and Conclusions.

This chapter has demonstrated that two dimensional mesh generating techniques can be extended successfully to surfaces without effecting their computational order. They can be applied in such a way that they generate grids of *good quality*, whilst effectively taking account of surface approximation error.

Part III
Three dimensional mesh generation.

This is the last section of this dissertation, which includes an explanation of a new three dimensional mesh generator with a description of the major algorithms that the technique is contingent upon. This section then concludes with a summary of major achievements and possible further extensions.

Chapter 6

3D Mesh Generation.

6.1 Introduction.

This chapter gives an overview of the research that has been carried out into three dimensional grid generation. An outline of the initial attempt at the recursive bisection mesh generation is given, together with geometry representation and bisection techniques.

Dealing with some key features of the geometry, such as cavity removal, is best carried out before the main task of grid generation. Therefore, the initial bisection planes are chosen to simplify the model by introducing interface elements that join the cavity regions to outer boundaries. The selection of bisection planes for mesh generation is then described, together with how this is implemented to generate rudimentary meshes. The problems associated with this basic technique are then covered, together with methods of improvement.

The simplification of a geometry into simpler parts, such as convex regions, and how this can aid nodal placement is described. Once the domain is sub-divided into simpler regions the nodal placement algorithm is then implemented, followed by the final stage of meshing. The final meshing of the domain was found more reliable and faster using a non-recursive method based on a boundary constrained local min-max algorithm. The min-max heuristic method was found to be best implemented using local tetrahedral transformations and operated in an average computational order of *nlogn*.

6.2 The initial recursive bisection mesh generator.

Below are the steps used in the initial three dimensional recursive domain bisection mesh generator. This was the starting point from which the final three dimensional grid generator was developed.

(1) Remove all cavities from polyhedral domains using the technique in section 6.5.

(2) Generate a surface triangulation over the polyhedral domains.

(3) Place all initial polyhedral domains in stack S,

(4) Remove domain D from stack S,

(5) Generate an order list L of face elements' angles,

(6) Choose an edge E from list L with largest edge angle, since these offer the greatest possibility for domain complexity reduction.

(7) Generate characteristic plane ρ, a plane which bisects the *inner wedge* defined on this edge.

(8) Attempt to find a bisection plane through this edge that divides the region into two, using the techniques in sections 7.4.3 and 7.3.3.

(9) If a valid bisection plane is not found, choose the next edge from the list L which does not have the same characteristic plane as previous bisection attempts on this domain, go to step 8.

(10) If more than ten attempts are made at bisecting this domain, apply binary mesh operators, section 3.3.

(11) Once the domain is bisected into two regions Φ_1 and Φ_2, they are added to the stack S, if they have more than four triangular elements. If a region has only four triangular faces they form a tetrahedral element of the mesh.

(12) Repeat steps 4 to 11 until the stack S is empty.

The internal nodal placement is achieved by generating a refined two dimensional triangular grid over each bisection interface polygon, using the length of the boundary edges of the cut interface as a guide to mesh density. It should also be noted that any new interfaces, generated by *binary mesh operations*, section 3.3, are meshed using the 2D grid generator in Chapter 4. An example of a mesh generated in this fashion is illustrated in Figure 1.2.1 Chapter 1.

6.3 Geometry representation.

The following section will give a brief overview of the basic methods and terminology used to describe three dimensional objects. The geometry representation of the model used, has a great effect on the types of geometry operations, which can be applied during the grid generation. Therefore, the geometry representation of the model effects not just the computational order of the grid generator, but also its reliability and robustness to computer arithmetic errors. This section is then concluded, with reasons, why a boundary model representation was selected as the primary input format for the mesh generator.

6.3.1 Surface based models.

It is easy to generate simple graphical models of real objects, but unfortunately, points and lines alone do not in general convey sufficient information for the application of complex algebra operations.Confusion can be avoided if we adopt a more rigorous view of modelling:

1) Physical objects :By means of models, our aim is to speak and argue about some real things of our three-dimensional real world. Unfortunately, assuming a Platonician view, we cannot even perceive a real-world object in its full complexity and sub-microscopic details, much less represent all aspects of it in a computer.

2) Mathematical objects : In order to have any hope of modelling objects in a computer, we must therefore adopt a suitable idealization of the real three-dimensional physical objects we are ultimately interested in. These idealized objects should have an intuitively clear connection with the real world, while being so simple that we can assign a computerized representation to them.

Therefore a mathematical geometry representation should possess the following qualities (Requicha's analysis [Req83]).

(a) Expressive power : What objects / forms of objects covered.

(b) Validity : Are all admissible representations valid.

(c) Unambiguity and uniqueness : Do some models have more than one representation?

(d) Description Languages / Descriptive power

(e) Conciseness : How large are representation of practical models

(f) Closure of operations : Do manipulations of objects generate valid models

(g) Computational ease, applicability and Reliability : Speed of operations and complexity of coding (order of operations)

The main types of modelling methods that possess the above characteristics are:-

(a) Decomposition models, that represent complex objects as a collection of simple objects from a fixed collection of primitive object types, combined with a simple 'gluing' operation.

(b) Constructive models, as above but objects can be combined with a number of complex operations such as union, intersection, difference etc.CSG (Chapter 2 section 2.1.1) are often built up of Binary trees of primitive objects [Dew88].

(c) Boundary models, which represent objects in terms of boundary data. The boundary of complex structures are represented as a collection of faces, which in turn, are often represented in terms of their boundary being a one-dimensional curve. Therefore, Boundary models may be viewed as a hierarchy of models.

The Expressive power of boundary models combined with the well defined family of geometry operators, Euler operations [Mar88], make this method the natural choice for three dimensional grid generation. Boundary models have all the required operations needed for grid generation by mesh bisection and are the natural way to represent surface meshes.

6.3.2 Polyhedral domains.

The particular subset of boundary representation used by the mesh generator is polyhedral models. These models are constructed from two-dimensional non-intersecting primitives, polygonal faces. Each face is in turn constructed from a set of co-planar points with connecting lines and a surface normal. Each polyhedral face can have any number of nodes, edges and holes, see Figure 6.3.1.

Figure 6.3.2 shows the hierarchical structure, which can be used to describe polyhedral models. The illustration demonstrates that it is possible to have a model, which consists of more than one polyhedral domain. This is necessary for the accommodation of multi material regions, a simple example of which is two adjacent cubes.

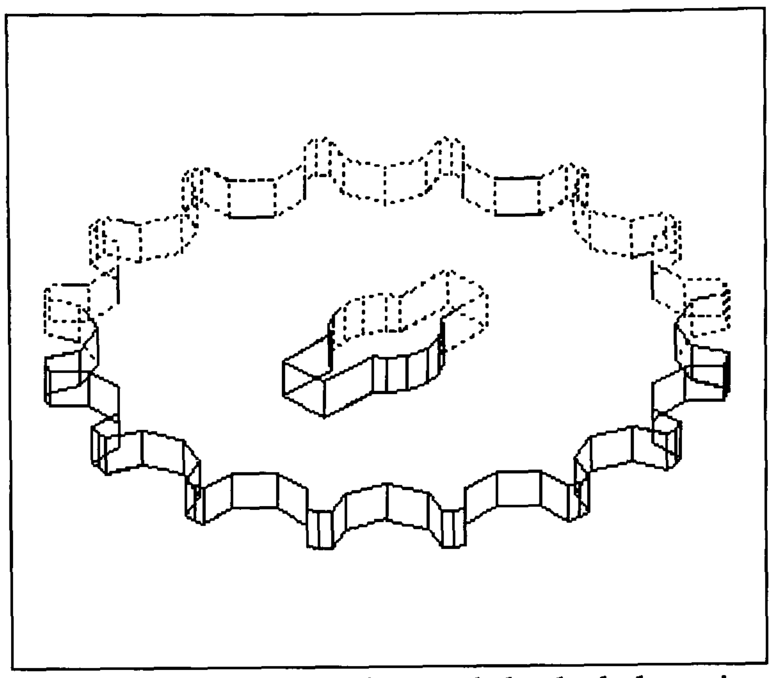


Figure 6.3.1 : Complex polyhedral domain.

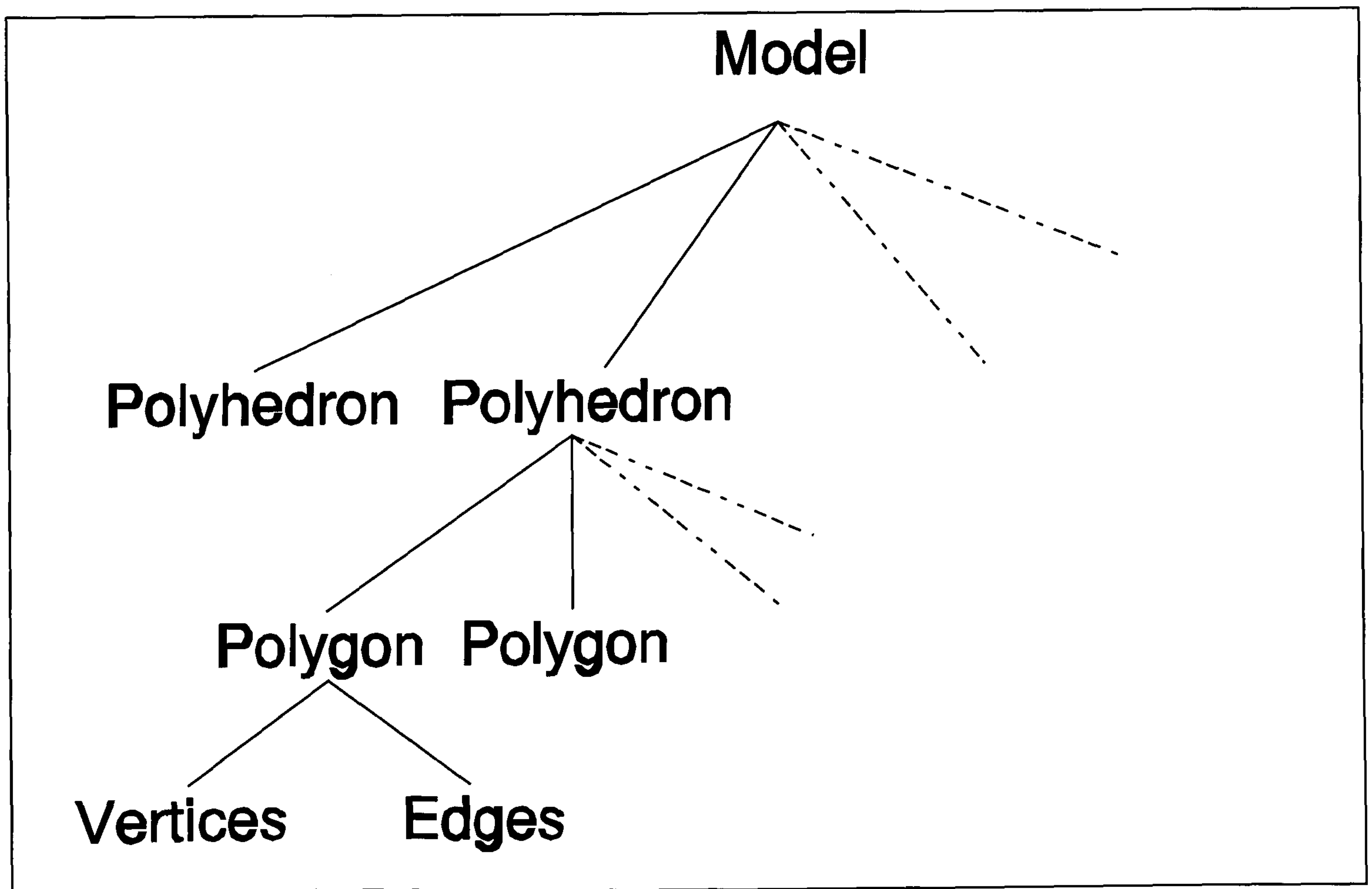


Figure 6.3.2 : Hierarchical structure of Polyhedral models.

6.4 Bisection method.

This section will outline the techniques used for the selection of bisection planes. The reliability and quality of the techniques used for the bisection of a geometry into disjointed parts are fundamental to the automatic mesh generator. Sub-dividing geometry is a highly complex task, therefore in the literature there exist a number of techniques, see [Man88][Cha84][BaD92] amongst others. These techniques, rely on finding all the intersections of the cut plane and the faces of the polyhedral domain first. These approaches [Man88][Cha84][BaD92], were found to be inappropriate for the mesh generation algorithm, since they often introduced acute angles into the domain [BaD92]. Therefore, techniques based on the above methods were developed, namely the *Polyhedral cut face algorithm* and *Edge following bisection,* and are described below.

These methods trace out the boundary of a cut face from an initial starting edge, lying on the bisection plane of the polyhedral domain. This edge tracing approach enables local bisection, Figures 6.4.2, where finding the intersection of all polyhedral faces of a polyhedral domain is inappropriate. Also, it enables the tracing out of non-planar bisections faces, Figures 6.4.3 and 6.4.4. The *Polyhedral cut face algorithm (Figures 6.4.1 and 6.4.2)* introduces a planar interface element, where the *Edge following bisection* (Figures 6.4.3 and 6.4.4) tries to follow the outer contours of the domain and for any single bisection generates many interface polyhedra. These two methods are described in more detail in Chapter 7.3.

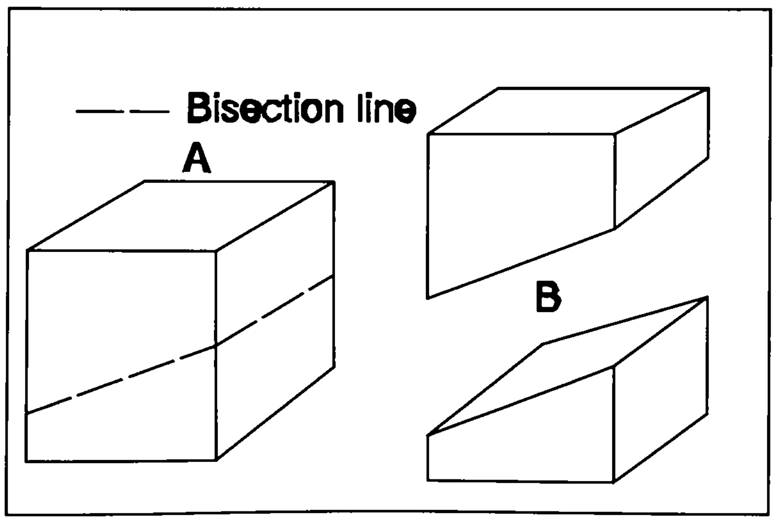


Figure 6.4.1 Bisection of a domain.

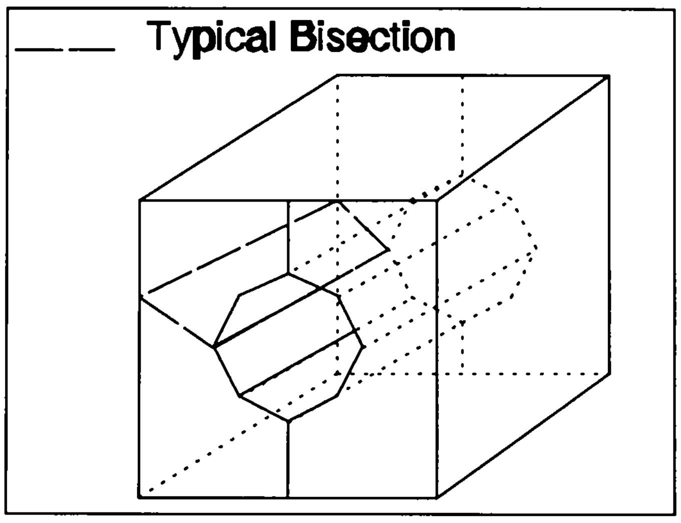


Figure 6.4.2 Internal bisection polygon.

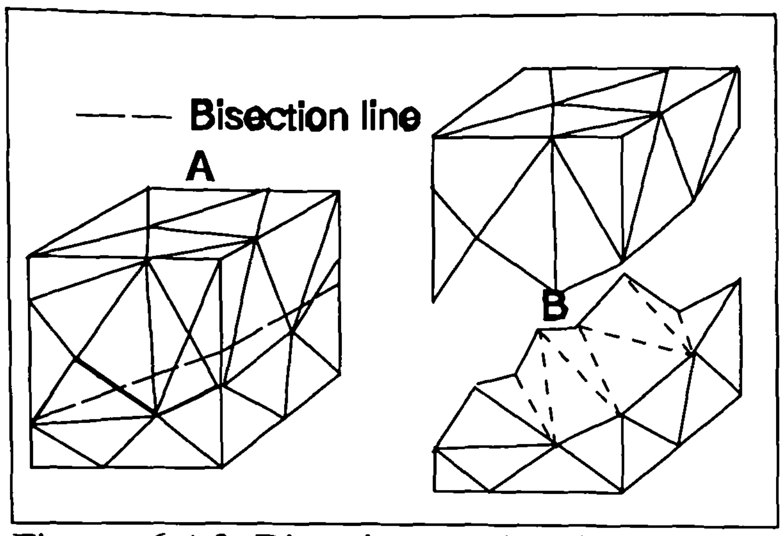


Figure 6.4.3 Bisection walk which cuts a domain into two independent regions.

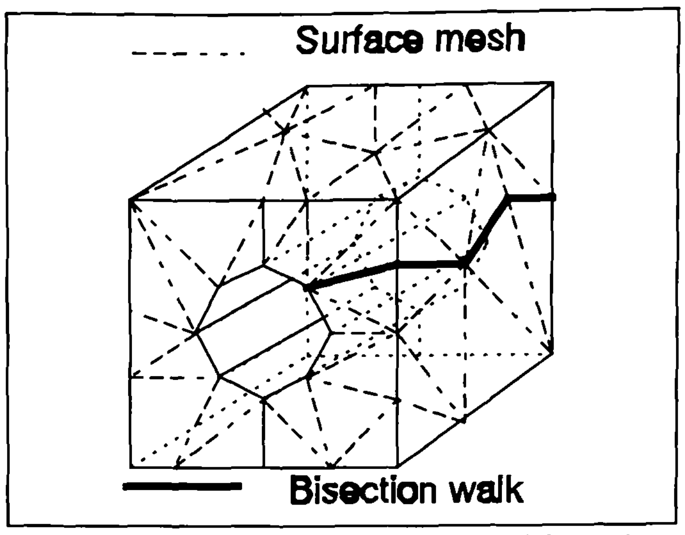


Figure 6.4.4: Internal bisection polygons generation

The *polyhedral cut face algorithm* is the preferred method, since it is the simplest and fastest, and only introduces one simple polyhedral interface into domain. However, this technique for various reasons, see Chapter 7.3.3, cannot always be applied, which provides the motivation for the second method *Edge following bisection* that can bisect all types of geometry without limitations.

6.4.1 Selection of cut face.

Algorithms for the selection of bisection planes for decomposing polyhedral regions are given in [Cha84] and [BaD92]. However, these algorithms are optimized for decomposing domains into convex regions and worst case time complexity, i.e speed of execution. However, we are concerned with the task of bisecting a domain into an arbitrary number of regions and the resulting quality of the bisection. This is achieved by avoiding the creation of small angles between polyhedral faces and bisection planes.

The selection of a bisection plane P_b given an initial edge *e* in a polyhedral region S is fundamental to the quality of the overall mesh. Edge *e* is the adjacent edge of the polygon faces P_0 and P_1 and is defined by the vertices V_0 to V_1. The initial stage is the identification of the inner wedge region. Figures 6.4.5 and 6.4.6 show an inner wedge region for convex and concave polygon pairs.

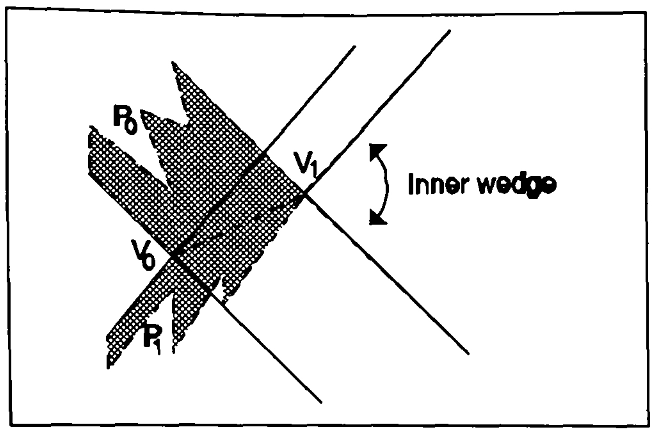


Figure 6.4.5 convex polygon's inner wedge.

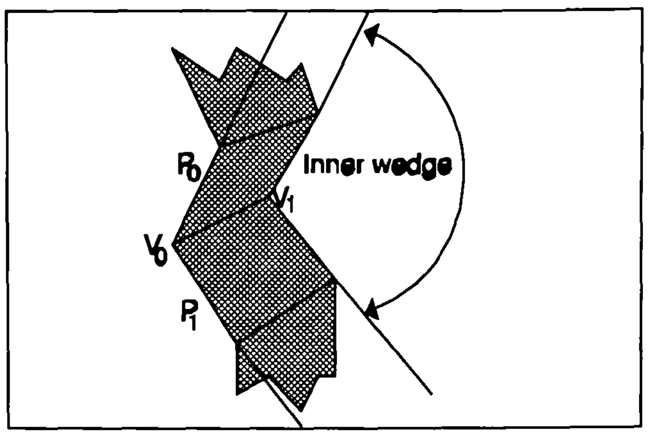


Figure 6.4.6 Concave polygon's inner wedge

The inner wedge region is then bisected into, at most, seven possible bisection planes, (Figure 6.4.7) with angles of at least ten degrees between them. This provides a reasonable spread of bisection planes for subdividing the domain. These bisection planes are used one after another, in the order given in Figure 6.4.7, in an attempt to bisect the domain using the polyhedral cut face algorithm in section 7.3.1. If all the attempts fail the process is repeated using the "*edge following bisection*" method of section 7.3.3

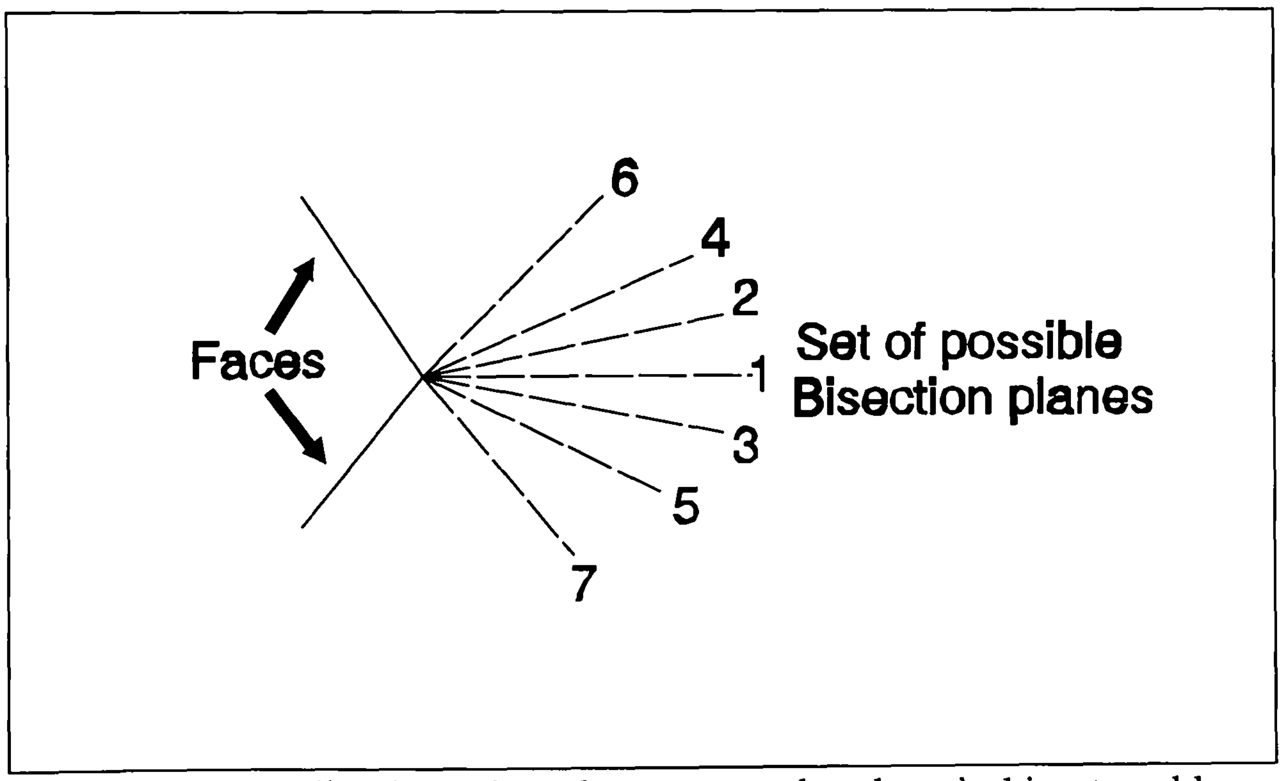


Figure 6.4.7 Possible bisection planes to resolve domain bisect problem.

6.5 The sub-region problem.

Other geometry bisection techniques, [Man88][Cha84][BaD92], handle multiply-connected regions without modifications. However, it was found by simplifying the geometry of the domain by introducing interface polyhedra, that join cavity regions up to the outer boundaries, this enables a simpler and more robust bisection algorithm to be applied during grid generation. This section will outline a technique that implements the two bisection techniques *Polyhedral splitting algorithm* (Chapter 7 section 7.3.2) and *Edge following bisection method* (Chapter 7 section 7.3.3).

Geometry simplification is achieved by finding a cutting plane, which bisects both the inner and outer polyhedral domains. The geometry is then separated into two parts and a set of interface polygons are generated between the two new independent domains. In Figure 6.5.1, a cube with a cavity is divided into two simpler polyhedral regions, using the *Polyhedral splitting algorithm* and Figure 6.5.2 illustrates this operation using the *Edge following bisection method.*

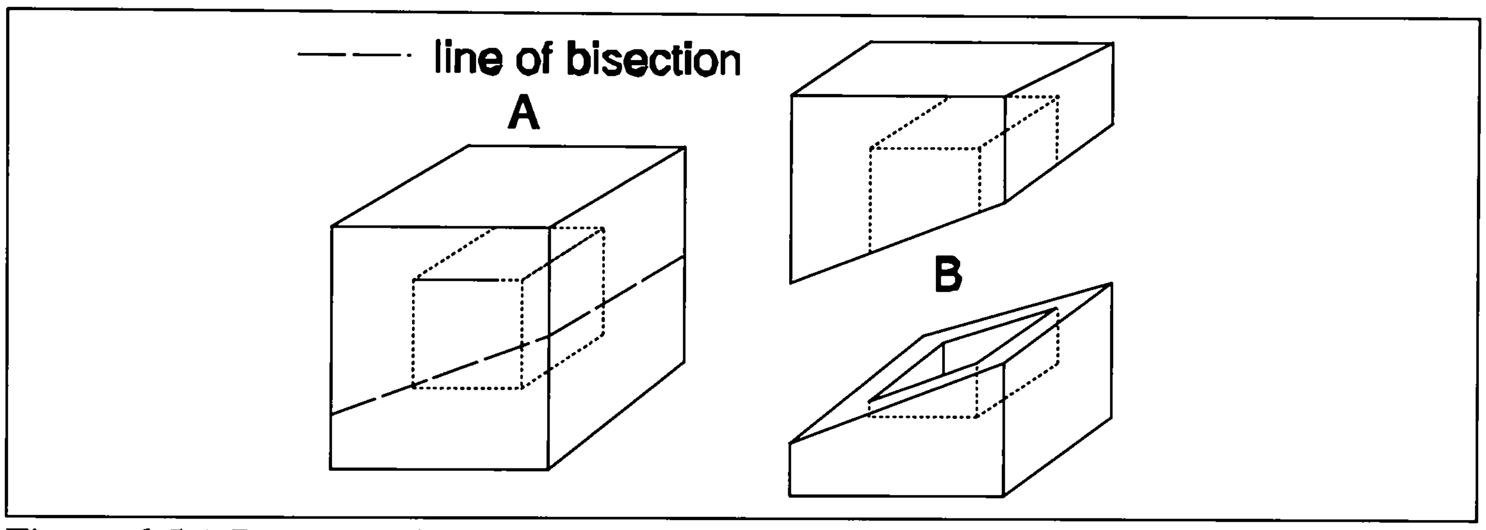


Figure 6.5.1 Decomposing a complex polyhedral region into two simple regions.

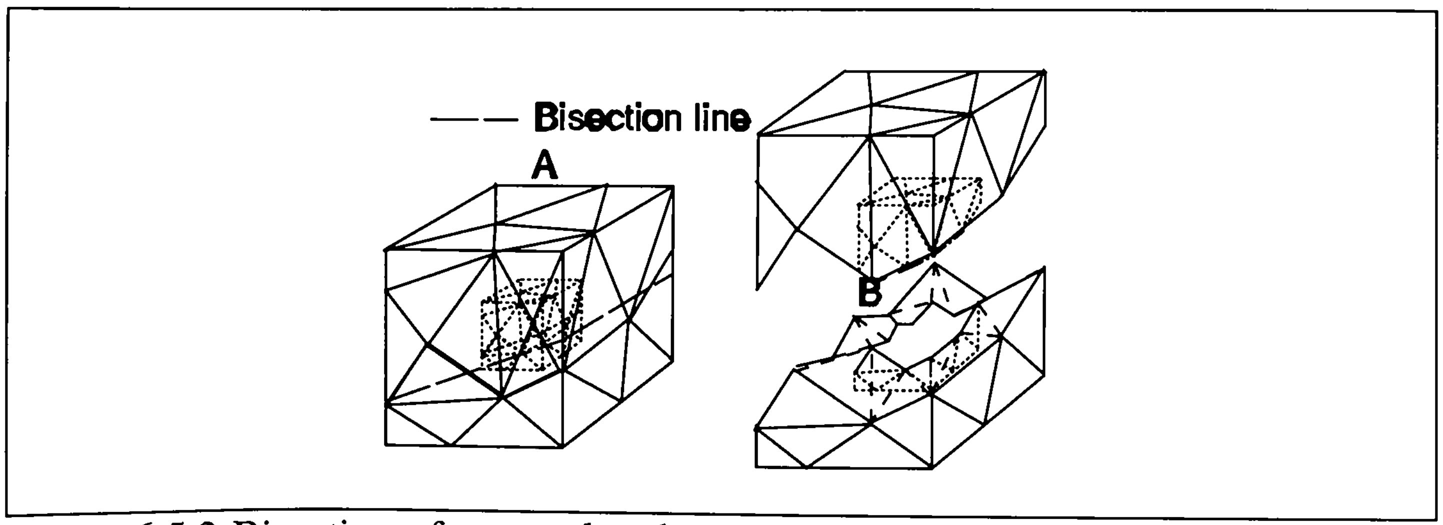


Figure 6.5.2 Bisection of a complex domain.

The selection of the bisection plane is vital if we want to avoid the creation of degenerate angles, short edges or narrow regions. However, these problems cannot always be avoided in complex geometries with several sub-domain regions.

A simple approach is applied for finding the bisection planes, which takes each face of a sub-polyhedral domain in turn and uses its face normal to define a cutting plane. This bisection plane is then used in the polyhedral splitting algorithm described in section 7.3.2 to divide the domain. Figure 6.5.3 shows a region which has been resolved using this technique.

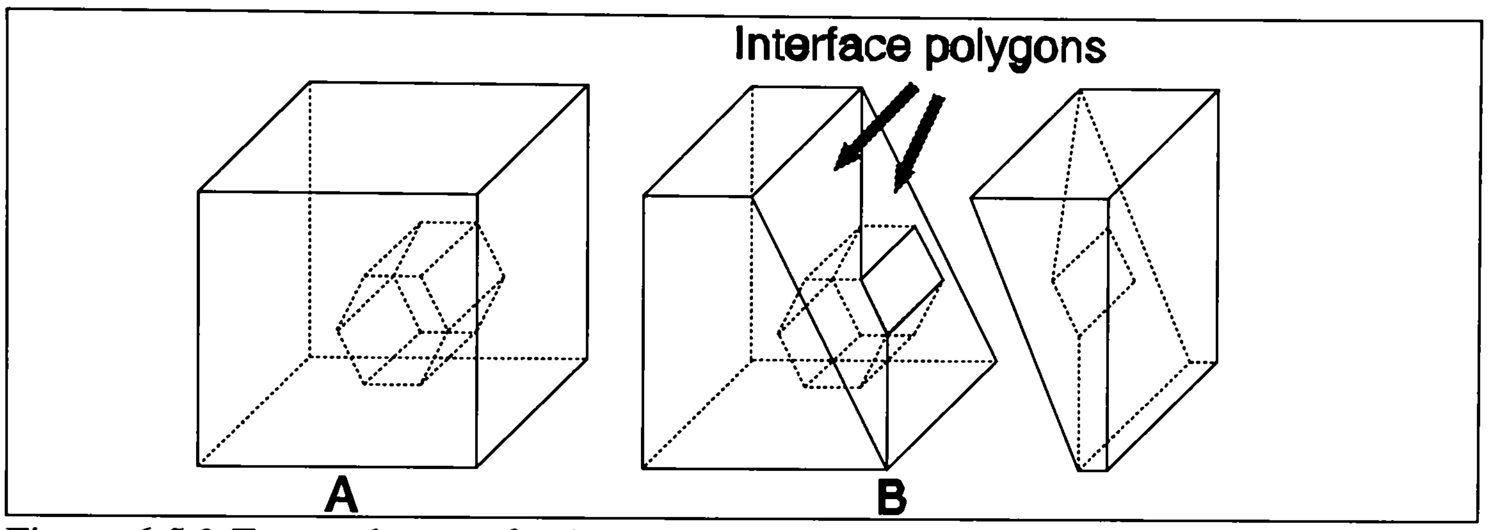


Figure 6.5.3 Face polygon of subregion used to bisect domain into two polyhedra.

However, the above algorithm may fail to find an acceptable bisection. If the above method fails, a second technique is applied where each reflex edge of the inner domain is examined in turn to see if it can be used to resolve the sub-region. Let P_0 and P_1 be two adjacent faces that form a reflex edge in the inner polyhedral with normal vectors N_0 and N_1, respectively. Let V_0 and V_1 represent the vertices which form the adjacent edge of the polygons P_0 and P_1. An inner wedge is defined, Figure 6.5.4, which is the intersection of the half space defined by two planes with normals N_0 and N_1 that pass through the edge V_0 to V_1. A set of planes are then calculated, which subtend the inner wedge angle between 20 and 30 degrees, and these are then used to resolve the *hole polygon*.

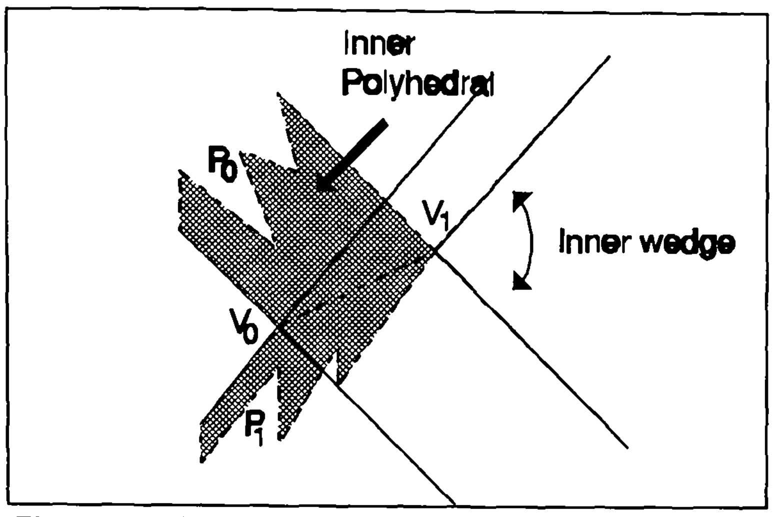


Figure 6.5.4 Inner wedge

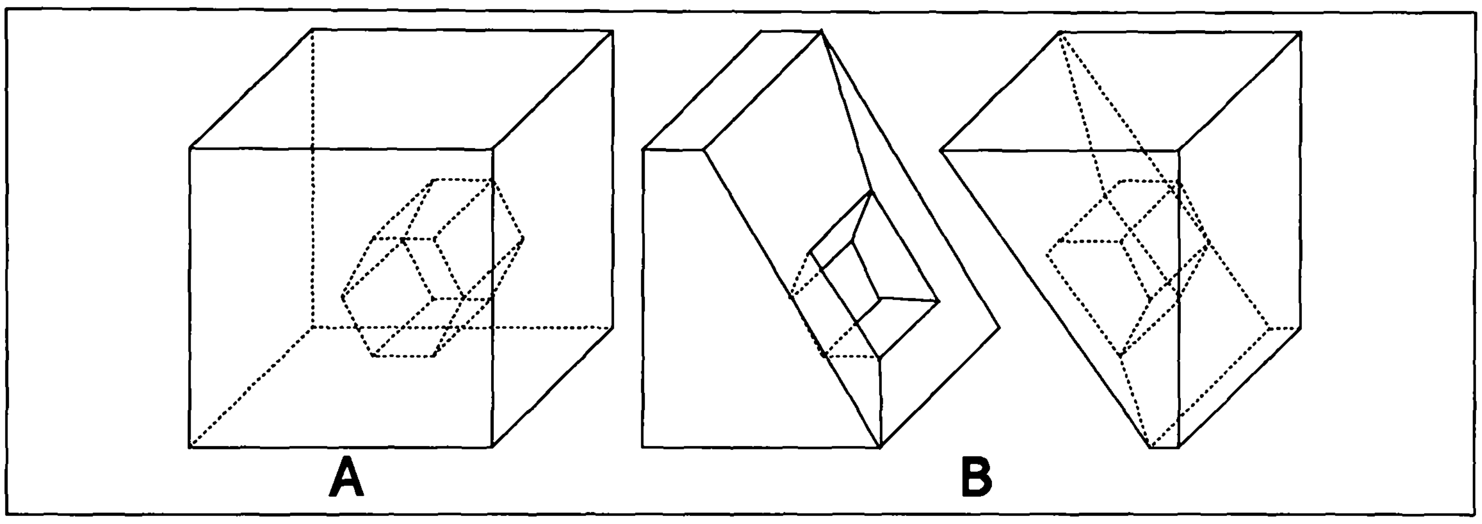


Figure 6.5.5 Example Bisection using the above technique.

If the above method fails the whole procedure is repeated using the alternative method of *edge following bisection*, described in section 7.3.4.

6.6 Problems with recursive domain bisections.

The initial recursive 3D mesh generator coupled with the 2D nodal placement algorithm (Chapter 4 section 4.6.5) was found sufficient to generate a three dimensional tetrahedral grid. However, it was found that this technique tended to generate grids of poor quality and offered little control over mesh density, often generating grids with over refined regions in a way similar to the British Isle's map depicted in Figure 4.5.6 Chapter 4. On regions with complex boundaries and a large number of surface elements there is a substantial number of rejections of bisection planes and cut interfaces that resulted in a dramatic increase in CPU time. Also, on complex domains the *edge following bisection method* (Chapter 7 section 7.3.3), is used as the main domain cutting algorithm which often increases the complexity of the domain, measured by the number of polyhedral faces, and results in highly complex surfaces.

The grid quality generated by this method is very similar to the 2D example of the British Isle's map depicted in Figure 4.5.5. Even when a min-max solid angle optimization algorithm (Chapter 7 section 7.5) was implemented little improvement in the grid could be achieved. This was mainly due the limitations of 3D local transformations.

From the initial work carried out it became clear that a method of nodal placement similar to the one in Chapter 3 section 3.6.4, which first decomposed the domain into a number of convex regions is required.

6.7 Decomposing polyhedral domains into convex regions.

For the nodal placement algorithm to work, the domain has to be decomposed into convex regions. Decomposing a region into convex parts is a well researched area, [Cha84][Bad92] and can be achieved by introducing bisection planes which subtend reflex edges of the model. The algorithm implemented in the mesh generator is outlined next, and is implemented using the bisection techniques described in Chapter 7 sections 7.3.1 and 7.3.3:

(1) Generate an order list of reflex angles, descending order of angle size. Any angle between two polygon faces larger than $\pi+\varepsilon$, where ε is machine tolerance, is regarded as reflex.

(2) Remove next "reflex angle" edge E from head of list.

(3) If edge E on a double occurring face, add to the tail of list of reflex angles, go to (2). Double occurring faces are polygons which form an internal interface within a region, but do not cut the domain completely into two parts. Therefore, they are identified by the fact that both sides of the polygon are in the same domain. Double occurring faces with reflex angles are left to the end after all other angles have been dealt with.

(4) Try to remove reflex angle with either the polyhedral cut face algorithm, section 7.3.1, or the edge following method section 7.3.3. If the edge following method is used, add any newly formed reflex edges to the tail of the relevant list. If this edge cannot be resolved add to the tail of the reflex edge list

(5) repeat from step (2) until all reflex angles are resolved.

In step 4 a reflex edge may not be resolved, the edge is then deferred until other reflex edges have been resolved; then the sub-polygon containing the difficult edge may be smaller or may be subdivided by other cut faces into two or more reflex sub-edges. The above algorithm is not guaranteed to resolve all reflex angles. Therefore, binary mesh operators [Chapter 3 section 3.3] are applied to any remaining unresolved edges.

6.8 Generation of grids within sub-regions.

Once the internal nodes are generated and inserted, described in section 6.11, the generation of the mesh can begin. An initial attempt at meshing the domain is given by:

(1) Generate a surface grid over each polygon face. This is done to ensure that the interface of the grids generated in each convex sub-domain is consistent.

(2) Take each region in turn and using the edge following bisection technique, section 7.3.3. attempts to generate a grid of tetrahedra.

Step (2) involved the inclusion of internal nodes that were less than $0.5B_w$, where B_w is the *band width* (see section 7.3.3) of the bisection from the bisection plane. Therefore, this involves the tracing out of the cut interface boundary along the edges of the domain being bisected, as in section 7.3.3. The band width of the bisection interface is calculated, and any internal nodes within half of this distance either side of the cut plane (Chapter 7 section 7.3.3) is included in the set of nodes to be meshed. This set of nodes are then projected onto the cut plane and rotated onto the Z=0 axis. They are then meshed using a 2D mesh generator, the connectivity of this mesh is then used to stitch the points in 3D space. From this bisection the domain would be split into two or more *halves* and the associated data structures are updated. This process is applied recursively to each region until valid tetrahedra are formed.

The above method was found to be slow, since bisection planes were often rejected. There was also the problem of regions which could not be bisected or tetrahedralized, which required the introduction of extra edges and / or vertices on the polygon faces and within the body of the domain. The method would often produce invalid and intersecting tetrahedra because of the difficulties involved in bisecting regions using an edge following bisection technique coupled with internal node inclusion.

However, in the literature [Joe92b][Bak92][GHS88][Joe92c] there are several algorithms published for generating meshes in convex regions using Delaunay triangulation with a computational order of *nlog(n)*. The method which was selected here is based on 3D vertex transformations and is described in section 7.5.

The initial attempt at meshing the convex regions was implemented using boundary constrained Delaunay triangulation, however, this was rejected due to the generation of degenerate tetrahedra, that can result in the failure of the algorithm. An idea by Joe [Joe89] can be adapted to use a local min-max angle criterion, instead of circum-spheres of tetrahedra, as a grid generation heuristic. This method of generating grids in convex regions will be described in Chapter 7.6.

The algorithm implemented is a local min-max meshing technique, which requires at least one interior node, since there always exists a boundary constrained triangulation for a convex polyhedral with one mesh node within its interior [Wör81].

The final meshing technique is still a recursive bisection technique, where the domain is recursively divided until the resulting sub-domains are sufficiently small so that a simple algorithm, *Boundary constrained* min-max triangulation, may be applied. The final sub-domain meshing algorithm may be extended to a full recursive-bisection technique, once improvements have been made to the bisection part of the program. This will also have to be done with improvements to the 3D min-max transformation algorithm, see section 7.6.1.

The benefits of making the algorithm a fully recursive technique are not yet clear, as the *Boundary constrained min-max* algorithm is highly efficient. Also, there is no current thought on how to extend a full recursive bisection technique to automatically generate quadrilateral and hexahedral elements.

6.9 Outline of final mesh generator and results.

The final mesh generator with example results, are presented in this section.

6.9.1 Input requirements.

(1) Input of geometry is in the form of a polyhedral domain, an example input is given in section 6.13.

(2) The user specifies a required number of elements, which the mesh generator will try to accommodate.

6.9.2 The mesh generator.

(1) Read data file; simplify model by removing cavities, section 6.5, and remove hole polygons in polyhedral faces, section 4.6.3.

(2) Decompose polyhedral domain into a set of convex polyhedral regions, see section 6.10 below.

(3) Generate nodes in polyhedral regions, see below section 6.11.

(4) Generate boundary grids over all polyhedral faces, this ensures that the mesh conforms between sub-regions.

(5) Generate boundary constrained local min-max triangulation in sub-regions, section 7.6.

The user then has the following options to optimize the mesh:

(1) Local min-max swapping routine applied over the whole domain, section 7.5.

(2) Laplace's smoothing.

The optimization methods can be applied in different combinations, as the ordering of these mesh operators can affect the overall quality of the final mesh. For a complete overview of how modules in the mesh generator can be interlinked, see Appendix A6.

6.10 Convex domain partitioning.

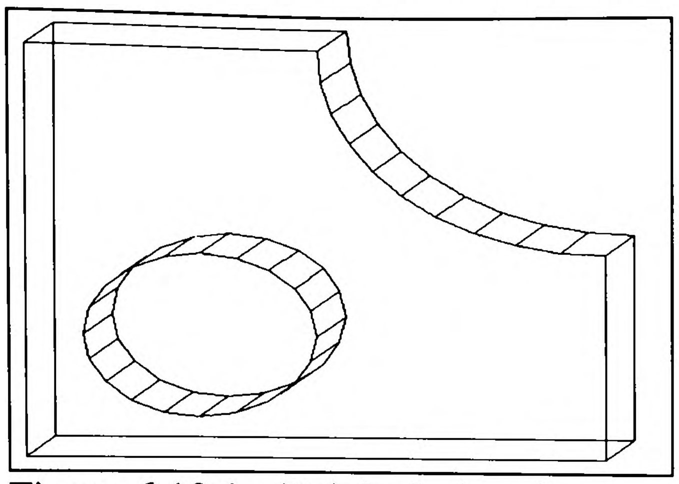


Figure 6.10.1 : Polyhedral region.

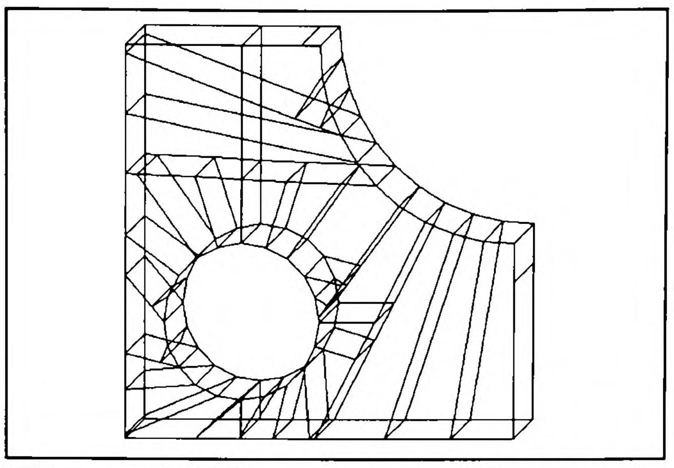


Figure 6.10.2 : Polyhedral domain bisected into convex parts.

The above figures represent a polyhedral domain, Figure 6.10.1, which is then divided into number of convex regions Figure 6.10.2.

The polyhedral decomposition into convex regions is dependent on the complexity of surface definition. Figure 6.10.3 illustrates a surface mesh of the above polyhedral domain depicted in Figure 6.10.1. Since a surface mesh already has a large number of nodes associated with it, the program will try and avoid the introduction of unnecessary face vertices, therefore, the edge following algorithm in section 7.3.3 is used. The resulting decomposition is illustrated in Figure 6.10.4, where a large number of bisections are necessary to generate a set of convex regions.

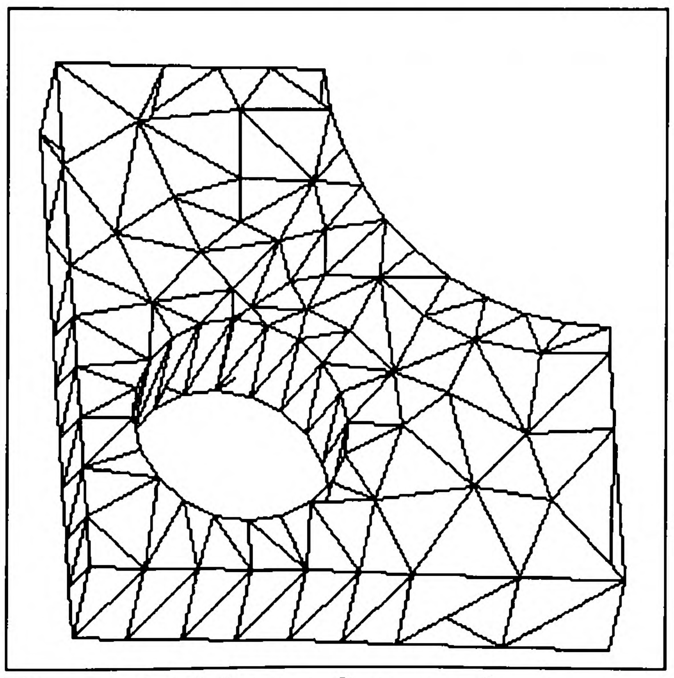


Figure 6.10.3 : surface mesh.

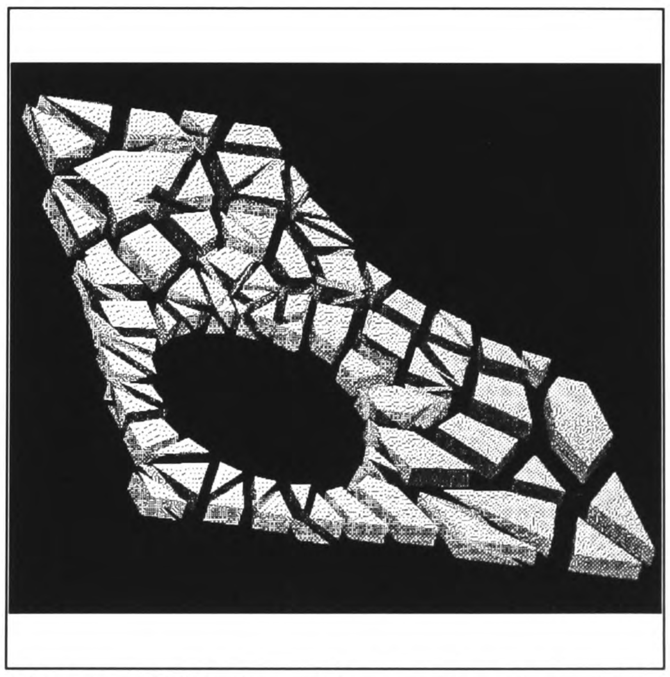


Figure 6.10.4 : Decomposition into convex parts.

6.10.1 Domain bisection.

To demonstrate the computation efficiency of the bisection method in section 6.4, a family of polyhedral domains were defined, which were derived from a common primitive. The family of polyhedral domains used were derived from taking a sphere, of any radius, and generating a given number of polyhedral faces over its exterior boundary, see Figure 6.10.5. Each polyhedral domain was then taken in turn, and the CPU time was recorded for the mesh generator to perform one bisection. The results obtained, were then plotted in the form of a graph, see Figure 6.10.6.

The graph below, shows a comparable linear computational growth for the number of polyhedral faces against the time taken to perform a domain bisection. However, since the primitive object is a sphere, we can only conclude that the graph below is representative of the bisection algorithm working on convex geometries.

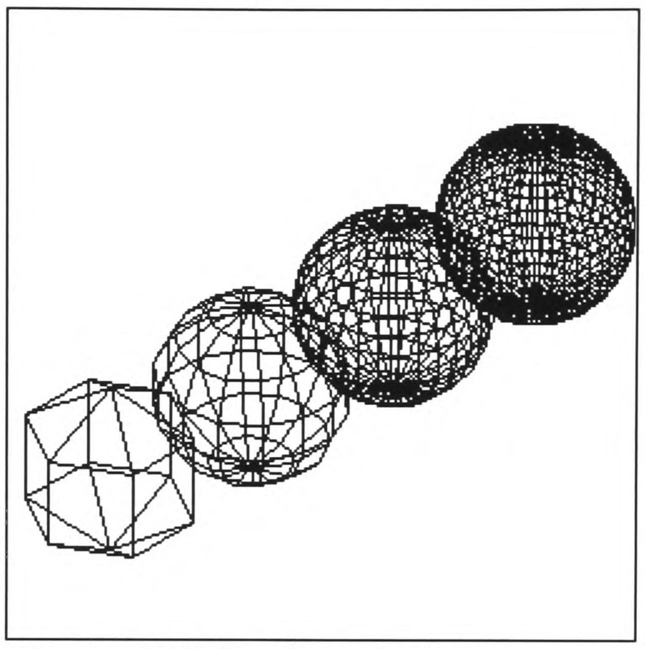


Figure 6.10.5 : Set of convex polyhedral domains

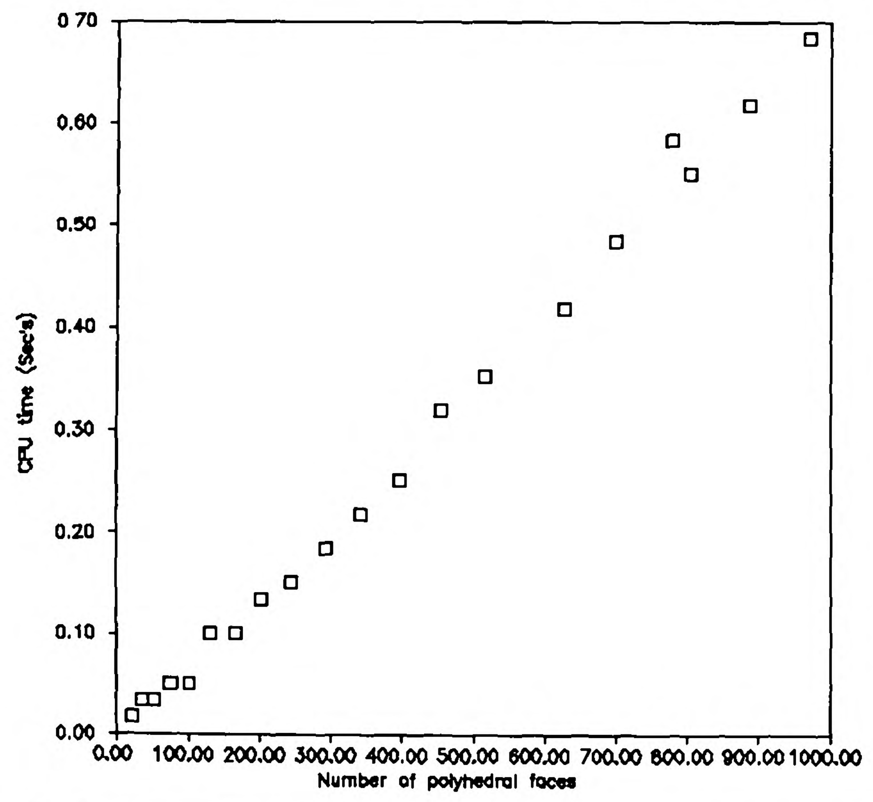


Figure 6.10.6 CPU time to bisect a polyhedral region with a given number of boundary faces. CPU times are for non-optimized code running on a Sun Sparc station 10.

6.10.2 Complex domain bisection.

Since the previous section 6.10.1, was based purely on a convex domain a new model was developed, see Figure 6.10.7.

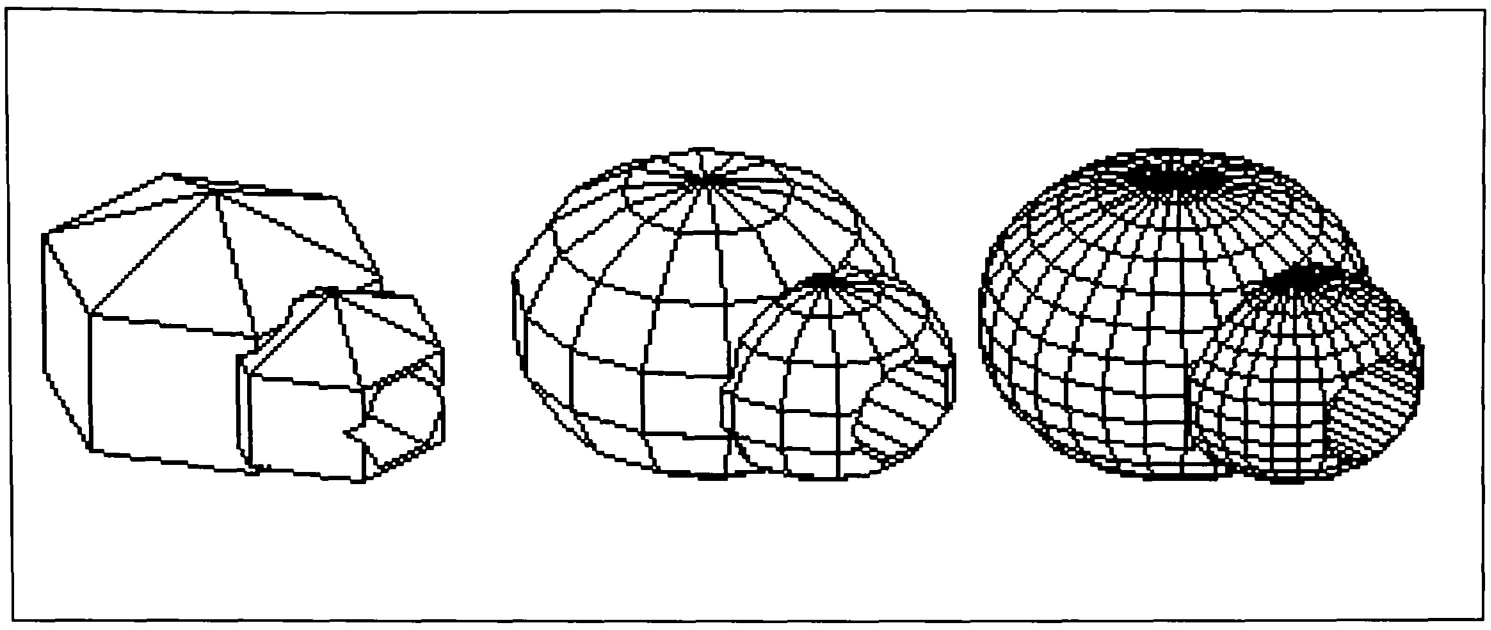


Figure 6.10.7 : Set of test domains.

The above domains are generated from a common object, which had a different number of polyhedral faces generated over it. The common primitive was generated from two intersecting spheres, of different radii, which has a cylindrical hole passing through both.

The graph below, is a plot of number of polyhedral faces, over the above object, against CPU time to preform one bisection of this domain. This graph shows a good computational comparison for time to preform a bisection of the domain against number of surface elements.

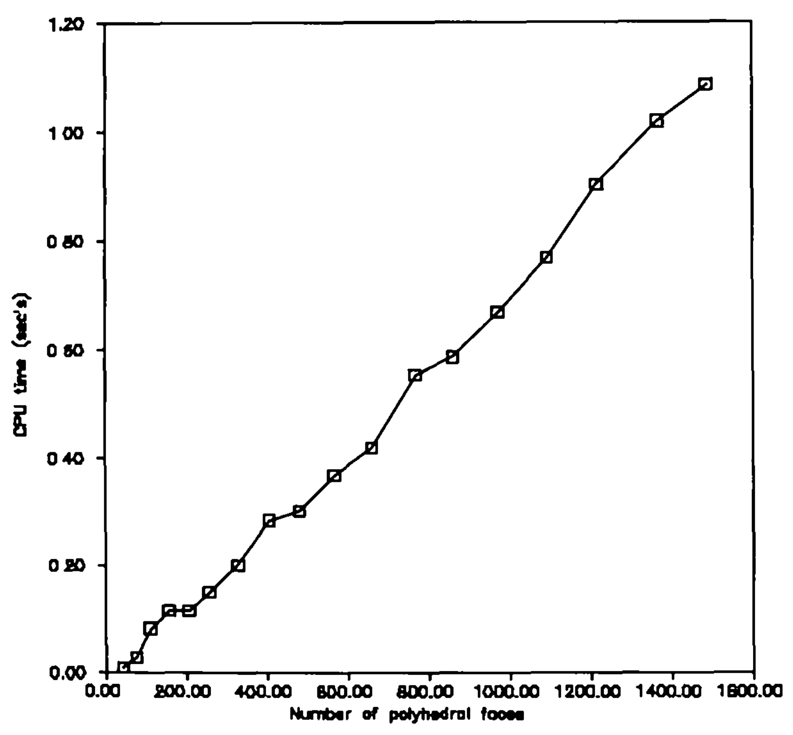


Figure 6.10.8 : CPU time to preform a bisection on a polyhedral domain with a given number of polyhedral faces.

6.11 Convex domain shrinking and nodal placement.

The requirement of the nodal placement algorithm is that the user specifies the required number of elements R and the mesh generator attempts to generate a grid to that given density. Once the domain is divided up into convex regions, the total volume V of the domain can be calculated [Sto91] and used to find the *ideal nodal spacing* N_p. To estimate the average tetrahedral volume V_t we divide the volume V by the required number of elements R. Therefore the nodal spacing N_p is calculate by :

$$N_p = \sqrt[3]{8V_t} \quad \textit{See appendix A2}$$

After the nodal spacing is calculated a method of shrinking convex regions is applied, as in the method explained in Chapter 4.6.4, which generates nodes on the surface of concentric domains, Section 7.4.

6.12 Geometry input requirements.

A 3D mesh generator must be able to cope with relatively complex boundary models, which consist of polyhedral domains. A typical polyhedral model can have face holes, interior holes, internal interfaces and / or hole interfaces, Figure 6.12.1.

Faces	: are regions defined by a set of planar points in counter clockwise order (when viewed from the outside)
Face holes	: regions defined by a set of planar points in an ordered list around the boundary of the face.
Interior holes	: independent polyhedral regions, which defines a hole inside another polyhedral.
Internal interfaces	: polygon faces, which define a boundary between two polyhedral regions.
hole interfaces	: polyhedral region, which defines the interface of two regions, of which one is completely contained inside the other.

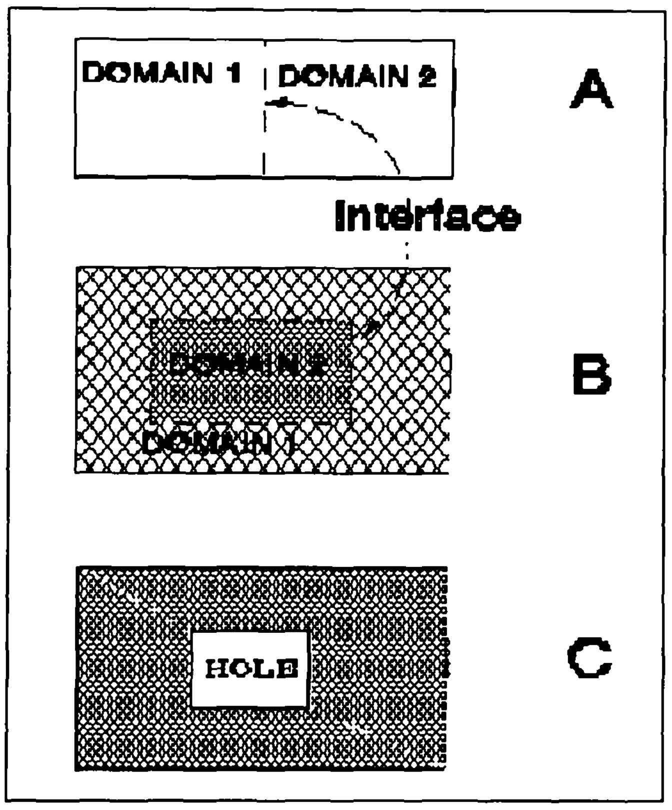


Figure 6.12.1 Three forms of interface polygons.
(A) Interface between two domains,
(B) Sub-polyhedral interface,
(C) Hole interface polygon region.

All the above information about a model must be ascertained before the generation of the grid can begin. This information can either be provided by the user or a CAD package, however the grid generator should minimize its input requirements.

Therefore the following information has to be obtained from the geometry description of the model:

(i) Which groups of polyhedra form interfaces and polyhedral regions,
(ii) The set of polyhedral regions that represent cavities,
(iii) The direction of surface elements' normals,
(iv) Sub-polyhedral domains and the polyhedral regions they are contained in.

The data input requirements of the new bisection method, is based on the following :

The automatic identification of which groups of polygons *(i)* form each polyhedral domain, can only be practically achieved when there are no *interface polygons*. Therefore, the user is required to specify which polygon faces belong to which polyhedral domains, however, interface polygons can be automatically identified by the program.

Whether or not a polyhedral region represents a cavity *(ii)* has to be either provided by the user or the CAD package that generated the model. However, the direction of face normals *(iii)* can be established, by taking any face K on each polyhedral domain. A ray can then be projected from one of the nodes on the face K, in the direction of the face K's normal. The number of times this ray intersects with the current polyhedral indicates the direction of the face normal. If there are any polyhedra faces that intersect the ray with a *small angle*, which may produce computational errors, a different node can then be selected on the face K, for the starting point of the ray. Once the direction normal of one face on a polyhedral domain is established, all other polyhedra normals can be set relative to this face.

Satisfying point *(iv)* the identification of sub-polyhedral and to which outer polyhedral they belong, is relatively straight forward. We can simply take one node of the polyhedral region that is not a member of an interface polyhedra. A ray testing routine [Rog85] can then be applied to the model to find which domains are contained in other regions.

6.13 Mesh generator's input format.

Taking the information in section 6.12 into account, resulted in a relatively simple data input requirement for the new bisection method. The algorithm just requires a list of points, polyhedral faces and a polyhedral domain face list. The polyhedra faces are defined as an ordered list of nodes, in either direction around the boundary. This data input format which is illustrated in the next two examples, can handle most forms of polyhedral domain with interface boundaries, cavities and sub-domains.

Example 1

An example cube with a cavity adjacent to another cube.

```
cube with a cavity in, adjacent to another cube.
17 20 3 26 --  No. faces,  No. nodes,
               No. of polyhedra regions, No. of required elements.

0.0 0.0 1.0       -- nodes X Y Z.
1.0 0.0 1.0
1.0 1.0 1.0
0.0 1.0 1.0
0.0 0.0 0.0
0.0 1.0 0.0
1.0 1.0 0.0
1.0 0.0 0.0
2.0 1.0 1.0
2.0 1.0 0.0
2.0 0.0 0.0
2.0 0.0 1.0
0.25 0.25 0.75
0.75 0.25 0.75
0.75 0.75 0.75
0.25 0.75 0.75
0.25 0.25 0.25
0.25 0.75 0.25
0.75 0.75 0.25
0.75 0.25 0.25
                  -- polyhedra faces.
 4  1  2  3  4    -- number nodes in face, node list.
 4  7  8  5  6
 4  1  4  6  5
 4  4  3  7  6
 4  1  5  8  2
 4  2 12  9  3
 4 10 11  8  7
 4 12 11 10  9
 4  3  9 10  7
 4  2  8 11 12
 4  2  8  7  3
 4  13 14 15 16
 4  17 18 19 20
 4  17 20 14 13
 4  19 18 16 15
 4  17 18 16 13
 4  20 19 15 14
                      -- polyhedral domains.
 6  1  2  3  4  5 11  -- number of faces, list of faces.
 6  6  7  8  9 10 11
-6 12 13 14 15 16 17  -- negative number of faces indicate a cavity
                         polyhedral region
```

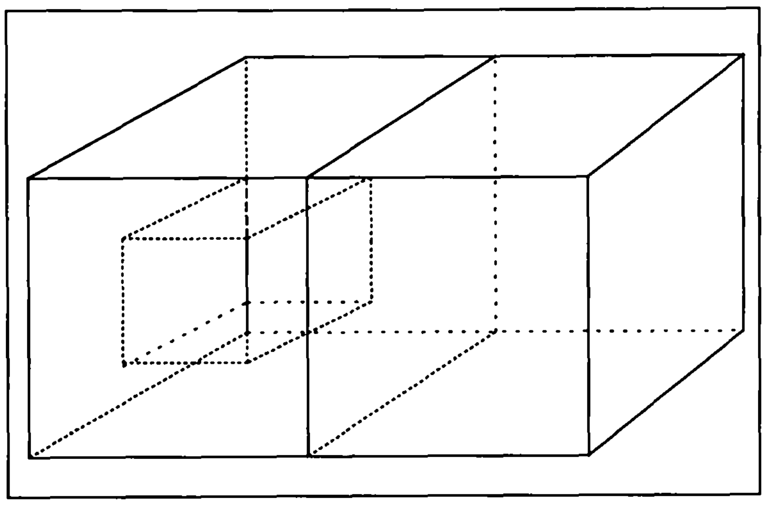


Figure 6.13.2 Box with a cavity adjacent to another box.

Example 2

Example of a Re-entry polyhedral with a complex top face.

```
Re-entry box
 12  16 1 26  -- No. of faces, No. of nodes , No. of Polyhedral domains,
               No. of required elements
 0.        0.        0.       -- Nodes X,Y,Z
 1.00000   0.        0.
 1.00000   1.00000   0.
 0.        1.00000   0.
 0.        0.        1.00000
 0.        1.00000   1.00000
 1.00000   1.00000   0.
 1.00000   1.00000   1.00000
 1.00000   0.        1.00000
 0.250000  0.250000  0.250000
 0.750000  0.250000  0.250000
 0.750000  0.750000  0.250000
 0.250000  0.750000  0.250000
 0.250000  0.250000  1.00000
 0.750000  0.250000  1.00000
 0.750000  0.750000  1.00000
 0.250000  0.750000  1.00000
```

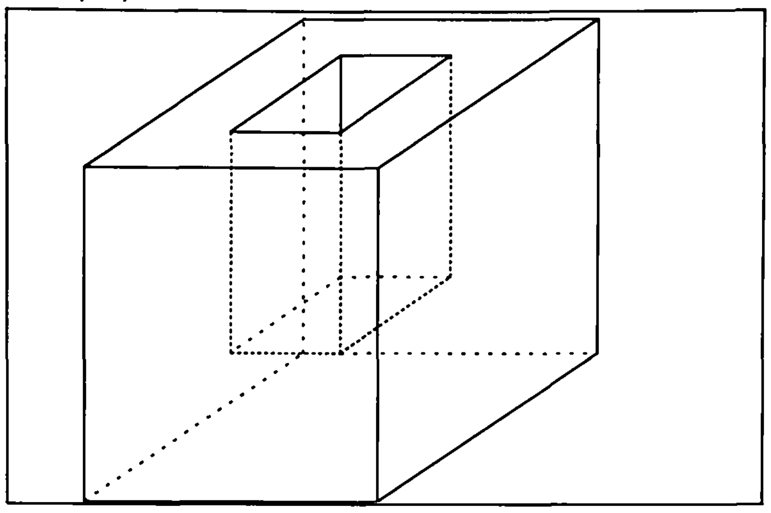


Figure 6.13.3 Re-entry Polyhedral domain with complex top face.

```
                -- polyhedra faces.
 4 1  4  3  2  -- number nodes in face, node list.
 4 1  5  6  4
 4 3  4  6  7
 4 2  3  7  8
 4 1  2  8  5
 4 5  6  7  8
 4  9 10 11 12
 4 11 12 16 15
 4 10 11 15 14
 4 12 16 13  9
 4 13 14 10  9
 4 16 13 14 15
                           -- Polyhedral domain face list
12 1 2 3 4 5 6 7 8 9 10 11 12 -- number of faces, list of faces.
```

As can be seen from the above two examples, the data input is so simple that it can be learnt quickly. The only meshing information is the required number of nodes provided by the user. However, this input format is still difficult to generate for complex shapes.

6.14 Summary and Conclusions

As described in this chapter, the initial idea of the recursive bisection mesh generator has been extensively modified and developed. This has resulted in a computational and reliable mesh generaton technique.

The method presented in this chapter is still a problem reduction technique, where an initial geometry is first simplified, by *cavity removal* and then *concavity removal.* Then once this geometry model is sufficiently simplified, a simpler algorithm can then be applied, based upon boundary constrained min-max tessellation.

The next chapter will now describe the fundamental algorithms on which the new bisection mesh generator, described in this section is contingent.

Chapter 7
3D Mesh Generation
Computational techniques.

7.1 Introduction

This chapter gives a detail description of the low level algorithms, which the last chapter is contingent on. The Data structure and method of storage used within the mesh generator, which is crucial for speed and robustness of code, is initially described.

Nodal placement within a mesh generator and its tetrahedra tessellation methods form the fundamental key to mesh quality. Therefore, a detail description of the four types of bisection techniques used within the code, which form the heart of the mesh generator, are presented. These bisection methods are used for domain decomposition and cavity removal as discussed in Chapter 6.

The nodal placement technique based on convex domain shrinking is described, together with how this is combined with a direct local boundary constrained min-max tessellation algorithm. The min-max tessellation method is based on local-transformations by Joe [Joe89] and is also described.

7.2 Data structures

The data structures within the code affect computational speed and ease of geometry manipulation [BaD92]. However, speed and the amount of information stored must be weighed against memory costs and the types of algorithms implemented. These issues are examined in this section together with a description of the data structures used within the mesh generator.

7.2.1 Data requirements of polyhedral domains.

The main elements of a boundary model are :-

faces :	are regions defined by a set of planar points and a surface normal.
face holes :	regions defined by a set of planar points, which defines a hole inside a face polygon.
interior holes :	independent polyhedral regions, which defines a hole inside another polyhedral
internal interfaces :	polygon faces that define a boundary between two polyhedral regions
hole interfaces :	a polyhedral region, which defines the interface of two regions, of which one is completely contained inside the other.

A data structure is required to represent all the above characteristics, plus enough information to enable the efficient application of complex geometry operations. Three of the main operations that must be preformed on this data structure are :

(1) Insertion of a *vertex* in the interior of an *edge*.

(2) Insert an *edge* in the interior of a *face*.

(3) Insert a *face* in the interior of a *polyhedral* domain.

Figure 7.2.1 is a graphical representation of the type of information which is deemed necessary for three dimensional mesh generation, this is based on a *winged-edge* data structure [Mar88][Bau75][Bra79][GuS85][Kar90].

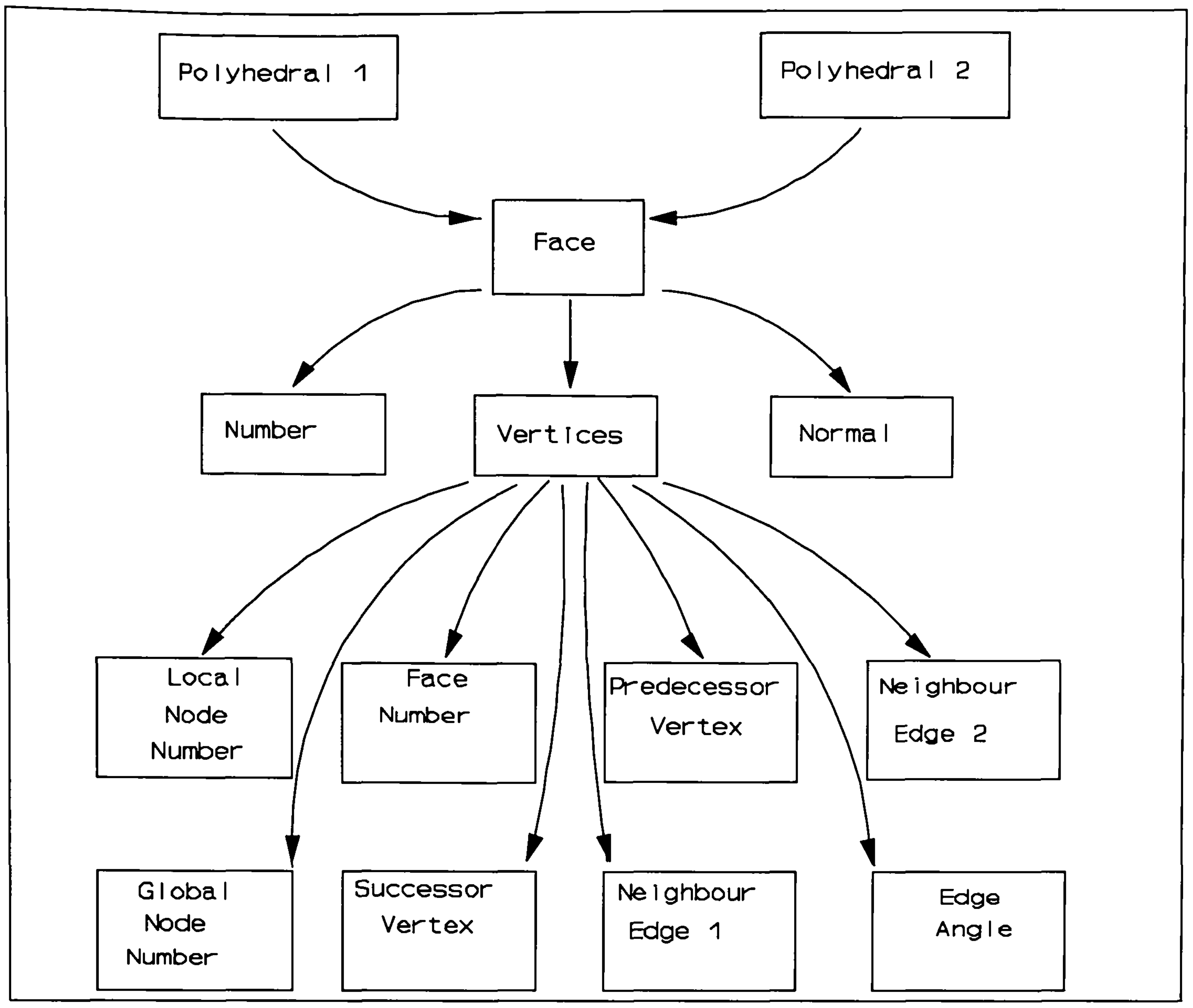


Figure 7.2.1 data structure used for polyhedral domains

From Figure 7.2.1 a large amount of information is required to enable efficient operation of geometry searching. However, the reader should bear in mind that a typical software analysis package would store far more information about a model.

7.2.2 Detail description of stored data.

Below is listed the data structure required for the representation of polyhedral domains for three dimensional mesh generation. This data structure is based on the winged-edge and half-edge data structures described in "an introduction to solid modelling" [Mar88][Bau75][Bra79].

The top part of a polyhedral data structure is a head list, which points to the relevant groups of polygon faces.

Head_Pointer_List:	Pointer to first polygon face (Polygon_Face_List).
	Pointer to next entry in Polygon_Head_List.
Polygon_Face_List:	Face_Pointer, negative if face normal points into domain.
	Pointer to next element in Polygon_Face_List.
Face_Pointer	Pointer to first node in Face_Vertex_list.
	Face normal.
	Sign Pointer of first domain, Domain_Ptr1.
	Sign Pointer of second domain, Domain_Ptr2.
	Domain pointers are negative, if face normal is pointing into the domain,
	hence Domain_Ptr1*Domain_Ptr2$\leq$0.
Face_Vertex_List	Local node number
	Global node number
	Face_Pointer
	Successor entry in Face_Vertex_List.
	Predecessor entry in Face_Vertex_List.
	Face_Vertex_List pointer of adjacent face clockwise.
	Face_Vertex_List pointer of adjacent face anticlockwise.
	Edge angle.

The majority of the above data, is stored to enable efficient traversal of polyhedral domains. Therefore, the above data structure enables the acquisition of any relationship between a geometry element with any associated item of the polyhedral domain. e.g given a node of face, just by traversing the data structure, it is possible to find all polygons in which it is contained.

Figure 7.2.2 illustrates that for an interface polygon A, there can exist two adjacent faces C and B. Therefore, Face_Vertex_List has two adjacent pointers, which is sufficient to cover all valid boundary models such as Figures 7.2.3 and 7.2.4. Figure 7.2.5 illustrates a non-manifold boundary model, which is also applicable for use in this data structure.

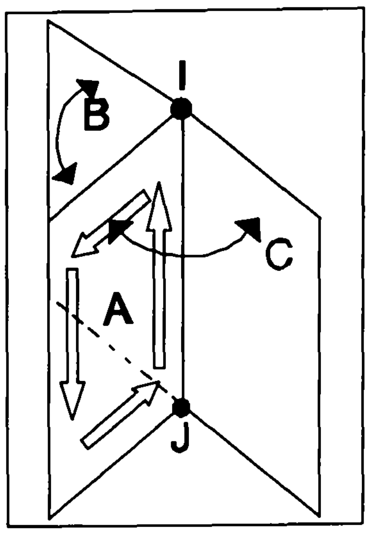


Figure 7.2.2

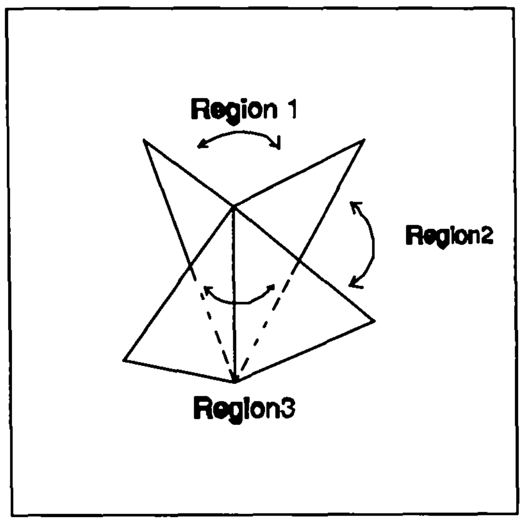


Figure 7.2.3

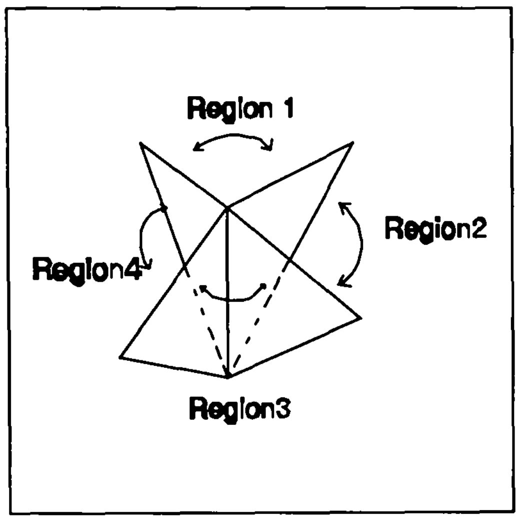


Figure 7.2.4

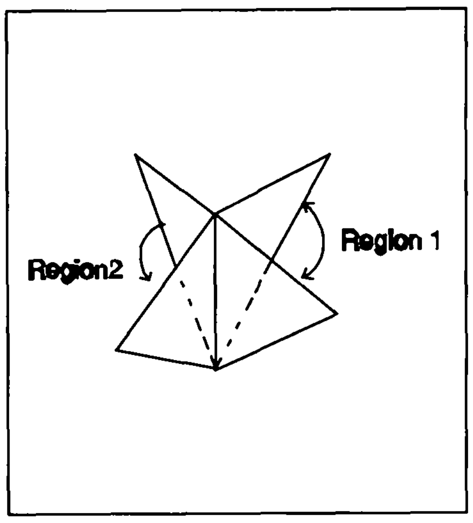


Figure 7.2.5

Another feature of this data structure is that only one face angle has to be stored with each vertex entry. The angle between two adjacent faces is stored in the entry which satisfies the condition $(J-I)\alpha>0$, where J,I are the global vertex numbers of the common edge and α the sign to indicate the direction of the face normal. α is negative if the face normal points into the domain; otherwise it is positive. Therefore, given a Face_vertex entry, to find the angle on the edge of this node J and its successor I, test the condition $(J-I)\alpha$. If this is positive the face angle is stored in this Face_Vertex_List entry else it is stored in the adjacent polygon face entry. Therefore, the angle between each face is only stored once, which optimizes memory usage and allows for the use of a fast edge angle testing routines.

7.3 Generation of cut planes.

The generation of a cut interface is a vital part of any bisection based method of triangulation. Sub-dividing geometry is a highly complex task, therefore in the literature there exist a number of techniques, see [Man88][Cha84][BaD89][BaD92][ChP92] amongst others. These techniques, rely on finding all the intersections of the cut plane with the faces of the polyhedral domain first. These approaches [Man88][Cha84][BaD92] [BaD89][ChP92], were found to be inappropriate for the applications in the mesh generator, due to the generation of acute angles. Therefore, techniques, based on the above methods were developed, namely the *Polyhedral cut face algorithm* and *Edge following bisection.*

However, the above two methods, *Polyhedral cut face algorithm* and *Edge following bisection*, were not readily extendable to multi-connected geometries. Therefore the methods, *Polyhedral splitting* and *Contour polyhedral splitting* algorithms, were derived from the previous two techniques to handle the special case of multi-connected regions. This section will outline these four bisection methods, which are used within the mesh generation code. The selection of the cut plane is described in the previous Chapter 6 section 4.

7.3.1 Polyhedral cut face.

This section will outline the primary bisection technique which generates a cut face polygon C_f in a simply connected domain. The method traces out a cut polygon given an initial edge e_0 and the plane of bisection P_b. This techniques differs from other methods, such as Chazelle [Cha84], in which all intersections between the bisection plane and the polyhedral domain are found first, which may be multi-connected.

The initial edge e_0 with vertices v_0 and v_1 will form one side of the interface polygon. The algorithm then proceeds to walk around the polyhedral domain, in a anticlockwise direction keeping the region to the left, generating new edges one at a time. This approach allows the rejection of the cut polygon at any time, if it generates any *small angles* or is *too near* to existing vertices. Figure 7.3.1 illustrates a cut polygon that divides a domain into two separate parts. However, Figure 7.3.2 is the more common case where the cut polygon introduces an internal or double occurring face into the polyhedral.

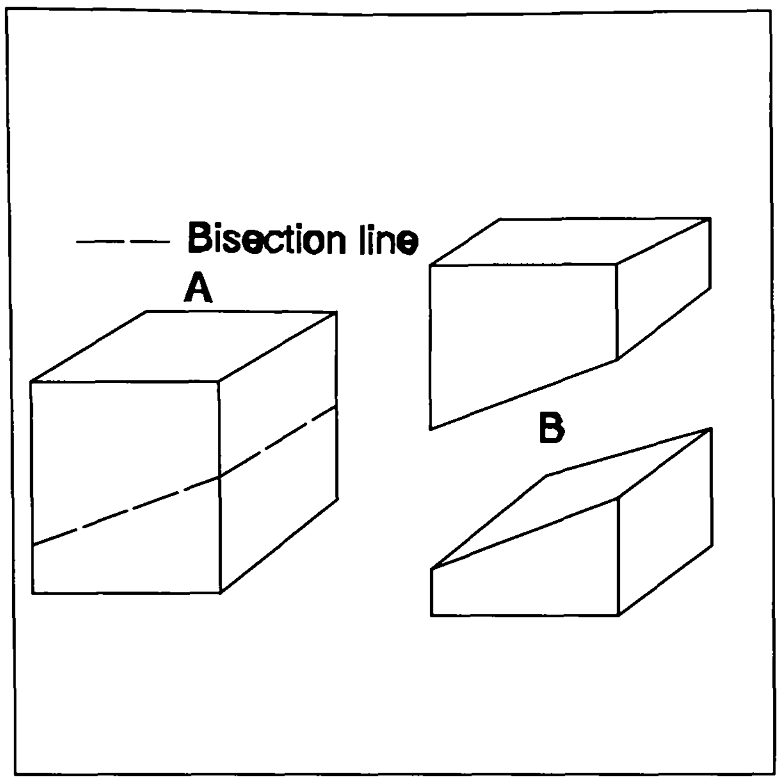


Figure 7.3.1 Bisection of a domain.

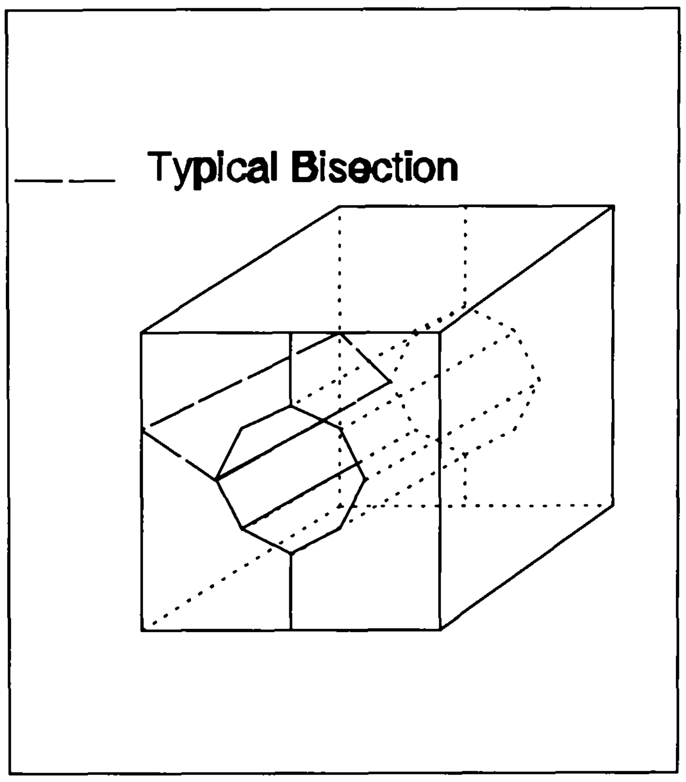


Figure 7.3.2 Internal bisection polygon.

In this technique it is vital to have an efficient method of establishing which element should be visited next. If the current element is J and it is assumed that all the polygons' nodes are in a counterclockwise order, Figure 7.3.3A, when viewed from

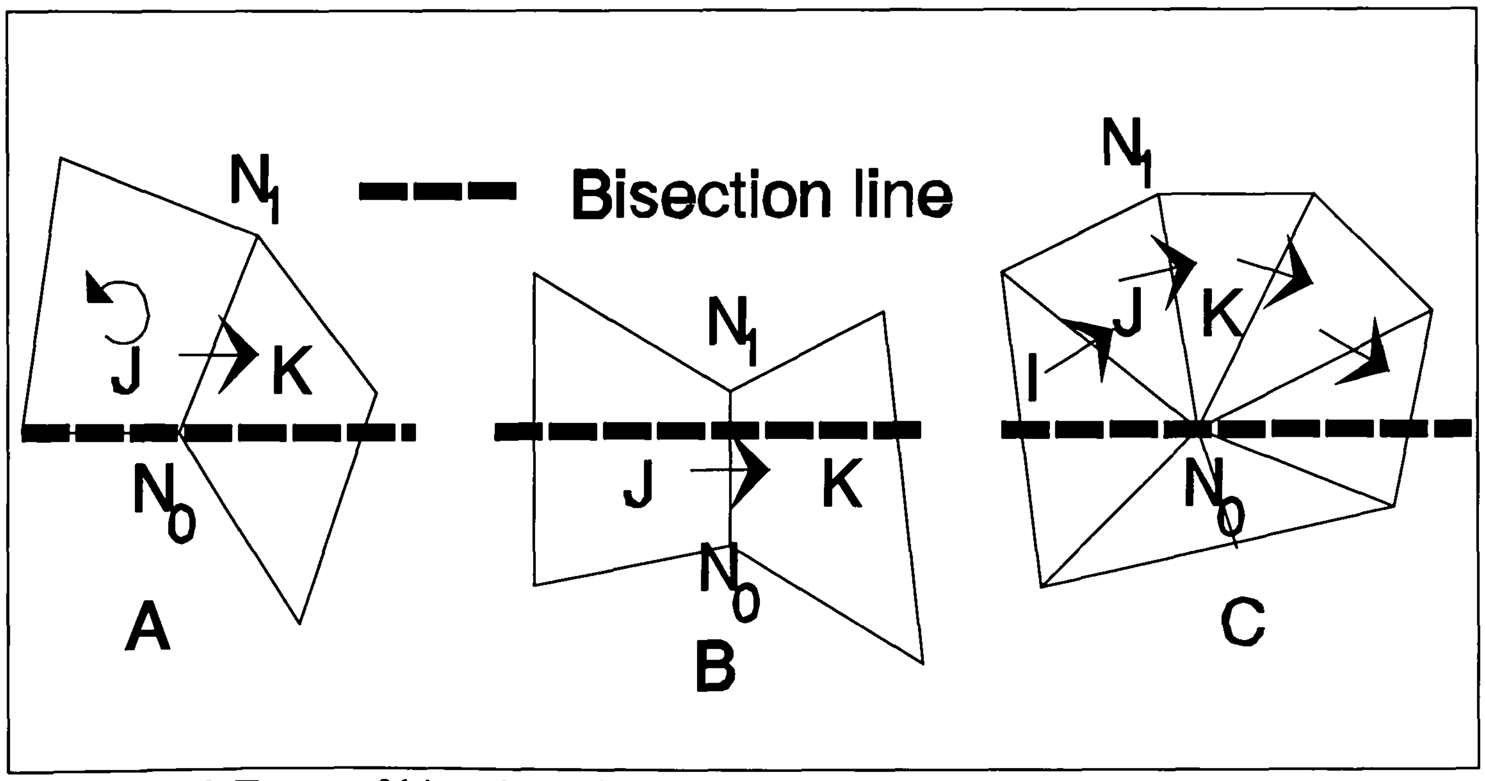


Figure 7.3.3 Types of bisection adjacency.

outside the domain. Then there exists an edge of *polygon J*, with vertices N_0 and N_1, such that N_1 is above and N_0 is either below or on the bisection plane. Therefore, the next element to traverse is adjacent to the edge N_0 to N_1 of polygon J. Figure 7.3.3 shows the three cases when the element J has a (A) touching edge, (B) bisecting edge and (C) vertex on the bisection plane. However, there are two degenerate cases in the walking algorithm. These are when N_0 and N_1 are both on the bisection plane, or N_1 is below. In this situation the cut polygon forms degenerate angles with the polyhedral and is rejected.

It is also possible for the bisection plane to cut a face polygon more than once, Figure 7.3.4. The nodes on the edge which touch or bisect the cut plane that is closest to the previous bisected edge is selected for N_0 and N_1. Polygons which bisect the cut plane in n places will be visited $n/2$ times during the algorithm's walk through the polyhedral domain.

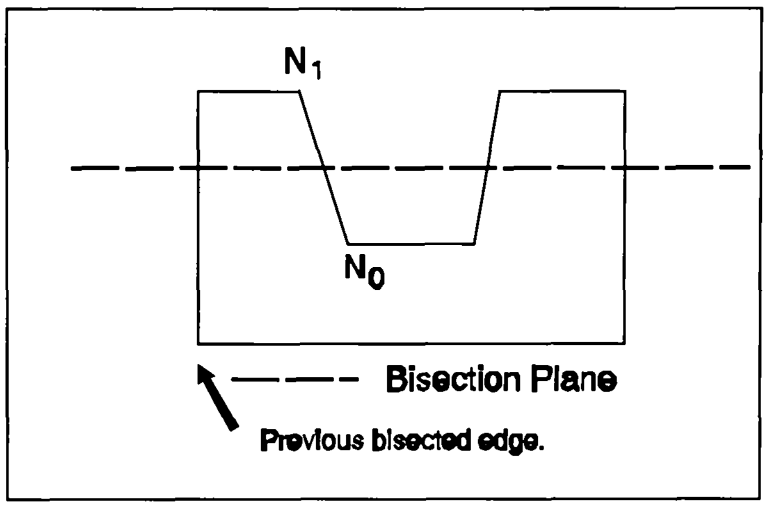


Figure 7.3.4 : Multi intersection.

The routine generates a list of nodes, which will form the cut polygon, by walking through the set of polygon faces that intersect the cut plane Figure 7.3.5. The algorithm walks through each polygon in turn, A to H, in the direction of the arrows generating new nodes on edges that intersects the cut plane. As the algorithm walks through each polygon, the angles generated between this element and the cut plane are calculated. If any of these angles are degenerate, the cut plane is rejected, or else the polygon face is added to a list of visited faces. In the case when a polygon edge is on the cut plane (Figure 7.3.3C) the angles between the cut plane and both these faces are calculated.

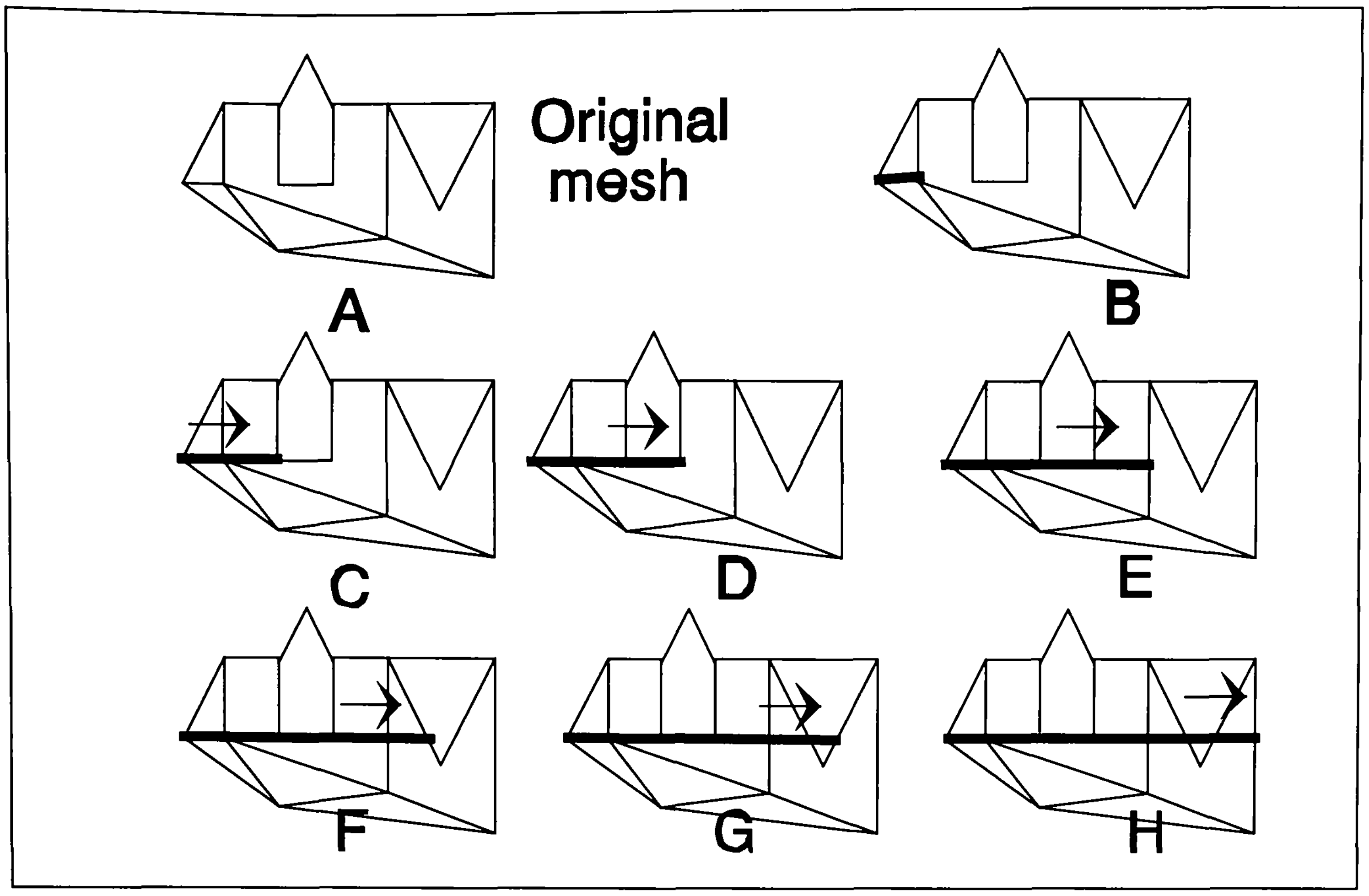


Figure 7.3.5 Bisection walk through the mesh.

Once a complete loop of the polyhedral domain has been completed, the algorithm terminates, and the interior angles of the cut polygon are calculated. If the interior angles are equal to -2π the polygon is external and rejected, see Appendix A4. If the cut polygon is interior to the polyhedral domain, a further check is carried out to ensure that it does not intersect or pass too close to any face polygons.

If the cut polygon is accepted the faces which intersect the bisection plane are then bisected by the insertion of new edges. The cut polygon is then inserted into the polyhedral domain. The domain is then processed to establish whether it has been bisected into two complete halves and the relevant data structures are updated.

7.3.2 Polyhedral splitting algorithm

The previous algorithm only works on polyhedral domains which have no sub-polyhedral regions. These occur when there is a hole or sub-domain contained in a polyhedral model Figure 7.3.6A. Figure 7.3.6B illustrates a typical bisection generated by this algorithm.

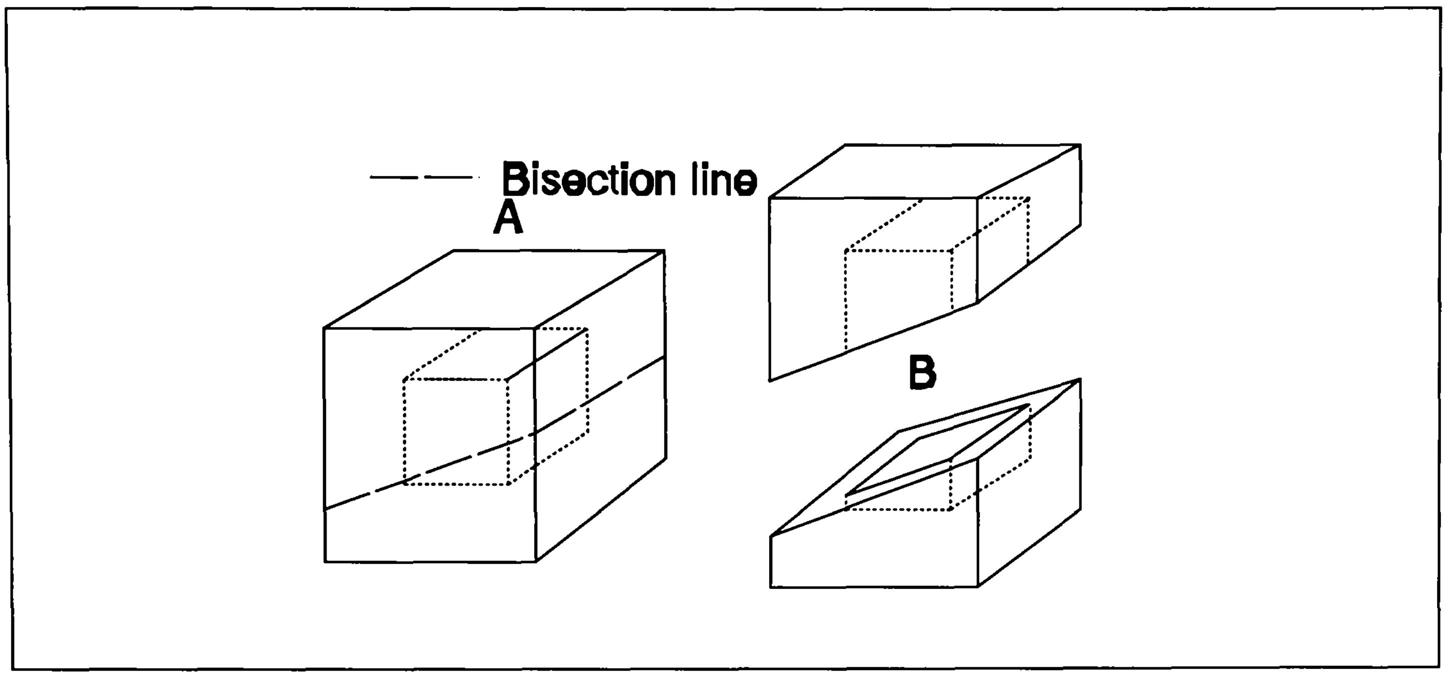


Figure 7.3.6 Bisection of a polyhedral with sub-polyhedral domain.

The algorithm is based on a standard bisection method [Man88] that requires a bisection plane P_b and polyhedral domain S. The method can be described as follows:

(1) Label all nodes as either on, above or below split plane P_b. Reject the cut polygon if there are any nodes that are *too near* the bisection plane which do not lie on it.

(2) Sub-divide all polygons that bisect the cut plane and update polyhedral data structure. Relabel bisected segments of *on polygons* as above or below.
For all polygons which lie on the bisection plane, label as *below* if their surface normal is in the same direction as the bisection plane, or else label as *above*.
For any polygons that are not co-planar to the bisection plane, however touch it along an edge, Figure 7.3.7 . Test to see if the adjacent element on the touching edge is on the same side of the bisection plane, if it is reject bisection plane.

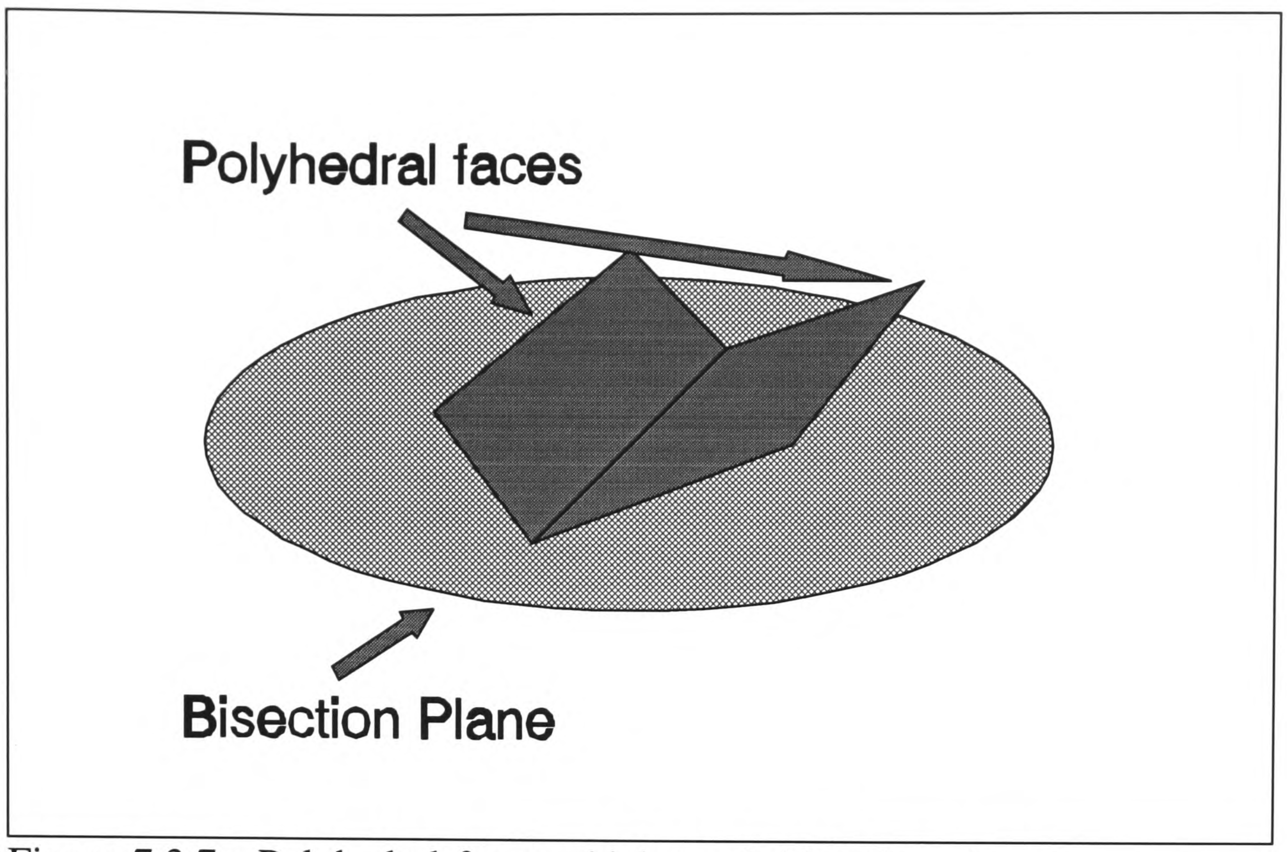


Figure 7.3.7 : Polyhedral faces which touch bisection plane.

(3) Generate list of edges which lie on the bisection plane. These edges are found by searching the adjacency list of the polygons that are label as *down*. An edge which lies on the bisection plane, is any polygon's side which has an adjacent element which is above the cut plane P_b.

(4) Find lists of vertices that form closed loops, these are polygon faces. Reject cut plane if any degenerate polygons found.

(5) Classify faces as outer or inner polygon regions. i.e find hole polygons. This process is simplify by the fact that there are no interface edges or sub-polygon regions except hole polygons.

(6) Introduce cut edges to remove inner polygon regions.

(7) Insert new interface polygons into polyhedral domain.

(8) Insert new polyhedral domains into polyhedral data structure.

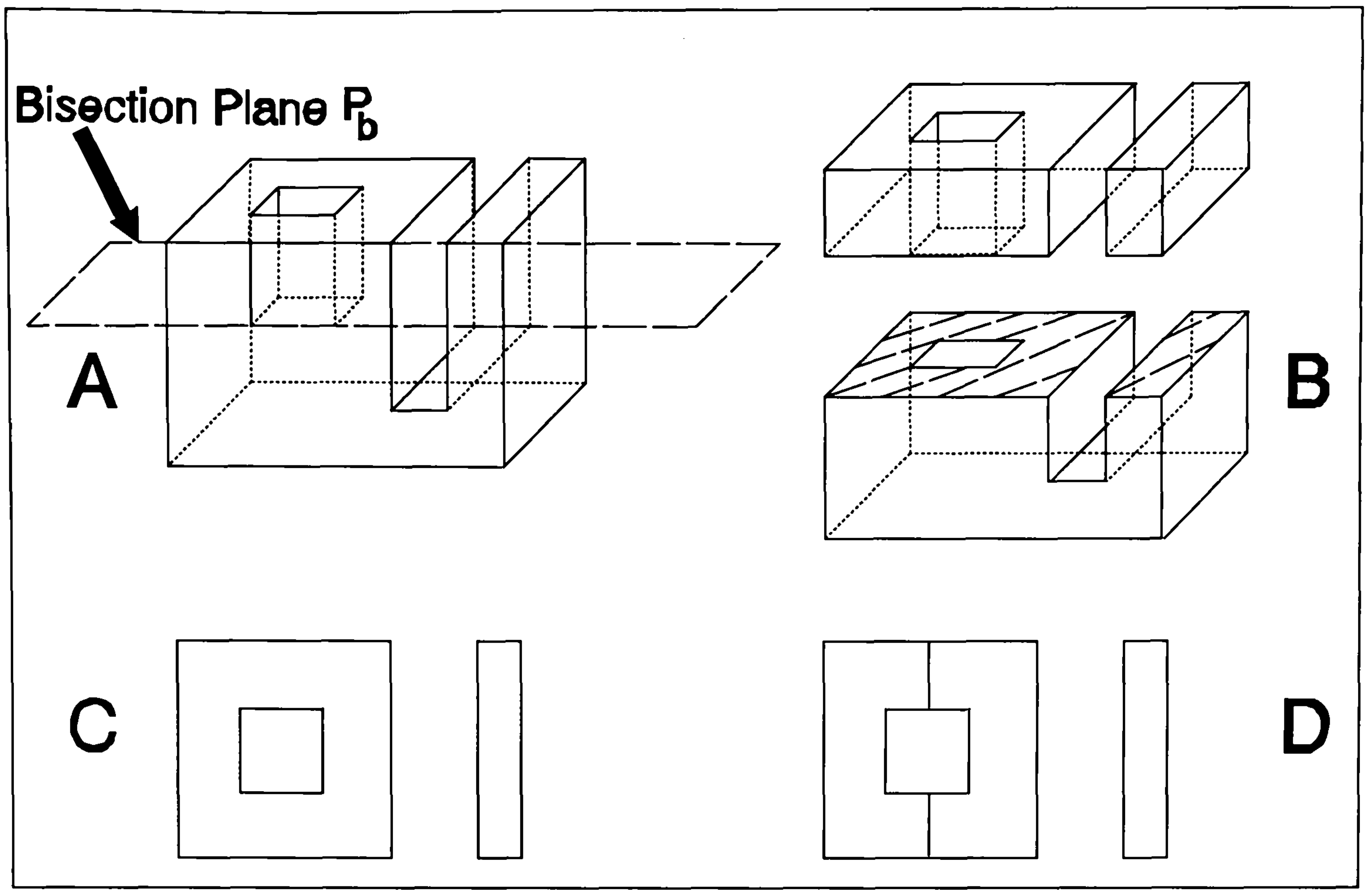


Figure 7.3.8 Example bisection of a domain.

Figure 7.3.8A represents a domain which is bisected along the cut plane illustrated and Figure 7.3.8B shows the resulting bisection. Figure 7.3.8C shows the list of edges, which lie on the bisection that form a set of three polygons of which one is a hole region. The last Figure 7.3.8D, show the polygons used to generate the cut interface, the *"hole polygon"* has been removed by the introduction of two new *cut edges* (See Chapter 4 section 4.6.4).

7.3.3 Edge following bisection method.

The previous bisection methods always introduce new nodes into the polyhedral model. However, when the surface is complex, as in the case of a surface mesh, this is often undesirable. Since introducing a large number of nodes in a complicated region can result in an over refined mesh. The basic idea is illustrated in Figure 7.3.9 where a domain (A) is cut in half along the edges of the polygons that results in the bisection (B).

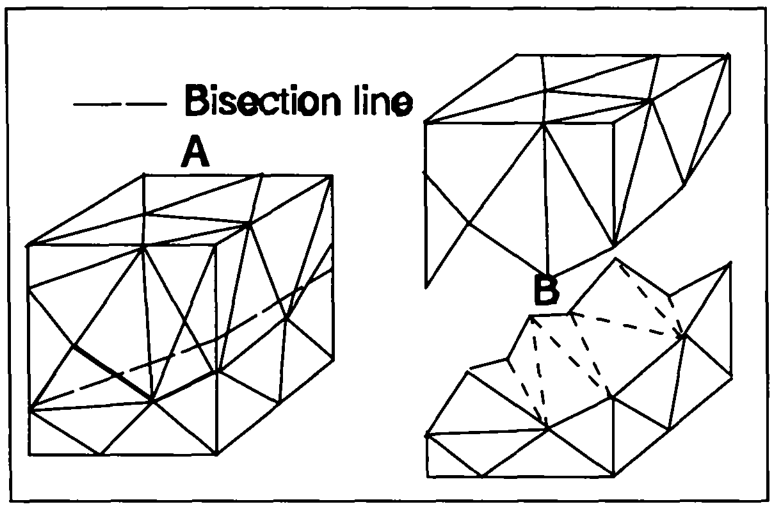


Figure 7.3.9 Bisection walk which cuts a domain into two independent regions.

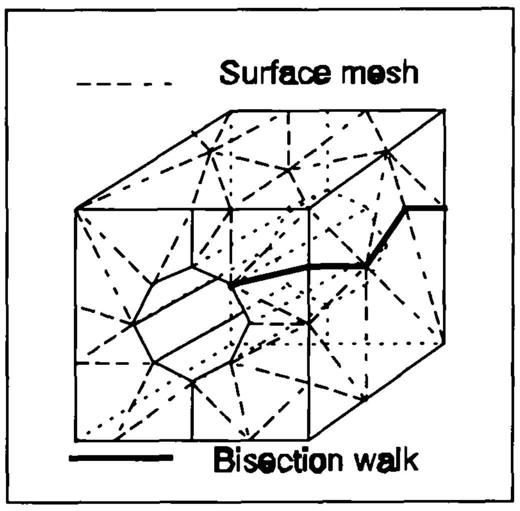


Figure 7.3.10: Internal bisection polygons generation

The method traces out a cut interface given an initial edge e_0 and a bisection plane P_b in a similar way to the method described in section 7.3.1. From the initial edge e_0 the method proceeds to walk around the region in a anticlockwise direction jumping from node to node of each polygon face.

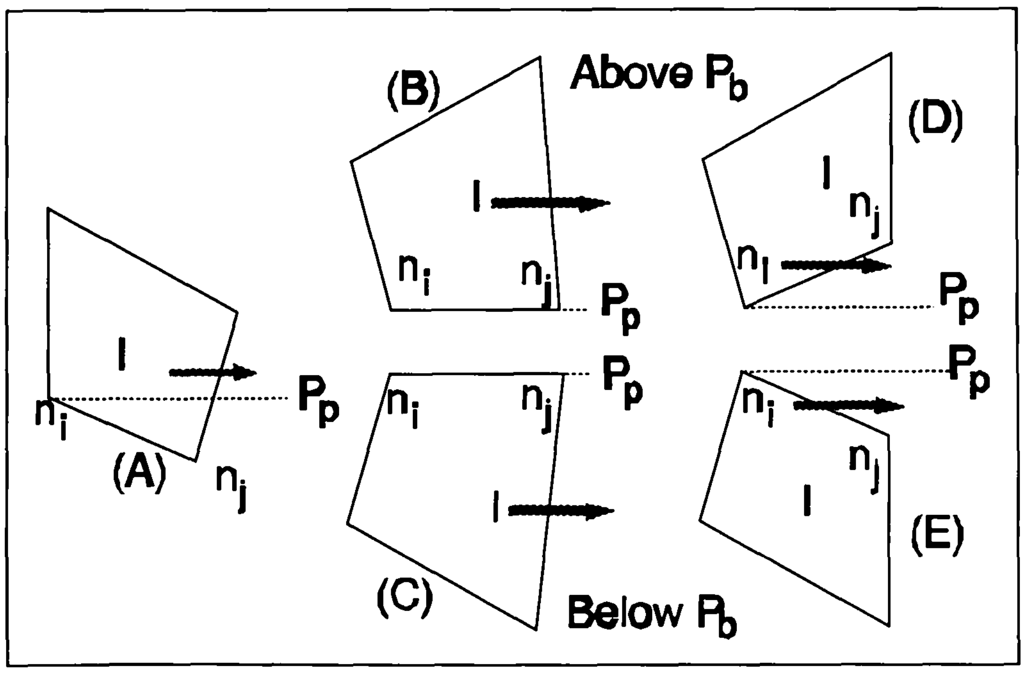


Figure 7.3.11 : Types of bisection elements.

If the current polygon is I and we are at the node n_i with local node number L_i, the next node n_j and polygon J have to be established. This is achieved by generation a plane P_p through the node n_j and parallel to the plane P_b. If P_p bisects an edge of polygon I (Figure 7.3.11A), then n_j is the node closest to the plane P_b on this edge and polygon J is the adjacent face on that side. However, if P_p is parallel to the edge with node n_i of polygon I, n_j is the next node from n_i keeping the domain to the left. Figure 7.3.11B and C shows examples of polygons below and above the bisection plane P_b.

The third case is when the plane P_p does not intersect any edges of polygon I. Node n_j is set to n_i and the following test is carried out:

```
If polygon I above $P_b$ then:
    Polygon J is the element adjacent to the edge $\{L_i,L_{i+1}\}$.

Else if polygon I is below $P_p$ then:
    Polygon J is adjacent to the edge $\{L_{i-1}, L_i\}$.
```

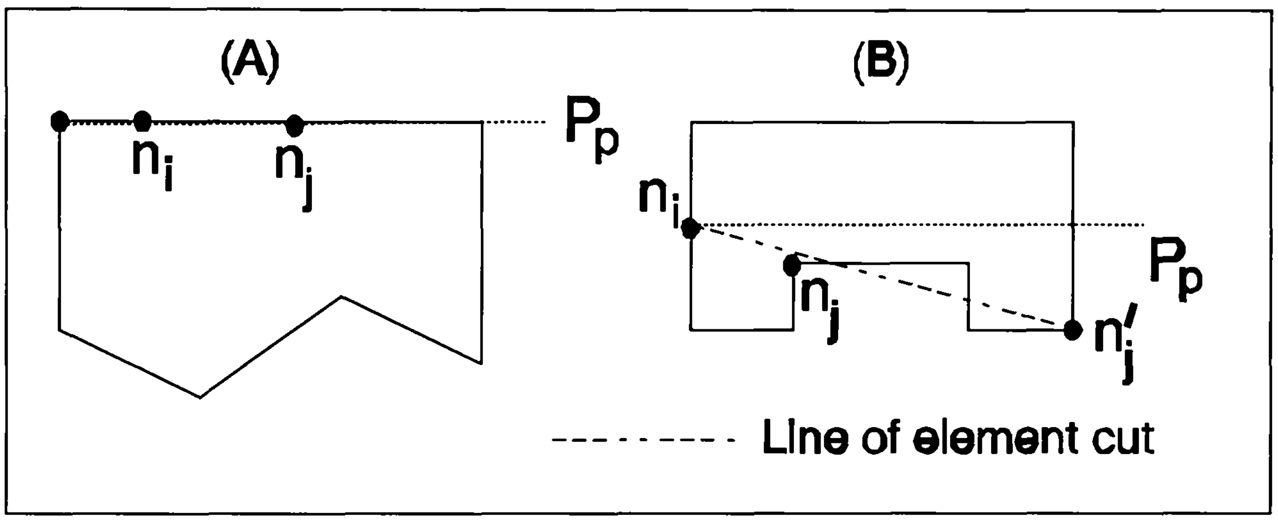


Figure 7.3.12 Special cases.

There are two special cases illustrated in Figure 7.3.12. In case (A) the local bisection plane is parallel to several line segments of polygon I. Therefore, if the angle between the line segments, which are adjacent on the node n_j, are 180 degrees, the current polygon J becomes I. Case (B) is when the line from n_i to node n_j' bisects more than one edge of the polygon I. In this case, the edge that is first bisected by the *line of element cut*, Figure 7.3.12B, has the node which is closest to this line selected as n_j.

This method walks around the domain using the above rules to generate a list of interface points I_p with elements adjacent to each interface line segment. The routine will reject a possible cut plane if any element adjacent to an interface line segment forms an acute angle with the bisection plane P_b. The bisection walk is terminated when a closed loop is found, this is when the node n_j is a member of the set I_p and not equal to n_i. The list I_p is then processed to remove any nodes, which are not a member of the closed loop, since the last n_j entry is not necessarily equal to the first entry in the list I_p.

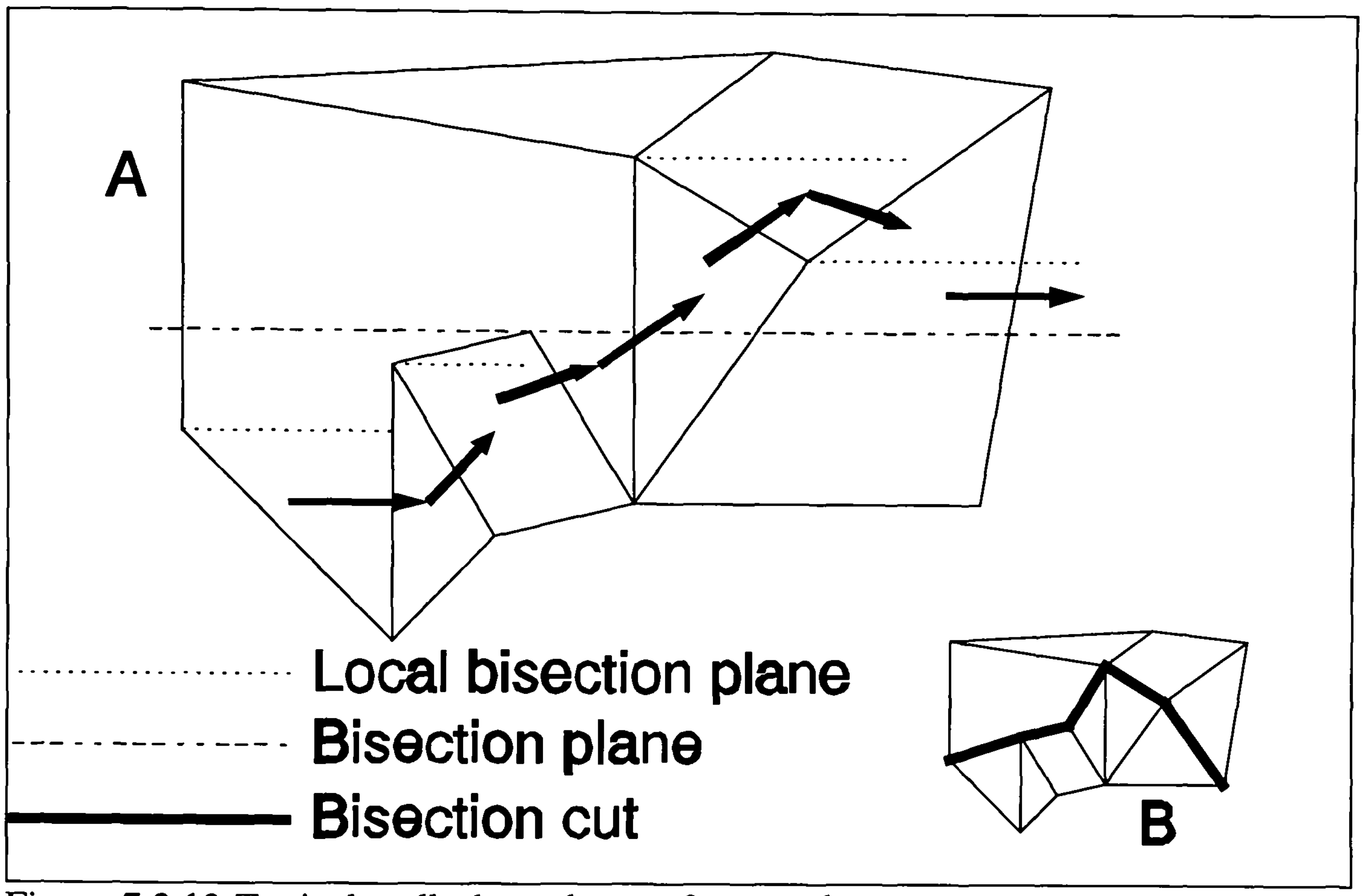


Figure 7.3.13 Typical walk through a surface mesh.

The above Figure 7.3.13 shows a typical walk through a surface grid, in which the algorithm follows the direction of the arrows. Figure 7.3.13B illustrates the bisection edge generated using this method.

The list of bisection nodes are non-planar, therefore a set of polygon interface elements are generated. The method used is to project the set of interface nodes onto the bisection plane as illustrated in Figures 7.3.14 and 7.3.15.

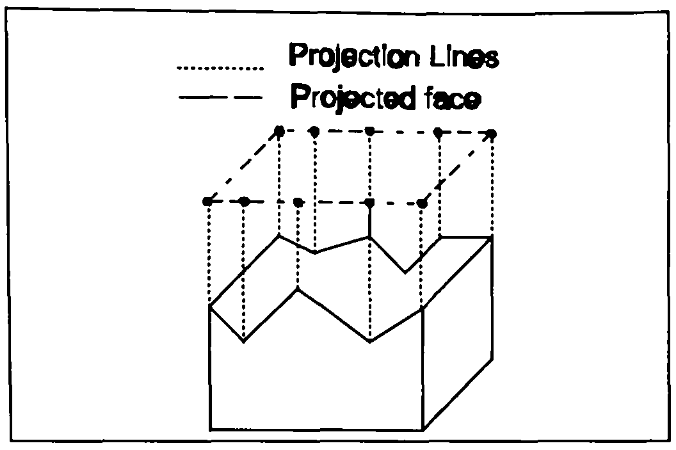


Figure 7.3.14 : Projection of cut face.

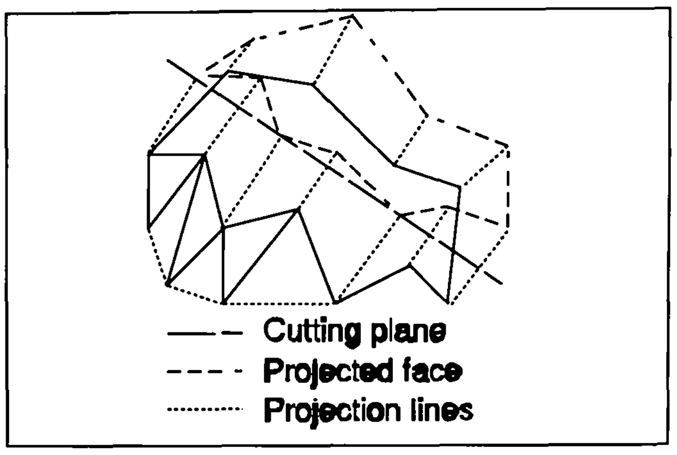


Figure 7.3.15:Projection of cut surface.

Two planes P_u and P_l are then generated that are parallel to the bisection plane P_b and enclose the set of interface nodes. P_u is of a distance $\max(d_1,d_2....d_n)+\psi$ from the bisection plane P_b. The parameters d_l to d_n are the signed distance of the interface nodes from the plane P_b, and Ψ is an additional spacing function. Ψ is defined as the minimum acceptable relative distance between cut plane and vertices multiplied by the average length of an edge on the surface of the domain. The relative acceptable distance is a constant with a value between 0.2 and 0.6 depending on quality of bisection required, in most cases a value of 0.4 is adequate. The plane P_l is defined as the distance $\min(d_1,d_2....d_n)-\psi$ from the bisection plane P_b.The term *bandwidth* of bisection is also defined as the distance between the planes P_u and P_l.

A polygon region is then generated with a thickness between the two planes P_u and P_l, with an outer boundary defined by the set of projected interface points. A test is then carried out on all nodes within the polyhedral domain that are not directly connected to an interface node. If any of these nodes are found to be contained within this region the bisection plane is rejected.

The projection is then rotated onto the X-Y plane where it is meshed using a standard two dimensional mesh generator. The connectivity of this mesh is then used to stitch the nodes together in 3D space, which form interface elements. The angle between all interface elements and their outer boundary are calculated and if any are acute the bisection is rejected.

Unfortunately the rule of adding no additional boundary nodes resulted in the generation of highly angled interface surfaces, which were often rejected by the mesh generator. Therefore, the option of generating a possible mid-node n_j' was introduced. The node n_j' is inserted into the mesh if the line $\{n_i,n_j\}$ makes an unacceptable angle with the bisection plane P_b and the line $\{n_i,n_j'\}$ is an improvement.

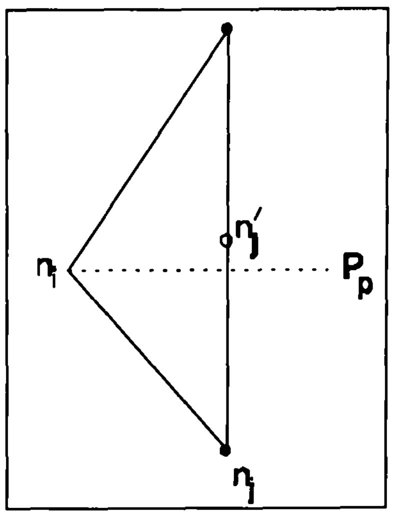


Figure 7.3.16

The mid-node n_j' is defined at the centre of the edge that is bisected by the local cutting plane P_p, Figure 7.3.16. Figure 7.3.17 shows a typical bisection walk using optional nodal placement to avoid highly uneven interface surfaces.

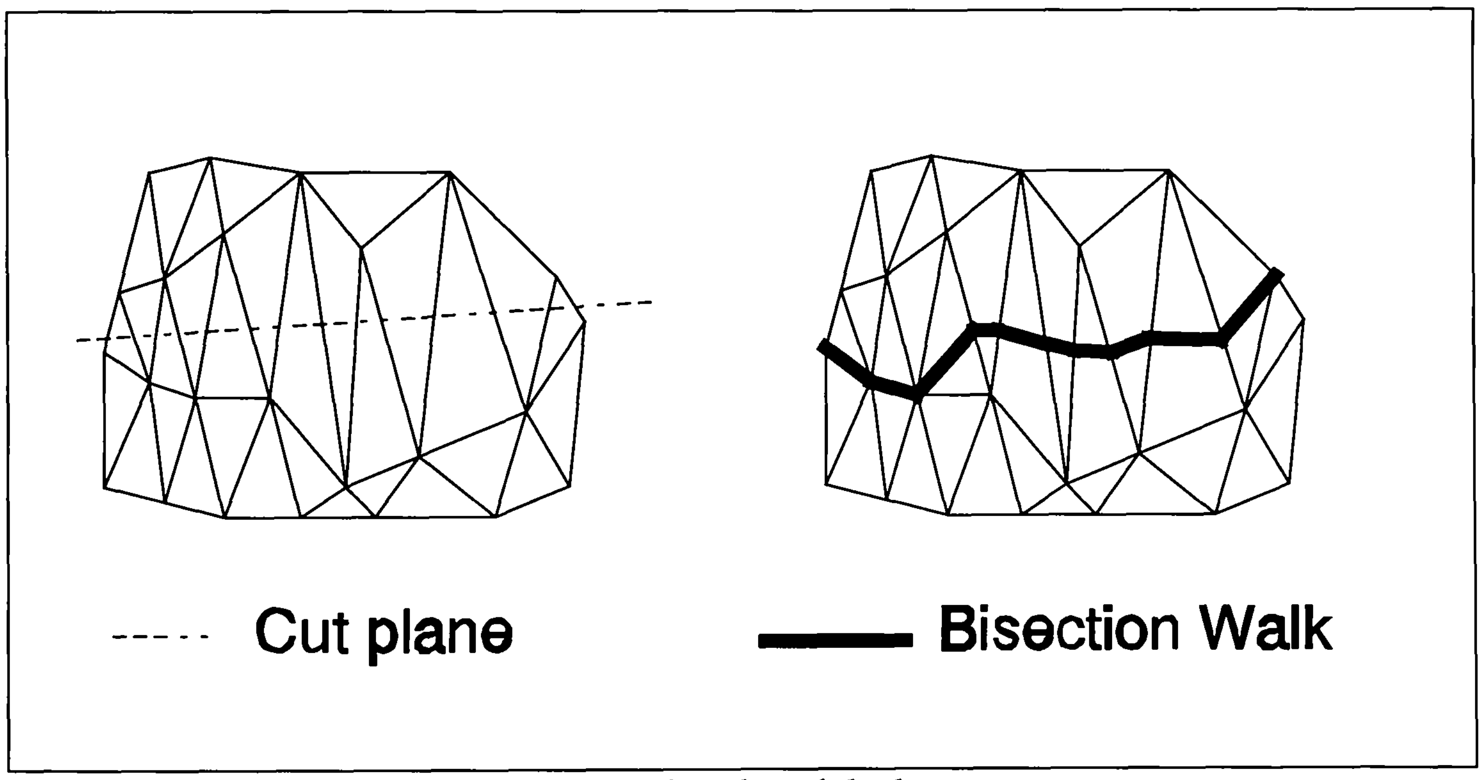


Figure 7.3.17 Bisection walk with optional nodal placement.

7.3.4 Contour polyhedral splitting algorithm

Once again the previous algorithm only works on non-multi connected regions. Therefore, the need for the following algorithm, which divides regions that have sub-polyhedra domains. Figure 7.3.18 shows a cube with a hollow section contained within.

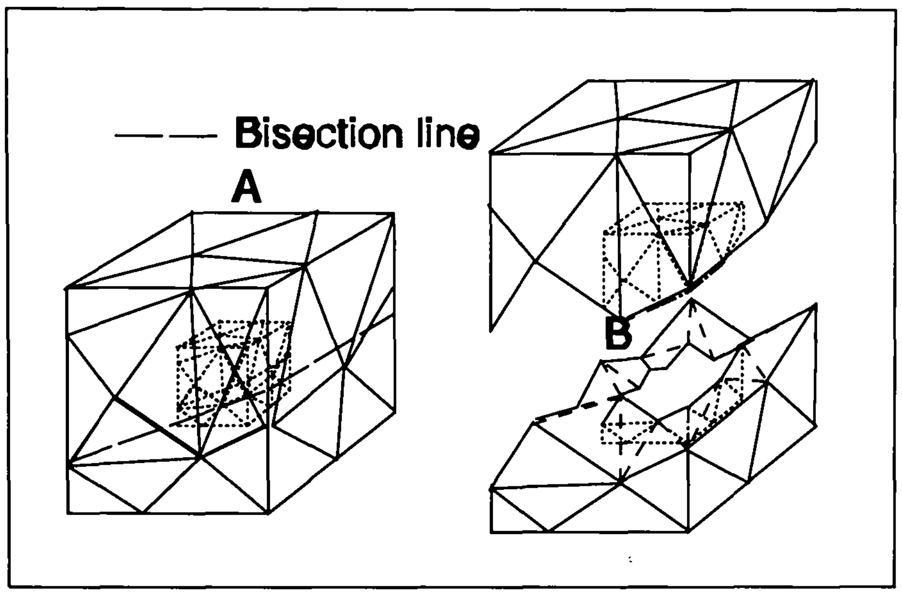


Figure 7.3.18 Bisection of a complex domain.

The first five steps in this algorithm are the same as the polyhedral splitting algorithm of section 7.3.2. Once the set of interface polygon loops have been established the method then proceeds to search each loop for a suitable starting edge. This is done by taking the set of polygon faces that lie on a loop, and then searching them to find the closest edge which is within an acceptable angle range. Once an edge is found for each loop a bisection walk is carried out using the method described in the previous section 7.3.3. Each loop is checked with its own set of P_u and P_l planes as described in section 7.3.3. However, the set of interface loops are projected and rotated as a group onto the XY plane, where they are meshed using a 2D mesh generator. The connectivity of the mesh is then used to stitch the points in 3D space which form interface polygons. The angle between all interface elements and their outer boundary are then calculated and if any angle is acute the bisection is rejected.

7.4 Nodal placement using 3D convex domain shrinking.

The nodal placement technique implemented in the mesh generator is akin to the method of *Normal offsetting* by Johnston and Sullivan [JoS93]. The Johnston and Sullivan technique could be applied to domains of arbitrary shape. However, in this research, it was deemed necessary to restrict this technique to convex domains, for implementation ease and computational efficiency.

A convex region is shrunk by a nodal spacing factor N_p which is described in Chapter 6 section 11. This is achieved by moving each polygon face P_i of the region in by a factor N_p along its surface normal, Figure 7.4.1, to define a new face s_i.
The set of faces s_0,s_1..s_k, form a set of half-spaces. The intersections of these half-spaces, if N_p is sufficiently small, defines a region *H* which is the shrunken polyhedral domain.

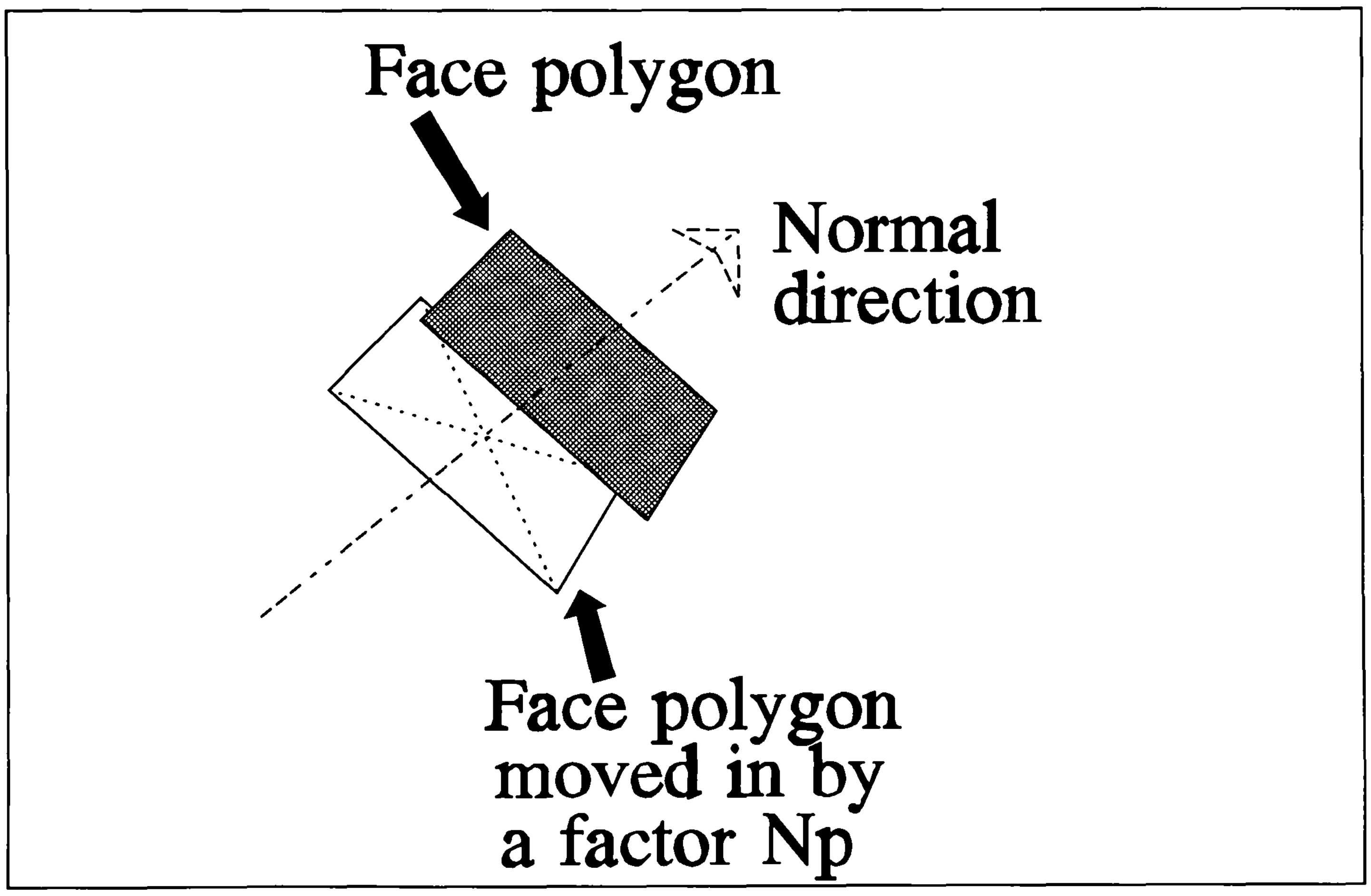


Figure 7.4.1 : Face polygon moved inwards by a factor N_p.

In the mesh generator all the polyhedral faces of the model are divided initial into a set of convex faces, to simplify the algorithm. Therefore, the convex shrinking algorithm is as follows.

```
(1)  Copy all faces of convex polyhedral domain into the shrunken polyhedral data
     structure S.
(2)  For each face I in the data structure S do:
(3)      Move the plane $P_i$ of the face I, inwards along its normal by a factor $N_p$.
(4)      Find the intersection of the plane $P_i$ with other polygons in the shrunken
         polyhedral data structure S. Generate new faces and update data
         structure S.
```

The routine which finds the intersection of face *I* with polygons in the shrunken polyhedral *S*, and updates *S*; is as follows:-

```
(1)  Let $P_i$ be the plane defined by the face I.
(2)  Determine if the intersection is empty or the entire polygon.
         (i)    Label all vertices as above or below plane $P_i$
                (Any vertices which lay on the plane $P_i$ are label as above)
                (Vertices are classified as on if their distance from the bisection
                plane is less than $N_p\varepsilon$, where $\varepsilon$ is machine tolerance)
         (ii)   If all vertices are above the plane $P_i$, the intersection is empty.

(3)  If intersection is not empty then:
         (i)    Insert new nodes where edges bisection the plane $P_i$.
         (ii)   Subdivide faces which have edges with newly inserted nodes,
         (iii)  Remove all edges which are above plane $P_i$,
         (vi)   Remove all polyhedra faces with deleted edges.
         (v)    Insert new polyhedral face, made up of the edges formed by the
                newly generated nodes, into the data structure S.
```

Once the shrunken polyhedral domain is generated, sharp angles are then removed to ensure an even nodal spacing. This is achieved by selecting adjacent faces which have an angle Φ, less than a minimal value, 30 degrees [JoS92], Figure 7.4.2A. A plane *P* is then generated, at a distance α into the domain (Figure 7.4.2B) to remove this acute angle, see Figure 7.4.2C. The value of α is set proportional to the angle Φ and nodal spacing H, and is calculated by $H/2tan(\Phi/2)$.

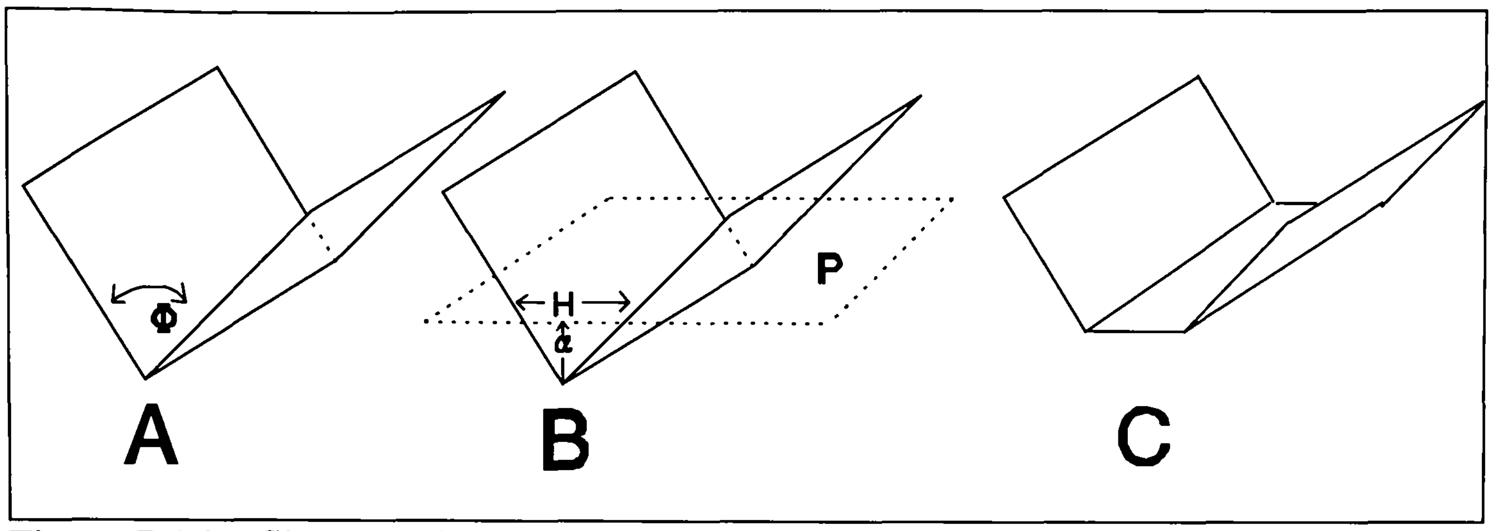


Figure 7.4.2 : Sharp angle removal

Each face of the shrunken polyhedral is then processed to remove narrowness or short edges, see Chapter 4 section 4.6.4. The two dimensional nodal placement algorithm is then applied to generate face and edge nodal points.

The whole process is then repeated, by shrinking this shrunken polyhedral again until a degenerate polyhedral is formed. In the case of a degenerate polyhedral domain a node will be placed at its centroid if the volume of the domain is larger then $(2n+1)V_t$ where n is number of face nodes and V_t the average tetrahedral volume of the mesh. This is done to ensure an even nodal density throughout the domain.

7.5 Three dimensional local transformations (Vertex Swapping).

Three dimensional vertex swapping [Joe89] is in effect a local transformation, since in reality it is the addition and removal of groups of vertices of tetrahedra. Local element transformation is based on the idea that groups of elements that form a convex region can have their internal edges transformed without effecting their external geometry.

The Four transformations which are used :-

(1) 2 to 3 Mapping (Only if 5 points form a convex domain)

Two tetrahedra with vertices {A,B,C,D} and {A,B,C,E} are transformed to three tetrahedra {A,B,D,E},{A,C,D,E} and {B,C,D,E} by the insertion of the edge {E,D}.

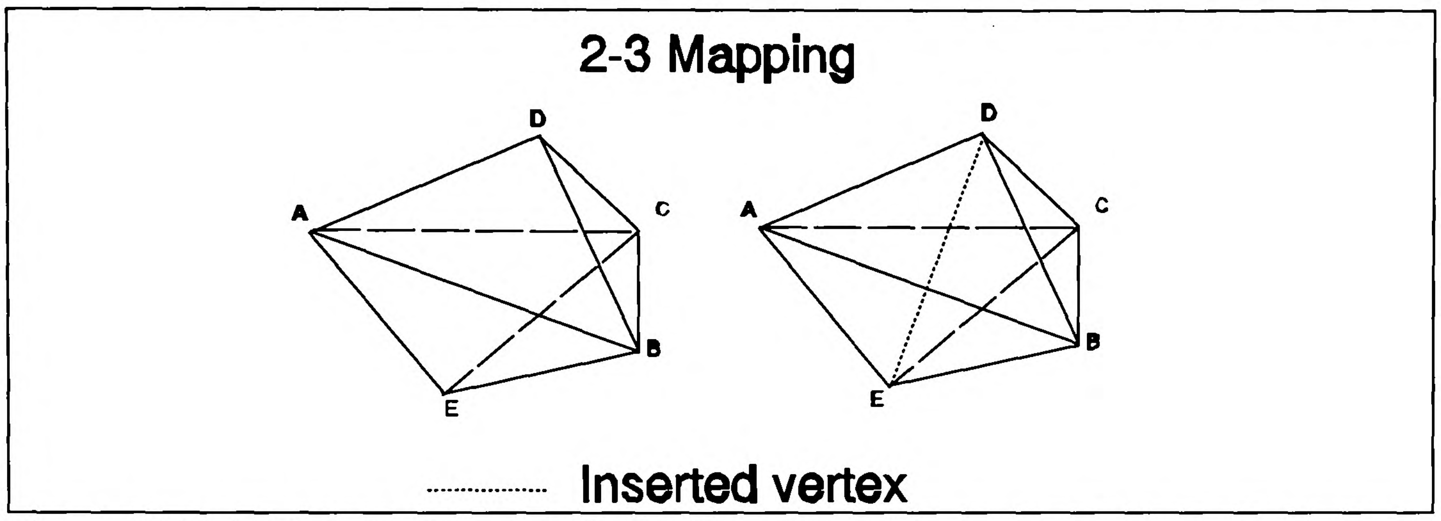


Figure 7.5.1A: Two to three tetrahedra transformation.

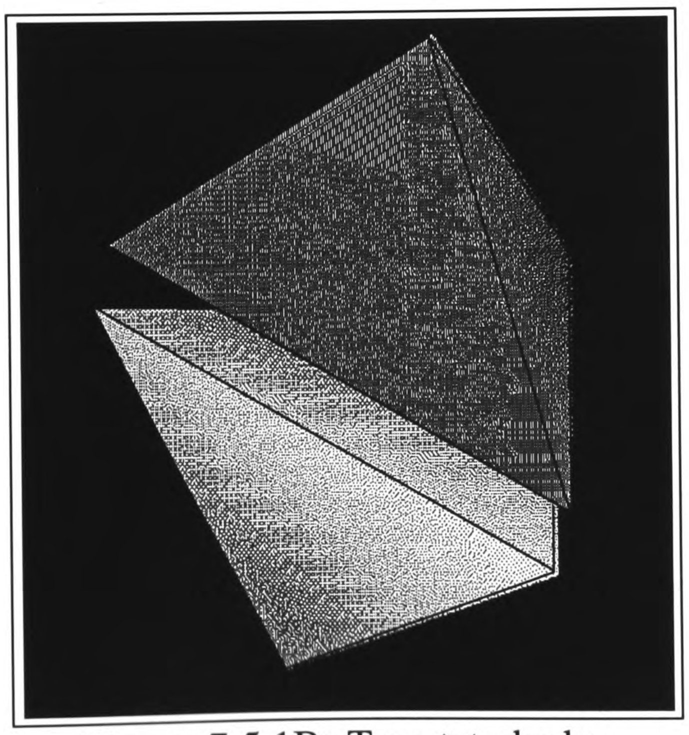


Figure 7.5.1B: Two tetrahedra.

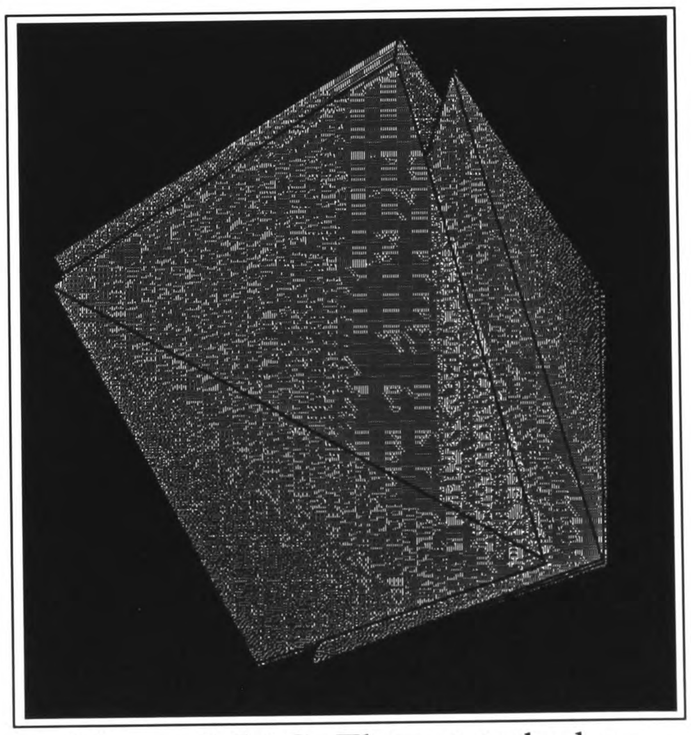


Figure 7.5.1C: Three tetrahedra.

(ii) 3 to 2 Mapping

Three tetrahedra {A,B,D,E},{A,C,D,E} and {B,C,D,E} are transformed into the tetrahedra {A,B,C,D} and {A,B,C,E} by the removal of the edge {D,E}. This is the reversal of the process shown in Figure 7.5.1.

(iii) 2 to 2 Mapping (only if the 4 nodes {A,B,C,D} lie on the boundary)

The tetrahedra {A,B,D,E} and {B,C,D,E} are transformed into tetrahedra {A,B,C,E} and {A,C,D,E} by swapping the edge {BD} to {AC}

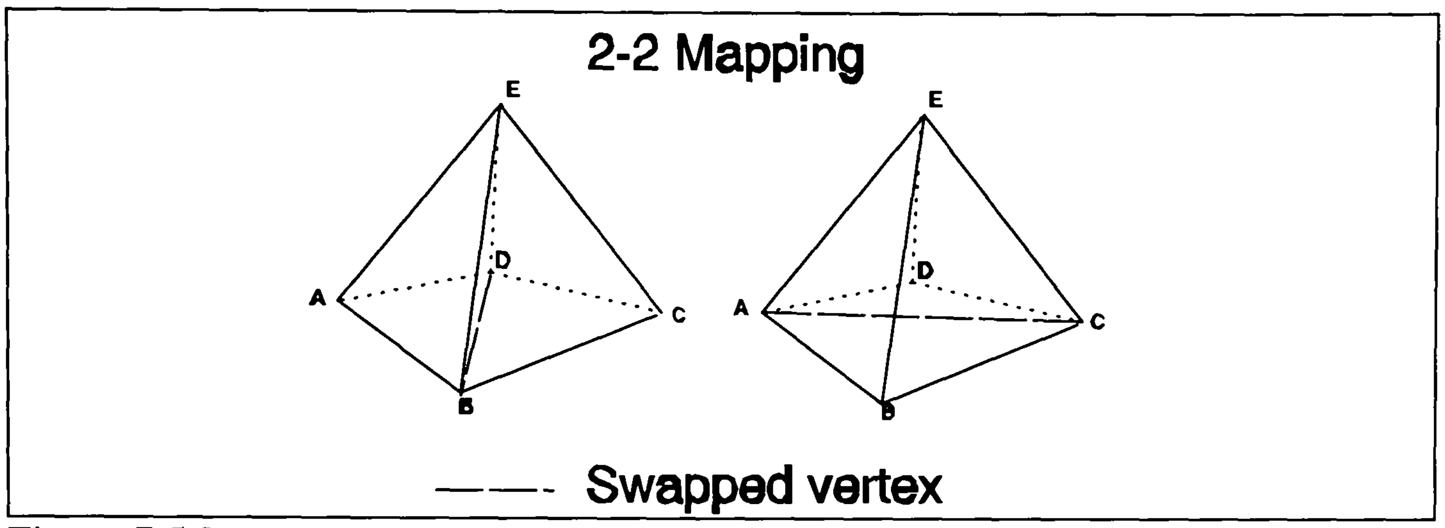


Figure 7.5.2

(iv) 4 to 4 Mapping :-

The tetrahedra {A,B,C,D} {A,B,C,E}, {A,B,F,D} and {A,B,F,E} can be transformed into {C,F,A,D}, {C,F,A,E}, {C,F,B,D} and {C,F,B,E} or {A,C,D,E}, {B,C,D,E}, {A,F,D,E} and {B,F,D,E}. This is achieved by moving the edge {A,B} to {C,F} then to {D,E}.

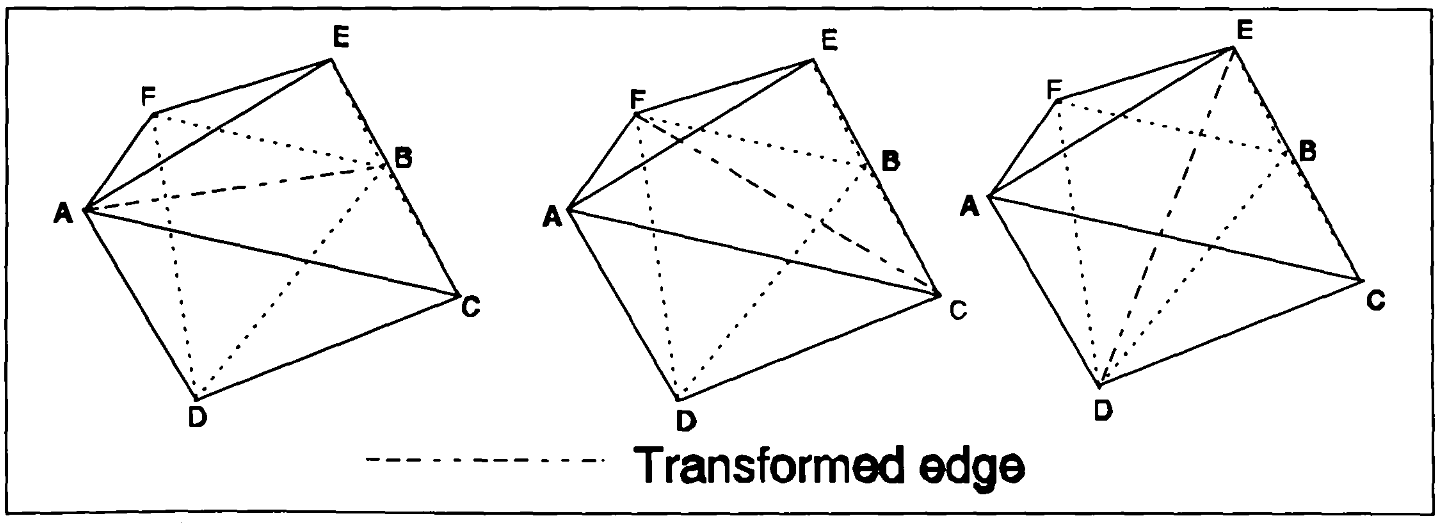


Figure 7.5.3 : Four to four translations.

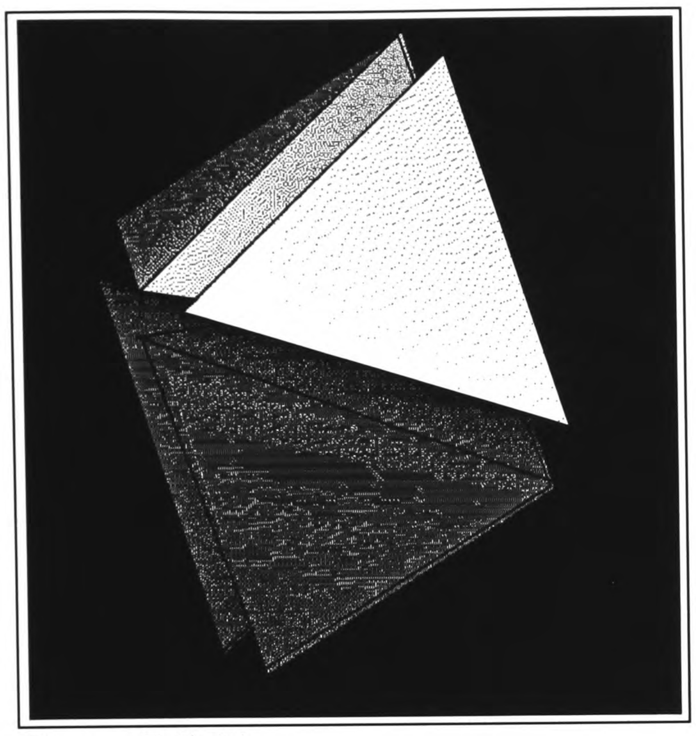


Figure 7.5.4A

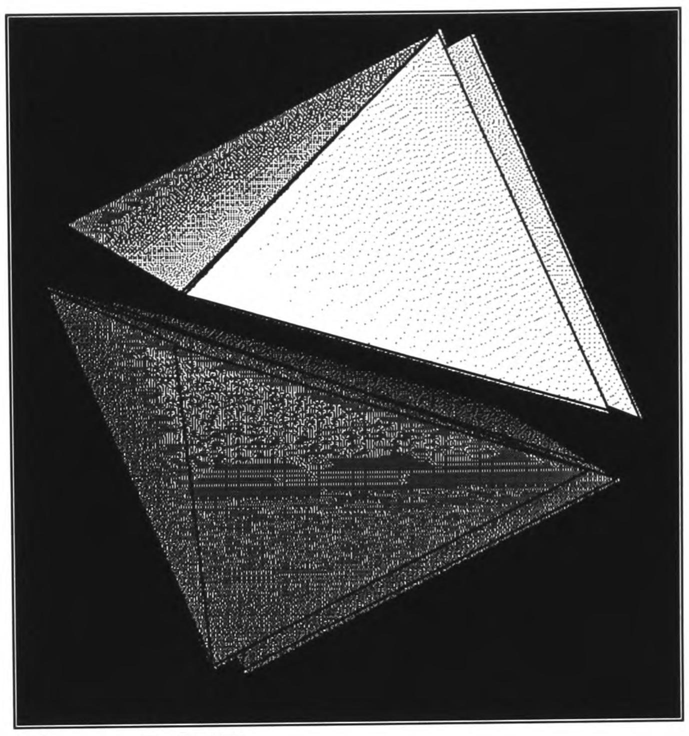


Figure 7.5.4B

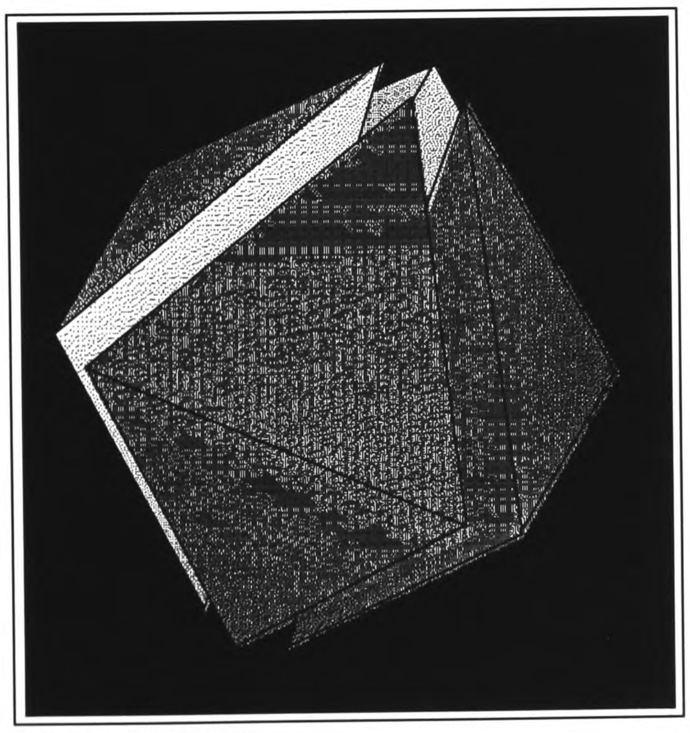


Figure 7.5.4C

Figures 7.5.4A to 7.5.4C show various 4-4 Mapping (In the order given in Figure 7.5.3).

The algorithm first generates a linked list *Q* of all the internal faces within the current mesh. The algorithm then repeats the following set of operations until the list *Q* is exhausted. The procedure is as follows:

(1) remove face *I* from the head of list *Q* with vertices *{a,b,c}*.

(2) Let *Itet* and *Jtet* be the two tetrahedral sharing the face I.
Itet and *Jtet* have vertices *{a,b,c,d}* and *{a,b,c,e}* respectively.

(3) If no four of the five vertices *{a,b,c,d,e}* are coplanar and the edge *{c,d}* intersects the interior of the triangle *{a,b,c}*.
let β = min of the solid angles of the tetrahedra *{a,b,c,d},{a,b,c,e}*
and 6= min of the solid angles of the tetrahedra *{a,b,d,e},{a,c,d,e},{b,c,d,e}*
if $6>\beta$ then apply the 2-3 transformation and add the faces *{a,d,e},{b,d,e}* and *{c,d,e}* to the tail of *Q*.

(4) If no four of the five vertices *{a,b,c,d,e}* are coplanar and either :
  (a) the edge *{a,b}* intersects the interior of the triangle *{c,e,d}*,
  (b) or the edge *{a,c}* intersects *{b,e,d}*,
  (c) or the edge *{b,c}* intersects *{a,e,d}*.
Then relabel the common face as *{a,d,e}* so the that the two
adjacent tetrahedra *Itet* and *Jtet* become *{a,b,d,e}* and *{a,c,d,e}*.
let β= min of the solid angles of the tetrahedra *{a,b,d,e},{a,c,d,e},{b,c,d,e}*
and 6 = min of the solid angles of the tetrahedra *{a,b,c,d},{a,b,c,e}*
if $6>\beta$ then apply the 3-2 transformation and remove the faces *{a,d,e},{b,d,e}* from *Q*.

```
(5)   The four of the five vertices {a,b,c,d,e} are co-planar ,
      relabel the planar face {a,d,b,e} such that the lines {a,b} and {d,e} intersect.
      if {a,b,d} and {a,b,e} are boundary faces then
        let $\beta$= min of the solid angles of the tetrahedra {a,b,c,d},{a,b,c,e}
        and $\delta$ = min of the solid angles of the tetrahedra {a,c,d,e},{b,c,e,d}
        if $\delta>\beta$ then apply the 2-2 transformation and add the faces {a,d,e} and {b,d,e}
        to the tail of Q and remove the faces {a,b,d} and {a,b,e}.
      if there exists two tetrahedra {a,b,d,f} and {a,b,e,f}.
        let $\beta$=min of the solid angles of the tetrahedra {a,b,c,d},{a,b,c,e},{a,b,d,f},{a,b,e,f}
        and $\delta$=min of the solid angles of the tetrahedra{a,c,d,e},{b,c,e,d},{a,c,d,f},{b,c,e,f}
        if $\delta>\beta$ then apply the 4-4 transformation and add the faces {a,d,e} and {b,d,e}
        to the tail of Q and remove the faces {a,b,d} and {a,b,e}.

(6)   If none of the previous mapping can be applied, the face is locally optimized.

(7) Repeat steps 1-6 until Q is empty.
```

The initial order of the faces in the list Q is often arbitrary, however this can be modified, so the routine start at a particular location within the mesh.

7.6 Boundary constrained three dimensional triangulation.

The initial attempt at meshing the convex regions was using boundary constrained Delaunay triangulation, however this was dropped due to the generation of degenerate tetrahedra which resulted in the algorithm failing. An idea by Joe [Joe89][Bak92] [Bak89][BSB92] was taken and modified to use a min-max angle criterion instead of circum-spheres of tetrahedra as a grid generation heuristic. This method of generating grids in convex regions will be described below.

The algorithm requires at least one interior point, since there always exists a boundary constrained triangulation for a convex polyhedral with at least one mesh vertex within its interior [Wör81]. Therefore, if a convex region has no interior points a vertex is generated at its centroid, which the algorithm will later on try to remove by using local transformations.

The vertex swapping algorithm is a min-max solid angle local transformation based method as described in the previous section. The algorithm is modified to check that before a possible transformation is carried out that no external edges are effected.

The initial stage in the generation of a boundary constrained mesh, is the connection of all the boundary triangular faces up to one internal point near the centre of the domain. This triangulation, after begin improved by applying local transformations based on a min-max angle criterion, forms the initial triangulation of the domain. Then the remaining internal points are then inserted into the region one at a time.

On the insertion of a vertex, its location within the previous triangulation is first determined. The vertex can be either on the interior of a tetrahedron, on an edge or a face. Initial tetrahedra are then formed in the grid that include this newly inserted vertex. The mesh is then improved using a local min-max angle criteria, which is initialised in the region where the newly formed tetrahedra were created.

In the case where an internal vertex v was generated for the above algorithm to work, a further set of transformations are applied. These transformations try to remove all faces/edges incident on *v*. Let N_{tet} be the number of tetrahedral with vertex *v*, this is equal to the number of boundary faces. A list *Q* is generated of all faces with the vertex *v*. The algorithm then repeats the following set of operations until N_{tet}=4:

(1) remove face *I* from head of *Q* with vertices *{a,b,v}*.

(2) Let *Itet* and *Jtet* be the two tetrahedra sharing the face I.
Itet and *Jtet* have vertices *{a,b,c,v}* and *{a,b,d,v}* respectively.

(3) If no four of the five vertices *{a,b,c,d,v}* are coplanar and the edge *{c,d}* intersects the interior of the triangle *{a,b,v}*. A 2-3 mapping is applied, the new face *{d,c,v}* is added to the tail of *Q* and N_{tet} is reduce by one.

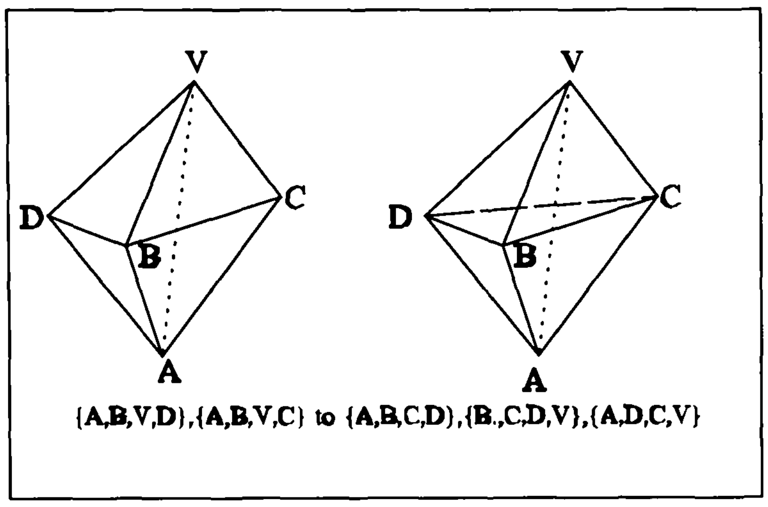


Figure 7.6.1 2-3 Mapping

(4) If no four of the five vertices *{a,b,c,d,v}* are coplanar, and the line *{a,v}* intersects the interior of the triangle *{b,c,d}*, and the tetrahedron *{a,c,d,v}* is present. Then apply a 3-2 mapping and remove the faces *{a,c,v}* and *{a,d,v}* from *Q*. Add the faces *{a,v,c}* and *{a,d,v}* to the tail of *Q* and reduce N_{tet} by 2.

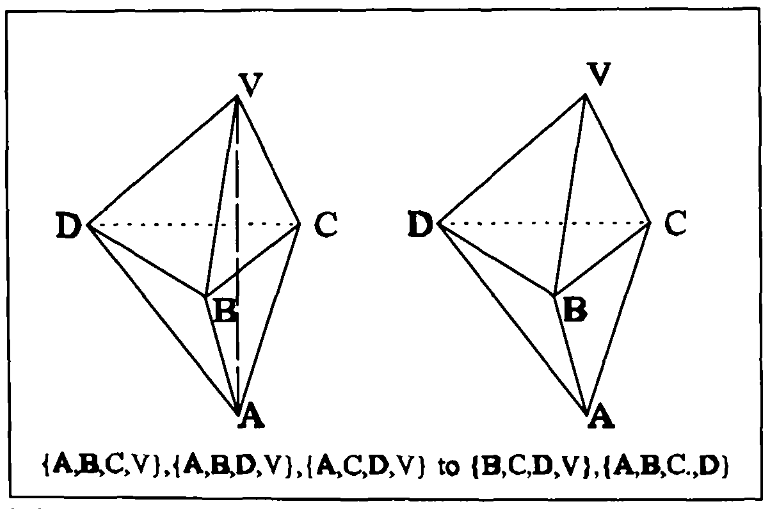
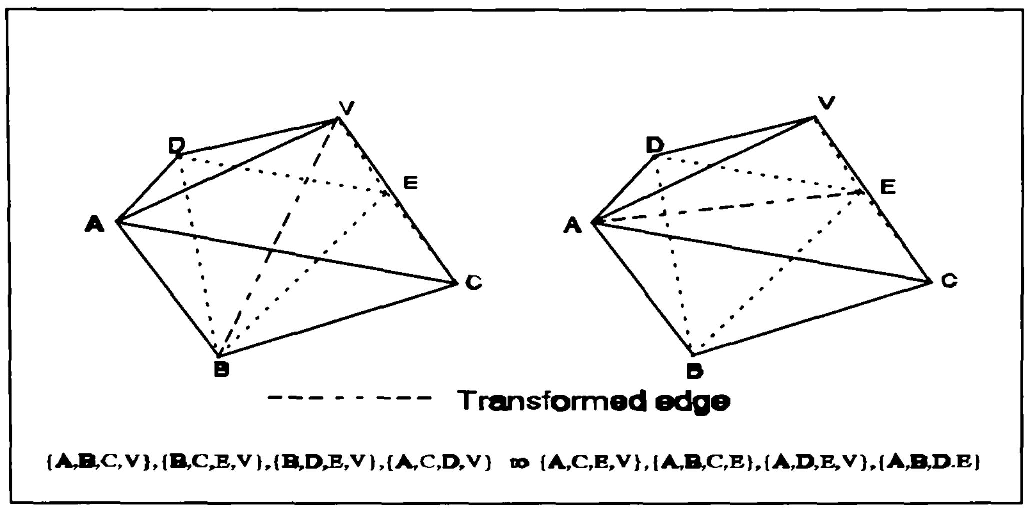


Figure 7.6.2 3-2 Mapping

(5) The vertices *{b,c,d,v}* are co-planar and the lines *bv* and *ae* are in the quadrilateral *{c,b,d,v}* and there exists two tetrahedra *{b,c,e,v}* and *{b,d,e,v}*. Apply the 4-4 mapping which swaps the edge *{v,b}* to *{a,e}*. Remove the faces *{b,c,v}*,*{b,d,v}* and *{b,c,v}* from *Q* and reduce N_{tet} by 2. The face *{a,e,v}* is added to the tail *Q*.


V
D
E
A
C
B
V
D
E
A
C
B
Transformed edge
{A,B,C,V},{B,C,E,V},{B,D,E,V},{A,C,D,V} to {A,C,E,V},{A,B,C,E},{A,D,E,V},{A,B,D.E}
Figure 7.6.3 4-4 mapping

(6) If none of the previous mapping can be applied, add face *I* to *Q*.

(7) Repeat steps 1-4 until N_{tet} is equal to 4 or none of mappings can be applied to the faces in *Q*

(8) if N_{tet} equals 4 the tetrahedron *{a,b,c,v}*,*{a,b,d,v}*,*{a,c,d,v}* and *{b,c,d,v}* can be removed to form one tetrahedron *{a,b,c,d}* which ineffectually removes the vertex *v*.

The above algorithm will always terminate as each mapping removes a tetrahedron incident on *v*. The method is not always guaranteed to remove an internal vertex even though a boundary-constrained triangulation exist from the set of boundary points.

7.6.1 Local minima

The min-max tetrahedral transformation algorithm implemented in the grid generator is a local min-max routine. This routine converges to one of the possible local optimal angle solutions for the transformations implemented, see section 7.5, which may be far from the global optimal solid angle tetrahedral mesh.

The graph below, Figure 7.6.1, is a plot of mean solid angles, before and after optimization. This graph is ordered from left to right, into increasing mean solid angles, *before optimization*. This graph is generated from a geometry with a fixed nodal distribution, with various tetrahedral meshes. The results were achieved by taking an initial mesh, and then applying sets of random transformations to the grid. The geometry was then optimized by the local-minmax optimization routine. Figure 7.6.2, presents minimal solid angles of the above meshes, which shows a general minimal angle improvement.

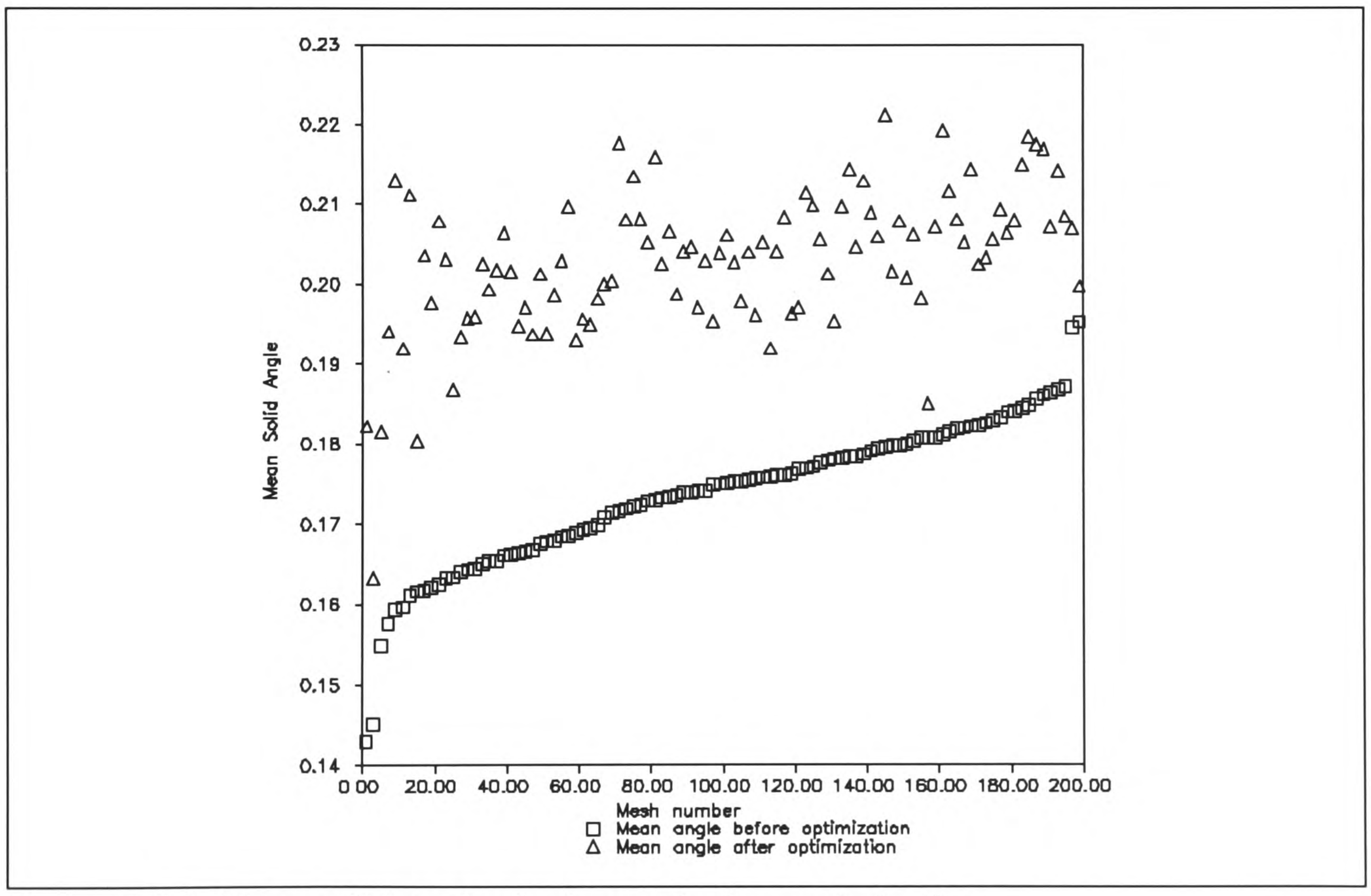


Figure 7.6.1 Mean solid angles before and after optimization.

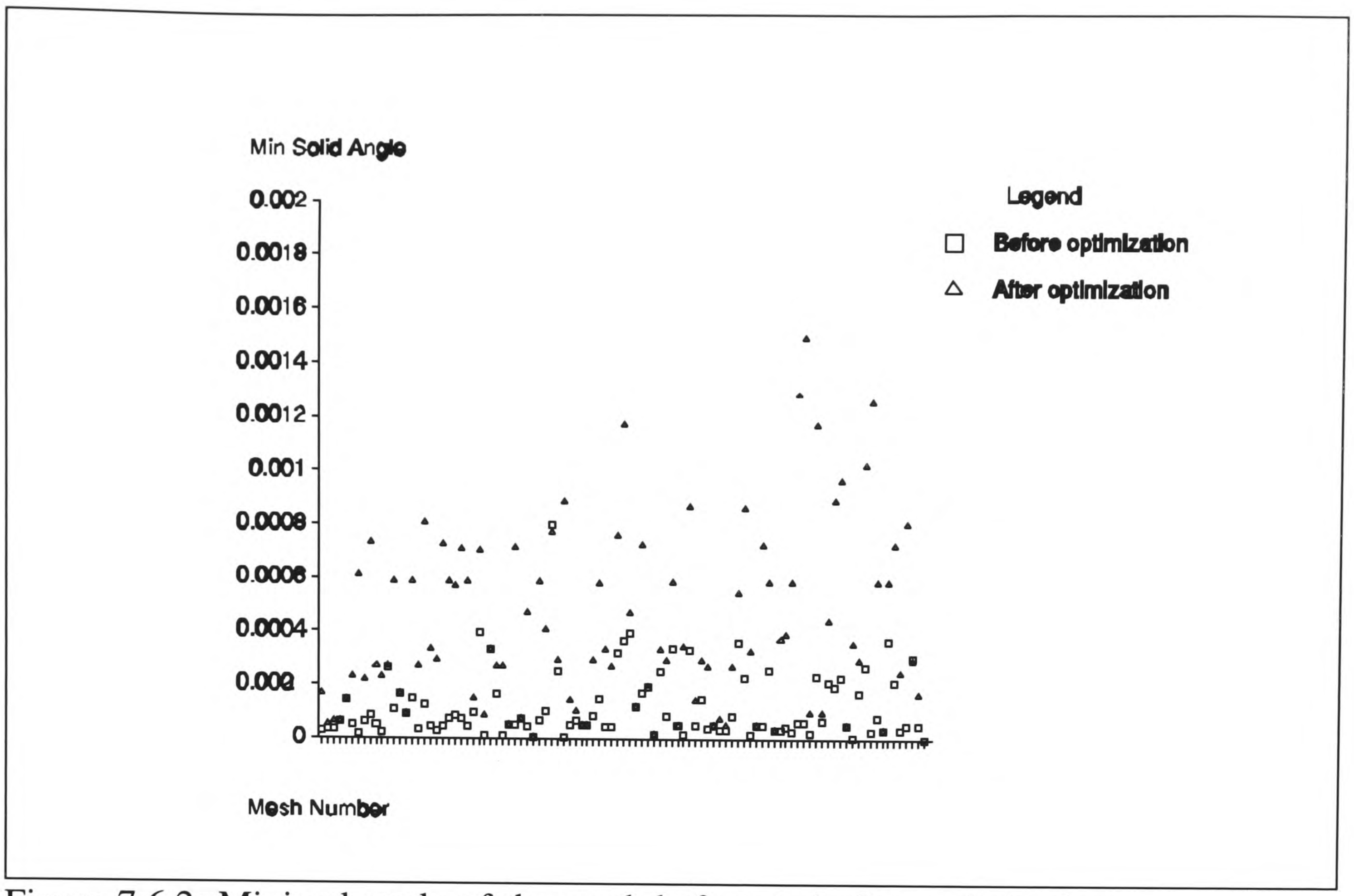


Figure 7.6.2: Minimal angle of the mesh before and after optimization.

One possible reason for the large number of possible local optima achieved, is that the optimization routine is limited by the number of transformations implemented. However, the local optimization algorithm does improve the quality of the final mesh [Joe91b], and adding new possible transformation would enhance the algorithm.

7.7 Conclusions.

The design and implementation of any algorithm for geometry operations must be done in conjunction with the model representation and data structure used within the program. The geometry representation and data structure of the code has a great effect on computational efficiency and robustness of the overall mesh generation tool.

The sub-division of geometry into simpler parts is a highly complex task, which is reflexed in this chapter by the number of different bisection methods presented. The correct mesh bisection technique has to be applied at the right time for the grid generator to work reliably and efficiently. The sub-division part of the mesh generator tool was found to be such a complex task that a rudimentary heuristic algorithm, Boundary constrained local min-max meshing algorithm, is used once the domain is sufficiently simple. Even the simple mesh generation technique of Boundary constrained local min-max meshing algorithm has to be implemented in conjunction with a robust and reliable technique of tetrahedral translations.

For a further discussion of finite precision problems and techniques to minimize their effects, see Appendix A5.

Chapter 8
3D Mesh Generation
Results and Conclusions.

8.1 Introduction

This chapter will present some example models, which have had three dimensional tetrahedral meshes generated over their geometry, using the techniques described in Chapter 6. The Chapter is divided into three sections, to try and give an indication of the power of the meshing technique, described in this thesis. The three sections are described next:

Basic Examples :	Demonstrates the mesh generators ability to cope with varies geometry features, computational efficiency and mesh quality.
Comparison Examples :	This section is an attempt to compare the bisection mesh generator's computational efficiency and mesh quality, with other mesh generators described in various journals.
Further Examples :	The final section illustrates some of the further examples to which the mesh generator has been applied to. This section also helps to re-enforce some of the conclusion drawn in the previous sections.

The results are presented with CPU time in seconds, for various size grids, with associate mesh quality measurements. The two mesh quality measures are element *goodness measure*, section 2.4.2, and *tetrahedral solid angle*, section 2.4.1. These are obtained on the completed mesh before optimization, as this gives an indication of the power of the basic meshing technique. The CPU times were obtained for un-optimized code running on a SPARC 10 with the Unix operating system SunOS release 4.1.3.

8.2 Basic examples

The next four example geometries illustrate the mesh generator's ability to cope with various basic geometry features, which were discussed in Chapter 7, section 7.1. Each example is presented with an illustration of the geometry with an indication to which feature it demonstrates. For each example the CPU time to generate various size meshes is presented, together with a table indicating how CPU time is spread between the various stages in mesh generation.

The overall CPU profile of the complete time to generate various size grids, over the example geometry presented in this section is comparable linear, see Figures 8.2.3,8.2.8,8.2.13 and 8.2.18. From the tables 8.2.1,8.2.2,8.2.3 and 8.2.4, of CPU break down, the bisection part of the code is highly efficient, only taking a fraction of the overall CPU time of mesh generation. The tables also indicates that the majority of the meshing time is being taken up by the min-max boundary constrained triangulation routine, with the nodal placement technique second most CPU intensive.

Each example has the two mesh quality measures, which were presented in Chapter 3 section 3.3, mean values plotted against mesh size. Both the *tetrahedral solid angle* and *element goodness* measures average values improve with mesh size. This is a highly desirable feature of any mesh generator, since increasing the number of tetrahedra should reduce the individually dependency of each element on the geometry of the problem.

8.2.1 Three dimensional box with a cavity (Example 1).

This example demonstrates the mesh generators ability to deal with domains with simple cavity regions (Multi-connected regions).

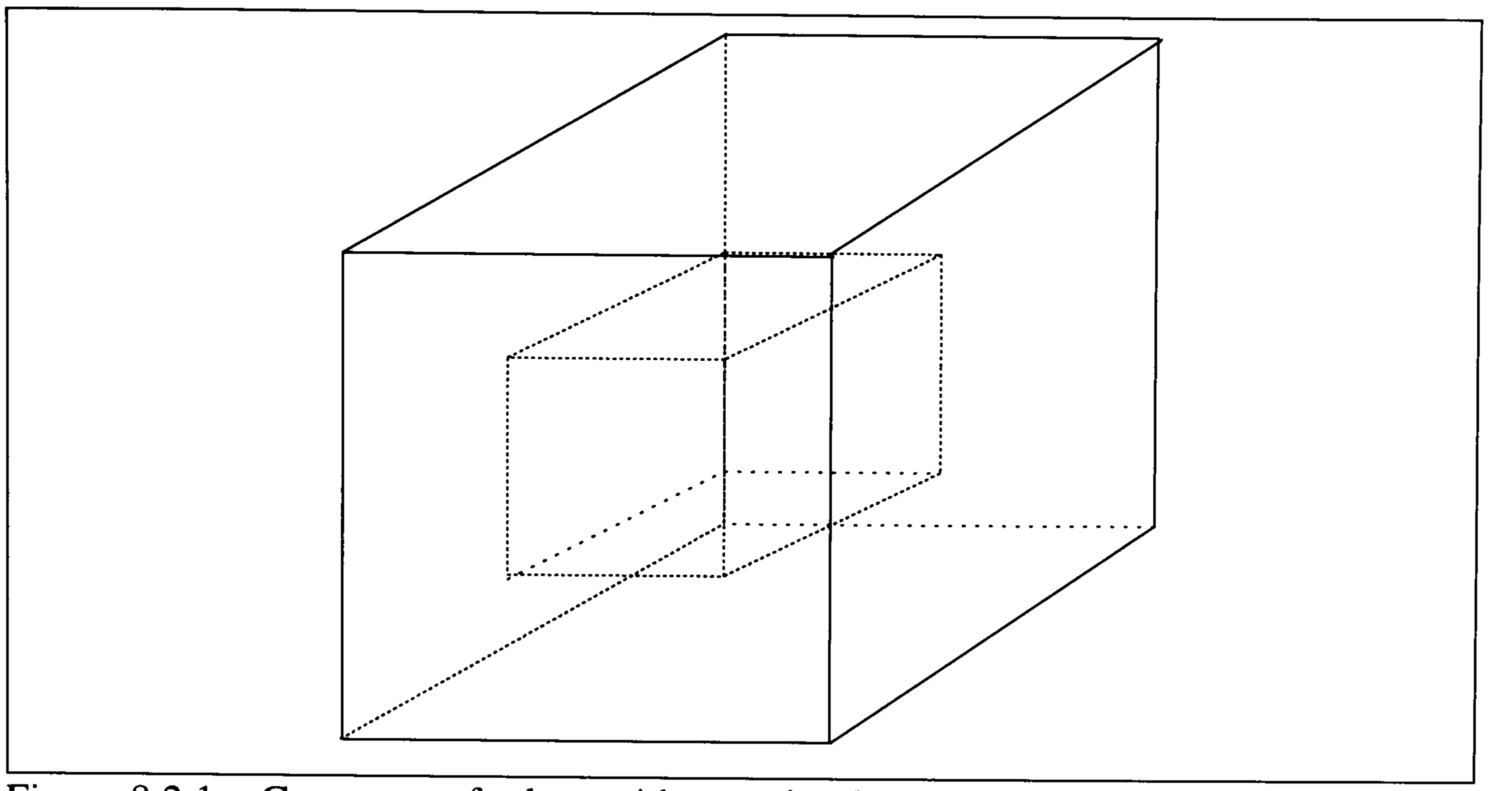


Figure 8.2.1 Geometry of a box with a cavity (Example 1).

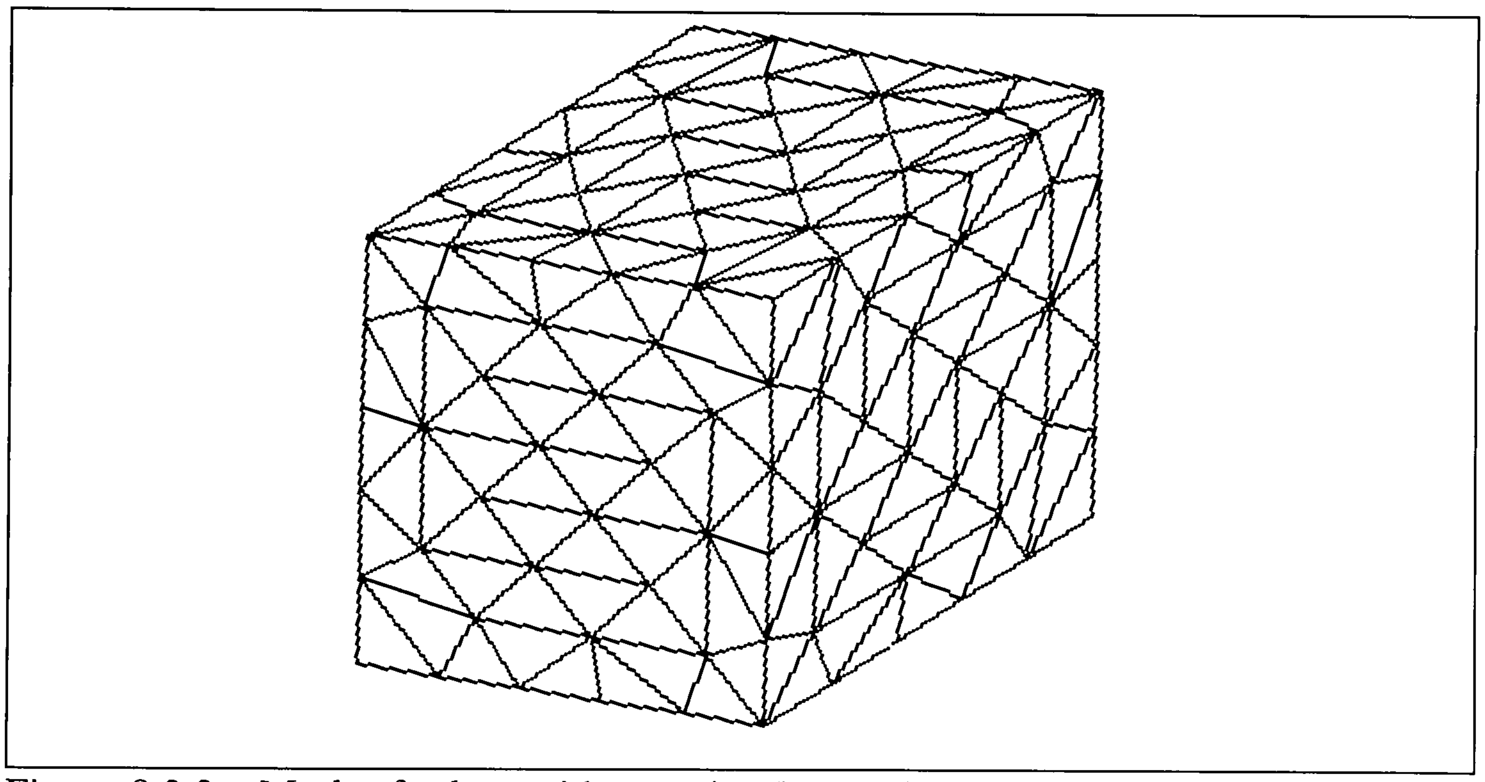


Figure 8.2.2 Mesh of a box with a cavity (Example 1).

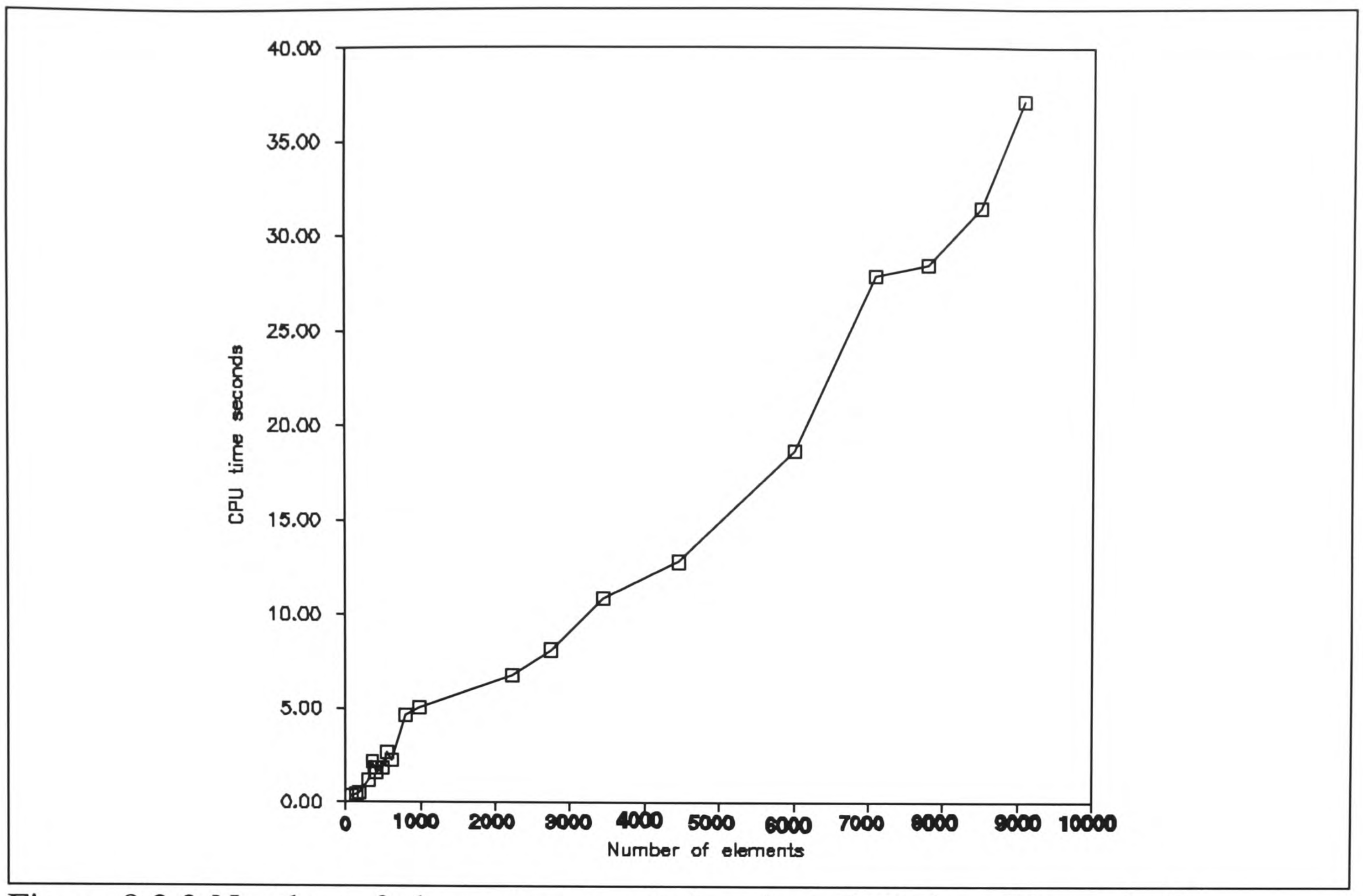


Figure 8.2.3 Number of elements generated against CPU time (Example 1).

Below is a table showing how CPU time is divided between the various routines in the mesh generator. The first and second columns are the number of elements and nodes generated in the grid. The third column, is the setup time of the geometry, such as hole polyhedral and cavity identification. This is followed by time to sub-divide the geometry into sub-regions, which then has nodes generated in them, and the times for this is in column 5. The last two columns are CPU time to generate the final tetrahedral meshes in the sub-regions and the overall CPU time for mesh generation.

Table 8.2.1: Break down of CPU time, in seconds, used in the code (Example 1).

Elements	Nodes	Init Geom	Bisect	Gen. Nodes	Gen. Tets	Total Cpu
173	67	0.02	0.02	0.1	0.35	0.49
994	313	0.021	0.021	0.17	4.92	5.132
4458	1138	0.02	0.022	0.54	12.63	13.212
9062	2112	0.02	0.21	0.76	37.03	38.02

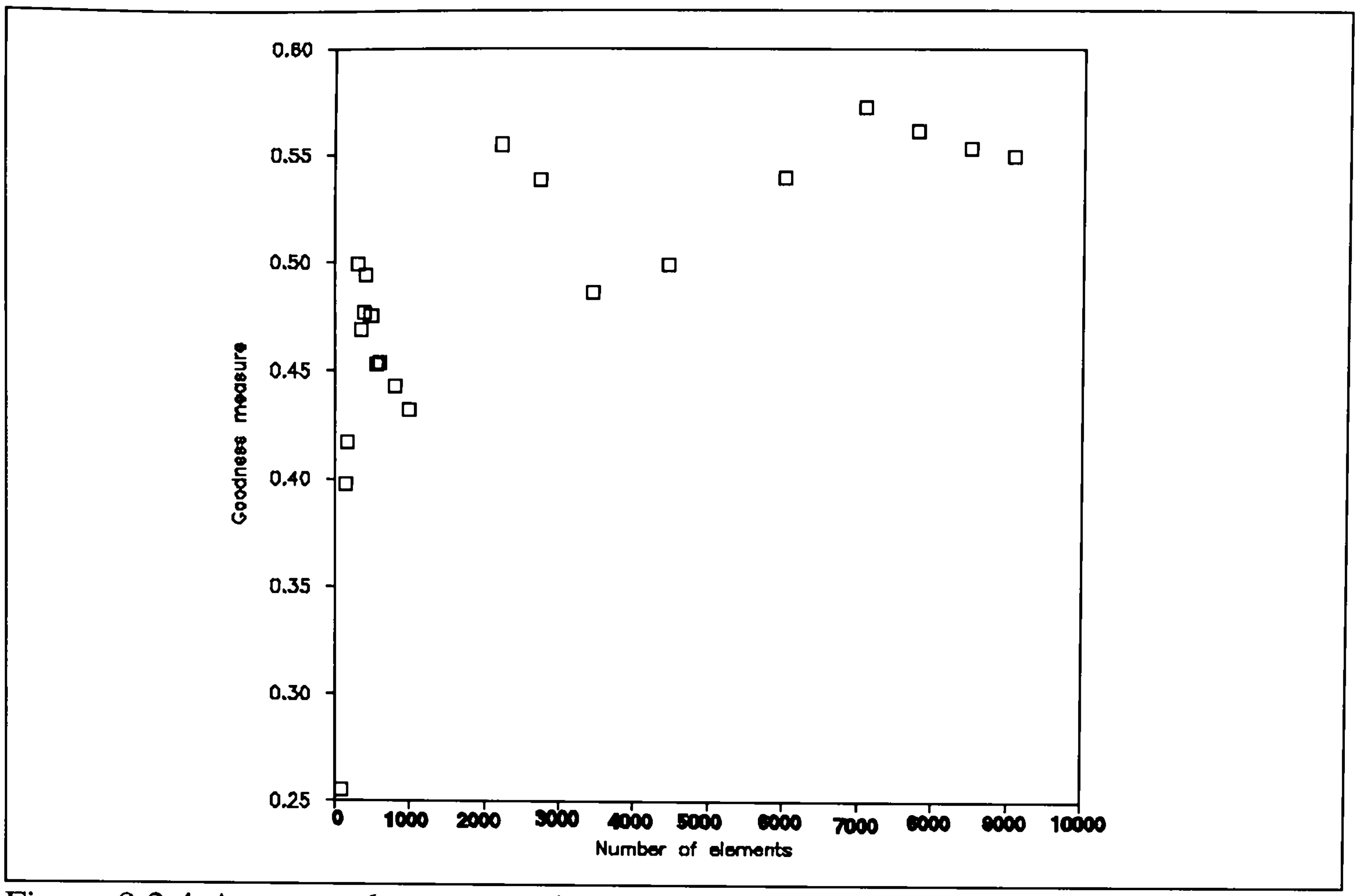


Figure 8.2.4 Average element goodness measure plot (Example 1).

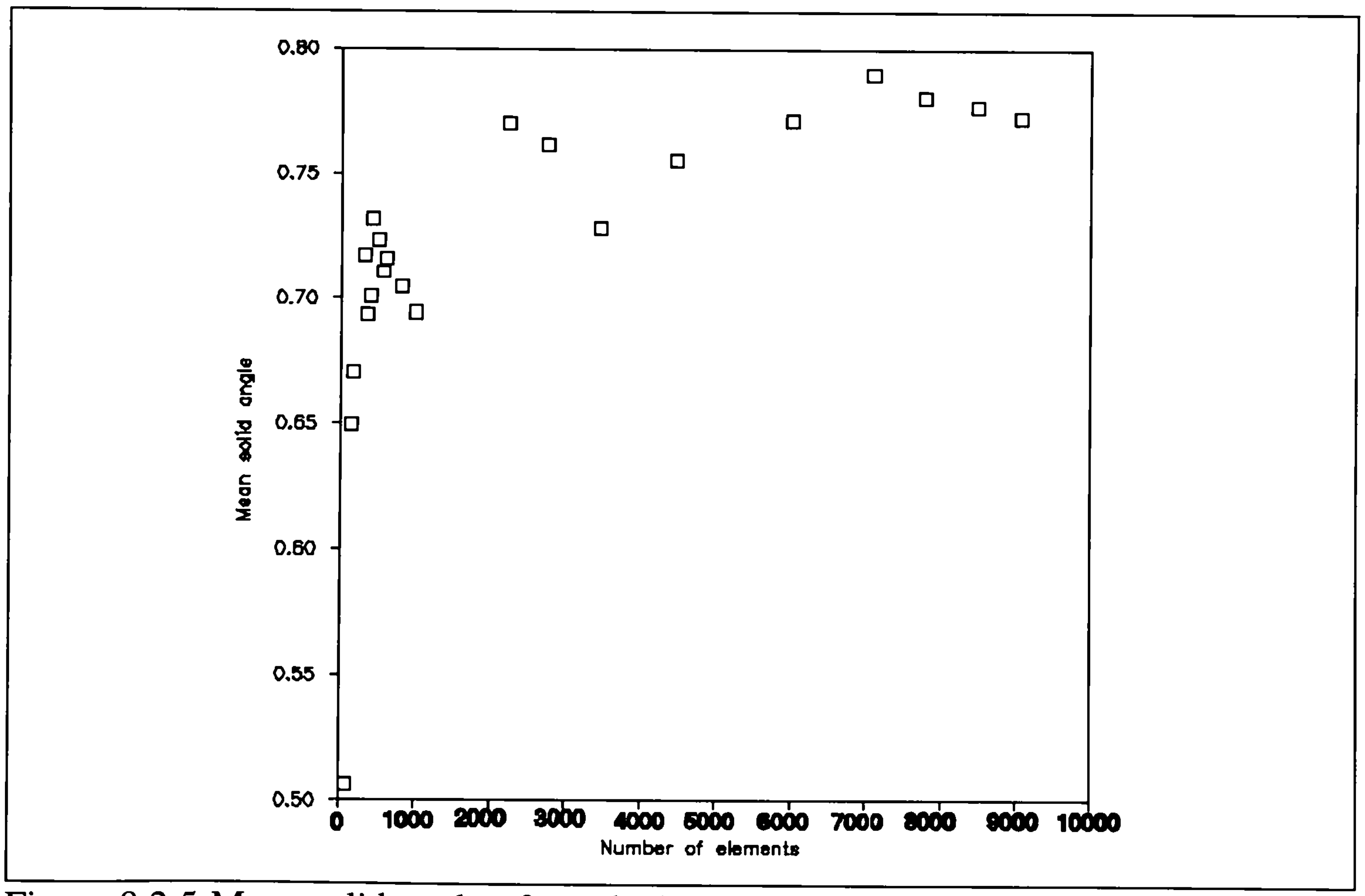


Figure 8.2.5 Mean solid angle of tetrahedra as mesh size increase (Example 1).

8.2.2 A box with a cavity adjacent to another box (Example 2)

This section presents a simple example of two adjacent independent domains which share a common boundary interface. One of the region also has a cavity (hollow) section.

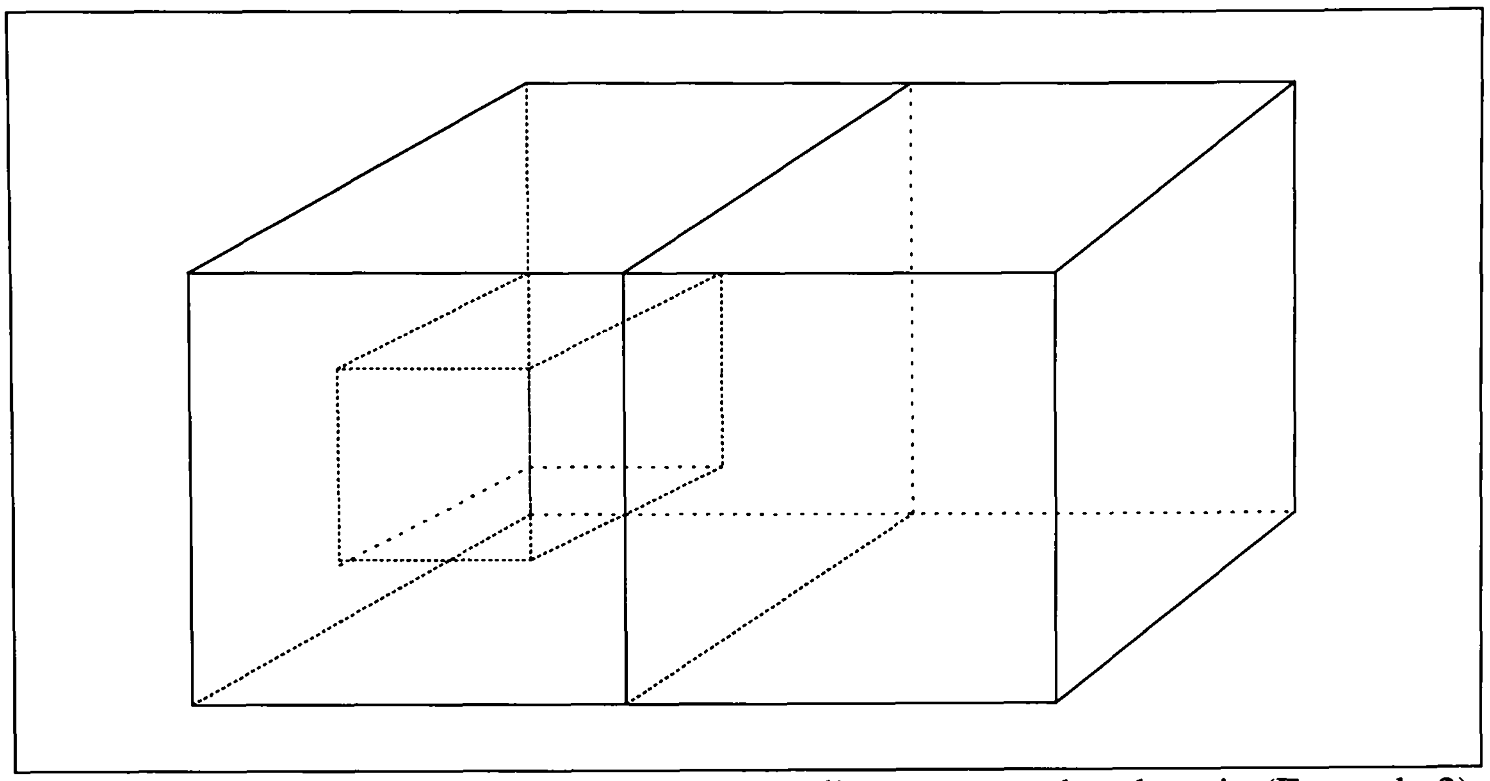


Figure 8.2.6 Geometry of a box with a cavity adjacent to another domain (Example 2).

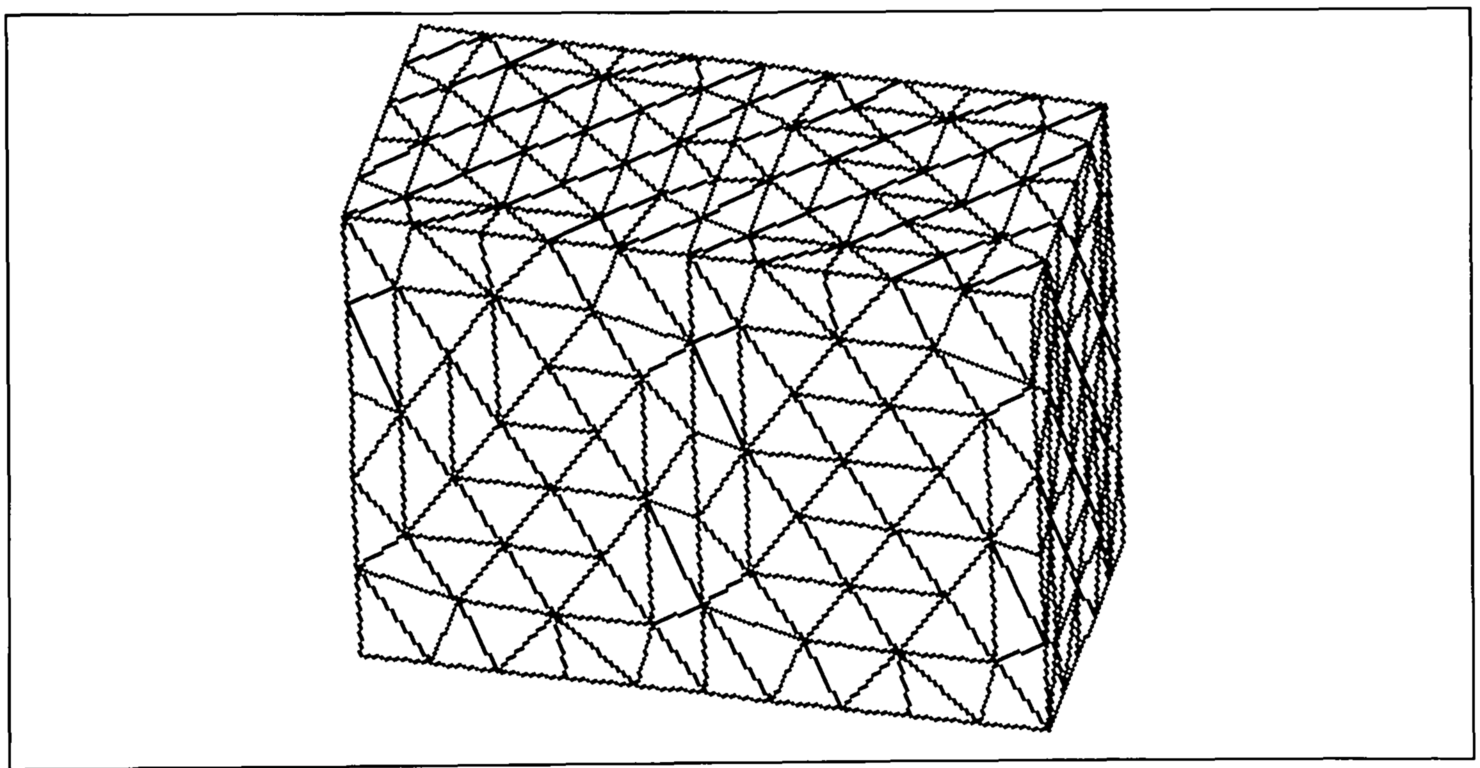


Figure 8.2.7 Mesh of a box with a cavity adjacent to another domain (Example 2).

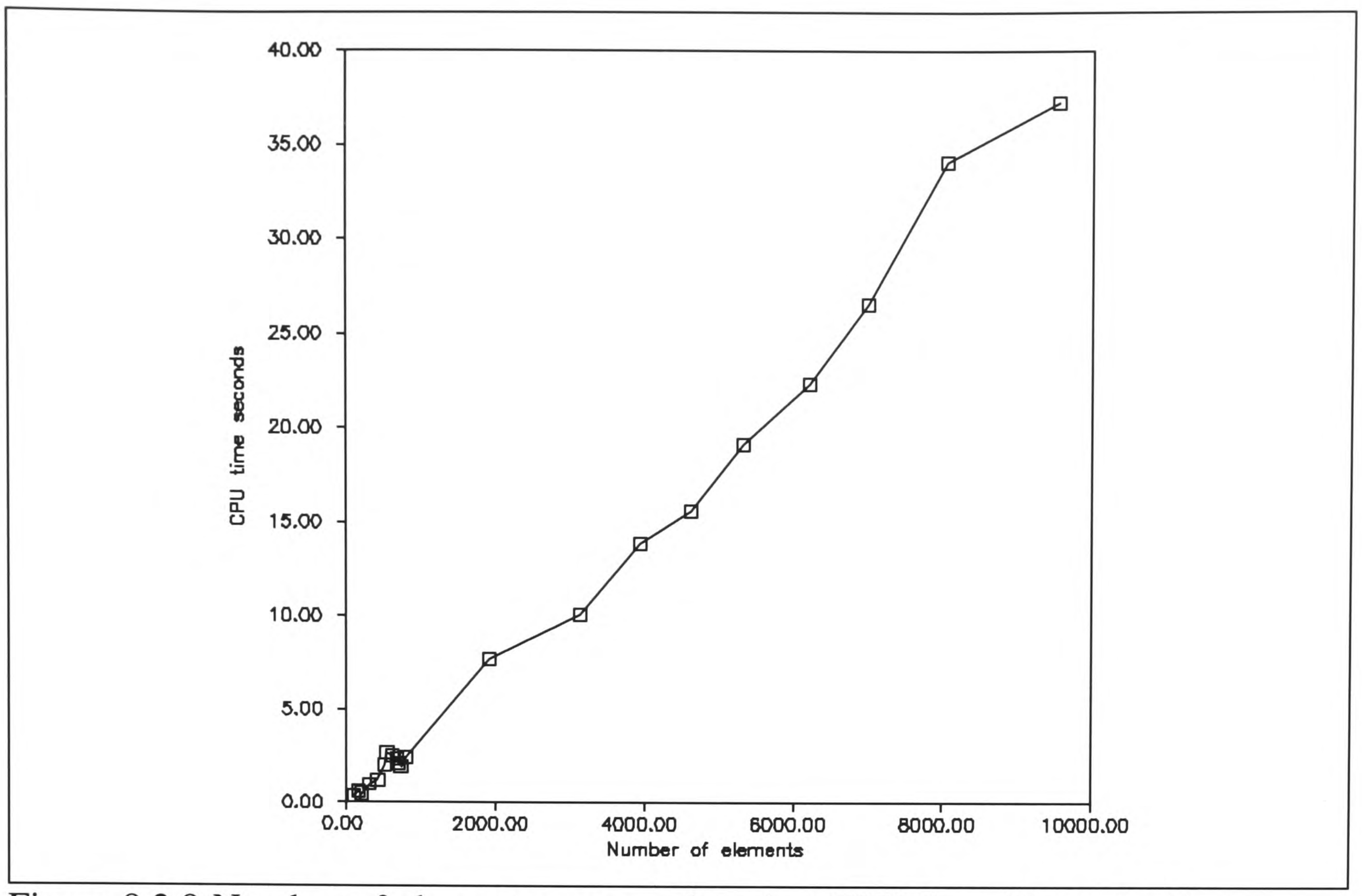


Figure 8.2.8 Number of elements generated against CPU time (Example 2).

Table 8.2.2 : CPU Break down for the cavity cube with interface (Example 2).

Elements	Nodes	Init Geom	Bisect	Gen. Nodes	Gen. Tets	Total CPU
200	72	0.025	0.022	0.2	0.22	0.467
803	253	0.024	0.02	0.18	2.04	2.264
4624	1146	0.027	0.022	0.57	14.46	15.079
9551	2172	0.025	0.021	0.77	34.43	35.246

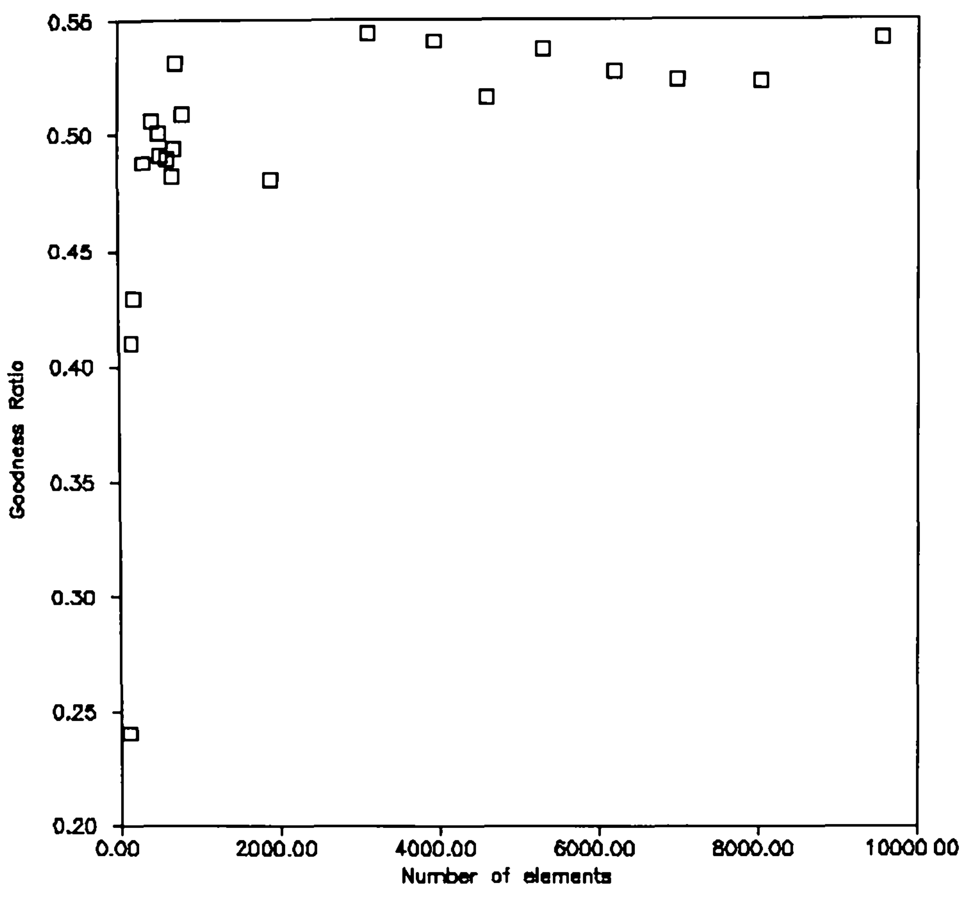


Figure 8.2.9 Number of elements against mean goodness factor (Example 2).

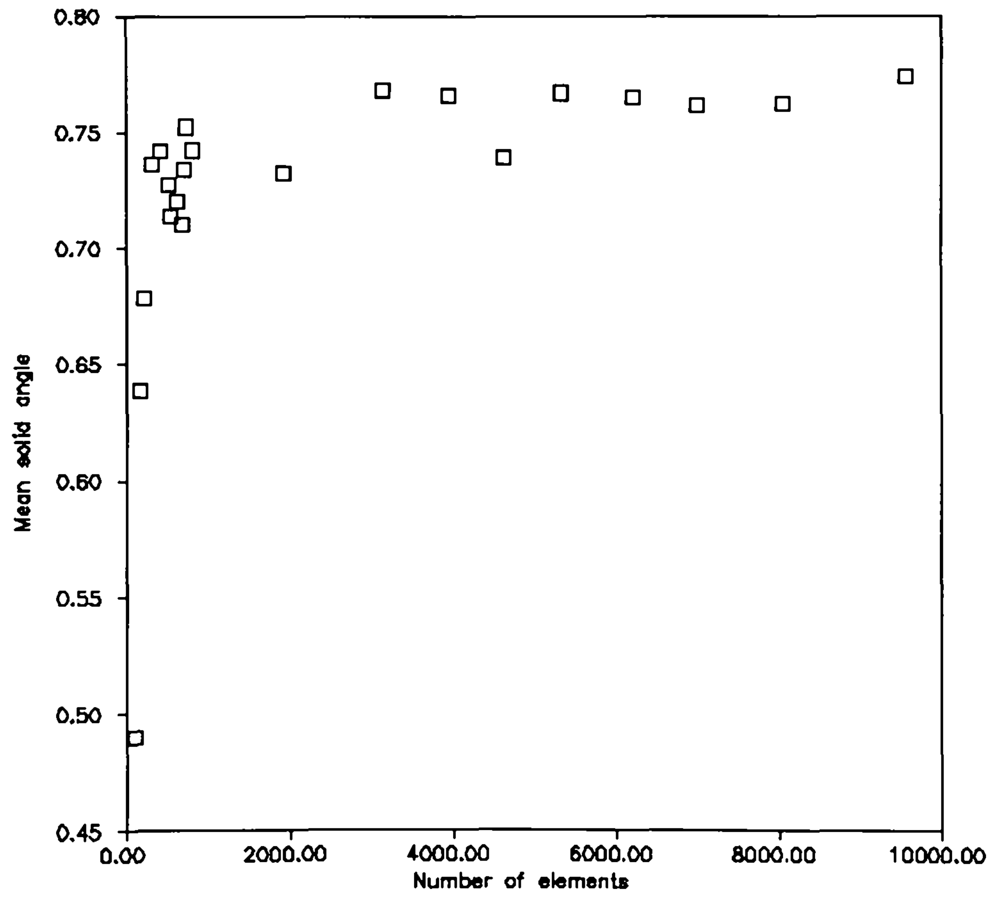


Figure 8.2.10 Number of elements against mean solid angle (Example 2).

8.2.3 Three dimensional lug geometry (Example 3).

This example illustrates the mesh generator's ability to cope with more complex domains with holes and concave regions.

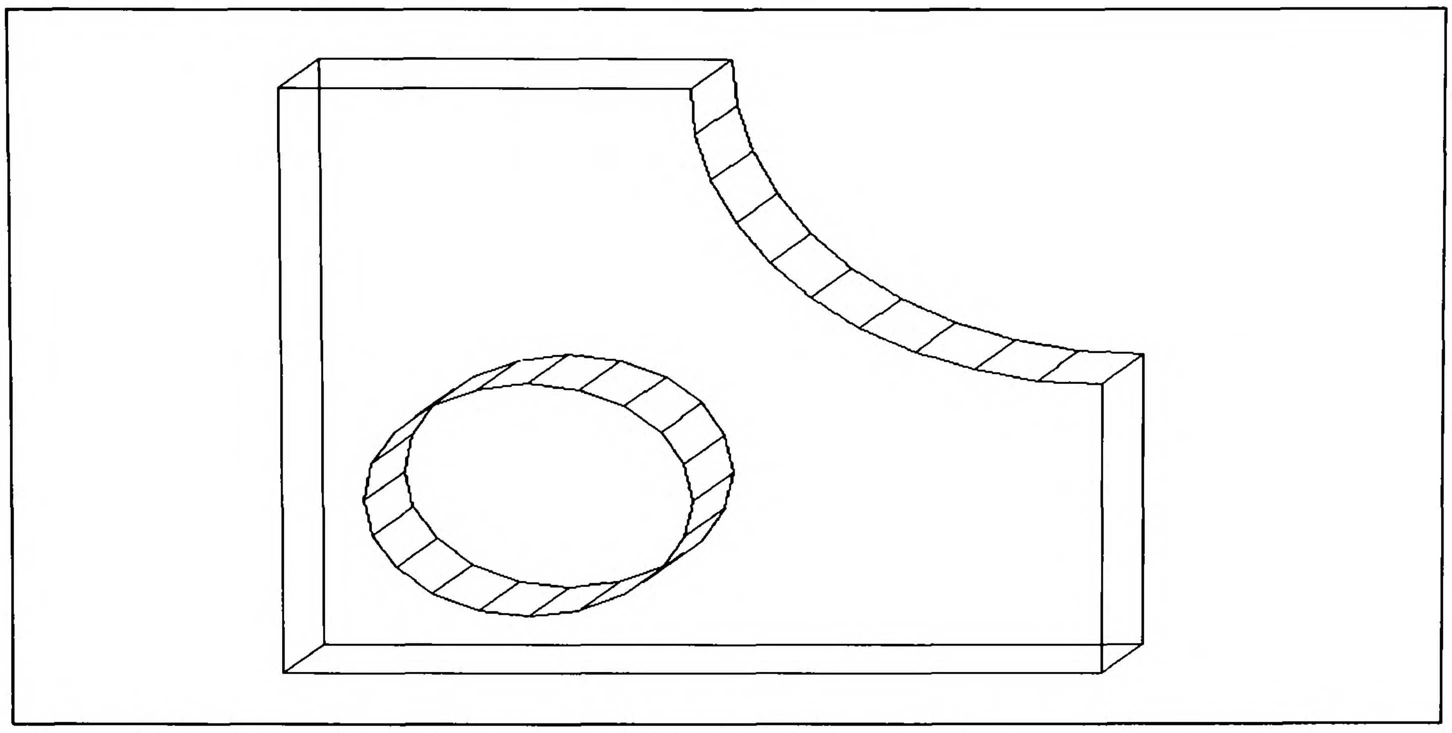


Figure 8.2.11 Polyhedral domain of 3D lug (Example 3).

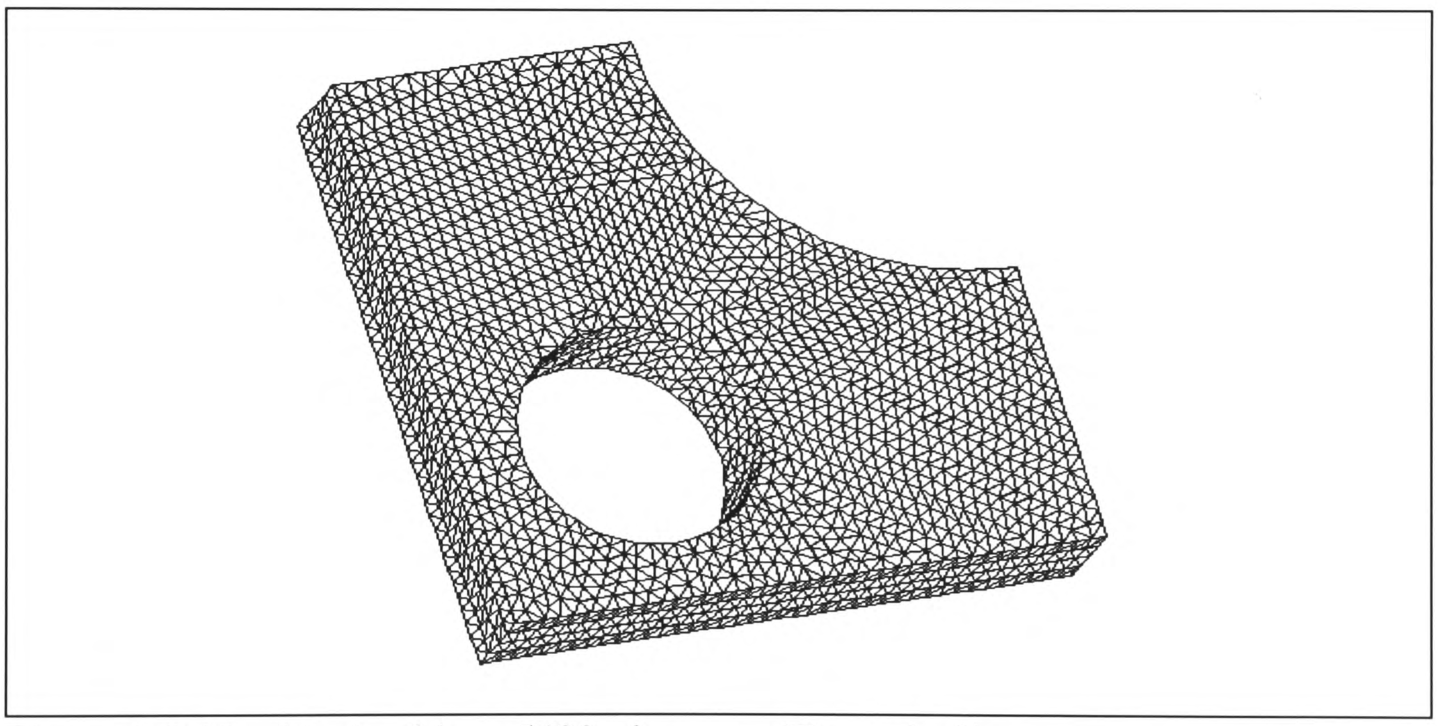


Figure 8.2.12: Mesh of lug, 5432 elements (Example 3)

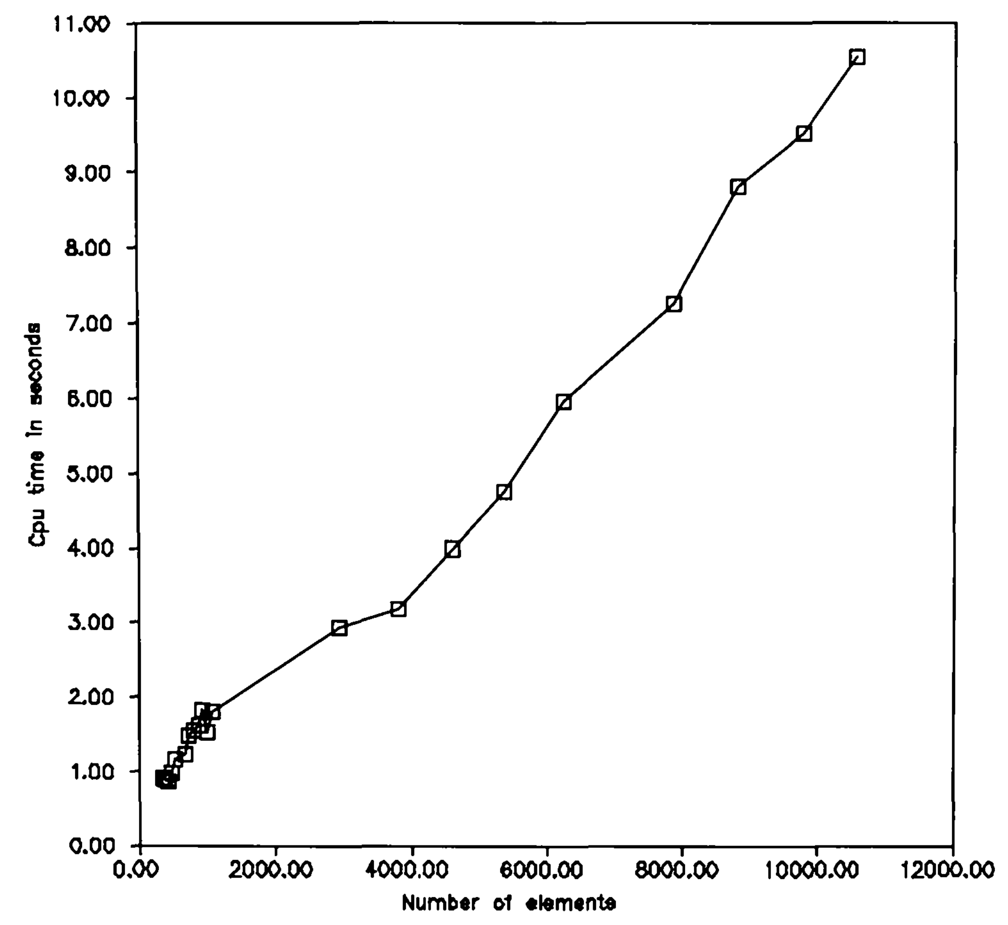


Figure 8.2.13 Number of elements generated against CPU time (Example 3).

Table 8.2.3 : CPU breakdown for the Lug mesh (Example 3).

Elements	Nodes	Init Geom	Bisect	Gen. Nodes	Gen. Tets	Total CPU
410	204	0.025	0.15	0.55	0.57	1.295
1134	412	0.025	0.15	0.64	1.15	1.965
5432	1407	0.026	0.17	1.35	4.85	6.396
10170	2387	0.025	0.15	1.81	9.68	11.665

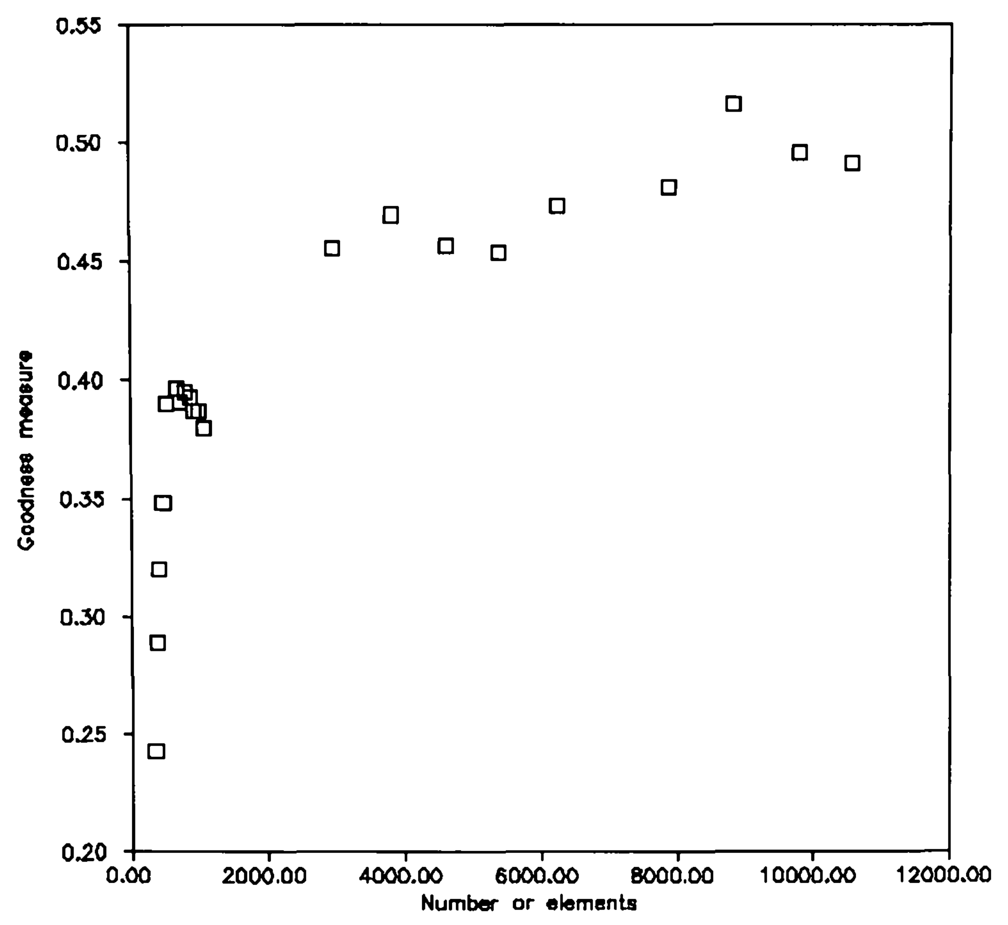


Figure 8.2.14 Average element goodness measure plot (Example 3).

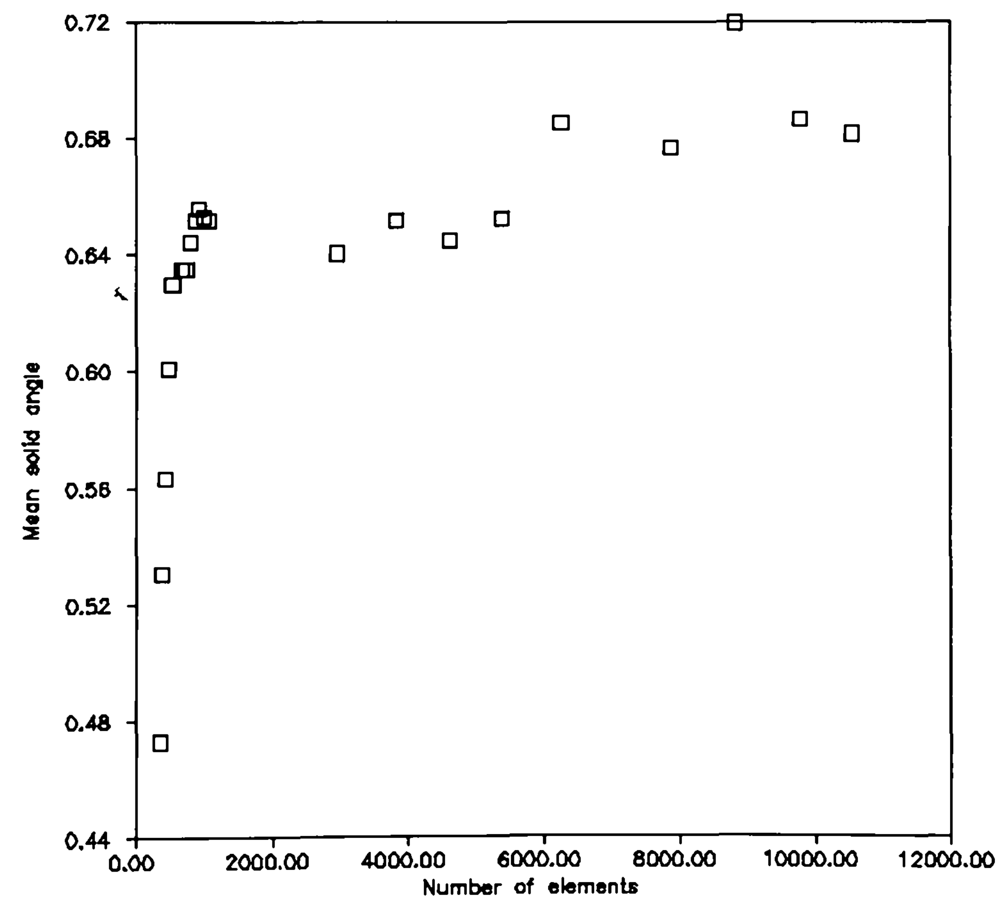


Figure 8.2.15 Mean solid angle of tetrahedra as mesh size increases (Example 3).

8.2.4 Cross section of a plane in a wind tunnel (Example 4).

Below, Figure 8.2.16, illustrates a simple plane geometry within a cube. This example shows the ability of the grid generator to cope with *hole polygon regions*, this is illustrated by the outer boundary of the aircraft forming a hole in one side of the surrounding cube.

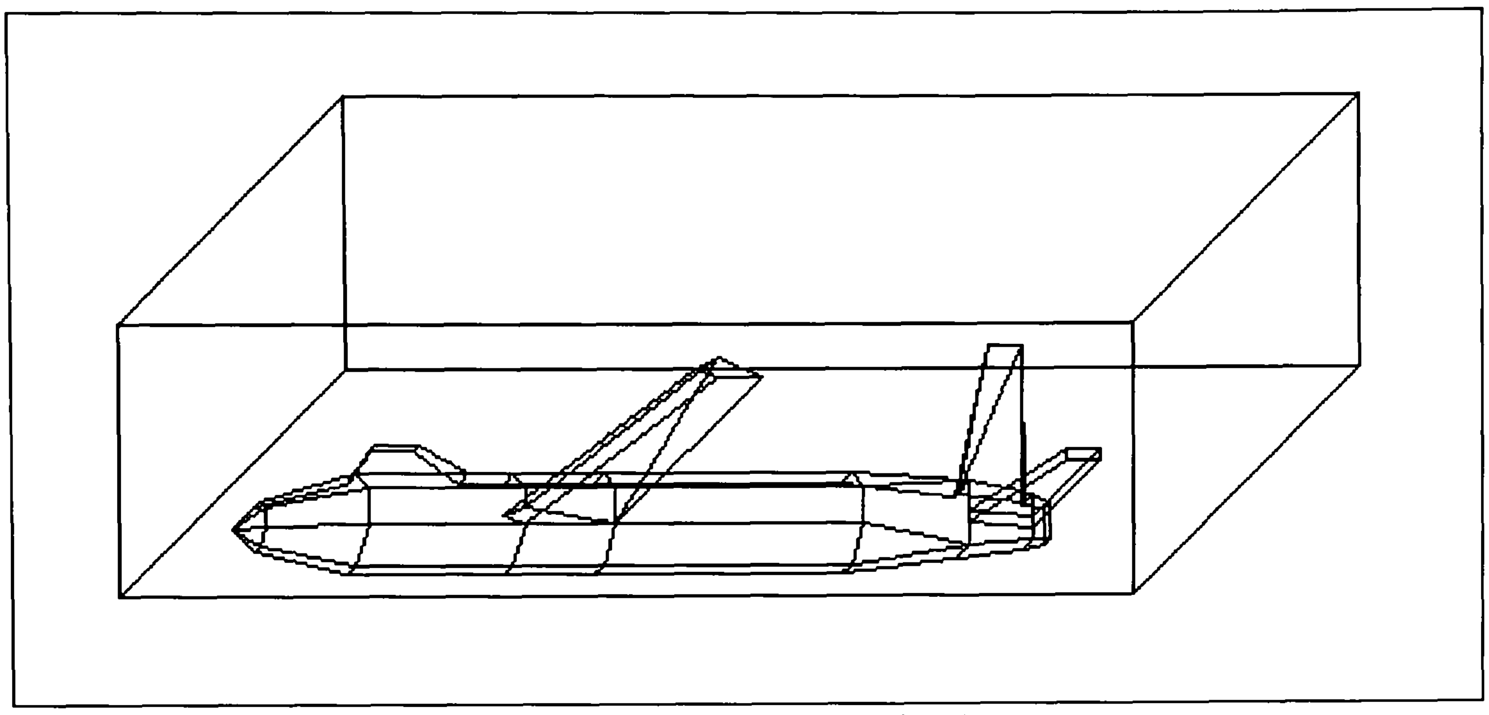


Figure 8.2.16: Example of plane geometry (Example 4).

Figure 8.2.17 illustrates the completed three dimensional tetrahedral mesh of the above geometry.

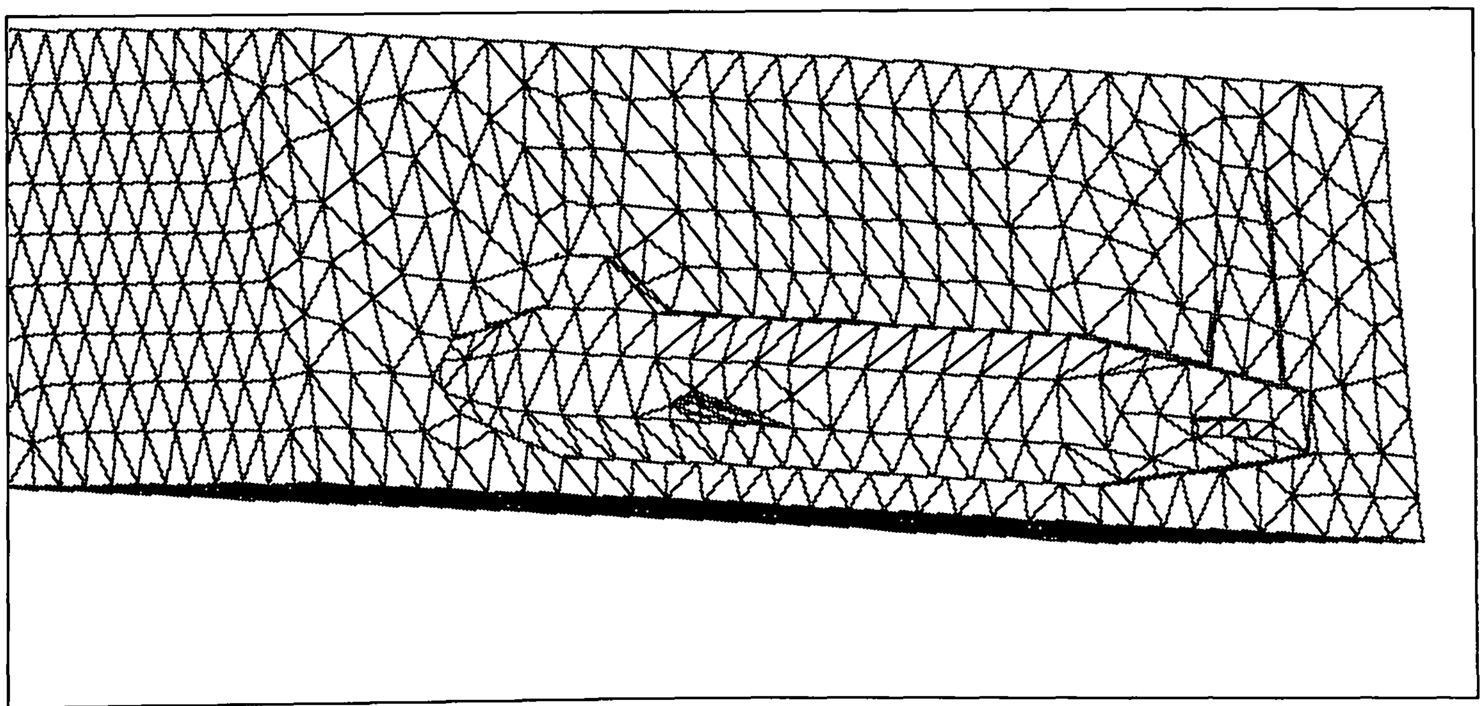


Figure 8.2.17: Example of plane tetrahedral mesh (Example 4).

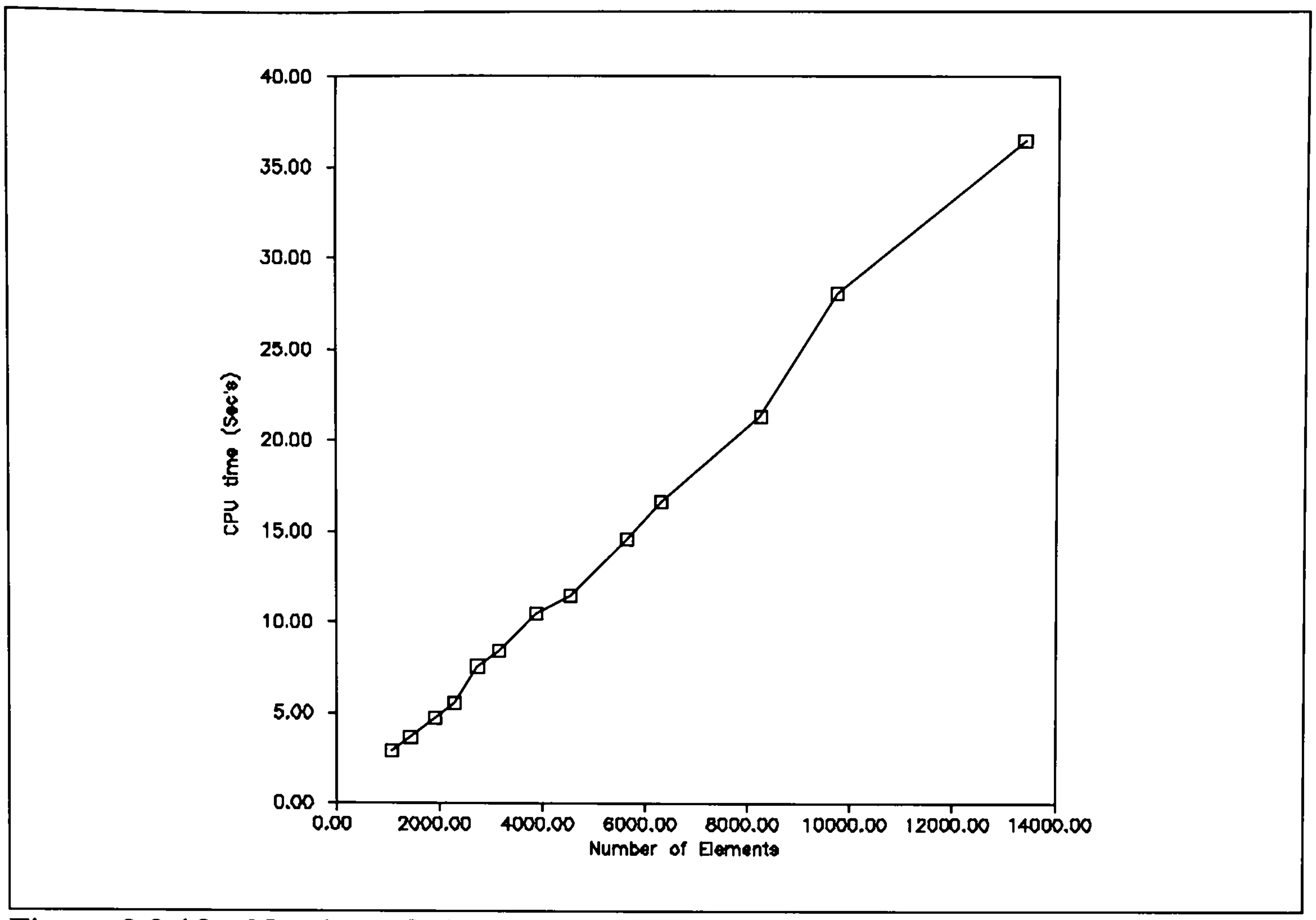


Figure 8.2.18 : Number of elements generated against CPU time (Example 4).

Table 8.2.5 : CPU use, in the generation of plane in tunnel mesh (Example 4).

Elements	Nodes	Init Geom.	Bisect	Gen. Nodes	Gen. Tets	Total CPU
1439	415	0.03	0.35	2.15	1.4	3.93
4124	1027	0.031	0.3	2.73	8.71	11.771
12599	2812	0.034	0.32	3.76	37.3	41.414
21292	4590	0.031	0.3	4.54	72.61	77.481

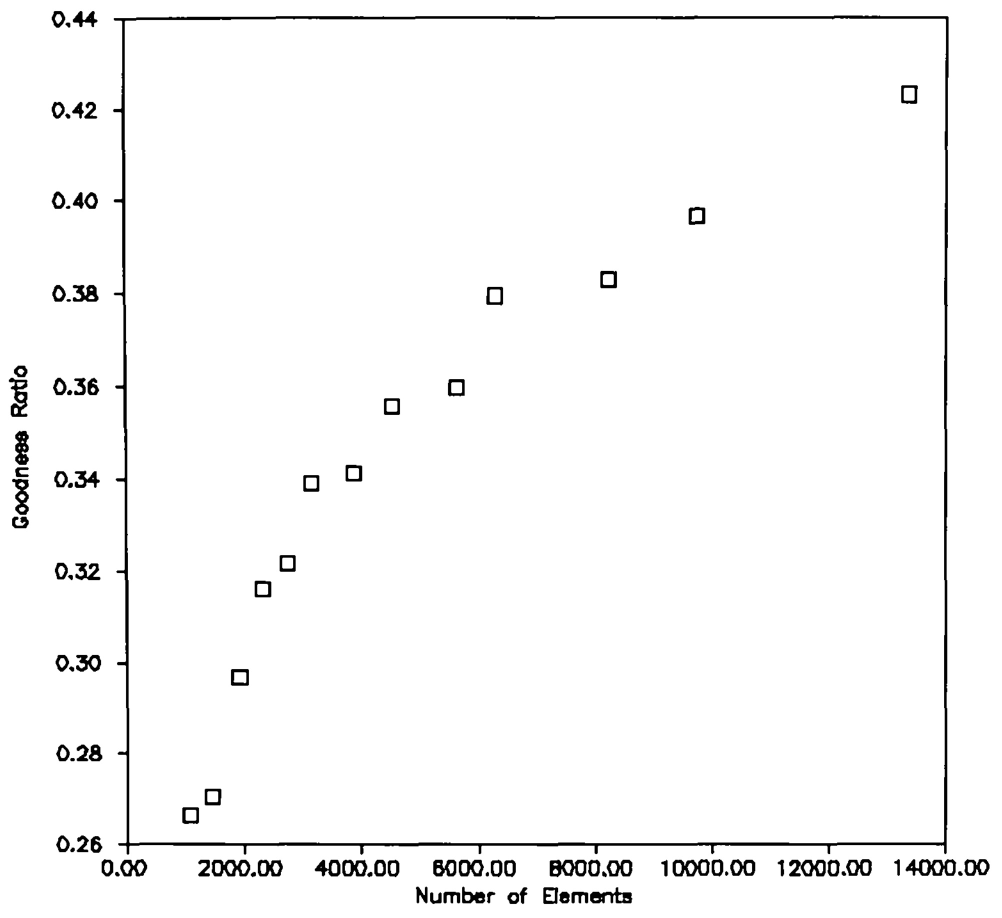


Figure 8.2.19: Number of elements against mean goodness factor (Example 4).

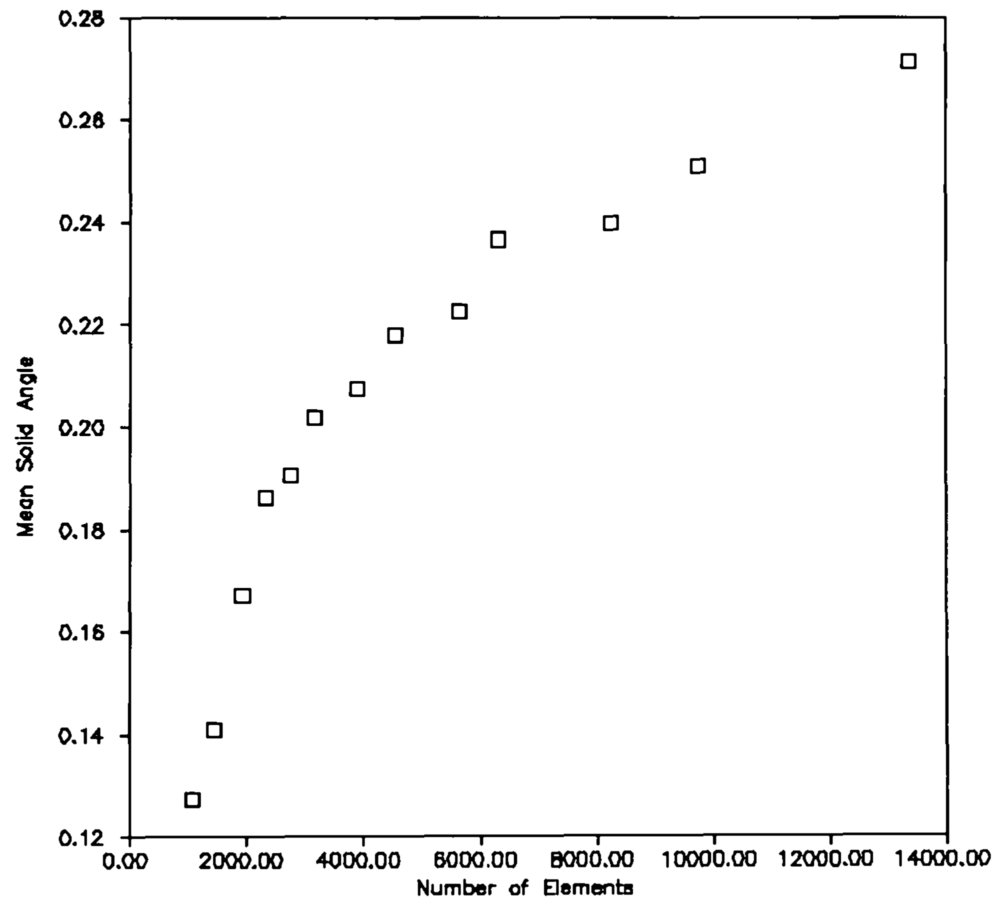


Figure 8.2.20 Number of elements generated against Mean solid angle (Example 4).

8.3 Comparison examples.

In literature there exist few, or possibly none, real test case examples for mesh generation. In the few papers, which present any meshes with grid quality measures or CPU timings, little information is provided about the geometry that their technique is applied to. Most papers have taken the approach - *Here is my algorithm; look what a good job it has done on a few examples*, Sabin M.A [Sab91]. However in this section, simple geometries from various papers have been reproduced as closely as possible.

8.3.1 Normal offsetting technique

In the paper by Johnston [Joh93] the *Normal offsetting* technique is described for three dimensional tetrahedral mesh generation. This paper contains three example meshes, together with numerical evaluation of the quality of the final solutions using the aspect ratio â, defined as the ratio of the radius of inscribed sphere to radius of circumscribed sphere. An ideal value for an equilateral tetrahedron is 0.333 [Bur90]. The first example, a cubic domain with a slender appendage, from the paper by Johnston [Joh92], being easy to reproduce, was selected as a test case for comparison with the mesh generator described in this thesis. Figure 8.3.1 depicts the geometry of the initial problem and Figure 8.3.2 is the surface mesh of the final solution.

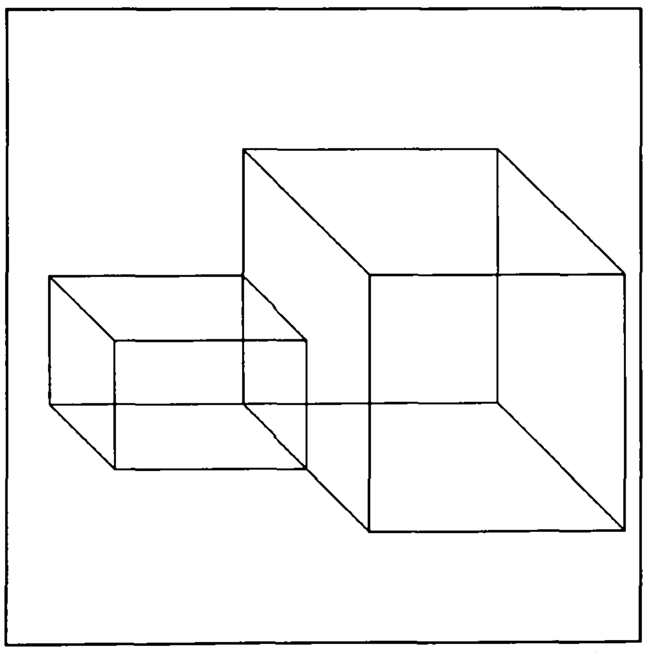


Figure 8.3.1 : Geometry of a cubic domain with a slender appendage.

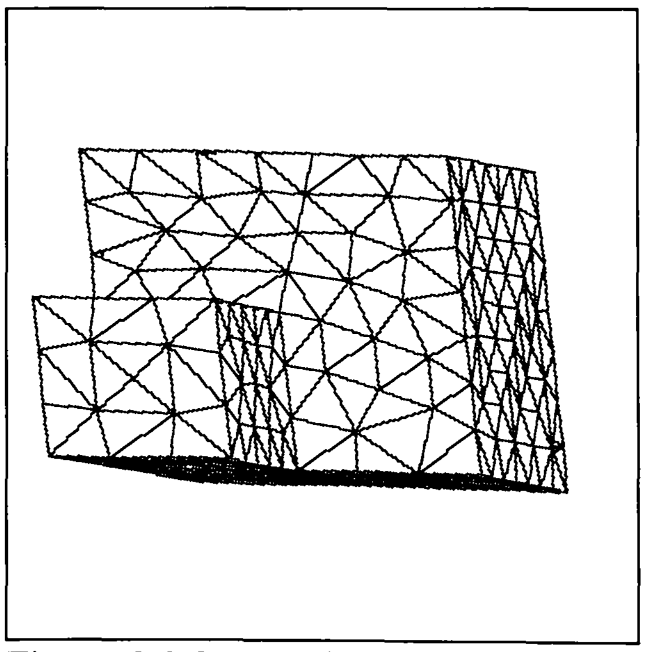


Figure 8.3.2 : Final tetrahedral mesh.

Table 8.3.1 shows a break down of the elements by their â values for the two techniques, normal offsetting and the bisection technique described in this thesis. Column two of the table is taken from the paper by Johnston [Joh93], in which 762 nodes and 3610 elements with a mean â value of 0.2398, were generated over the domain in Figure 8.3.1. The bisection technique, in this thesis, generated 837 nodes and 3593 elements with a mean â value of 0.2183.

Table 8.3.1 : Comparison of results.

â ratio ranges	Normal offsetting	Domain Bisection
00.000 -> 0.010	0.20	0.12
0.010 -> 0.033	1.00	3.86
0.033 -> 0.067	1.60	3.34
0.067 -> 0.100	2.00	4.51
0.100 -> 0.133	2.80	5.12
0.133 -> 0.167	3.30	6.96
0.167 -> 0.200	7.00	7.40
0.200 -> 0.233	12.30	10.05
0.233 -> 0.267	33.60	24.02
0.267 -> 0.300	26.30	25.77
0.300 -> 0.333	9.90	8.85

The two techniques generate meshes, with *about the same* distribution of tetrahedral aspect ratios. It could be argued that the *normal offsetting technique* generates a better quality mesh. However, if you consider that normal offsetting technique is an order $n^{1.66}$ method [Joh92] and the mesh generator in this thesis is at worst order *nlog(n)*. Couple this with the fact that with a few passes of a Laplacian smoothing routine [Her76], a linear computational algorithm, the mesh generated by the bisection technique could become as good, if not better, than the normal offsetting technique. Then the bisection mesh generator becomes more attractive.

8.3.2 Mesh generation by binary mesh operations

Shephard and Lo [Shl92] presented a technique of generating coarse meshes using *binary mesh operators*, see Chapter 3 section 3.3. In this paper there are four examples, of which one is provided with some numerical evaluation of the quality of the grid generated, using a tetrahedral goodness function λ_i, see Chapter 2 section 2.4.2. The domain they used in their analysis is depicted in Figure 8.3.4, and the resulting mesh generated by the mesh generator in this thesis, Figure 8.3.5.

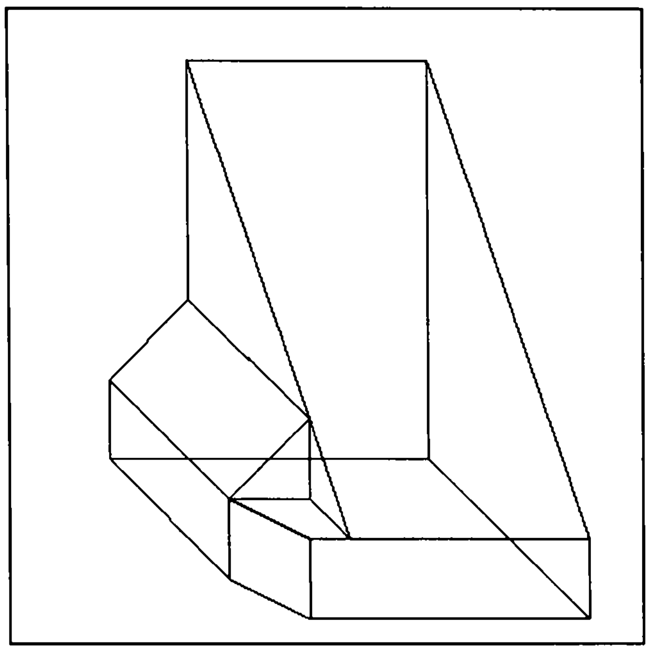


Figure 8.3.3: Shephard and Lo [ShL92] test geometry.

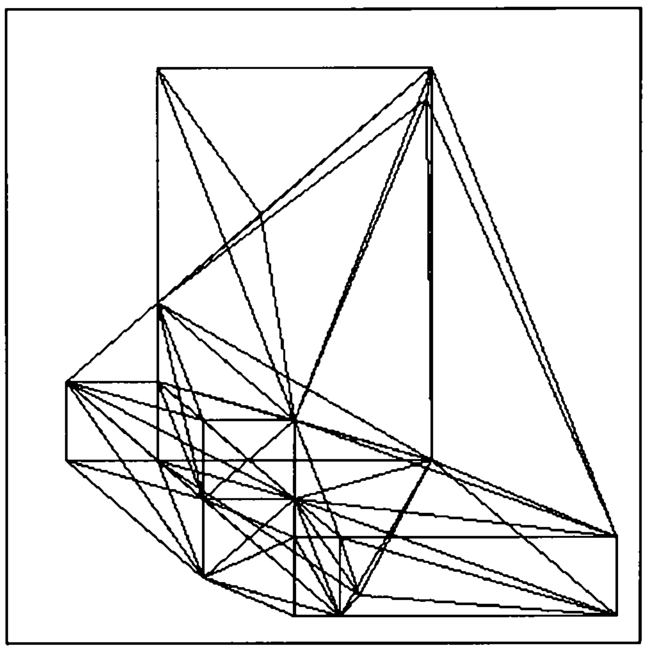


Figure 8.3.4 Tetrahedral mesh of Figure 8.2.3.

Shaphard and Lo obtained a mesh with 21 elements, and a λ_i minimum and average values of 0.1493 and 0.4410, respectively. The bisection mesh generator, presented in this thesis, obtained a mesh with 47 elements, and a minimum value of λ_i 0.0059 and average value of 0.3548.

The 47 elements, was the minimal amount of tetrahedra the bisection technique could generate for the above domain, Figure 8.3.3. This is largely down to the requirements *of the Boundary constrained min*imax algorithm, which needs an internal node to be placed at the centre of each sub-domain. These results demonstrate that the Shaphard and Lo technique can generate a grid with fewer elements and of better quality than the *Bisection technique* in this thesis. However, it was found that if the number of elements was increased to 66, the λ_i minimal and mean values become 0.1601 and 0.4725, respectively, which is an improvement on Shaphard and Lo technique.

This result indicates that the bisection technique of mesh generation, presented in this thesis, is not a minimal grid generation method. However, Shephard and Lo method is of exponential computational growth, and the bisection mesh generator is a far more computational efficient technique.

8.3.3 Delaunay and Min-max triangulation.

In this section the two techniques, Min-max and Delaunay, of triangulating a set of points in E^3 are compared. The Min-max triangulation technique forms an integral part of the bisection mesh generator presented in this thesis.

The method used to compare the two techniques, was to generate several sets of uniform randomly distribution nodes in E^3. The analysis was repeated several times to generate the information in Figure 8.3.5,and tables 8.3.2 and 8.3.3. The graph, in Figure 8.3.5, is a plot of the averages of the mean solid angles of meshing the sets of random uniform nodes. From the graph it can be observed that the min-max technique generates grids with a better average solid angle value than Delaunay triangulation.

Tables 8.3.2 and 8.3.3 were also generated to demonstrate that the min-max triangulation routine generates a grid with an overall better distribution of *solid angles* than the Delaunay technique. These tables present a break down of solid angle measurements in the grids for Delaunay and Min-max triangulation, for different number of nodes. It can be clearly seen from these tables that Delaunay generates more solid angles in the lower ranges than the Min-max technique.

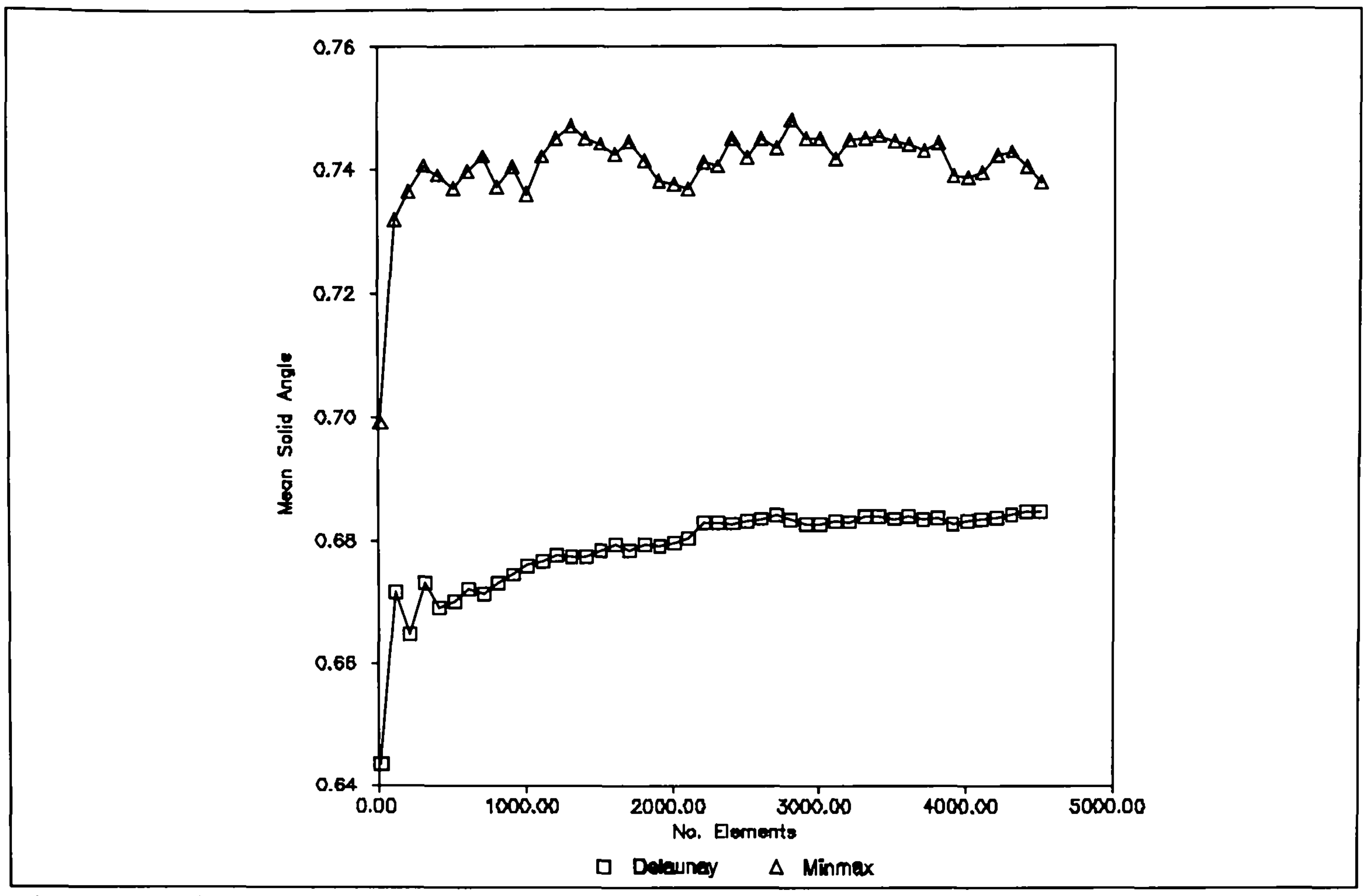


Figure 8.3.5 : Plot of mean solid angles of meshes, for *Delaunay* and *Local min-max* triangulation routines for various size grids.

Table 8.3.2 : Break down of Solid angle ranges for Delaunay and Minmax triangulation.

No. Nodes	10		110		510	
Range	Del.	Minmax	Del.	Minmax	Del.	Minmax
0.00 - 0.31	0.45	0.09	0.46	0.29	0.47	0.29
0.31 - 0.63	0.27	0.00	0.13	0.06	0.14	0.05
0.63 - 0.94	0.09	0.00	0.03	0.02	0.02	0.01
0.94 - 1.26	0.00	0.00	0.01	0.00	0.01	0.00
1.26 - 1.57	0.00	0.00	0.00	0.00	0.00	0.00
1.57 - 1.88	0.00	0.60	0.00	0.44	0.00	0.42
1.88 - 2.20	0.10	0.20	0.30	0.12	0.31	0.15
2.20 - 2.51	0.00	0.10	0.07	0.03	0.06	0.03
2.51 - 2.83	0.00	0.00	0.03	0.01	0.02	0.01
2.83 - 3.14	0.00	0.00	0.00	0.00	0.00	0.00

Table 8.3.3 : Break down of solid angle ranges for Delaunay and Minmax triangulation.

No. Elements	1010		2020		4010	
Range	Del.	Minmax	Del.	Minmax	Del.	Minmax
0.00 - 0.31	0.45	0.31	0.46	0.30	0.45	0.30
0.31 - 0.63	0.13	0.06	0.14	0.06	0.14	0.06
0.63 - 0.94	0.02	0.01	0.03	0.01	0.03	0.01
0.94 - 1.26	0.01	0.00	0.01	0.00	0.01	0.00
1.26 - 1.57	0.00	0.00	0.00	0.00	0.00	0.00
1.57 - 1.88	0.00	0.46	0.00	0.48	0.00	0.42
1.88 - 2.20	0.28	0.14	0.26	0.12	0.30	0.15
2.20 - 2.51	0.06	0.03	0.06	0.04	0.07	0.03
2.51 - 2.83	0.02	0.02	0.02	0.02	0.02	0.02
2.83 - 3.14	0.00	0.00	0.00	0.00	0.00	0.00

8.3.4 Finite Octree mesh generation.

In the paper [ShG91] on the technique of Finite Octree grid generation by Shephard and Georges, there are several meshes presented with minimum and maximum dihedral angle measurements. The technique which they presented was based on an Octree division of the domain, and the meshing of each octane region was achieved by *binary mesh* operators. The method is quoted to be of computational order *nlog(n)*, however in practice the method exhibits a linear computational growth [ShG91].

The geometry in Figure 8.3.6, is an example taken from the paper by Shephard and Georges [ShG91], which they used to demonstrate the computational efficiency of their technique. This domain has been applied to the bisection mesh generator, Figure 8.3.7, and the graph of *normalized CPU* is presented in Figure 8.3.8. The graph, in Figure 8.3.8, also has the normalized CPU plotted for the *Finite Octree technique*, from the paper by Shephard and Georges [ShG91].

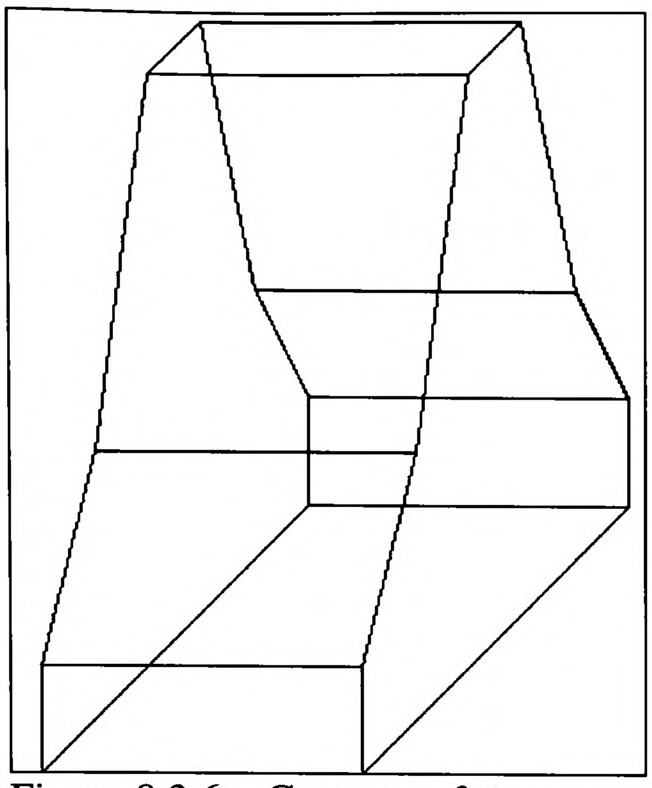


Figure 8.3.6: Geometry from paper on Finite Octree technique [ShG91].

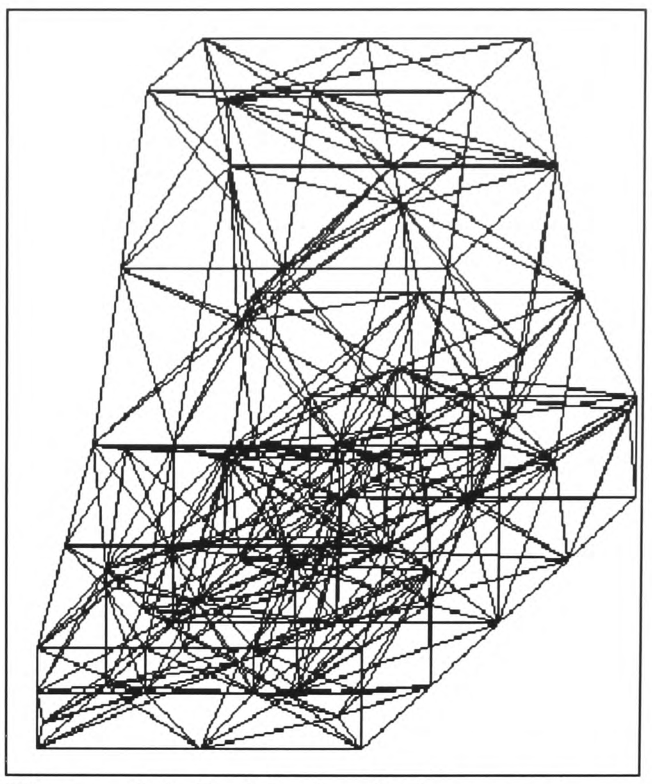


Figure 8.3.7: Mesh of Geometry.

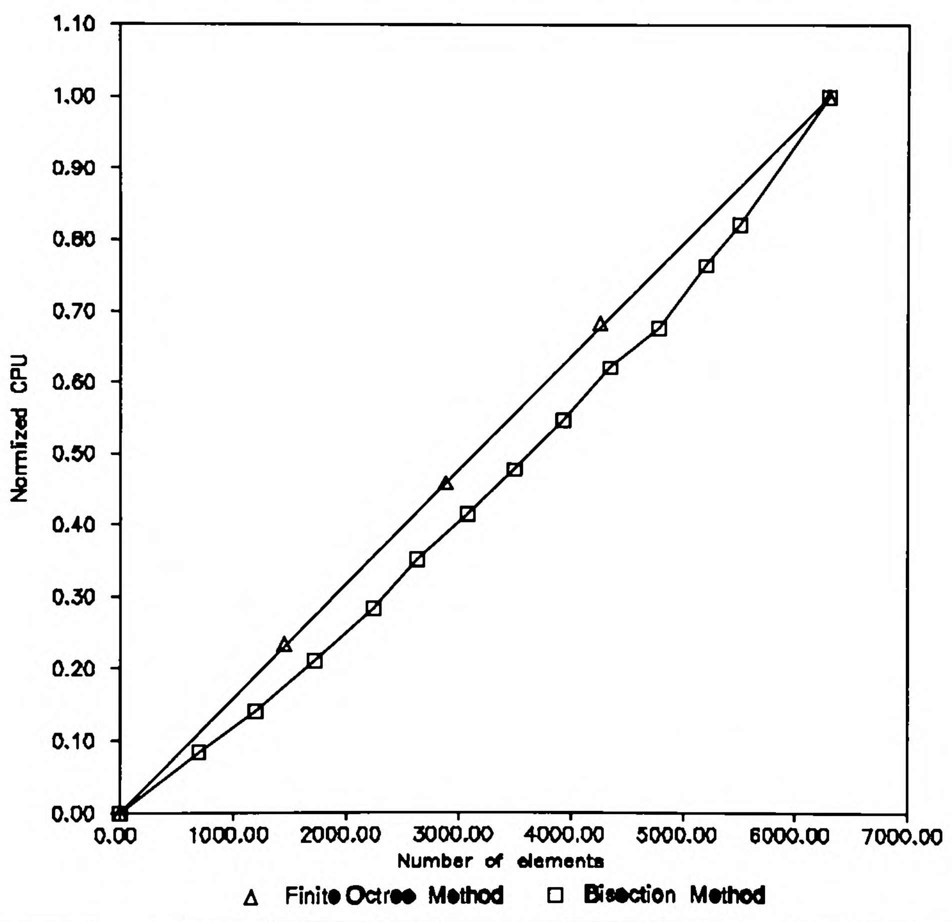


Figure 8.3.8: Plot of normalized CPU time for Finite Octree and Bisection mesh generators.

The above Figure 8.3.8, shows that the Finite Octree technique from the paper [ShG91] exhibits a far more linear characteristic for this problem. However, the profile of the computational efficiency of the bisection method is near linear for this problem. Other examples in this chapter show a more linear CPU speed up, see Figure 8.4.13, section 8.4.

A further example, presented for the Finite Octree technique, is depicted in Figure 8.3.9, which generated a mesh of 71 elements. In the paper by Shephard and Georges [ShG91], they give the maximum and minimal dihedral angles for the mesh of the geometry of Figure 8.3.9, generated by the Finite Octree technique, of 26 and 124 degrees, respectively. This geometry was applied to the bisection mesh generator, Figure 8.3.10, and generated 73 elements with a minimal and maximal dihedral angles of 17 and 130 degrees.

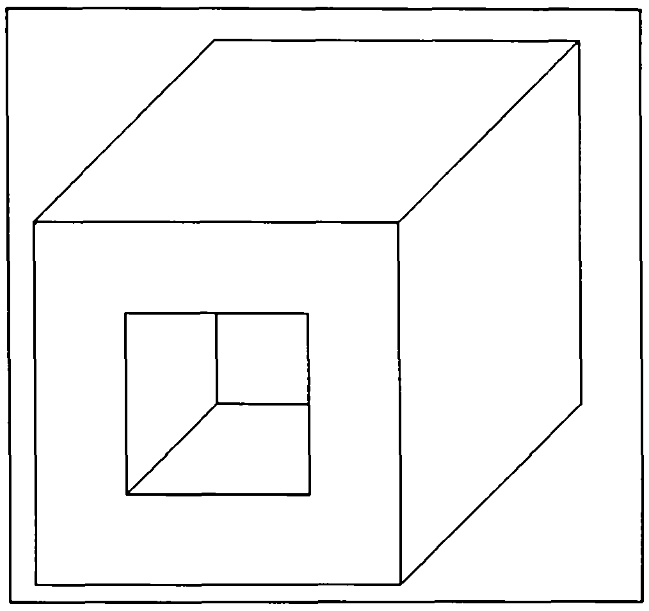


Figure 8.3.9 Example Geometry.

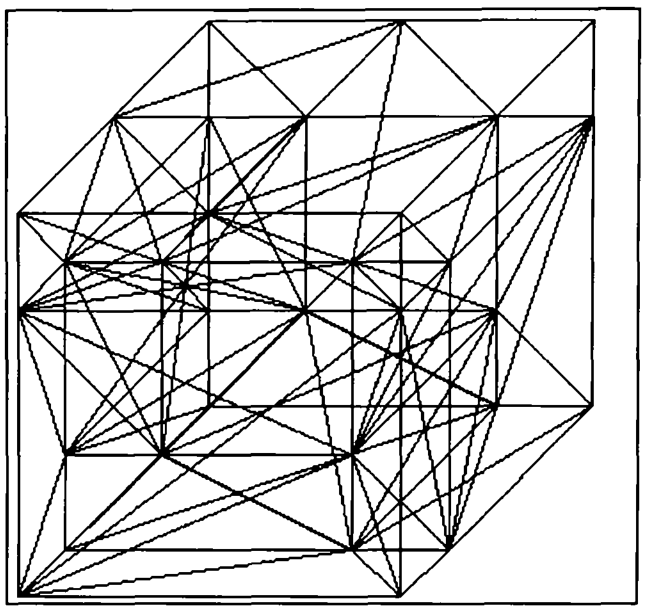


Figure 8.3.10: Mesh of 73 Elements.

From these two examples, it seems that the Octree technique is slightly better in mesh quality and computational efficiency. However, the data on computational efficiency on the Finite Octree technique, from the paper [ShG91], is inadequate, since no real CPU data timings were provided, and just quoting a minimal and maximum dihedral angle provides little information about overall mesh quality.

The two examples demonstrated in this section, to compare the bisection mesh generator, presented in this thesis, with the Finite Octree technique, provide some interesting results. The two techniques are very closely related in computation order and mesh quality. The Finite octree method implements a highly computational efficient octree division algorithm, with an order n^2 binary mesh operation technique in the sub-regions. Since the order n^2 algorithm is confined to very small subregions, the more computational efficient Octree algorithm dominates the overall computational order of the mesh generator. The Bisection mesh generation technique, implements a computational efficient bisection algorithm with an order *nlog(n)* meshing technique in the sub-regions, however these sub-regions are larger than the ones used in the Octree technique. Hence, more detail data is required on the Finite Octree mesh generator to achieve a better comparison between the two meshing techniques.

8.3.5 Summary.

The bisection mesh generator has preformed well against other techniques, documented in literature. These results are especially encouraging, if you consider that these test geometries are taken from others papers. Therefore, the test geometries have not been adaptive to suit the 3D Bisection mesh generator. The reader should also note that the 3D Bisection mesh generator has not been manipulated to suit each test geometry.

As an example, of how a mesh generator can be manipulated, so that it generates a good quality grid for a particular geometry, we take the second example geometry in section 8.3.4, Figure 8.3.9. This geometry was used to compare the Bisection mesh generation technique with the Finite Octree method, by Shephard and Georges [ShG91]. Shephard and Georges method generated a grid with minimal and maximum dihedral angles of 26.0 and 124.6 degrees, respectively. The Bisection method generated a grid with minimal and maximal dihedral angles of 17 and 170.6 degrees. However, if the Bisection mesh generator is tweaked, by adjusting some constants within the mesh code, a comprehensive improvement in grid quality is achieved. This resulted in a mesh with minimal and maximum dihedral angles of 30 and 120 degrees, with 76 elements. Therefore, an improvement can be achieved, in grid quality, for a particular geometry, by adjusting parameters in the automatic mesh generator.

The problem of comparing mesh generation techniques is a highly complex area, and there is the need for some standard bench marks (geometries). These bench marks should be accompanied with some clearly defined criteria set out on how to compare the results from different grid generators, and what forms a quality mesh over these test geometries. Many of these problems have been discussed in the paper by M.Sabin [Sab91], *Criteria for comparison of automatic mesh generation methods.*

8.4 Further examples.

This section presents some further examples to which the mesh generator has been applied to. These examples do not present any new features, which have not already been discussed in the previous section. However, they do demonstrate that the mesh generator has been applied successfully to a wide range of geometries. This section also provides further examples to re-enforce the ideas discussed in the previous sections.

8.4.1 Example 1, simple plane model.

This example demonstrates the mesh generators ability to cope with typical three dimensional geometries, such as a full three dimensional model of a plane.

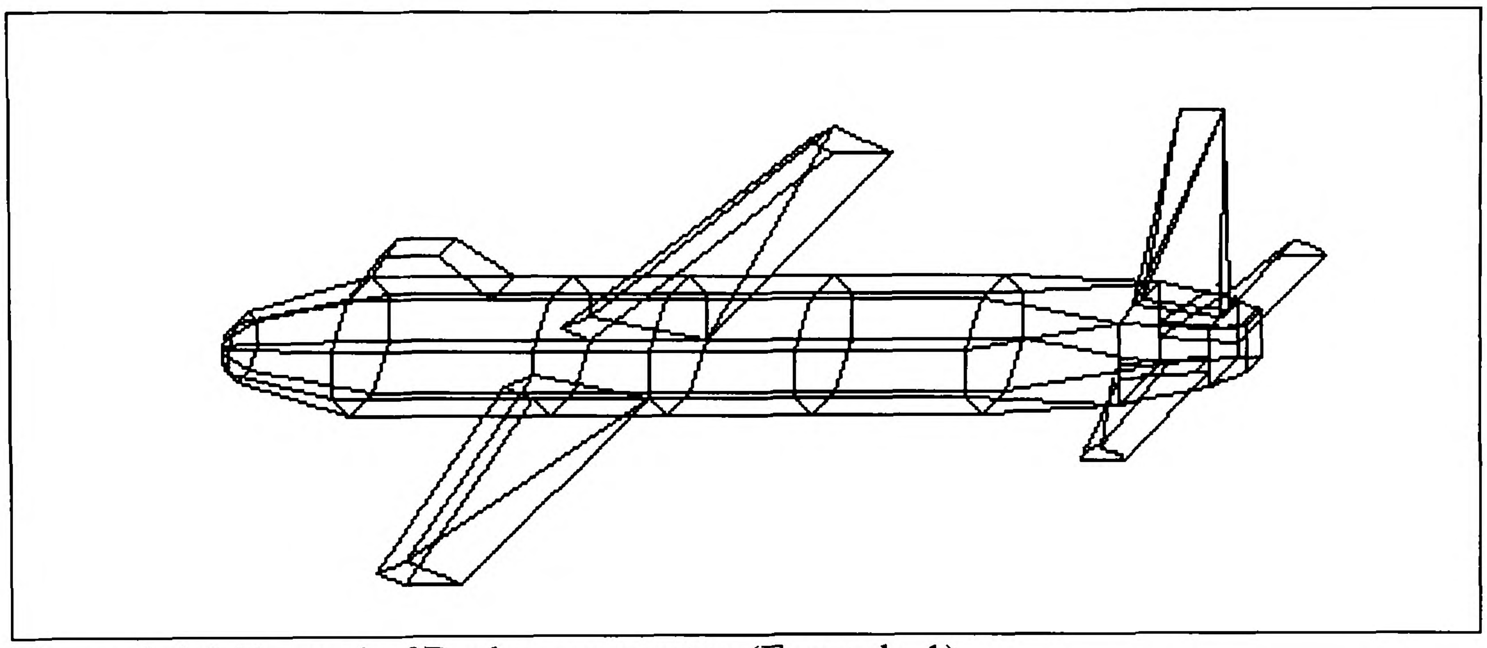


Figure 8.4.1 Example 3D plane geometry (Example 1)

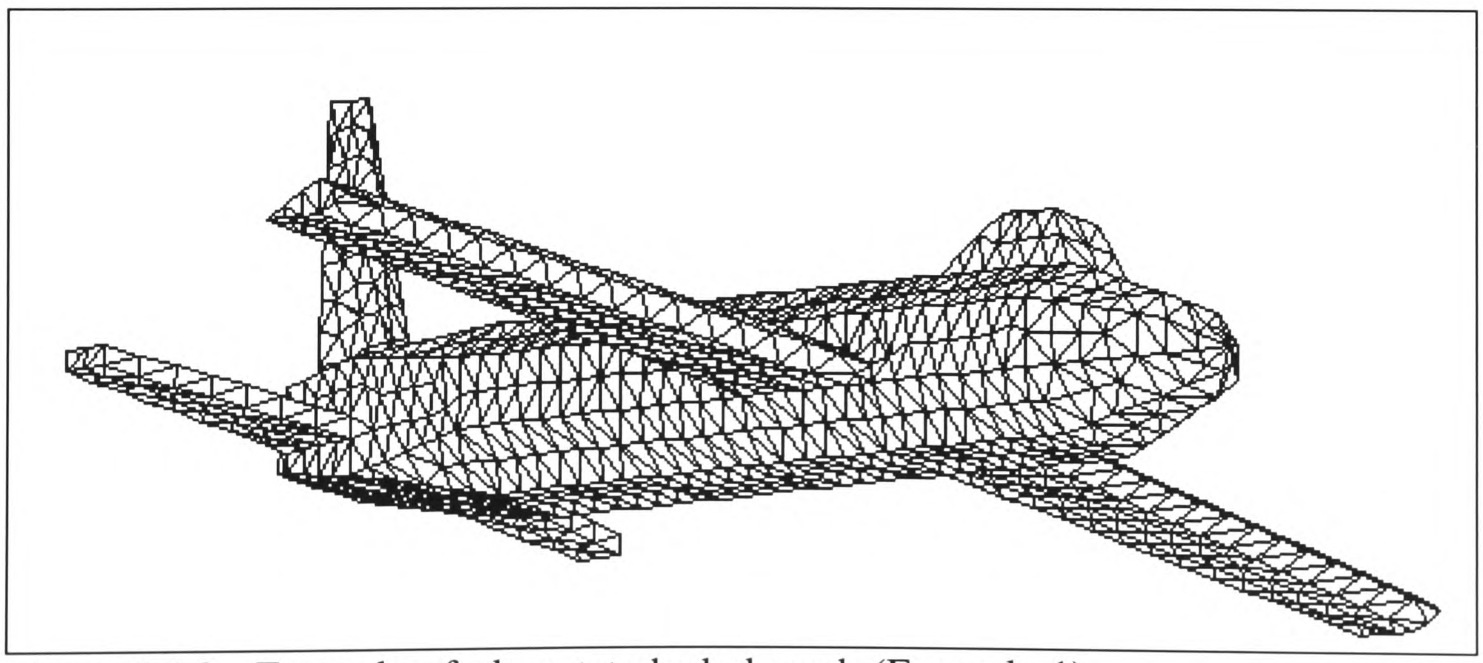


Figure 8.4.2 . Example of plane tetrahedral mesh (Example 1).

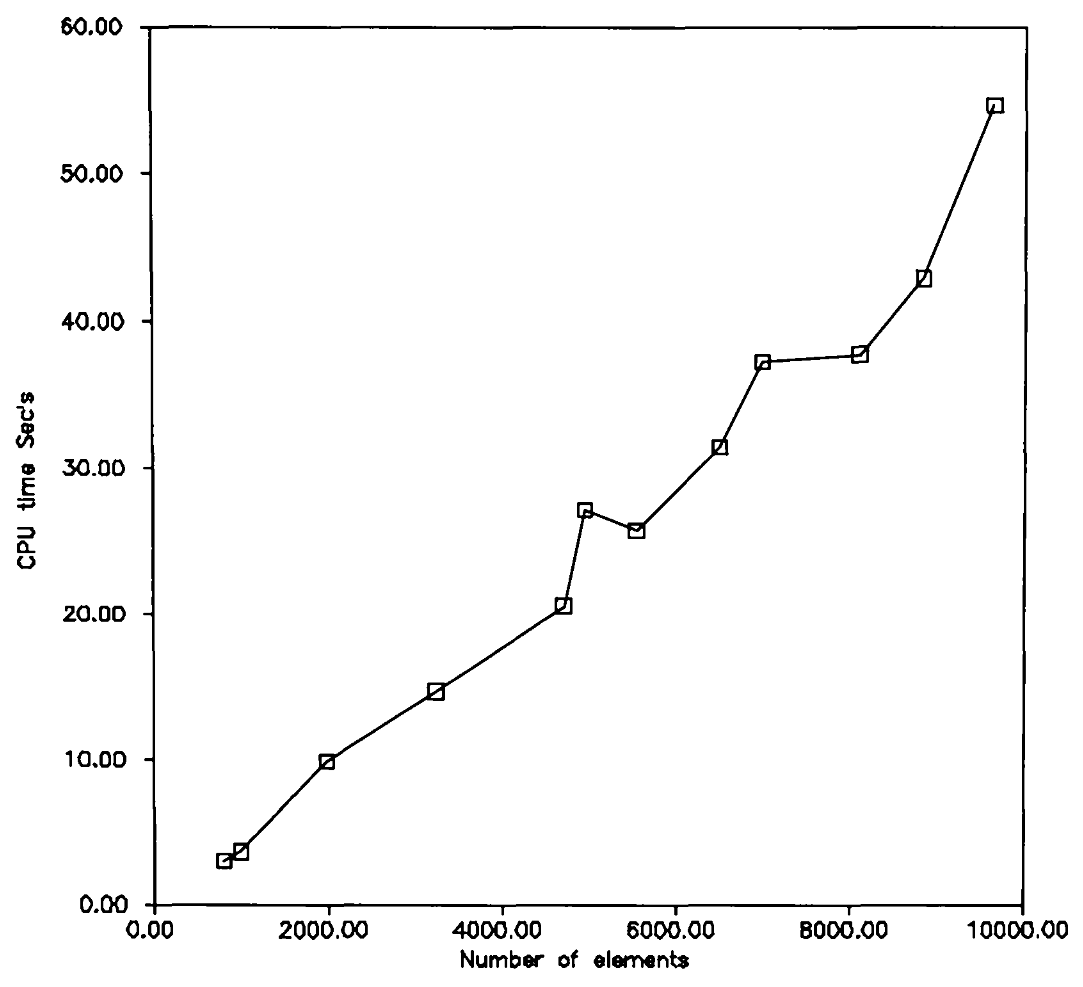


Figure 8.4.3 : Number of elements generated against CPU time (Example 1).

Table 8.4.1 : Break down of CPU use in the code for example 1 section 8.4.

Elements	Nodes	Init Geom.	Bisect	Gen. Nodes	Gen. Tets	Total CPU
959	306	0.025	0.4	1.3	1.23	2.955
3124	843	0.022	0.42	1.9	9.25	11.592
10194	2403	0.021	0.4	2.46	41.05	43.931
18197	4018	0.023	0.45	3.03	91.23	94.733

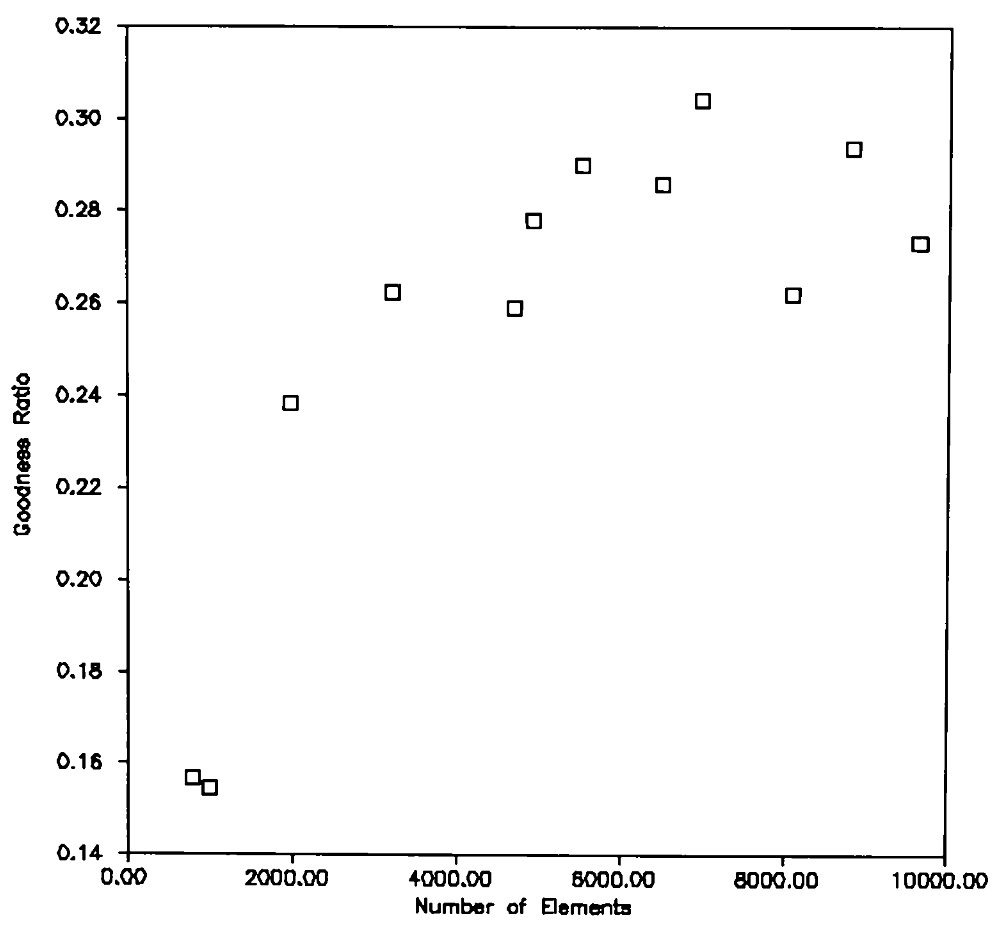


Figure 8.6.4 : Number of elements against mean goodness factor (Example 1).

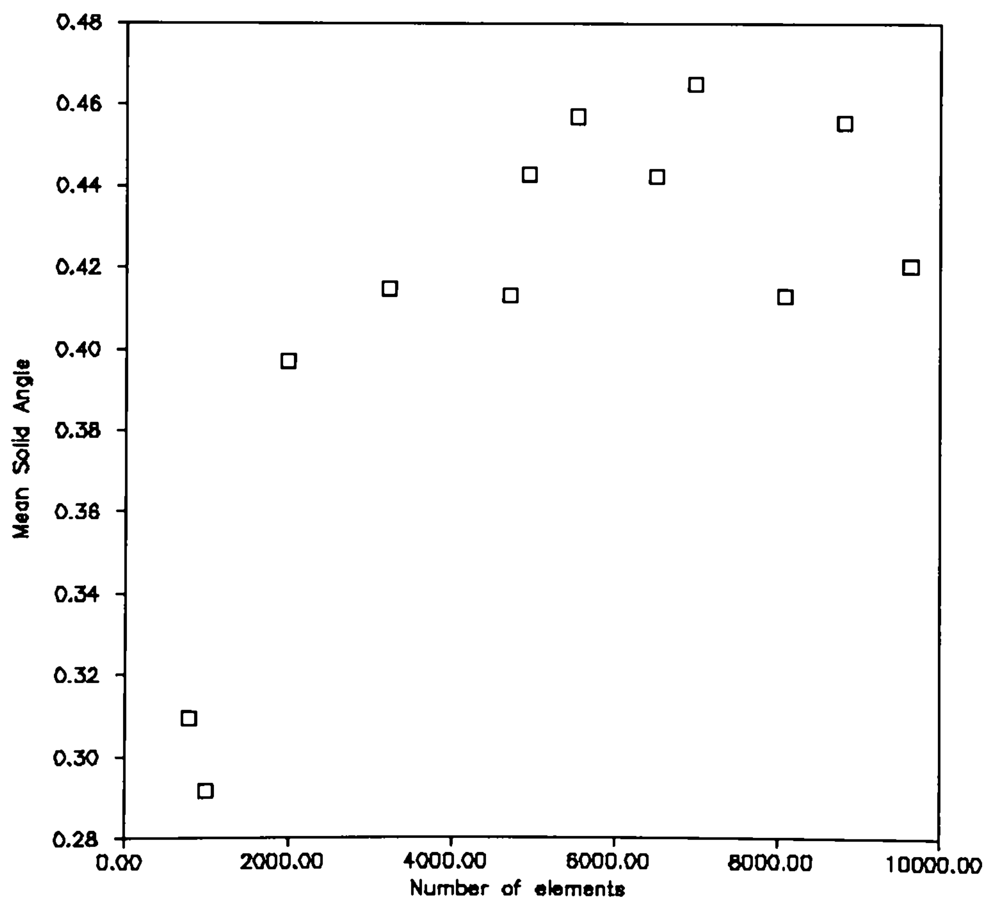


Figure 8.4.5 Number of elements generated against mean solid angle (Example 1).

8.4.2 Example 2, car model.

This car example shows the mesh generators ability to cope with sallow curves, such as on the sides of the car model.

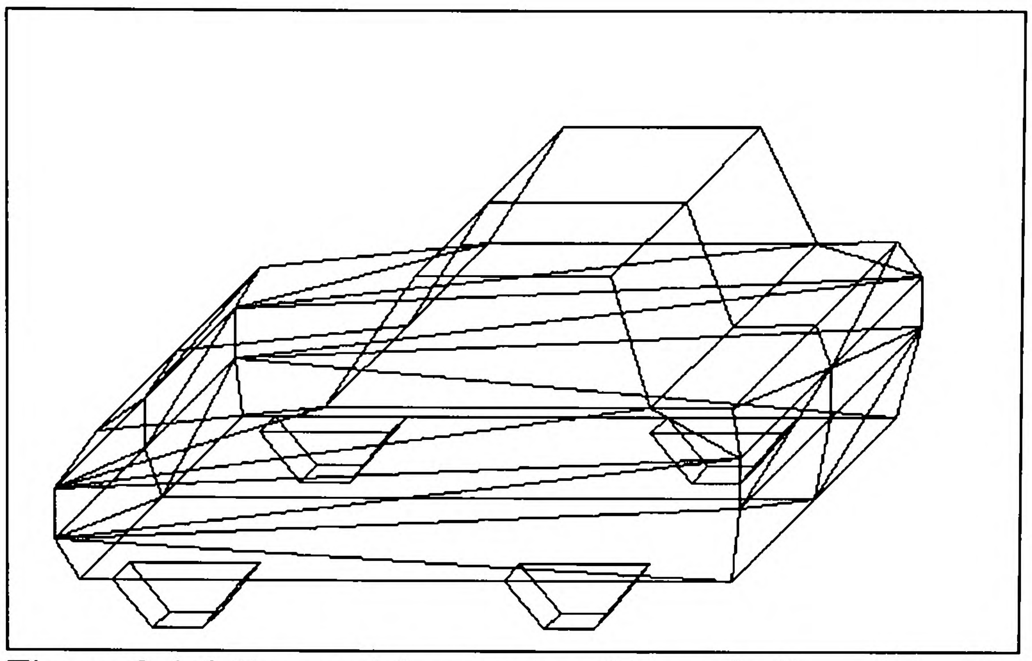


Figure 8.4.6 Car model geometry (Example 2).

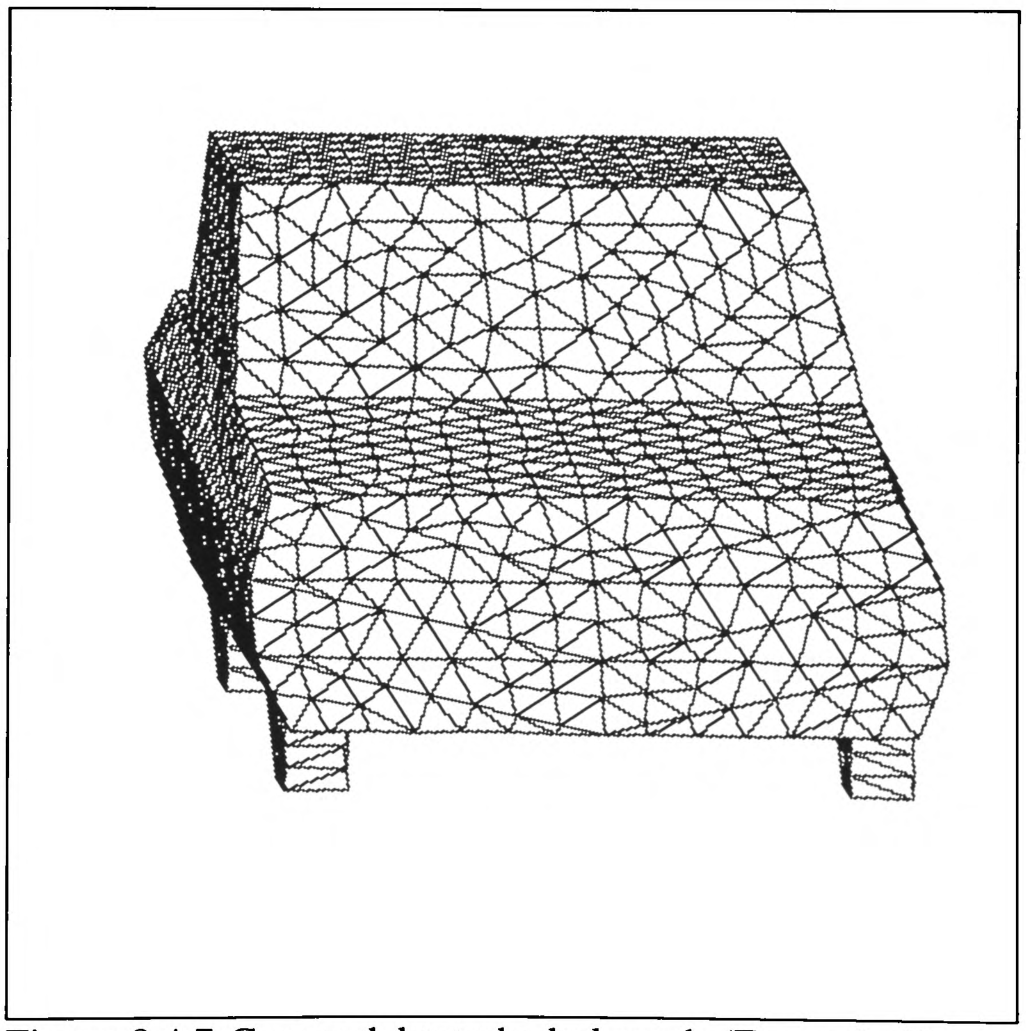


Figure 8.4.7 Car model tetrahedral mesh (Example 2).

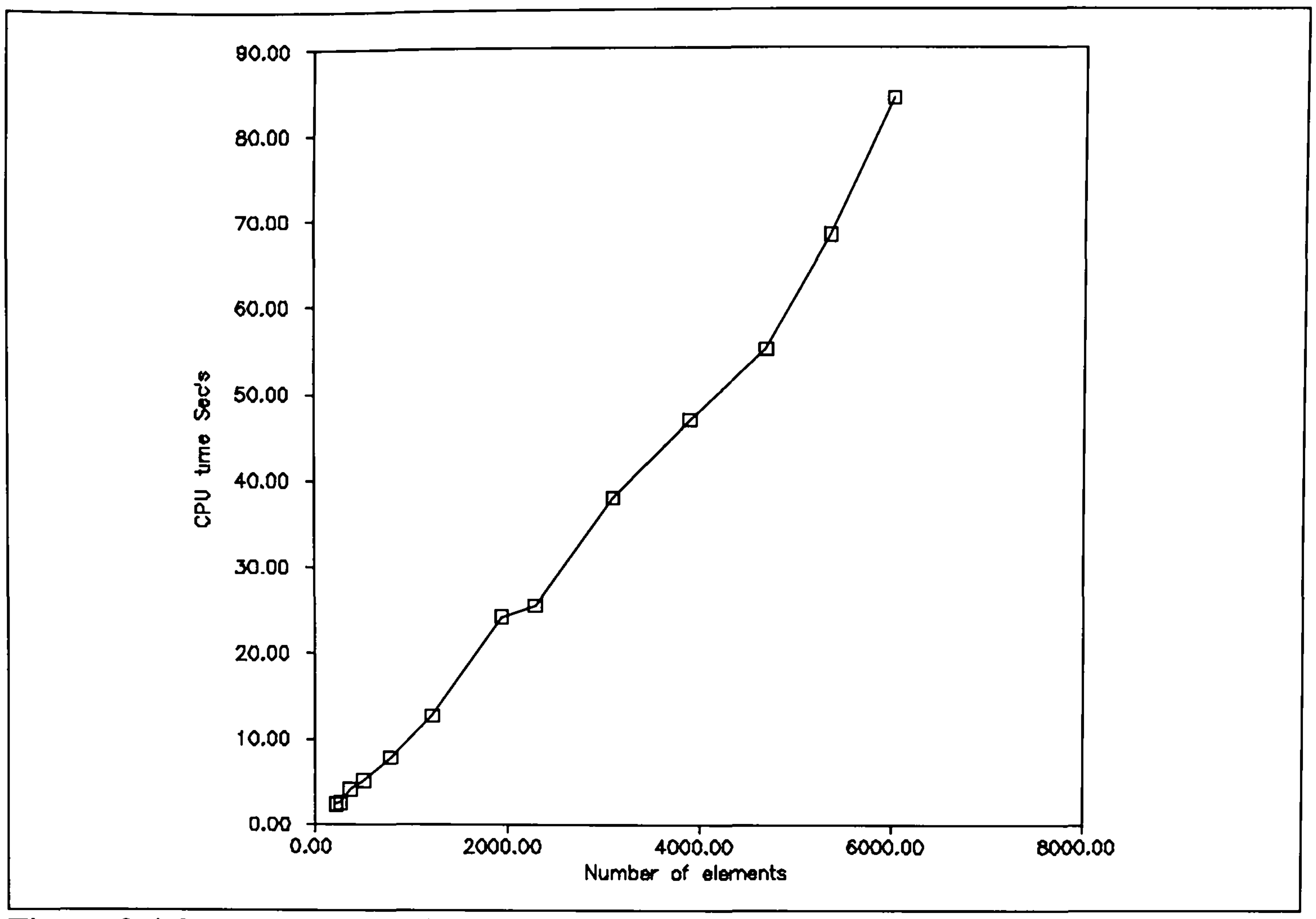


Figure 8.4.8 : Number of elements generated against CPU time (Example 2).

Table 8.4.2 : Break down of CPU use, for the generation of car mesh.

Elements	Nodes	Init. Geom.	Bisect	Gen. Nodes	Gen. Tets	Total CPU
1048	326	0.02	0.16	1.48	1.92	3.612
3878	1024	0.02	0.1	2.78	9.58	12.51
5148	1304	0.02	0.1	3.34	13.16	16.646
6538	1616	0.02	0.14	3.86	17.6	21.652

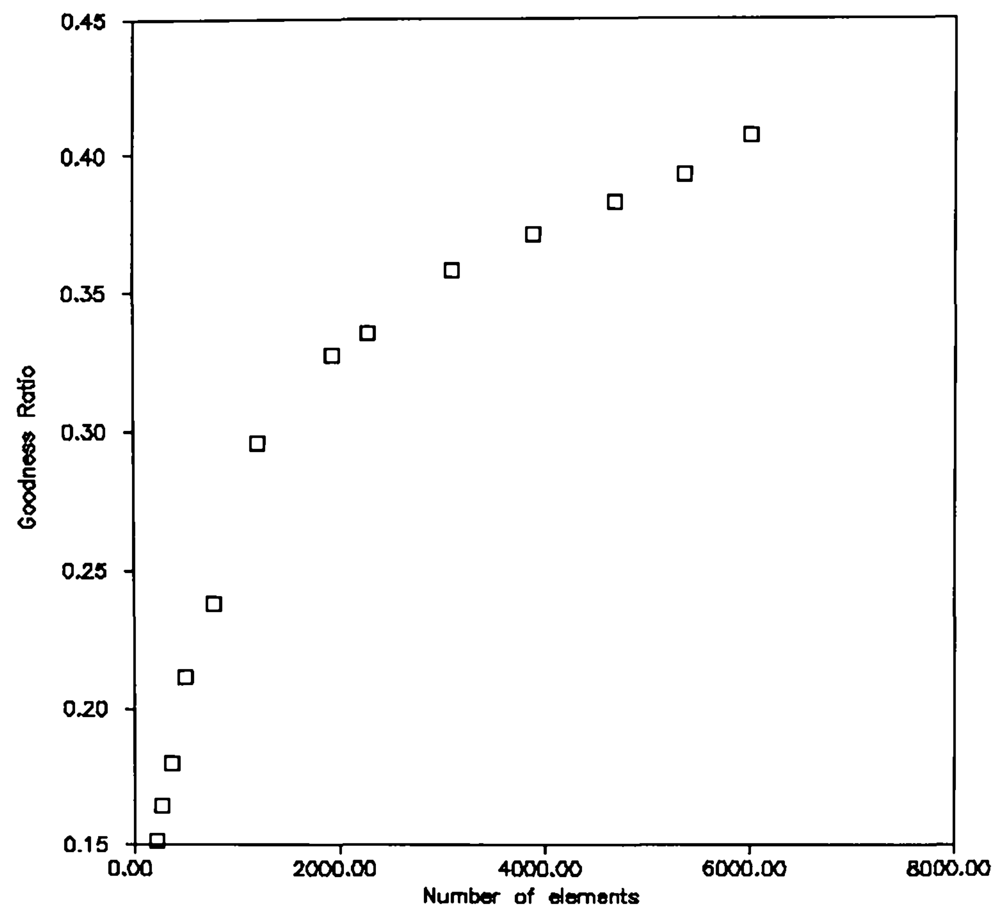


Figure 8.4.9 : Number of elements against mean goodness factor (Example 2).

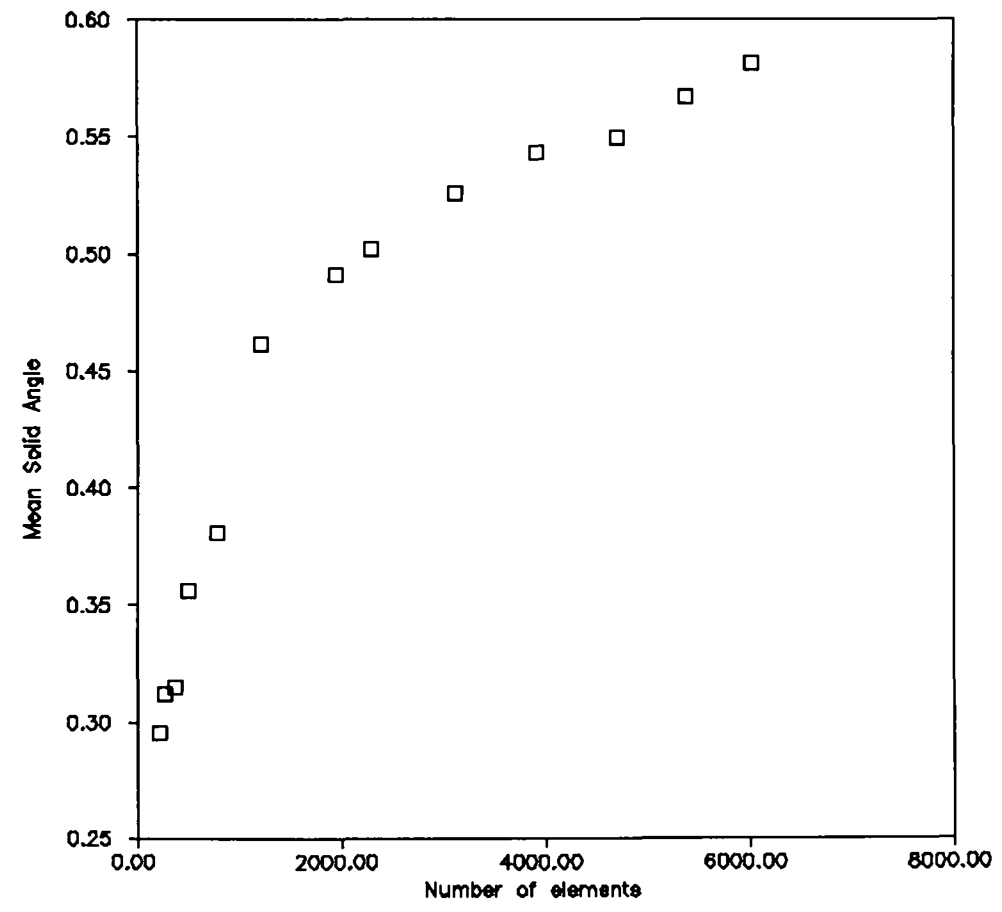


Figure 8.4.10 Number of elements generated against mean solid angle (Example 2).

8.4.3 Example 3, Cross section of car in a wind tunnel.

This car in a tunnel geometry is similar to the plane in a wind tunnel, Example 4 section 8.2.2, and shows a further example of a complex polyhedral face existing in one side of the model. The complex face is where the car body makes a hole in the outer hexahedral box.

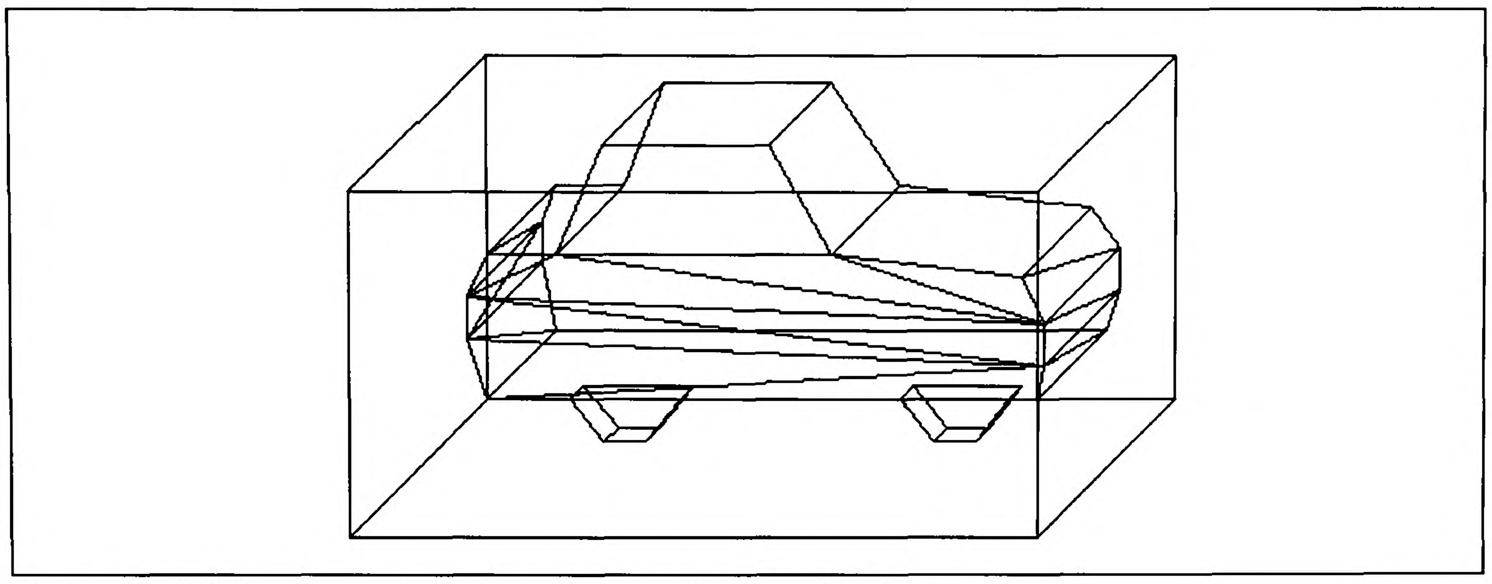


Figure 8.8.1 Car in a tunnel geometry (Example 3).

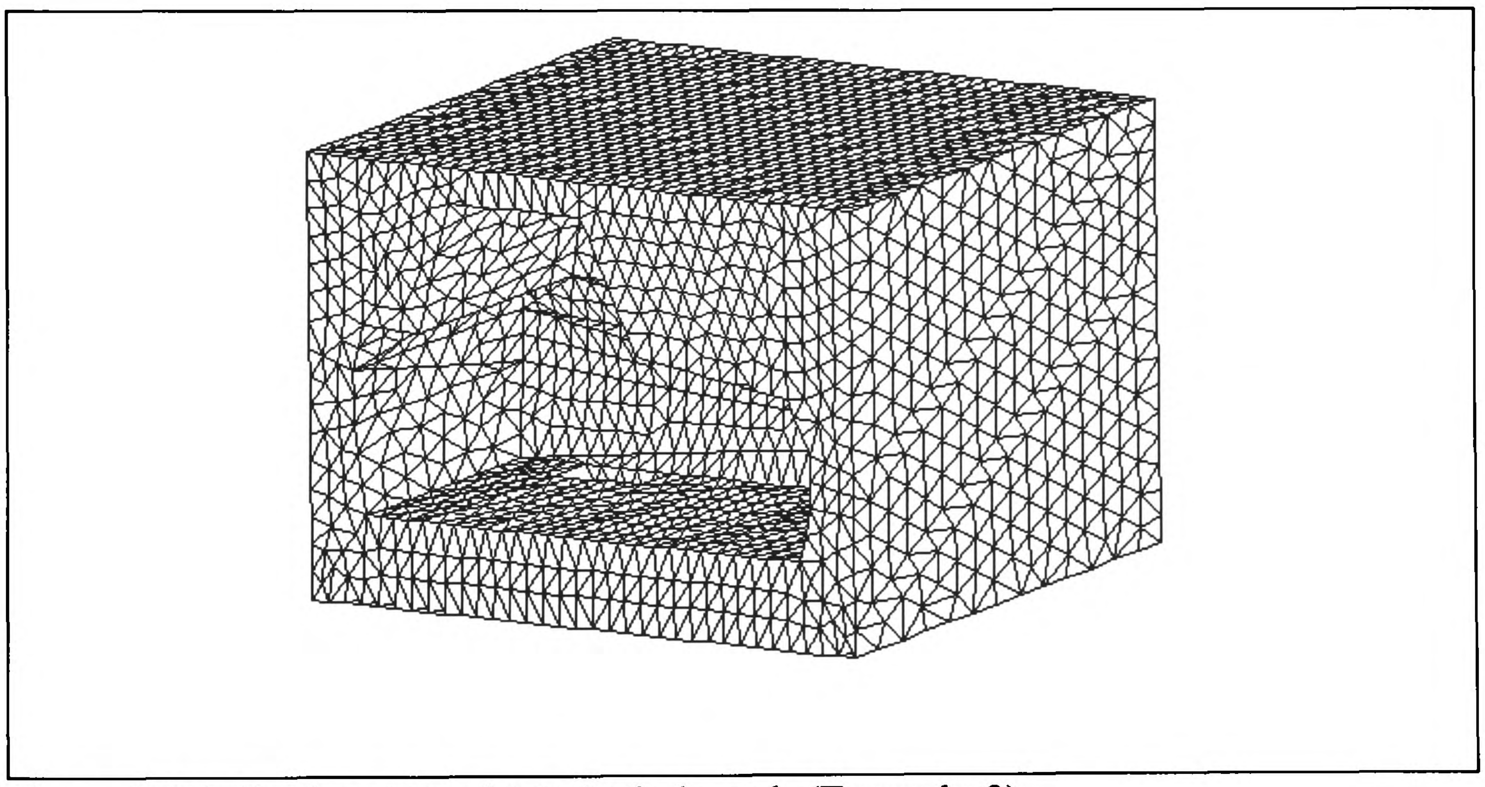


Figure 8.8.1 Car in a tunnel tetrahedral mesh (Example 3).

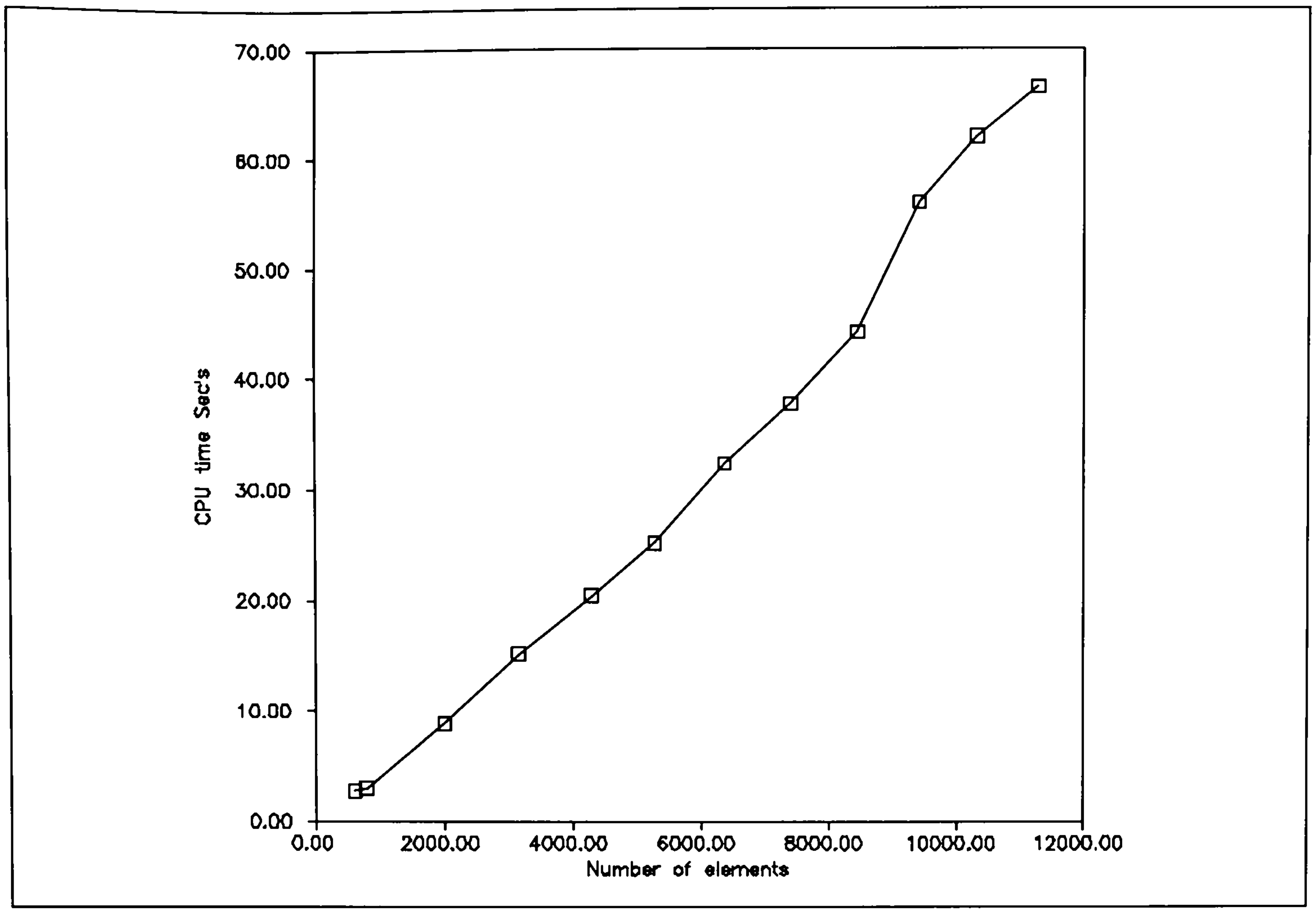


Figure 8.4.13 : Number of elements generated against CPU time (Example 3).

Table 8.4.3 : Mesh generation CPU use for car in tunnel example.

Elements	Nodes	Init. Geom.	Bisect	Gen. Nodes	Gen. Tets	Total CPU
1064	294	0.029	0.2	1.46	1.17	2.859
3375	838	0.03	0.13	3.51	4.92	8.59
4449	1086	0.031	0.18	4.23	6.9	11.341
5888	1437	0.03	0.17	5.37	10.27	15.84

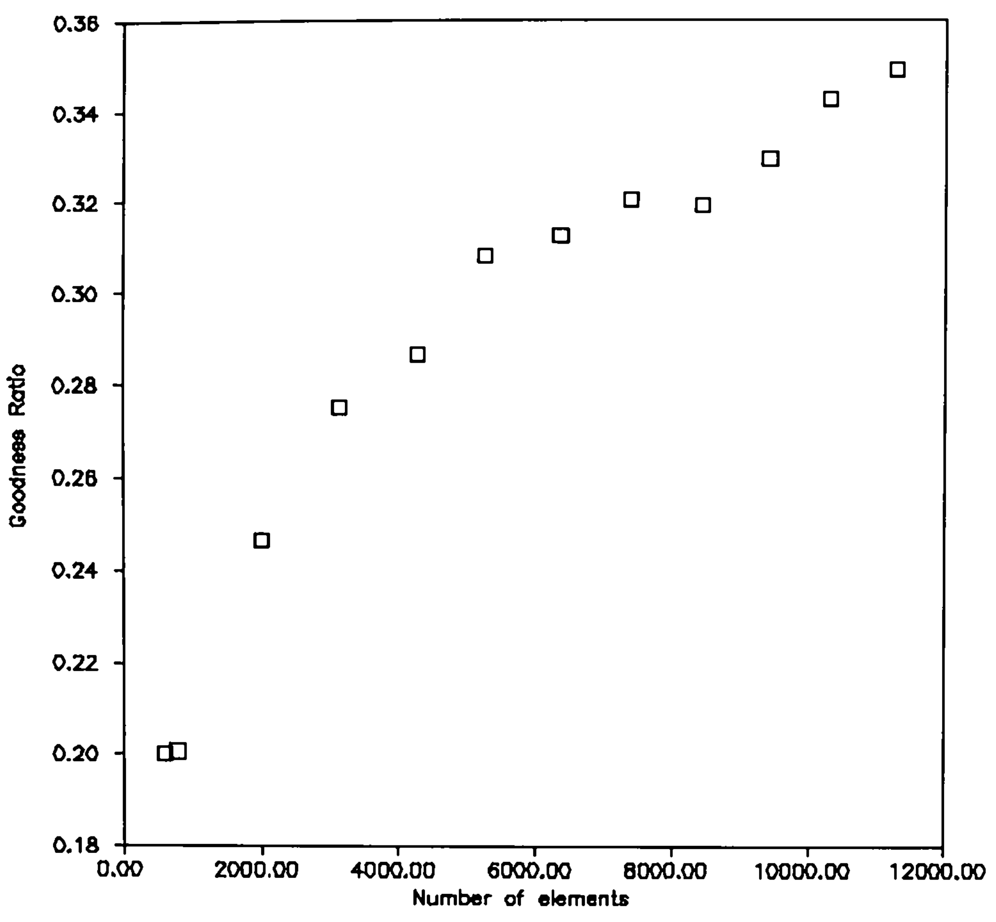


Figure 8.4.14 : Number of elements against mean goodness factor (Example 3).

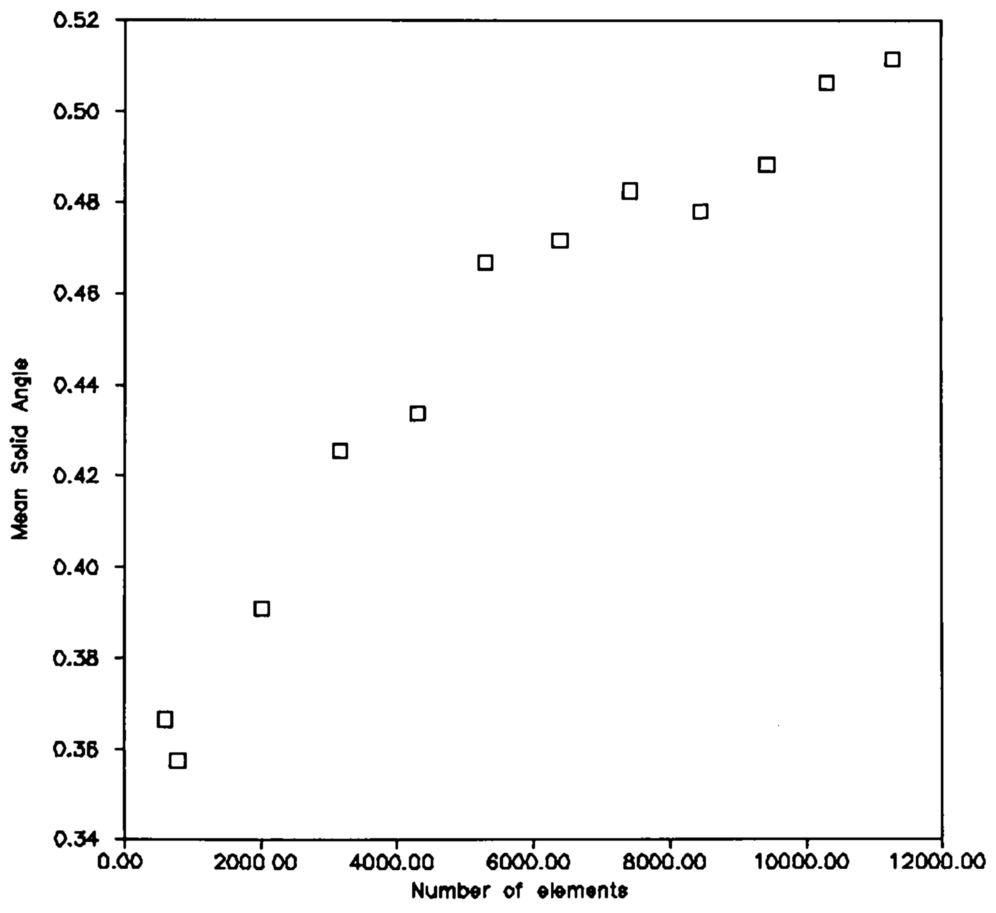


Figure 8.4.15 Number of elements generated against mean solid angle.

8.9 Discussion of results.

The graphs of CPU time against mesh size demostrate a good comparable *linear* relationship. Therefore, any high order algorithms within the code are swamped by the linear components, which gives the overall computational profile. If the CPU time is broken down into parts, the majority of the time is accounted for in the triangulation routine. The nodal placement is the next most CPU intensive, with the bisection of the domain into disjointed parts being a minor component.

The element mean quality measurements indicate that the element aspect ratio is poor for coarse grids. However, there is a general steep increase initial in tetrahedral shape measurements, that then tails off sharply as the density of the grid increases. The rate at which the element shape indicators improve from the initial coarse triangulation are more sharply notable for simpler geometries. We can then conclude that the element's aspect ratio is govern by the geometry of the model for coarse grids and improves as the number of tetrahedra increase. This is a desirable feature, as increasing the number of elements should reduce the dependency of the elements quality on the geometry.

The bisection technique introduces new nodes on the surface of a model even when a surface mesh is provided. Therefore, the minimal triangulation achieved by this method is not necessarily the minimal possible triangulation, and is governed by the initial bisection of the geometry. However, when comparing the mesh generator with other comparable meshing techniques it performs well, and generates meshes of acceptable quality.

The results show that a computational efficient algorithm based on a bisection technique can generate grids of workable quality. However, this method cannot be described as an optimal minimal meshing technique, rather a method that can provide an initial mesh, for computational purposes, of a given density.

Chapter 9

Conclusions.

9.0 Introduction.

This chapter will give a brief overview of some of the conclusions of the research carried out in an attempt to generate a reliable three dimensional tetrahedral mesh generator. The importance of bisections to simplify models for grid generation will be discussed, together with major achievements. This chapter will then be concluded by making comparisons with other major techniques and a discussion of possible improvements and extensions.

9.1 Conclusions.

9.1.1 Preparation of geometry.

Preparation of the geometry is extremely important, not only to improve the speed of execution, but also to avoid problems with rounding errors. For example, the storing of face normals for each polyhedra face, enables a fast in-out test to be carried out, without the need for slow and unreliable ray testing [Rog85]. Also, when a face is bisected the need to calculate new face normals is eliminated as the old face normal can be duplicated for each new polyhedral face. This ensures that any faces generated from an initial polyhedral region have identical face normals, which are not affected by machine rounding errors.

Since the mesh generator's code was not designed for any particular CAD package, and to allow for the manual preparation of geometries, the package requires the minimal amount of information. This is the reason why the preparation of geometry and identification of key features, such as cavities of the model, is carried out in the initial stages of mesh generation. If there are any problems with the geometry of the model, this can be spotted before the actual mesh generation begins.

Another important area is user interaction. The user should have complete control over mesh generation, with the ability to step in and modify the grid at any point. However, the mesh generation code should be completely automatic with reasonable defaults. Hence the mesh generator should be capable of being directly integrated with a CAD package, and be fast enough to allow for user interaction.

9.1.2 Quality of algorithms and calculations

During this research the importance of quality and robustness of algorithms was found to be highly critical. Even for basic calculations, such as the intersection of lines and planes in 3D space, three or four different methods had to be evaluated. The main criteria for selecting a method for a routine, was based on its robustness to either accumulated rounding errors or points of singularity.

Most of the problems associated with any algorithm, within the code, were caused by the finite precision of the computer. For example in the bisection routine, it had to be ensured that any newly generated bisection plane did not introduce any ill-conditioned faces, near parallel lines or planes and surfaces with small angles between them. For a further discussion of finite precision problems, see Appendix A5, Numerical precision.

9.1.3 The importance of domain Bisection.

Bisection of a domain into simpler regions is the *key to mesh generation.* Many CAD packages require the user to divide the model up into simpler regions so that a grid can be generated. In many ways the key goal is the automation of the process of dividing a region up into simpler parts, so that it can be meshed using any standard meshing technique. For example, the bisection part of the technique, presented in this thesis, was used to sub-divide a region, so a code such as FEMGEN could be used to generate the final grid, see section 9.3 below.

The importance of domain decomposition is reflected in many major mesh generation projects that utilize techniques, which simplify geometry by sub-dividing domains, e.g Medial axis [TaA91], Octree Quadtree [ScS90].

9.2 Summary of achievements and major contributions.

The bisection of geometry for the generation of unstructured meshes has been demonstrated to be a practical and a valid technique. The technique of *Recursive mesh bisection* is an advantageous and computational efficient method of generating two dimensional unstructured triangular grids. This technique has been adapted to handle all 2D geometries without limitations and can generate high-order polygon elements.

It has also been established that many 2D element quality optimization techniques can be extended to surface mesh generation without affecting their computational order, and can produce acceptable results. The definition of surface Delaunay triangulation, described in Chapter 5, generates grids with good quality elements in order *nlogn*, whilst taking account of the surface approximation error.

The subdivision of 3D geometry is a highly complex task, and the order in which bisections are applied to a domain can have a dramatic effect on the final quality of the grid. This thesis has demonstrated, that with the careful selection of bisection planes, a *Bisection Technique* of generating three dimensional meshes of good quality is practical and computationally efficient. This research has also demonstrated that applying a direct local min-max routine can be as effective as the 3D Delaunay algorithm for computational order, and has the advantage of not producing any degenerate tetrahedra.

9.2.1 Comparison with other methods.

It is difficult to compare the grid quality of the new bisection mesh generation technique with other existing methods, since there seems to be no general bench marks available [Sab91]. The first three examples of 3D grids in Chapter 8, being simple and easy to reproduce, form ideal bench marks. There is the need for more complex bench marks, but these three basic geometries, cavity box, cavity box adjacent to another cube and lug with circler hole region, form a set of basic requirements of any meshing algorithm. Also these examples are presented with two different element shape measures, which should make it easy for others to compare their algorithms with the results presented in this dissertation.

However, from the literature available [CFF85][Joe91][ShL91][YTH91], the new bisection method's results compare favourably for element quality, see Chapter 8. Comparison of the algorithms, in this dissertation, with reported computational [CFF85] [Löh85][ScS89][Joe91] time order of alternative methods, indicate that the routines presented are amongst the best.

9.3 Further work and enhancements.

This section outlines some possible enhancement and improvements to the grid generator. However, any enhancements to the code must be carried out in conjunction with continuing work on improving the reliability, optimization and updating of the algorithms.

9.3.1 Curved surface mesh generation.

The initial attempts at surface meshing suffered from the problem of poor curve distance estimating routines, which did not allow the use of the *convex polygon shrinking nodal placement* technique. Therefore, the preliminary nodal insertion method of section 4.5.2 was implemented. Although this resulted in poor surface meshes, it did achieve reasonable angle quality. Hence a fast and reliable method of finding a point on a curve at a distance δ from a point p in the direction of a given vector n is required.

The bisection of a parametric surface into two or more independent parametric surfaces is a vital requirement. This would enable the bisection part of the mesh generator to be extended from just polyhedral domains to full boundary structures with parametric surfaces. However, this is a complex area that involves surface fitting and slope continuity [BaM91].

9.3.2 Combining the two bisection methods.

The bisection part of the code implements two different bisection techniques. One method bisects the region with a planar cut, where the other technique follows the outer contours of the domain and generates a non-planar bisection. The first method is preferred since it does not add a complex interface, just one extra polyhedral face into the model. Therefore, this bisection will have less effect on any subsequent bisections of the domain. The edge following bisection introduces a complex surface into the model, with many polyhedral faces, which have to be considered by any proceeding task.

A relaxation parameter could be introduced into the *edge following* technique, that will restrict the extent to which the bisection cut deviates from the plane. This relaxation parameter could then be varied according to the complexity of the geometry. An interface mesh could then be generated, and groups of planar elements joined to form one planar polyhedral face. The identification of groups of planar interface elements would make the first bisection algorithm obsolete. It should also be possible to set the relaxation parameter to zero, in the second algorithm, which should result in an identical bisection to the first bisection algorithm.

9.3.3 Hexahedra mesh generation.

Once a domain is divided up into several elementary regions, hexahedra elements could then be generated in a similar way to Medial axis mesh generation. For instance, when generating a hexahedral mesh, this could be done by generating quadrilaterals on the surfaces of all *convex polyhedral* faces. Then using algebraic methods, hexahedral elements could then be generated within convex regions.

The following examples presents some methods of generating quadrilateral meshes, using a technique of reducing the domain complexity to simpler regions. The domain (Figure 9.3.1) is first simplified, using the algorithms used in the mesh generator, into simply connect regions, Figure 9.3.2. The domain can then be further divided into convex regions, Figure 9.3.3, and then meshed using parametric mesh generation tools, to generate the final quadrilateral mesh, Figure 9.3.4

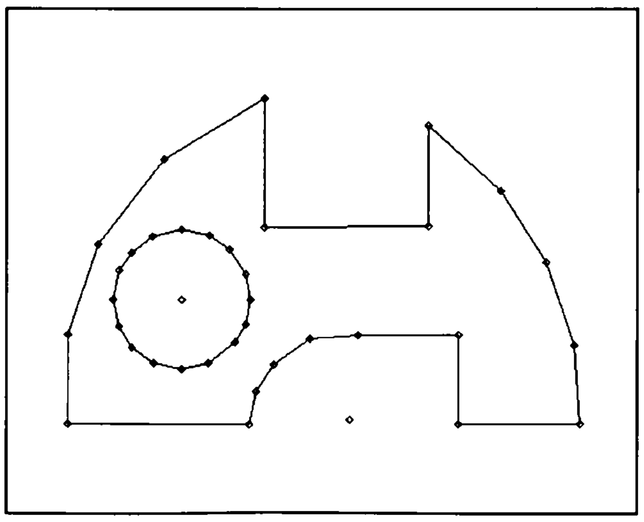


Figure 9.3.1: Geometry of problem.

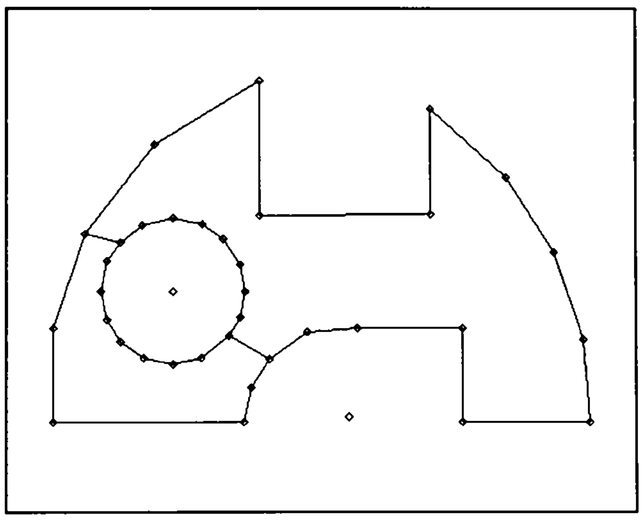


Figure 9.3.2: Division into simply connected regions

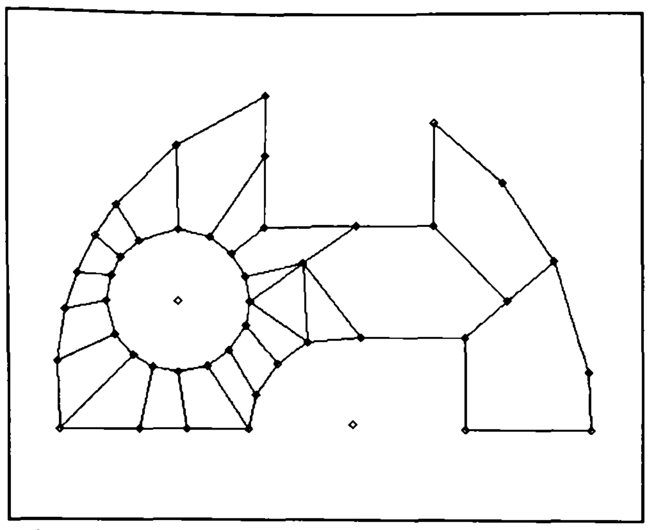


Figure 9.3.3 : Convex division.

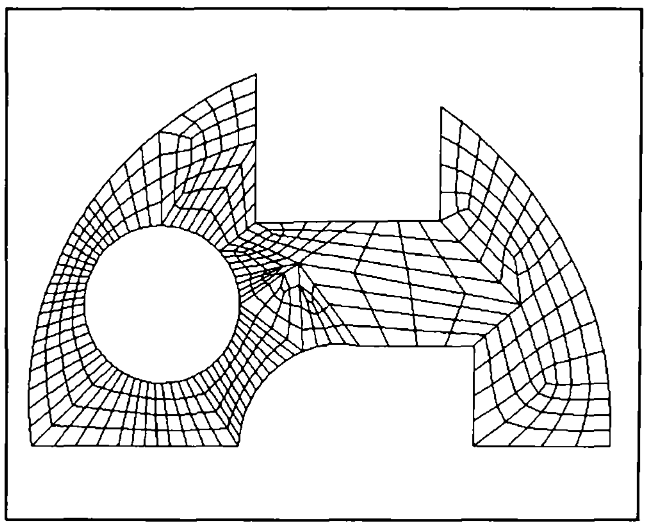


Figure 9.3.4: Quadrilateral mesh.

The bisection of the geometry does not have to be strictly convex, Figure 9.3.5. The subdivision in Figure 9.3.5 was obtained as follows:

```
(1) Place all simply connected polygons on the stack $S$.
(2) Take the next polygon off the top of the stack $S$.
(3)         Choose an initial node $i$, at random, on the boundary of the polygon.
(4)         Set the sum of angles $\alpha=0$
(5)         Let $\alpha=\alpha+\mathrm{MAX}\ (0, \Phi_i-180^\circ)$, where $\Phi_i$ is the angle at node $i$
(6)         If $\alpha> 44^\circ$ Then
                Subdivide polygon by introducing a bisection edge at node $I$.
                This bisection line should not introduce any reflex edges into the
                polygon.

(7)         Move to the next node adjacent to the node $i$ in an anti-clockwise
            direction, and let this become the new node $i$
(8) If all nodes are not processed in the current polygon, goto step 5.
(9) If the stack $S$ not empty goto step 2.
```

A quadrilateral mesh generated using the subdivision of the geometry in Figure 9.3.5 is given in Figure 9.3.6.

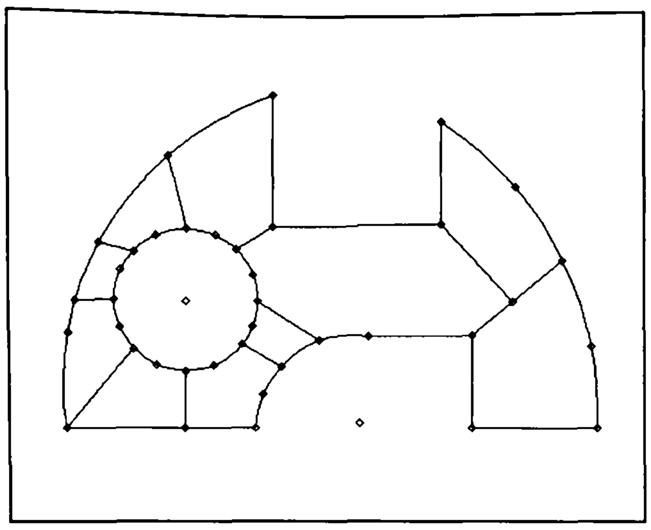


Figure 9.3.5: Parametric subdivision.

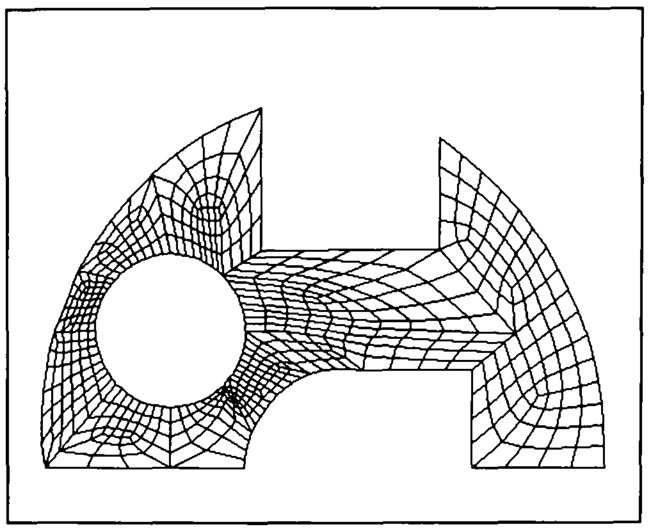


Figure 9.3.6: Quadrilateral mesh.

If the value at which we generate a bisection edge is increased from 40 to 80 degrees, in step 6 in the above algorithm, a geometry subdivision is generated as in Figure 9.3.7. The resulting mesh of the geometry sub-division in Figure 9.3.7, is presented in Figure 9.3.8.

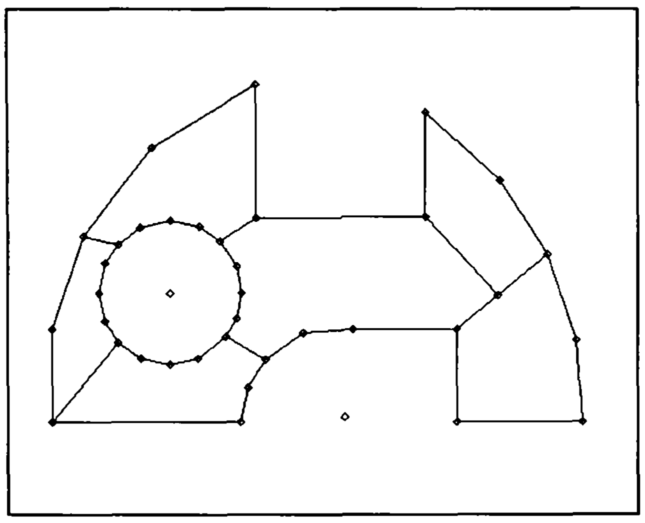


Figure 9.3.7 : Geometry subdivision

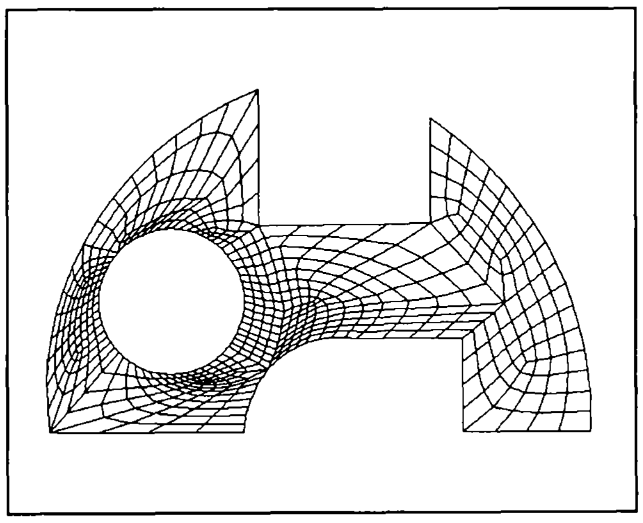


Figure 9.3.8: Quadrilateral mesh 3.

All the above geometries were bisected by algorithms implemented in the bisection mesh generator. The final meshes were achieved by importing the subdivided geometries into FEMGEN. The bisected geometries were meshed in *full automatic mode* in FEMGEN with line division 4, which accounts for the poor quality of nodal distribution in the final meshes.

9.3.4 Integration of a user nodal spacing on each sub-region.

Special meshing points in 3D space could be given a grid density requirement. The bisection method could then ensure that all mesh density control points are divided into individual regions. Any region without a mesh density point, could have a mesh density interpolated, using various nodal distribution functions, from regions that have nodal spacing specified within them. The mesh density of faces in sub-regions could then also be interpolated from the two mesh density points on either side. Once the boundaries have been triangulated the internal mesh density of each sub-region could be graded internally from the region's boundary triangulation.

The difficulty here is placing the mesh density control points. One way in which to give more control over grid generation, to the engineer designing the model, is to integrate the mesh generator directly with the CAD package. This should be done in such a manner as to let the engineer manipulate the geometry, to allow for the easy positioning of grid mesh control points. The ability to view and modify initial sub-division of geometry and to mark sections of the mesh for refinement or de-refinement is also required. Once the domain is divided up satisfactory the user should then be given several options to which methods and types of elements to generate over each region.

9.4 Final remark

The bisection method of generating three dimensional unstructured meshes, has been proven to be a most valuable technique with many path ways for further development. It is hope that this project will continue and expand into a fully commercial mesh generation tool, incorporated within appropriate software products.

9.4 REFERENCES

[BaB83] R.E. Barnhill and W. Boehm (1983), Surfaces in computer aided geometric design, North-Holland publishing.

[BaD89] C.L. Bajaj and T.K. Dey (1989), Robust decompositions of polyhedra, Foundations of Software Technology and Theoretical Computer Science, Ninth Conference Proceedings, Springer-Verlag, Pages 267-279.

[BaD90] C.L. Bajaj and T.K. Dey (1990), Polygon nesting and robustness, Information Processing Letters, Vol: 35, Iss. 1, Pages 23-32.

[BaD92] C.L. Bajaj and T.K. Dey (1992), Convex decomposition of polyhedra and robustness, SIAM Journal of Computing, Volume 21, Pages 339-364.

[Bak92] T.J. Baker (1992), Tetrahedral mesh generation by a constrained Delaunay triangulation, Proceedings IMACS '91, 13th World congress on computation and applied mathematics, July 22-26,1991 Trinity college Dublin Ireland, Vol. 1 , Pages 114-115.

[BaM91] C. Bajaj and K. Myung-Soo (1991), Convex hulls of objects bounded by algebraic curves, Algorithica, Vol. 6, Iss 4, Pages 533-553.

[Bay73] Bays, C. (1973), Some Techniques for structuring chained hash tables, Computer Journal 16:2 (May) Pages 126-131.

[Bau75] B. Baumgart (1975), A polyhedron representation for computer vision, In National Computer Conference, pages 589-596, AFIPS Conf. Proc.

[Bey81] W.H. Beyer (1981), CRC standard mathematical tables, 26th edition, CRC press Boca Raton Florida.

[BKK84] F.W. Burton, V.J. Kollias and J.G. Kollias (1984), Consistency in Point-in-Polygon tests, The computer journal, Vol 27, No. 4, Pages 375-376.

[BoP91] J. Bonet and J. Peraire (1991), An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems, International Journal for Numerical Methods in Engineering, Vol. 31, Pages 1-17.

[Bow81] A. Bowyer (1981), Computing Dirichlet tessellations, The computer journal Vol. 24, No. 2, Pages 162-166.

[BoW83] A. Bowyer and J Woodwark (1983), A programmers geometry, Butterworths publications.

[Bra79] I.C. Braid, (1979) Notes on a Geometric Modeller, CAD Group Document 101, Computer Laboratory, University of Cambridge, June 1979, Revised 1980.

[Bre87] J.E. Bresenham (1987), Ambiguities in incremental line rastering, Theoretical foundations of computer graphics and CAD, Editor R.A. Earnshaw, NATO ASI Series, Pages 329-358.

[BrC92] H. Bronnimann and B. Chazelle (1992), How hard is halfspace range searching, Proceeding of the Eighth Annual Symposium on Computational Geometry, Publisher ACM, Pages 271-275.

[Bro92] K. Brodlie (1992), An assessment of general purpose visualization software, SERC CFD community club, Visualization in computational fluid dynamics, Professor P. Huctchinson (Chairman), Proceeding from the seminar at RAL on 9th March 1992.

[BSC91] T.D. Blacker, M.B. Stepshenson and S. Canann (1991), Analysis automation with paving: A new quadrilateral meshing technique, Advances in Engineering Software, Volume 13, numbers 5/6, Pages 332-337.

[Bur90] E.K. Buratynski (1990), A fully automatic three-dimensional mesh generator for complex geometries, International Journal for Numerical Methods in Engineering, Vol. 30, Pages 931-952.

[Byk76] A.Bykay (1976), Automatic generation of triangular grids: 1-subdivision of a general polygon into convex subregions; 2- triangulation of convex polygons. International Journal for numerical methods in Engineering, Vol 10, 1329.

[CaM91a] J.H. Cavendish, and S.P. Marin (1991), Feature-based design and finite element analysis of functional surfaces, The mathematics of finite elements and applications VII, Academic Press Limited, Pages 129-140.

[CEG92] B. Chazelle, H. Edelsbrunner, L.J. Guibas, R. Pollack, R. Seidel, M. Sharir and J. Snoeyink (1992), Counting and cutting cycles of lines and rods in space, Computational Geometry: Theory and Applications, Vol. 1, Iss. 6, Pages 305-323.

[CeS85] Z.J. Cendes and D.N. Shenton (1985), Complementary error bounds for foolproof finite element mesh generation, Mathematics and computers in simulation 27, Pages 295-305, North-Holland.

[CFF85] J.H. Cavendish, D.A. Field and W.H. Frey (1985), An approach to automatic three-dimensional finite element mesh generation, International Journal for Numerical Methods in Engineering, Vol. 21, Pages 329-347.

[CFM91b] J.H. Cavendish, W.H. Frey and S.P. Marin (1991), Feature-based design and finite element mesh generation for functional surfaces, Advances in Engineering Software, Volume 13, numbers 5/6, pages 226-237.

[Cha92] B. Chazelle (1992), An optimal algorithm for intersecting three-dimensional convex polyhedra, SIAM Journal on Computing, Vol 21, Iss. 4, Pages 671-696.

[Cha91] B. Chazelle (1991), Triangulating a simple polygon in linear time, Discrete and Computational Geometry , Vol 6, Iss, 5, Pages 485-524.

[Cha91a] B. Chazelle (1991), An optimal convex hull algorithm and new results on cutting, Proceeding 32nd Annual Symposium on Foundations of Computer Science, IEEE Computing Soc. Press, Pages 29-38.

[Cha89] B. Chazelle (1989), An optimal algorithm for intersecting three-dimensional convex polyhedra, IEEE Computing Soc. Press, Pages 589-591.

[Cha84] B.Chazelle (1984), Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm, SIAM Journal of Computing , Vol. 13, Pages 488-507.

[ChG92] B. Chazelle and H. Edelsbrunner, An optimal algorithm for intersecting line segments in the plane (1992), Journal of the Association for Computing Machinery, Vol 29, Iss 1, Pages 1-54.

[ChG89] B. Chazelle and L.J. Guibas (1989), Visibility and intersection problems in plane geometry, Discrete and Computational Geometry, Vol. 4, Iss 6, Pages 551 - 581.

[Cho93] M-Y Chow (1993), Control Volume Unstructured Mesh Procedure for Convection-Diffusion Solidification Processes, PhD dissertation, University of Greenwich, London U.K.

[ChP92] B. Chazelle and L. Palios (1992), Decomposing the boundary of a non-convex polyhedron, Algorithm Theory - SWAT '92. Third Scandinavian Workshop Proceedings, Springer-Verlag, Pages 364-375.

[ChP90] B. Chazelle and L. Palios (1990), Triangulating a nonconvex polytope, Discrete and Computational Geometry, Vol.5, Iss. 5, Pages 505-526.

[CJL89] F. Cheng, J.W. Jaromczyk, J. Lin, S. Chang and J. Lu (1989), A parallel mesh generation algorithm based on the vertex label assignment scheme, International Journal for Numerical Methods in Engineering, Vol. 28, Pages 1429-1448.

[CoJ87] M.A. Corthout and H. Jonkers (1987), A new point containment algorithm for B-Regions in discrete plane, Theoretical foundations of computer graphics and CAD, Editor R.A. Earnshaw, NATO ASI Series, Pages 279-306.

[Con89] J.J. Connor (1989), A knowledge based approach for Boundary element mesh design, Supercomputing in Engineering Structures, Editors P.Melli and C.A. Brebbia, Computational Mechanics publishing Pages 255-267.

[Cox93] J.Cox, and M.E. Byu (1993), To appear in the Proceeding of the IMA workshop on Numerical grid generation, July 1993, Minosota.

[DBS92] T.K Dey, C.L. Bajaj and K. Sugihara (1992), On good triangulations in three dimensions, International Journal of Computational Geometry and Applications, Vol: 2, Iss: 1, Pages 75-95.

[Dew88] B.R. Dewey (1988), Computer Graphics for Engineers, Harper and Row publishers Inc.

[DSB92] T.K. Dey, K. Sugihara and C.L. Bajaj (1992), Delaunay triangulation in three dimensions with finite precision arithmetic, Computer-Aided Geometric Design, Vol. 9, Iss: 6, Pages 457-470

[ELJ91] M.G. Everett, P.J. Lawrence, B. Jones and M.Cross (1991), Software tools for aspects of computational modelling codes for materials processing, Mathematical Modelling for Materials Processing, Sept 1991, Editors M.Cross, J.F.T. Pittmain and R.D. Wood, Pages 529-538.

[EOD93] M.G. Edwards, J.T. Oden and L. Demkowicz (1993), An h-r adaptive approximate Riemann solver for the Euler equation in two dimensions, SIAM Journal for scientific computing, Jan 93, Vol 14, No. 1, pages 185-217.

[Fem91] Femgem/Femview, User manual, Femview limited, July 1991

[FaP79] I.D. Faux and M.J. Pratt (1979), Computational geometry for design and manufacture, Ellis Horwood (pub), John Wiley & Sons, New York, 1979.

[FaR92] B. Falcidieno and O. Ratto (1992), Two-manifold cell-decomposition of r-sets, EUROGRAPHICS 1992, Vol. 11 number 3, Pages 393-404, Editors A. Kilgour and L. Kjelldahl, B. Blackwell publishers.

[FeP75] H.F. Feng and T. Pavlidis (1975), Decomposition of polygons into simpler components: feature generation for syntactic pattern recognition, IEEE transactions in computing, Vol. C-24, pages 636-650.

[FHL90] P. Finnigan, A Hathaway and W. Lorensen (1990), Merging CAT and FEM, Mechanical engineering 32 (July 1990) ,Pages 32-38.

[Flo87] Leila De Floriani (1987), Surface representations based on triangular grids, The Visual Computer Vol. 3, Pages 27-50

[For87] A.R. Forrest (1987), Geometric Computing Environments, Theoretical foundations of computer graphics and CAD, Editor R.A. Earnshaw, NATO ASI Series, Pages 185-197.

[Fry87] W.H. Frey (1987), Selective refinement: A new strategy for automatic node placement in graded triangular meshes, International Journal for Numerical Methods in Engineering, Vol. 24, pages 183-220.

[Fry93] Y.D. Fryer (1993), A Control Volume Unstructured Grid Approach to the Solution of the Elastic Stress-Strain Equation, PhD dissertation, University of Greenwich, London U.K.

[Gad52] J.W.Gaddum (1952), The sums of dihedral and trihedral angles in a tetrahedron, American mathematics monthly 59, Pages 370-371.

[Gas83] C.Gasson (1983), Geometry of Spatial forms, Ellis Horwood Ltd.

[GHS90] P.L. George, F. Hecht and E. Saltel (1990), Fully automatic mesh generator for 3D domains of any shape, Impact of Computing in Science and Engineering, Vol. 2, Iss 3, Pages 187-281.

[GHS88] P.L. George, F. Hecht, and E. Saltel (1988), Constraint of the Boundary and Automatic mesh generation, Proceeding Second international conference on numerical grid generation in computational fluid, Miami, Pages 589-597, December 1988.

[GiA81] H.El Gindy and D. Avis (1981), A linear algorithm for computing the visibility polygon from a point, Journal of Algorithms, Vol. 2, pages 186-197.

[GrY86] D. Greene and F.F. Yao (1986), Finite-Resolution Computational Geometry, Manuscript, Xerox PARC.

[Gui90] M. Guiggiani (1990), Error indicators for adaptive mesh refinement in the Boundary element method - a new approach, International Journal for Numerical Methods in Engineering, Vol. 29, Pages 1247-1269.

[GuP91] H.N. Gürsoy N.M. Patrikalakis (1991), Automated interrogation and adaptive subdivision of shape using medial axis transform, Advances in Engineering Software, Volume 13, numbers 5/6, Pages 287-302.

[GuS85] L. Guibas and J. Stolfi (1985), Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, ACM Trans. Graphics 4, Pages 75-127

[HaA82] R.Haber and J.F. Abel (1982), Discrete transfinite mapping for the description and meshing of three-dimensional surfaces using interactive computer graphics, International journal for numerical methods in engineering , Volume 18, Pages 41-66.

[Her77] L.R. Herman (1977), Laplacian-isoparametric grid generation scheme, Journal for mechanical division ASCE, Vol 102 , pages 749-759.

[Hoa61] C.A.R. Hoare (1961), Quicksort. Algorithm 63: Partition, and Algorithm. CACM 4:7 (July).

[Hoa62] C.A.R Hoare (1962), Quicksort, Computer Journal 5:1, Pages 10-15.

[Hof89] C.H. Hoffmann (1989), The problem of accuracy and robustness in geometric computation, Computer , March 1989 Pages 31-41.

[HoJ93] C.M. Hoffmann and R. Juan (1993), Erep - An editable, high-level representation for geometric design and Analysis, To appear in Geometric and production modelling, Editors P. Wilson , M. Wozney, M. Pratt , North-Holland.

[JiW90] H.Jin and N.E. Wiberg (1990), Two dimensional mesh generation, adaptive remeshing and refinement, International Journal for Numerical Methods in Engineering, Vol. 29, Pages 1501-1526.

[Joe86] B. Joe (1986), Delaunay triangular meshes in convex polygons, SIAM Journal of Sci. Stat. Comput., Vol. 7, Pages 514-539.

[Joe89] B. Joe (1989), Three-dimensional triangulations from local transformations, SIAM Journal of Sci. Stat. Comput., Vol. 10, Pages 718-741.

[Joe91a] B. Joe (1991), Construction of three-dimensional Delaunay triangulations using local transformations, Computer Aided Geometric Design, Vol. 8, Pages 123-142.

[Joe91b] B. Joe (1991), Delaunay versus max-min solid angle triangulations of three-dimensional mesh generation, International Journal for Numerical Methods in Engineering, Vol. 31, Pages 987-997.

[Joe92a] B. Joe (1992), GEOMPACK - a software package for generation of meshes using geometric algorithms, International Journal for Numerical Methods in Engineering, Vol. 24, Pages 325-331.

[Joe92b] B. Joe (1992), Three-dimensional boundary-constrained triangulations, Artificial intelligence, Expert systems and Symbolic computing, Editors E.N. Houstis and J.R. Rice, Elservier science publishers B.V. (North-Holland), Pages 215-222

[Joe92c] B. Joe (1992), Three-dimensional boundary constrained triangulations, Proceedings IMACS '91, 13th world congress on computation and applied mathematics, July 22-26,1991 Trinity college Dublin Ireland, Vol. 1 , Pages 116-117.

[Joe93a] B. Joe (1993), Construction of k-dimensional Delaunay triangulations using local transformations, to appear in SIAM J. Sci. Comput., 14th November issue.

[Joh92] B.P. Johnston (1992), Fully automatic two dimensional mesh generation using normal offsetting, International journal for numerical methods in engineering, Vol. 33, Pages 425-442.

[JoS86] B. Joe and R. B. Simpson (1986), Triangular meshes for regions of complicated shape, International Journal for Numerical Methods in Engineering, Vol. 23, Pages 751-778.

[JoS93] B.P. Johnston and J.M. Sullivan (1993), A normal offsetting technique for automatic mesh generation in three dimensions, International journal for numerical methods in engineering, Vol. 36, Pages 1717-1734.

[JSK91] B.P. Johnston, J.M. Sullivan and A. Kwasnik (1991), Automatic conversion of triangular finite element meshes to quadrilateral elements. International journal for numerical methods in engineering, Vol. 31 Pages 67-84.

[JuL93] Y.H. Jung and K. Lee (1993), Tetrahedron-based octree encoding for automatic mesh generation, Computer-aided design, Vol. 25 No. 3, Pages 141-153

[KaE70] H.A. Kamel and Eisenstein K.K. (1970) Automatic mesh generation in two and three dimensional interconnected domains, Symp. high speed computing, Elastic structures, Liege, Belgium, 1970, pages 455-475.

[Kar90] M. Karasick (1990), Constructing strongly convex hulls using exact or rounded arithmetic, Proceedings of the sixth ACM Symposium on Computational Geometry, Berkeley Canada, Pages 44-52.

[Knu73] D.E Knuth (1973), The art of computer programming; Vol 3: searching and sorting, Addison-Welsey, Reading Massachusetts.

[KSP87] A. Kela , M. Saxena and R. Perucchio (1987), A hierarchical structure for automatic meshing and adaptive FEM analysis, Engineering Computing Vol 4 June 1987, Pages 104-112.

[Law72] C.L. Lawson (1972), Transforming triangulations, Discrete mathematics 3,1972 Pages 365-371.

[Law77] C.L. Lawson (1977), Software for C_1 surface interpolation, Mathematical software III, editor J.R. Rice, Academic press, New York, Pages 161-194.

[Law91] P.J. Lawrence and M. Cross (1991), Development of an automatic three-dimensional mesh generator, CFD news, Vol 2, Number 2 Sep 1991, Pages 8-14.

[LeC78] B.A. Lewis and M. Cross (1978), IFECS-an interactive finite element computing system, Applied Mathematical modelling, Volume 2, September 1978 , Pages 165-175.

[Lee83] Y.T.Lee (1983) Automatic Finite Element mesh generation based on constructed solid geometry, PhD thesis, University of Leeds.

[LeR76] B.A. Lewis and J.S. Robinson (1976), Triangulation of planar regions with applications, The computer journal, Vol 21, No. 4, Pages 324-331.

[LiJ93] A. Liu and B. Joe (1993), Relationship between tetrahedron shape measures, submitted for publication.

[Löh88] R.Löhner (1988), Some useful data structures for the generation of unstructured grids, Communications in applied numerical methods, Vol 4, Pages 123-135.

[Lo85] S.H.Lo (1985), A new mesh generation scheme for arbitrary planar domains, International Journal for numerical methods in Engineering, Vol. 21, Pages 1403-1426.

[Lo88] J.A. Lo (1988), An approach to automatic course three dimensional finite element mesh generation, Master's thesis 1988, Rensselaer Polytechnic Institute, U.S.A.

[LoS91] S.H. Lo (1991), Automatic mesh generation and adaptation by using contours, International Journal for Numerical Methods in Engineering, Vol. 31, Pages 689-707.

[LPG88] R.Löhner and P.Parikh, C. Gumbert (1988), Interactive generation of unstructured grids for 3-D problems, Second Int. Conf. on Numerical grid generation in Computational Fluid Dynamics, Miami Beach, Florida U.S.A. 5-9th December 1988, Pineridge Press.

[Lyu63] L.A. Lyusternik (1963), Convex figures and polyhedra, New York, Dover Publications.

[Män89] M.Mäntylä (1989) Boolean operations of 2-manifolds through vertex neighbourhood classification, Transactions on Graphics, Volume 5, Number 1, Pages 1-29, Jan 86.

[Mar88] Mantyla, Martti (1988), An introduction to solid modelling, Computer science press.

[Mas93] G. Masotti (1993), Floating-point numbers with error estimates, Computer aided design journal, Vol. 25, No. 9, Pages 524-538.

[MCL81] A.O. Moscardini, M. Cross and B.A. Lewis (1981), Assessment of three automatic triangular mesh generations for planar regions, Advances in engineering software, Vol 3, No. 3, Pages 108-114.

[MeP77] B. Mercier and O. Pironneau (1977), Some examples of implementation and application of finite elements method, Rapport de Recherche No. 248, IRIA, Le Chesnay, France 1977.

[Mid87] A.E. Middleditch (1987), The representation and manipulation of convex polygons, Theoretical foundations of computer graphics and CAD, Ed. R.A. Earnshaw, Nato ASI Series.

[Mil75] W Miller (1975), Software for round-off errors analysis, ACM transactions on Mathematical software, Vol. 1, No. 2, Pages 118-128

[MLC83] A.O. Moscardini, B.A. Lewis and M. Cross (1983), Agthom-automatic generation of triangular and higher order meshes, International Journal for Numerical Methods in Engineering, Vol. 19, Pages 1331-1353.

[MoB83] R. Mohr and R. Bajcsy (1983), Packing volumes by spheres, IEEE transactions on pattern analysis and machine intelligence, Vol PAMI-5, No. 1, Jan 1983, Pages 111-116.

[MTC92] R.B. Morris, Y. Tsuji and P. Carnevali (1992), Adaptive solution strategy for solving large systems of p-type finite element equations, International Journal for Numerical Methods in Engineering, Vol. 33, Pages 2059-2071.

[NoP88] A.K. Noor and J.M. Peters (1988), Error indicators and accuracy improvements of finite element solutions, Engineering Computing Vol 5 March 1988, Pages 39-49.

[Pat89] PATRAN Plus 2.4 released, Infografik , No. 6, Page 30.

[Per89] A. Perronnet (1989), Proceeding of the fifth international sylupotrium on numerical methods in Engineering, R.Gruber, J.Periaux,R.P.Shaw editors, Springer Verlag.

[PPF88] J.Peraire, J. Periro, L. Formaggia, K. Morgan, O.C. Zienkiewicz (1988), Finite element EULAR computations in 3-D AIAA, 26th Aerospace Science Meeting ,January 11-14 1988 Reno Nevada U.S.A.

[Pie91] L A Piegl (1991), On NURBS: A Survey, IEEE Computer Graphics & Applications, 11(1):55-71, January 1991.

[PPM92] J. Peraire, J, Peiró and K. Morgan, Adaptive remeshing for three-dimensional compressible flow computations, Computational physics , Vol. 103, No. 2, December 1992, Pages 269-285.

[Ram82] L.H. Ramshaw (1982), The Braiding of Floating Point Lines, Xerox Palo Alto Research Centre, CSL Notebook entry.

[Ran91] E. Rank (1989), Adaptive h-,p- and hp- Versions for boundary integral element methods, International Journal for Numerical Methods in Engineering, Vol. 28, Pages 1335-1349.

[RaR93] H Ratschek and J Rokne (1993), Test for intersection between plane and box, Computer aided design journal, Vol. 25, No. 4, Pages 249-250.

[Rec73] Rechenberg I (1973), Evolutionsstrategie, Friedrich Froman Verlag, Stuttgart, 1973.

[Req80] A.A.G. Requicha (1980), Representations of solid objects-theory and, methods, and systems. ACM Computing Surveys, 12(4):437-464 Dec. 1980.

[Rob87] J. Robinson (1987), CRE method of element testing and the Jacobian shape parameters, Engineering Computing, Vol 4 June 1987 pages 113-127

[Rog85] D.F. Rogers (1985), Procedural elements for computer graphics, London, McGraw-Hill 1985.

[Sab85] M.A. Sabin (1985), The state of Art, Pages 411-482 in fundamental algorithms for computer graphics, ed R.A. Earnshaw, NATO ASI Series F17, Springer Verlag (1985)

[Sab91] M.A. Sabin (1991), Criteria for comparison of automatic mesh generation methods, Advances in engineering software, Volume 13 No. 5/6, Pages 220-225.

[Sar83] R.F. Sarrage (1983), Algebraic methods for intersections of Quadric surfaces in GMSOLID, Computer vision, graphics, and image processing 22, Pages 222-238.

[SBS79] A.J.G. Schoofs, L.H.T.M. Van Beukering and M.L.C. Sluiter (1979), A general purpose two-dimensional mesh generator. Advances in Engineering Software, Vol 1,No.3,Pages 131-136.

[Sch78] B. Schachter, Decomposition of polygons into convex sets (1978), IEEE Transactions in computing, Vol. C-27, Pages 1078-1082

[ScS88] W.J. Schrodeder and M.S. Sherhard (1988), Geometry-based fully automatic mesh generation and the delaunay triangulation, International Journal for Numerical Methods in Engineering, Vol. 26, Pages 2503-2515.

[ScS89] W.J. Schrodeder and M.S. Sherhard (1989), An O(N) Algorithm to automatically generate geometric triangulations satisfying the Delaunay circumsphere criteria, Engineering with computers, Vol 5, Pages 177-189.

[ScS90] W.J. Schrodeder and M.S. Sherhard (1990), A combined Octree/Delaunay method for fully automatic 3-D mesh generation, International Journal for Numerical Methods in Engineering, Vol. 29, pages 37-55.

[ShL91] M.S. Shephard and J.A. Lo (1991), Automatic generation of coarse three-dimensional meshes using the functionality of a geometric modeller. Advances in Engineering Software,Volume 13,numbers 5/6, Pages 273-286.

[ShG91] M.S. Shephard and M.K. Georges (1991), Automatic three-dimensional mesh generation by the finite octree technique, International journal for numerical method in engineering, Vol. 32, Pages 709-749.

[SlH82] M.L.C. Sluiter and D.L.Hansen (1982), A General purpose automatic mesh generator for shell and solid finite elements, Computers in Engineering, Vol.3 (L.E Hulbert,Ed), Book Bo. G00217,ASME,1982, Pages 29-34.

[SlH84] S.W. Sloan, G.T. Houlsbyan (1984), Implementation of Watson's algorithm for computing 2-dimensional Delaunay triangulations, Advances in Engineering Software, volume 6, number 4, Pages 192-197.

[Sol85] B.J. Solomon (1985), Surface Intersections for Solid Modelling, University of Cambridge, PhD. Thesis 1985.

[Sto91] I. Stojmenovic (1991), Bisection and ham-sandwich cuts of convex polygons and polyhedra, Information processing letters, Vol 38, Iss. 1, Pages 15-12.

[TaA91] T.K.H. Tam and C.G. Armstrong (1991), 2D Finite element mesh generation by medial axis subdivision, Advances in Engineering Software, Volume 13, numbers 5/6, Pages 313-323.

[Tha80] W.C. Thacker (1980), A brief review of techniques for generating irregular computational grids", International journal for numerical methods in engineering, Vol. 15, 1980, Pages 1335-1341.

[Thm85] J.F. Thompson (1985), A survey of dynamically-adaptive grids in the numerical solution of partial differential equations, Applied numerical Mathematics 1, Pages 3-27.

[Tho80] Thamas A. Standish (1980), Data structure techniques, Addison-wesley publishing company.

[Tou85] T. Tossaint (1985), Computational Geometry, Elsevier science publishers,

[TPA93] T.K.H. Tam, M.A. Price, C.G. Armstrong and R.M. McKeag, Computing the critical points on the medial axies of a planar object using Delaunay point triangulation algorithm, submitted to IEEE PAMI.

[Wat81] D.F. Watson (1981), Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes, The compute journal, Vol 24, No.2, Pages 167-172.

[Wei88] K.J. Weiler (1988), Boundary graph operators for non-manifold geometric modelling topology representations, Geometric Modelling for CAD Applications, Wozny, Mclaughlin and Encaracao (eds) North Holland 1988, Pages 37-66.

[Wör81] B. Wördenweber (1980), Automatic mesh generation of 2 and 3 dimensional curvilinear manifolds, PhD dissertation , St John's college , University of Cambridge.

[YeS84] M.A. Yerry and M.S. Shephard (1984), Automatic three-dimensional mesh generation by the modified-octree technique, International Journal for Numerical Methods in Engineering, Vol. 20, Pages 1965-1990.

[YTH91] M.M.F. Yuen, S.T. Tan and K.Y. Hung (1991), A Hierarchical approach to automatic finite element mesh generation, International Journal for Numerical Methods in Engineering, Vol. 32, Pages 501-525.

[ZiP71] O.C. Zienkiewicz and D.V. Phillips (1971), An automatic mesh generation scheme for plane and curved surfaces by isoparametric co-ordinates, International journal for numerical methods in engineering, Vol. 3, Pages 519-528.

APPENDIX

A1 Calculating length of a side of an equilateral triangle given its area.

One of the requirements of the mesh generator is to calculate the length of the side of an equilateral triangle given its area *A*.

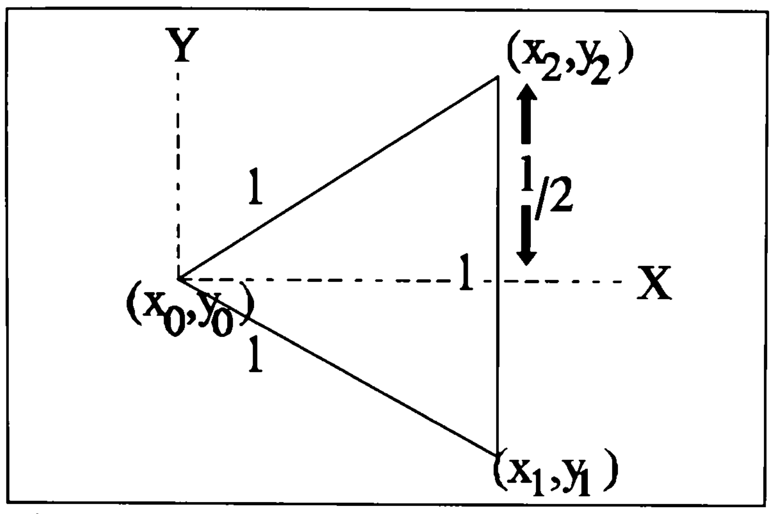


Figure A1.1 Equilateral triangle.

The area of a triangle is calculated by the cross product of the direction vectors $\underline{a},\underline{b}$ [Dew88] of two sides :

$A=0.5\ \underline{a}X\underline{b}$ where $\underline{a}^T=[x_1-x_0,y_1-y_0]$

$\underline{b}^T=[x_2-x_0,y_2-y_0]$

If we assume (x_0,y_0) is at the origin, and the altitude of the triangle is along the axes X=0. Hence :-

$2A=x_1y_2-x_2y_1$ but $x_1=x_2$

$2A=x_1(y_2-y_1)$ from the diagram $y_2=l/2$ and $y_1=-l/2$

$2A=lx_1$ however $l^2= l^2/4+x_1$ therefore $x_1=(3l^2/4)^{0.5}$ where $l>0$

$A=3^{0.5}l^2/4$

Therefore the length of a side of an equilateral triangle given its area is:

$$l=\sqrt{\frac{4A}{\sqrt{3}}}$$

A2 Calculating length of an edge of an equilateral tetrahedral given its volume.

For an equilateral tetrahedron Figure A2.1 the length of an edge has to be calculated from its volume *V*.

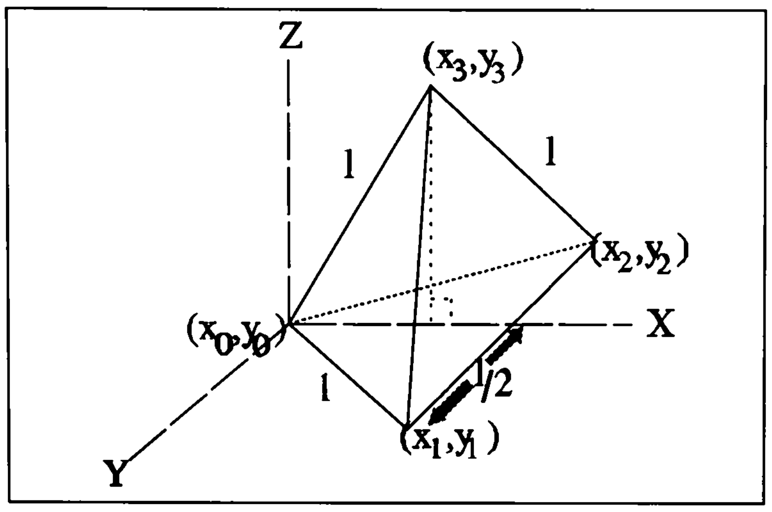


Figure A2.1 Equilateral tetrahedra.

The volume of a tetrahedral is calculated from the equation V=($\underline{a}$X$\underline{b}$.$\underline{c}$)/6 [Dew88] where $\underline{a}$,$\underline{b}$,$\underline{c}$ are unit direction vectors:

$\underline{a}^T=(x_1-x_0,y_1-y_0,z_1-z_0)$

$\underline{b}^T=(x_2-x_0,y_2-y_0,z_2-z_0)$

$\underline{c}^T=(x_3-x_0,y_3-y_0,z_3-z_0)$

If we assume that the base of tetrahedron is on the z=0 axes, with the altitude of the base along the x=0 axis and (x_0,y_0,z_0) is the origin, then:

$\underline{a}X\underline{b}=3^{0.5}l^2/2$ from Appendix A1.

$V=(0,0,3^{0.5}l^2/2).(x_3,y_3,z_3)/6$

$V=z_3 3^{0.5}l^2/12$ however $l^2= l^2/4+z_3$ therefore $z_1=(3l^2/4)^{0.5}$ where l>0

$V=l^3/8$

Therefore, the length of an edge of an equilateral tetrahedron given its volume is:

$$\sqrt[3]{8V}$$

A3 Inner and Outer Boundary

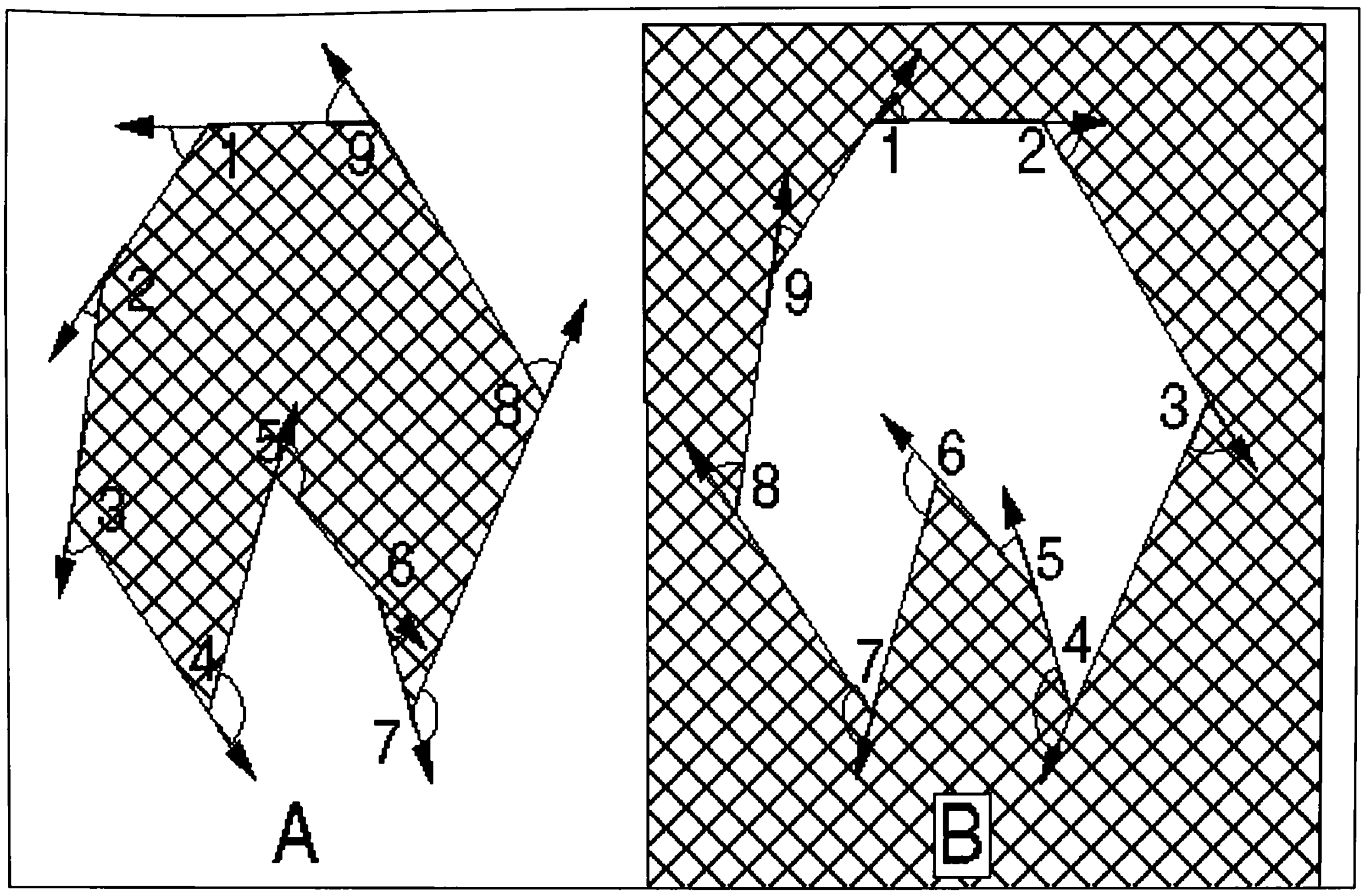


Figure A3.1

In the above diagram two identical boundaries are represented, one is a closed domain as in Figure A3.1A and the other is an inner boundary, Figure A3.1B representing a hole. The mesh generator has to be able to distinguish between the two types, and this can be done by summing the angles of the vectors that form the boundary. Each angle has a value of between -π to 0 if it is a convex angle, or 0 to π if it is concave [Mar88]; Figure A3.2.

If we sum the α_j angles in the above domain:

$$\sum_{i=1}^{n} \alpha_i = \pm 2\pi$$

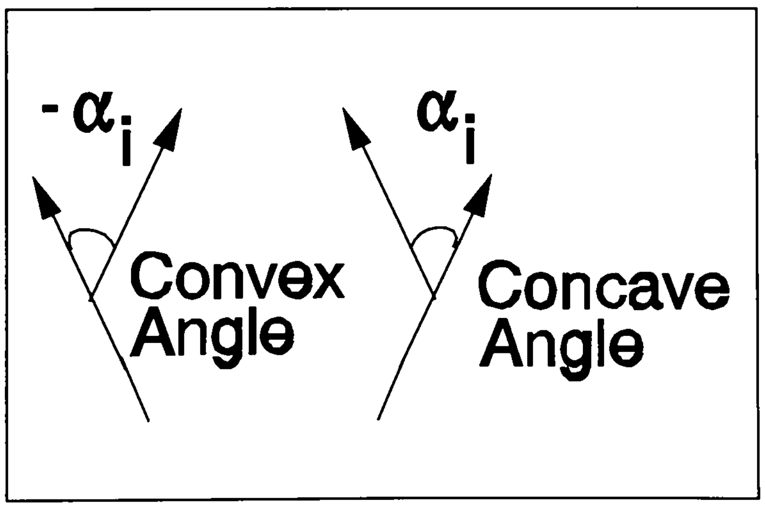


Figure A3.2

A value of -2π indicates an exterior boundary where 2π is an interior boundary.

A4 Adjacent edge searching.

One common operation carried out in mesh generation is the initialization of adjacency tables. For example, in the generation of an adjacency table for polyhedral domains, in which faces sharing a common edge must be established.

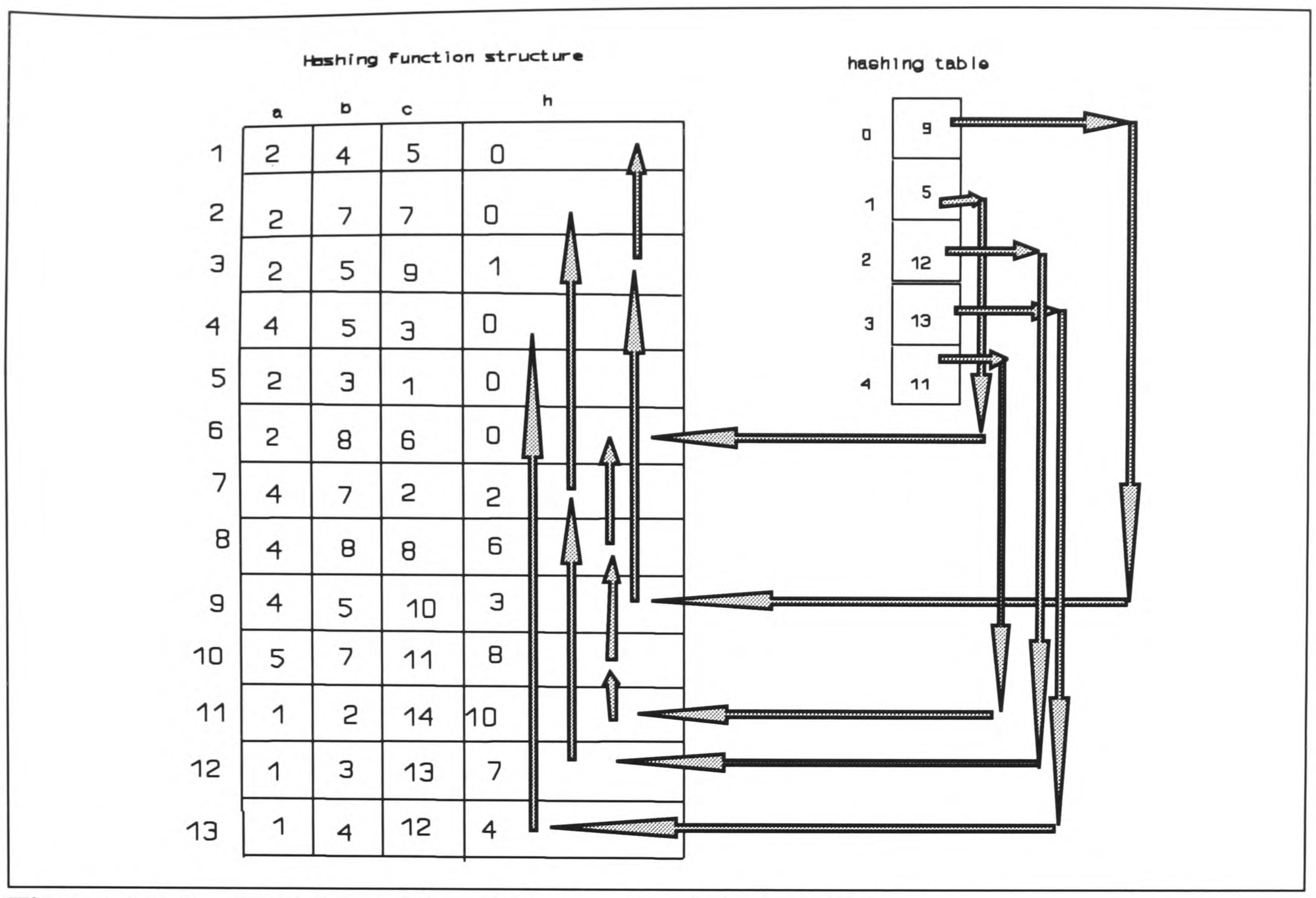
A simplistic adjacency searching technique is:

```
(i)     for each face $f_i$ in the polyhedral domain
(ii)        loop over each edge $e_j$ of the face $f_i$
(iii)           for each face $f_j$,$f_j \neq f_i$, of polyhedra domain
(iv)                loop over each edge $e_j$ of the face $f_j$
(v)                     If the two edges $e_i$ and $e_j$ match
(vi)                        Update adjacency table and return to step (ii)
```

The above algorithm is slow and of order n^2; a far superior method is to implement a hashing table technique which results in a procedure of computational order of nlog(n).

```
(i)     for each face $f_i$ in the polyhedra domain
(ii)        loop over each edge $e_j$ of the face $f_i$
(iii)           search the hashing table for an entry $h_i$ that matches $e_i$.
(iv)            If entry $h_i$ exists, get $f_j$ and $e_j$ from the hashing table
                    Update adjacency table and remove the entry $e_j$ from the
                    hashing table.
                Otherwise add entry $e_i$ to hashing table.
```

At the completion of the above algorithm the hashing table should be empty, if an adjacency table is successfully generated. The polyhedral faces are stored in a one dimensional array *VL*, section 7.2.2, with a face pointer structure *FP* pointing to the head of each face vertex list in this vector. Therefore the information store in the *hashing function data structure* is the two vertices *a,b* of edge e_i, where $a<b$, a pointer *c* which points to the edge e_i in the array *VL*, together with the hash function chaining list *h*. The hashing table together with hashing table structure is illustrated in Figure A4.1. The hashing table *ht* is of a size *Htsize*, where *Htsize* should be a prime number greater than number of edges *n* in the mesh, this is to minimize collision of the hashing function. The hashing function used to find the hash key in the hashing table is $(a*n+b) \bmod htsize$.



	a	b	c	h
1	2	4	5	0
2	2	7	7	0
3	2	5	9	1
4	4	5	3	0
5	2	3	1	0
6	2	8	6	0
7	4	7	2	2
8	4	8	8	6
9	4	5	10	3
10	5	7	11	8
11	1	2	14	10
12	1	3	13	7
13	1	4	12	4

0	9
1	5
2	12
3	13
4	11

Figure A4.1 : Hashing table with associated data structure.

When an associate entry in *FV* is required for the edge e_j. The two vertices *a* and *b* are used to generate a hashing key value *k*. The pointer *ht[k]* gives the starting location in the *hashing function structure* of a linked list of edges, which generate the same key *k*. The link list is then search for an entry that matches the edge e_i. If a matching edge is found, the value of *c* gives the location of the adjacent edge e_j in the array *VL*. This entry is then removed from the hashing structure and returned to the free storage. Column *a* is used as a free storage pointer link list when the entry is not used for storing edge data, a free storage pointer is indicated by a negative value in this column.

A fuller description of hashing tables and functions is given in *Data structure techniques* by Standish [Tho80].

A5 Numerical Precision

Geometric algorithms are notorious in practice for numerical instability. Many algorithms are far from robust for numerical reasons. A common fiction is that computer's floating-point arithmetic is "accurate enough". Therefore, a common "cure" for numerical problems is an increase in floating point precision. Obviously, this is not a real cure, but the effect should be a reduction in the frequency of numerically unstable cases. In the case of intersecting line segments, increasing precision would permit accurate intersection of lines at shallower angles. However, the root of many problems lie in the elementary ill-conditioning of the equations.

The problem with many systems is that we are not dealing with random configurations, and lines that are nearly parallel are relatively frequent, e.g. computer-aided design applications. Therefore, shallow intersection angles are common, and merely increasing the precision is not a substitute for correct handling of special cases, using appropriate numerical tolerances. Forrest [For87] quotes Solomon [Sol85] in saying that it is best to carry out all geometric computations as near to the origin as possible. This is simply that we should not expect significance if we subtract two large floating point numbers.

Another well known phenomenon in floating point arithmetic, is obtaining different results from geometric computations depending on the order of evaluation. For example, if one evaluation order causes numerical overflow, the user often finds that a slightly different approach eliminates the problem. More dangerously, if numerical overflow does not occur, we can still obtain inconsistent results which can be difficult to reconcile.

Even restricting the geometry is not enough to guarantee correct results: Ramshaw [Ram82] reports difficulties which arose in computation with simple line segments defined by integer end points, even when using double precision floating point arithmetic. These problems have been considered by many works [GrY86][Bre87][CoJ87]. Corthout [CoJ87] technique is to map to an integer grid at the precision of the user's modelling space. This has the merit of regularising the finite grid we are forced by the computer to employ.

To try and deal with the above problems the *new bisection mesh generator*, has various techniques implemented. The simplest approach applied is a combination of *Relative and absolute error* [Hof89]. This is primary used for the comparison of two numbers [Mas93], e.g. test if x is equal to y:

$$|x-y|<\varepsilon(1-M) \quad where\ M=MAX(|x|,|y|), \quad \varepsilon\ is\ machine\ tolerance.$$

Other examples are testing if the determinant of a matrix is zero:

$$\varepsilon MAX[1,a_1,a_2.....a_n]>\det[A]$$

Where $a_1,a_2...a_n$ are the entries in the matrix A.

Algorithms such as line and plane intersection are based on robust standard techniques [RaR93][Män88][BKK84] using *interval arithmetic*, see [Hof89][Mas93]. The bisection algorithms, Chapter 7 section 3, uses techniques to try and avoid geometry features which may cause numerical problems in later geometry operations. This is achieved by rejecting cut planes which introduce any short edges or acute angles into the model. This of cause results in a cost in CPU time, however short or acute angles are not a desirable feature of a tetrahedral mesh. This is also coupled with *geometric reasoning [BaD92]*. Therefore, if there is a geometry operation which may result in an inconsistent decision, a choice is taken to which outcome is less detrimental to the program. For example, the bisection algorithm implemented in the convex shrinking technique, Chapter 7.4, labels all nodes which are *close* to the bisection plane, as *above*. This many result in some polyhedral faces becoming non-planar, however *slightly* non-planar faces can be easily tolerated in the subsequent computation of nodal placement.

The projection of points and faces onto planes, as in the *Edge following cutting* routine Chapter 7 section 7.3.3, utilizes methods of *Perturbation* [Hof89]. This technique is where the initial data is slightly altered, during computation, to ensure a well conditioned set of geometric entities. This is achieved by *data normalization* [Hof89] where vertices positions are slightly altered so that they are not *too close* to other geometry entities.

However, many of the more complex error estimation techniques have not been implemented in the mesh generator, see [Mas93][Mil75]. Since these methods were considered unnecessary, and can in some cases result in a CPU time increase of 100% [Mil75]. Therefore, the main weapon used against *rounding errors* is avoiding degeneracies [Hof89].

The most dangerous area for ill-conditioned data, is the geometry provided by the user. This is where *near* degeneracies cannot always be avoided, and typical models will possess acute angles or narrowing. The only defence which can be provided against this problem is for the correct handling of special cases, using appropriate tolerances.

Numerical problems will hinder all attempts to provide a general purpose meshing tool. As long as we stick to generality, we must realise that there are many perfectly normal geometric constructions that cannot be represented correctly unless we use infinite precision arithmetic [For87].

A6 Mesh generator's modules.

The mesh generator, presented in this thesis, is constructed from a number of different modules. These modules can be taken and interlinked in different ways, to build other mesh generators. Figure A6.1 (Page 219), depicts the various modules and how they interlink. The key below describes the various modules and types of links.

Module/Link	Description
Arrow	Direction of data flow.
Dashed Arrow	This module uses another module. e.g 3D geometry preparation module uses the "Polyhedral Splitting Algorithm" and the "Contour Polyhedral Splitting Algorithm".
Double Arrow	Modules which interact, e.g the 2D Recursive bisection technique interacts with the "Line Node Generator"
2D Geometry Prep.	Routine which identifies holes/sub-domains and corrects direction of boundaries. This routine also removes sub-polygon domains by introducing separators which divide these domains into a number of simpler regions. See Chapter 4 sections 4.6.1 and 4.6.4.
3D Geometry Prep.	Routine which identifies holes/sub-domains, calculates and corrects direction of face normals, and removes sub-polyhedral by introducting separator faces which divide these domains into simpler regions. See Chapter 6 sections 6.5 and 6.12.
Polyhedral Splitting Algorithm	See Chapter 7 section 7.3.2.
Contour Polyhedral Splitting Algorithm	See Chapter 7 section 7.3.4
2D Convex Routine	This routine sub-divides the 2D geometry into a number of convex regions. See Chapter 4 section 4.6.3.
3D Convex Routine	This routine sub-divides 3D domains up into a number of convex regions. See Chapter 6 section 6.7.

Module/Link	Description
Polyhedral Cutface method	See Chapter 7 section 7.3.1.
Edge Following bisection method	See Chapter 7 section 7.3.3.
2D Convex nodal placement	Routine which automatically generates nodes by shrinking convex domains. See Chapter 4 section 4.6.5.
2D Node insertion	Once the initial mesh is generated from the boundary nodes, this routine inserts the internal nodes. See Chapter 4 section 4.5.2.
Line Node Generator	This routine works in conjunction with the recursive bisection mesh generator, to automatically generate internal nodes. This is achieved by generating nodes alone each bisection line. See Chapter 4 section 4.5.2.
3D Convex nodal placement	This routine automatically generates nodes by shrinking 3D convex domains. See Chapter 7 section 7.4.
3D Node insertion	Once the initial mesh is generated from the boundary nodes, this routine inserts the internal nodes.
Planar node generator	This routine automatically generates nodes over a cut face, which is generated by the 3D recursive bisection mesh generator. This routine is the 2D recursive mesh generator using the "Line node generation" routine. See Chapter 5 section 5.2.
2D Delaunay	2D Delaunay routine using Lawson's swapping algorithm, See Chapter 3 section 3.5.
2D Recursive Bisection	2D recursive mesh generator, see Chapter 4 section 4.3.
Surface Delaunay	Surface Delaunay mesh generator, see Chapter 5 section 5.3.
3D Delaunay	3D Delaunay using vertex transformations, see Chapter 7 section 7.5.

Module/Link	Description
3D Min-max	3D Min-max solid angle meshing, using local 3D transformations, See Chapter 7 section 7.5.
3D Recursive bisection	The initial 3D recursive mesh generator, described in Chapter 6 section 6.2.
Binary mesh operations	This routine is used if the 3D recursive mesh generator fails to find a bisection. See Chapter 3 section 3.3.
2D Vertex Swapping	2D vertex swapping, see Chapter 4 section 4.4.1.
2D Laplace smoothing	2D Laplace smoothing, see Chapter 4 section 4.4.2.
Surface vertex swapping	See Chapter 5 section 5.5.
Surface Laplace Smoothing	See Chapter 5 section 5.4.
3D Vertex swapping	See Chapter 7 section 7.6.
3D Laplace Smoothing	See Chapter 4 section 4.4.2.

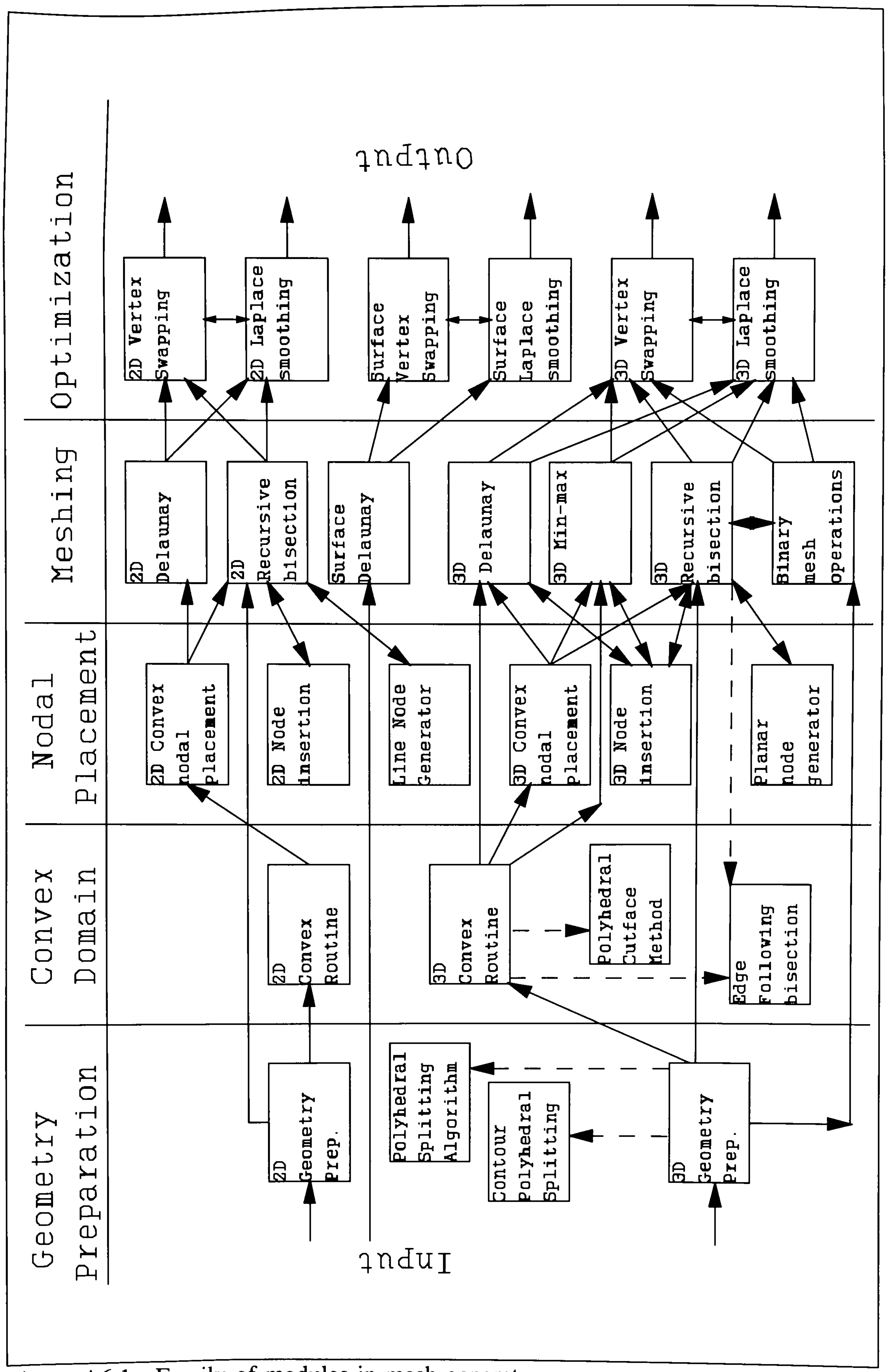


Figure A6.1 : Family of modules in mesh generator.