Science and Engineering Research Council

Rutherford Appleton Laboratory

Chilton DIDCOT Oxon OX11 0QX

RAL-92-059

A Mesh Generator for Tetrahedral Elements Using Delaunay Triangulation

J S Yuan and C J Fitzsimons

September 1992

A Mesh Generator for Tetrahedral Elements Using Delaunay Triangulation

J.S. Yuan, C.J. Fitzsimons

September 1992

Abstract

A tetrahedral mesh generator is introduced in this report. The generator is based on the Delaunay triangulation which is implemented by employing the insertion polyhedron algorithm. Some new methods to deal with the problems associated with the three-dimensional Delaunay triangulation and the insertion polyhedron algorithm are presented, such as the degeneracy, crossing situation, identifying of the internal elements and internal point generation. The generator works both for convex and non-convex domains, including those with high aspect-ratio subdomains. Some examples are given in this report to illustrate the capability of the generator.

Mathematical Software Group
Computational Modelling Division
Rutherford Appleton Laboratory
Chilton, Didcot
Oxfordshire OX11 0QX

Contents

List of Figures

1 Introduction

Efficient mesh generators for complex three-dimensional domains are becoming a crucial requirement in computational modelling with the finite element method. A wide of variety of algorithms have been devised for mesh generation. In this report we survey the Delaunay algorithm[1] and the *advancing-front* algorithm, and describe a new implementation of the Delaunay algorithm.

The Delaunay algorithm [6, 25] has been used extensively in many different different areas of computational modelling, e.g. field analysis problems in electrical, electronic, and mechanical engineering, astrophysics, biology, ecology, and archaeology. The procedure to implement a Delaunay algorithm may be divided into two stages: the initial mesh generation and the internal point placement. The initial mesh is based on the geometrical points defining the domain. The internal point introduction refines the mesh to obtain sufficient elements for the field analysis. There are two approaches to implementing the Delaunay algorithm, i.e. for adding the points to the mesh subject to the Delaunay criterion: the *insertion polyhedron* method [6, 2, 25] and the *edge swapping* method [6, 26].

In the advancing-front algorithm the internal mesh point generation and the connection of the points into the mesh is carried out at the same time. There exist a few methods for creating new mesh points and connecting them into meshes [22, 18, 19].

After comparing the two algorithms, we have developed a three-dimensional mesh generator MSH3D, based on the Delaunay algorithm. We have developed an efficient and versatile implementation for both convex and non-convex domains, including those with high aspect-ratio subdomains.

In next section of the report we review the Delaunay and the advancing front algorithms and provide a brief comparison. The results of this are that the computer storage needed by the two algorithms is approximately the same but the operations needed by the advancing-front algorithm are about four times those needed by the insertion polyhedron method.

In the Section 3 we introduce a new implementation of the Delaunay algorithm in three dimensions, in which we discuss in some detail soultions to problems such as how to deal with a non-convex domain or a multi-material domain, how to identify and remove elements which cross the boundary of the domain and elements which lie outside the domain, and how to generate internal mesh points.

In Section 4 we give some examples to illustrate the method.

2 A Review of Some Mesh Generation Procedures in Three Dimensions

In three-dimensional mesh generation, meshes with tetrahedral elements are geometrically flexible and allow complete polynomial approximation functions to be defined in the finite element method. Two methods which produce such meshes are the Delaunay algorithm [6, 26, 2, 25] and the advancing-front algorithm [22, 18, 19]. The Delaunay triangulation has been used extensively in many different fields, either directly or through its dual, the Voronoi tessellation [6, 25].

In this section of the report we review the Delaunay and the advancing-front algorithms and provide a comparison. In the Section 3 we present a new implementation of the Delaunay triangulation algorithm in three dimensions.

[1] We use the term *Delaunay algorithm* to denote a mesh generation algorithm in which the elements of the generated mesh satisfy the Delaunay criterion.

2.1 The Delaunay Algorithm

A Delaunay triangulation is one in which each element satisfies the Delaunay criterion, i.e. the interior of the circumsphere of any element in the mesh contains no mesh points. The Delaunay triangulation of a set of points is unique in the absence of the degeneracy [6, 2, 23], i.e. when more than four mesh points are co-spherical.

The Delaunay triangulation of a set of points in a domain may be formed as follows:

1. Define a large *box*, which contains the set of points and the domain. This box may be a tetrahedron, or a hexahedron that is divided into five or six tetrahedra.
2. Choose a point from the set.
3. Add this point to the mesh in the box, ensuring that the resulting mesh satisfies the Delaunay criterion.
4. Return to step 2 until all points have been added to the mesh.

This algorithm generates a mesh for the original set of points and the vertices of the box. The implementation of Step 3 is central to this algorithm and will be discussed in detail later.

In order to obtain a mesh of the set of points, two of the problems to be addressed are: how to deal with a non-convex domain or a multi-material domain and how to identify and remove elements which cross the boundary of the domain and elements which lie outside the domain. For a convex domain, no element crosses the boundary of the domain and all the elements outside the domain are connected to one of the vertices of the box. Therefore it is easy to obtain the Delaunay mesh for a convex domain. For a non-convex domain, some elements may cross the boundary of the domain or the interfaces in a multi-material domain (c.f. Figure 1 for an example in two dimensions). Also some parts of the boundary may not appear in the mesh, and some elements outside the domain may not be connected to any vertex of the box. Therefore special techniques to remove this *crossing situation* and to identify the elements inside the domain must be developed. The stitching method is presented in [24] to recover the boundary of the domain and to remove the crossing situation. Once a crossing situation is identified, supplementary points are introduced to resolve the situation in the stitching method. One such strategy for choosing where to place the points is to compute the intersection of incompatible mesh edges with the model geometry. Perronnet [23] also suggests this technique to resolve the crossing situation. In our experience, it is not very efficient to remove the crossing situation by introducing points on the boundaries for some complex cases in three dimensions, especially for multi-material domains.

A mesh derivation method is presented in [10] to recover the boundaries lost in the generated meshes. It is proved that a mesh with some elements crossing the boundaries can be transformed into an equivalent mesh without elements crossing the boundaries in the same domain with the same points, which means that it is unnecessary to introduce extra points to remove the crossing situation. But it is clear that equivalent mesh does not satisfy the Delaunay criterion.

A theory of graphs is presented in [12] to remove the elements outside the domain; this identifies and deletes the elements with a triangular surface which is not used by other elements and does not belong to the boundaries of the domain. The technique can be described as deleting the exterior elements in the existing mesh gradually. Unfortunately this theory fails to deal with the domains in which holes exist.

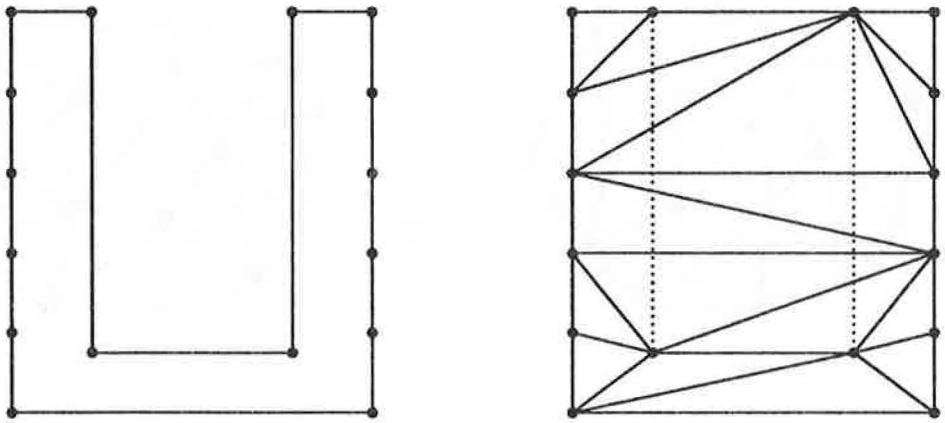

Figure 1: The insertion polyhedron method

We present an efficient method in §3 to identify the elements inside the domain, which works for all kinds of geometrical models.

Besides the above difficulties in the implementation of the Delaunay algorithm, efficient techniques must be developed to add each point into the existing mesh. There are two methods for adding a point into the existing meshes which use the Delaunay criterion: the insertion polyhedron method and the edge swapping method.

2.1.1 The Insertion Polyhedron Method

The insertion polyhedron method is described in [6, 2, 25]. When a new point is being inserted into the existing mesh, first the elements whose circumspheres contain the new point are found. The deletion of these elements results in a star-shaped[2] insertion polyhedron surrounding the new point. Connecting the new point to each vertex of the polyhedron forms a new mesh, each of whose elements satisfies the Delaunay criterion. Figure 2 shows an example in two dimensions. In the insertion polyhedron method, degeneracy occurs when a newly inserted point lies on the circumsphere of an existing tetrahedron, in which case overlapping elements or gaps may be produced. In this case, because of computer round-off error, an incorrect — or inconsistent — decision about whether the point lies in the circumsphere of an existing element may be made, and it is possible that the insertion polyhedron is not star-shaped in practice and some non-Delaunay elements result [6].

Cavendish [2] presents a method to overcome the degeneracy problem, in which he moves slightly the coordinates of the newly-inserted point when it lies ambiguously on a circumsphere associated with an existing tetrahedron. At the completion of the triangulation, all points which have been moved are restored to their original position.

Perronnet [23] presents another method to overcome the degenerate case which guarantees that the elements found whose circumsphere contain the newly inserted point are continuous and the insertion polyhedron is star-shaped. First, the tetrahedron containing the new point is found and is added to the polyhedron. Then each neighbour of the insertion polyhedron is checked until the

[2] A region R is star-shaped about a point p if $\forall x \in R$ then $[px] \subset R$.

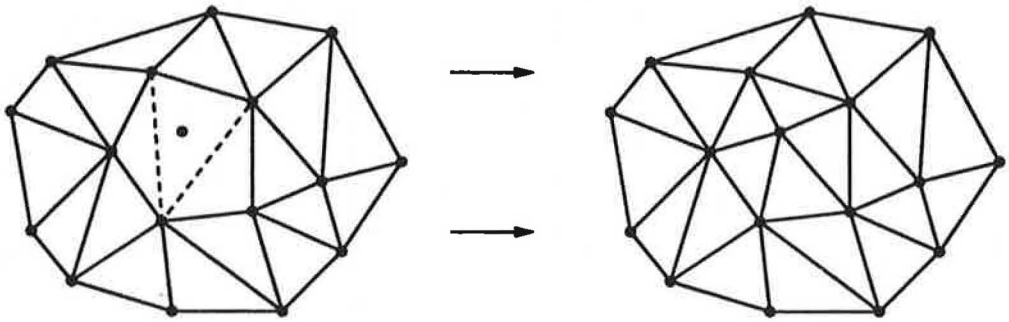

Figure 2: The insertion polyhedron method

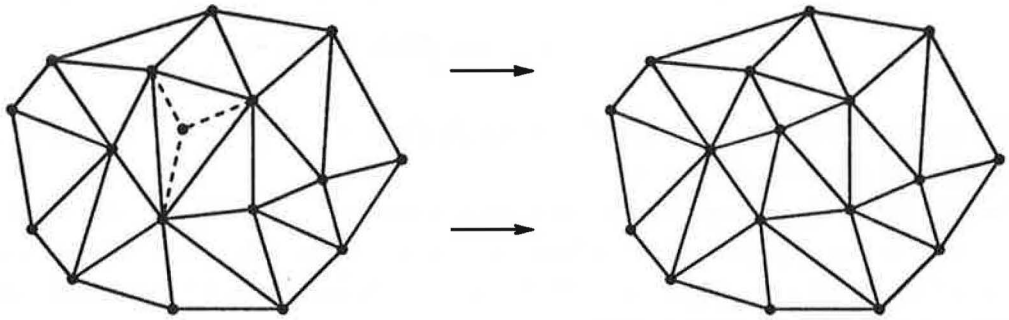

Figure 3: The edge swapping method

circumspheres of all neighbours of the polyhedron do not contain the new point. A neighbour whose circumsphere contains the point is added to the polyhedron.

2.1.2 The Edge Swapping Method

Until recently the edge-swapping method [6, 26] was applied in two dimensions only. In [6] Ferguson describes its successful extension to three dimensions. In the two-dimensional edge method, when a new point is inserted into the existing mesh the element E in which it lies is found. Then the new point is joined to the three vertices (cf. Figure 3). The newly-formed triangles are put in a list. Each item on the list is checked to see if it and its neighbours satisfy the Delaunay criterion. If the criterion is not satisfied, the common edge of the two triangles being checked is swapped and the newly-formed triangles are added to the list. When the list is empty the resulting mesh satisfies the Delaunay criterion. This precedure is local, i.e. it is necessary to check at most the elements which lie in the star-shped polygon of the insertion polyhedron method. In two dimensions, a diagonal can be swapped directly, which only changes the two elements concerned.

In three dimensions, the edge swapping often affects not only the two elements under consideration, but three or four in some cases. When swapping the diagonals or surfaces, the following three cases should be considered, based on the location of the intersection point (denoted by P) between the diagonal and the plane defined by the common triangular surface (denoted by T) of the two associated tetrahedra.

1) If P lies inside T, two tetrahedra are replaced by three new ones.

2) If P lies outside T, four tetrahedra are replaced by four new ones.
3) If P lies on the edges of T, three tetrahedra are replaced by two new ones.

2.2 The Advancing Front Algorithm

In the two-dimensional advancing-front algorithm [20, 21, 17], a line segment on the boundary is chosen as a basic front segment, e.g. segment AB shown in Figure 4. Then a suitable point is found, point C for instance, to create a triangular element ABC. Segments AC and CB are new fronts which are advanced by choosing suitable points to create new elements and new fronts. This advancing procedure continues until all of the fronts move onto the boundaries of the region or onto other fronts.

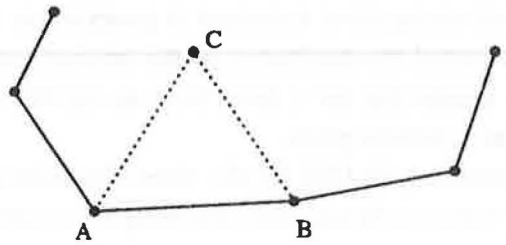

Figure 4: Advancing Front algorithm

In the three-dimensional advancing-front algorithm [22, 18, 19], the procedures of the algorithm to deal with a single domain are:

1. Generate triangular elements on the boundary surfaces of the domains. These are called front triangles and are stored in a list.
2. Choose an existing front triangle T from the list as a basic front.
3. Look for a suitable point to create a tetrahedron by connecting this point to triangle T.
4. Add any of the faces of this tetrahedron, which are not in the front list, to that list and remove the basic front from the list.
5. Return to Step 3 until the list is empty.

There are a few methods for defining the suitable points [20, 21, 17, 19]. A suitable point P for a basic front T is defined to satisfy the following requirements:

(a) Point P lies in the front-half space to which the process advances, i.e. the tetrahedron P–T has a positive volume.

(b) Tetrahedron P–T does not cross any existing fronts.

(c) Point P forms the greatest spherical excess to the basic front [19]. The spherical excess, Q, is:

$$Q = A + B + C - \pi$$

where A, B and C are the three angles of the spherical triangle, i.e. a curved triangle, formed by the penetration point of the connection lines between the suitable point and the vertices of the front triangle, which lies on the unit sphere centred at the suitable point.

With the advancing-front algorithm, a compatible mesh results when the list of the front triangles is empty. In some cases, the geometry cannot be completely divided into tetrahedral elements based on the boundary triangular elements generated previously. For instance, consider a hexahedron each of whose surfaces comprises two triangles from the front. It may not be possible to tetrahedralise the hexahedron without adding at least one interior point, depending on the pattern of triangles on the hexahedron surface.

[21] describes how to choose a suitable point for a basic front in two-dimensional advancing-front algorithm and gives a Fortran program for the two-dimensional advancing-front algorithm.

The two-dimensional advancing-front algorithm is discussed in detail in [17]. A method to generate internal points, which needs a rectangular grid to control their distribution is described. The generation of the internal points and their connection are completed separately.

A three-dimensional advancing-front technique is presented in [22]. This technique makes use of a *background mesh* to control the distribution of the internal points which are created when the front is advancing. This means that for a basic front an internal point is created, rather than an existing point is chosen, as a suitable point.

Data structures are presented in [18] for the three-dimensional advancing-front technique to avoid excessive computer time overheads when searching for a suitable point. A technique to check the intersection of surfaces is also described.

2.3 A Comparison of the Delaunay and the Advancing Front Algorithms

We make the following observations about the Delaunay and the Advancing Front Algorithms:

1. Both the algorithms can be implemented for convex and non-convex domains.
2. The computer storage needed by the two algorithms is approximately the same.
3. The operation count for both methods is dominated by the operation count for testing the crossing situation. In the insertion polyhedron algorithm, most of the work is spent finding the tetrahedra whose circumspheres contain a newly-inserted point and by testing the crossing situation. In the advancing-front algorithm, most of the work is spent testing if the generated elements contain other points and if the elements cross the existing fronts, which is much more laborious than testing the crossing situation in the insertion polyhedron algorithm. Therefore the operations needed by the advancing-front algorithm are about four times those needed by the insertion polyhedron method.

3 Some New Developments on Delaunay Triangulation

We have developed an automatic three-dimensional mesh generator (MSH3D) for use in EVEREST [14, 15, 16], a software package for the analysis of semiconductor devices in three dimensions, and for use in other engineering applications. The insertion polyhedron method is adopted in the generator. We have made some developments to deal with problems in the Delaunay algorithm and the insertion polyhedron method: degeneracy, the crossing situation, the identification of internal elements, mesh refinement and data structures.

In EVEREST the problem domain is defined as the union of a number of convex subdomains. We deal with the subdomains separately to reduce the occurrence of elements crossing the boundary and insert the points in a certain order in the insertion polyhedron method to avoid *surface element*

crossing for the degenerate case. We present a new method to identify the elements inside the domain, which is valid for all kinds of geometrical models. A new weight function is developed to generate the additional mesh points. The generator MSH3D based on the methods described in this report works both for convex and non-convex domains, including those with high aspect-ratio subdomains.

3.1 Introduction to the Mesh Generator MSH3D

The mesh generator MSH3D can be applied to both convex and non-convex domains, including those with high aspect-ratio subdomains. First the initial mesh, which only uses geometrical points, is generated. Subsequently additional points are generated automatically to refine the mesh until a suitable mesh is obtained.

The generator reads the geometrical information from the neutral file [5] which defines the subdomains of the domains. Each subdomain is meshed separately. The overall algorithm of MSH3D is:

1. Read the geometrical information from the neutral file.
2. Look for all the points on each surface, including those which are not used to define the geometry.
3. For each domain in turn:
   (1) define a big hexahedron as a *box*, which contains all of the points associated with the current domain, and dividing the hexahedron into six tetrahedral elements.
   (2) order those points by a natural numbering sequence according to the number of each point. This ordering ensures that the points on an interface are introduced in the same sequence when the two subdomains sharing the interface are meshed; this avoids surface elements crossing in the degenerate case as described in §3.4.2.
   (3) insert the points one-by-one into the existing mesh based on the order of the points using insertion polyhedron method.
   (4) delete the elements connected to one of the vertices of the box.
   (5) test for and remove the elements which cross the boundaries of the domain.
   (6) identify the elements inside the domain and delete the elements outside the domain.
4. Test if the two sets of the triangular elements on the interfaces formed by the tetrahedral elements in different domains cross each other and resolve the crossing situation.
5. Generate additional points and refine the mesh.
   1) calculate integer weights at the existing points by interpolation from those weights at geometrically important points which are specified by the user.
   2) create an additional point at a suitable position determined by the weights at the existing points. This is described in Section 3.6.2.
   3) insert the newly created point in the existing mesh using the Delaunay criterion.
   4) return to step 2 until no additional point can be created by comparing the weights at the points and the length of the element sides.

6. Write the mesh data to the neutral file.

Step 3.3 is the main part of the Delaunay algorithm. Step 3.5 is the most computationally intensive in the generator. In the following sections we discuss in more detail the implementation of Steps 3.3, 3.5 and 3.6 and highlight ways in which we have been able to reduce the computational effort required to generate a mesh.

3.2 Data Structures

A data structure which stores the neighbours of each tetrahedron is used. This makes some of the procedures in the generator more efficient, e.g. formation of the insertion polyhedron, identification of elements inside the domain, and testing of the crossing situation. We also store the coordinates of the centre and the square of the radius of the circumsphere for each tetrahedron which avoids their recalculation and saves a lot of computer time. Data structures which store the definition of the geometry in terms of points, lines, surfaces and volumes are used.

The data structure for the neighbour is that the four neighbours of each tetrahedron are stored in the array NEIBR(4, NELEMT). NEIBR(IP, NT) stores the number of the element which is opposite vertex IP of tetrahedron NT. The neighbours stored are only considered in the same domain. If the tetrahedron has no neighbour opposite point IP inside its subdomain, NEIBR(IP, NT) is set to point to the surface in the geometry definition which is opposite the point. In this case NEIBR(IP, NT) is given the negative value of the number of the surface to indicate that this element is on a boundary surface.

3.3 The Degenerate Case

Because the problem in the degenerate case is usually caused by computer round-off error, a small value ϵ can be given to control the effect of the round-off error when identifying the elements whose circumspheres contain the newly inserted point. As long as a suitable ϵ is chosen for each geometrical model, the degeneracy problem can be solved completely.

The precision for a computer is fixed, but the scale of the geometries and the radii of the circumspheres may vary greatly. Therefore the strategy to choose ϵ should depend on the precision and the scale of the circumsphere radius (R) when testing if this circumsphere contains the new point. In MSH3D, an arbitrary small value R/ER where ER= 1.D8 is given as ϵ, which can be adjusted automatically in the program, depending on whether the formed insertion polyhedron is compatible. If the insertion polyhedron is neither closed nor star-shaped, ER is adjusted gradually until a correct insertion polyhedron is obtained and this ER is used as the initial value in the following tests.

After all the elements whose circumspheres contain the newly inserted point are found, the surfaces forming the insertion polyhedron can be selected using the neighbour data structure. If the value of NEIBR for a triangular surface of a found tetrahedron does not appear in the numbers of the found elements, this surface must be part of the insertion polyhedron. Each triangular surface must have three neighbours in the surfaces of the insertion polyhedron. Otherwise the insertion polyhedron is not closed.

3.4 The Crossing Situation

It is computationally expensive to test if the generated elements cross the boundaries. Therefore, we endeavour to reduce the number of such occurrences. One approach is to define the surrounding box to be sufficiently large [11]. The success of this method relies on choosing the box to be sufficiently large in advance of starting the computation. We have adopted another approach, which is to deal with each subdomain separately. We have found that this reduces the number of elements crossing the boundaries, which we call tetrahedron-boundary crossing, but it produces *surface element crossing*, i.e. the two sets of the triangular elements on the interfaces formed by the tetrahedral elements in neighbouring domains cross each other, which is easier to deal with.

3.4.1 Tetrahedron Boundary-Crossing

The surfaces of the geometrical domain may be arbitrarily shaped. It is not easy to make a program to test if the tetrahedral elements cross these arbitrary surfaces. Therefore triangular elements with the geometrical points are generated on each boundary surface of the geometry in MSH3D. Then the tetrahedron boundary-crossing test is implemented by only considering the tetrahedra and the triangular elements on the boundaries. In [27] some methods to create boundary triangular elements are described.

In order to save the computer time, We have developed a three-layered approach to test if a tetrahedron crosses a triangle:

1. **Max/min tetrahedron/triangle test.** A tetrahedron ABCD cannot cross a triangle EFG if only one coordinate component satisfies one of the following inequalities,
   (a) $\max(X_A, X_B, X_C, X_D) < \min(X_E, X_F, X_G)$
   (b) $\min(X_A, X_B, X_C, X_D) > \mathrm{man}(X_E, X_F, X_G)$
2. **Max/min triangle/triangle test.** A triangle ABC cannot cross another triangle EFG if only one coordinate component satisfies one of the following inequalities
   (a) $\max(X_A, X_B, X_C) < \min(X_E, X_F, X_G)$
   (b) $\min(X_A, X_B, X_C) > \mathrm{man}(X_E, X_F, X_G)$
3. **Edge-triangle test.** An edge P_1P_2 and a triangle ABC cross each other if $P_1A \cdot P_1C \times P_1P_2$, $P_1C \cdot P_1B \times P_1P_2$ and $P_1B \cdot P_1A \times P_1P_2$ have the same sign.

If an element crosses the boundaries, a mesh point is introduced on the intersection to refine the mesh. Tetrahedron boundary-crossing does not happen for the convex domains and it only happens in some complex non-convex domains.

3.4.2 Surface Element-Crossing

The triangular elements on a boundary formed by tetrahedral elements satisfy the two-dimensional Delaunay triangulation, which is unique in the absence of the degeneracy. (Since the circumsphere of a tetrahedron in the mesh contains no other mesh points and the intersection of the boundary with the sphere is a circle, the triangles in the boundary mesh all satisfy the Delaunay criterion.) Therefore if the same points and the same region are used as an interface for two domains and on the interface the distribution of the points has no degeneracy, i.e. there are no more than three

points located on one circle, the two sets of triangular elements on the interface must be the same and surface element crossing can not happen.

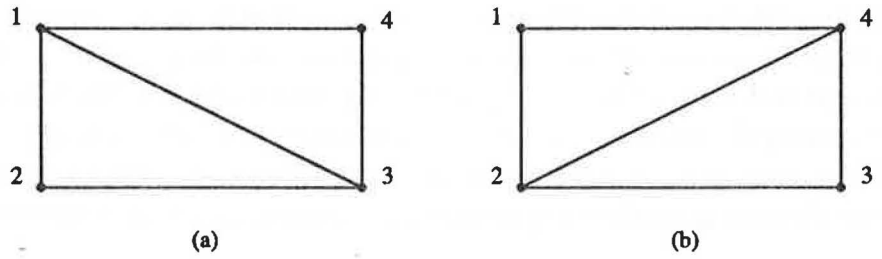

Figure 5: A degenerate case

In two-dimensional degeneracy, i.e. when more than three points lie on a circle, the Delaunay triangulation is not unique, e.g. two valid triangulations of four co-circular points are shown in Fig. 5. If that region is an interface or a part of an interface of two domains, surface element-crossing may happen. In practice, as a small value ϵ is introduced to control the computer round-off error, the point inserted finally is not contained by the existing element formed by the other three points in Figure 5, i.e. the mesh in Figure 5(a) must be generated by finally inserting Point 4 (or 2), that in 5(b) by finally inserting Point 3 (or 1). If the points are introduced in a fixed order, the mesh is unique in degenerate case. Therefore, surface-element crossing does not arise in the degenerate case if we insert the points in a definite order and choose an appropriate value for ϵ, as discussed in §3.3.

Surface element-crossing may occur when an interface between two subdomains contains a different set of points, depending from which subdomain it is viewed. A simple example of this is shown in Figure 6. In Figure 6(a) Surface 1267 is a boundary surface of one subdomain, but Surface 1234567 is the surface viewed from the other subdomain; therefore the two sets of triangular meshes as shown in Fig. 6(b) and (c) will cross each other. Supplementary points should be introduced on the intersection to remove the crossing. In the neighbour data structure, it is indicated clearly which surfaces of the tetrahedral elements are located on which subdomain boundary.

3.5 Element Deletion

After all the points are inserted into the big box with the insertion polyhedron method and the tetrahedron boundary-crossing situation is resolved, the elements inside and outside the current domain are identified. The procedure to identify the elements inside the domain is:

- Find a suitable surface so that all of the involved geometrical points in the current domain are located in the same half space that is defined by the extension of this suitable surface.
- Find an element one of whose triangular surfaces is on the suitable surface. This element must be inside the domain.
- Based on the first element found inside the domain, the others inside the domain can be found quickly by means of the neighbour relationship and the idea of the advancing-front algorithm in the mesh generation. Each surface of the tetrahedra, except for those whose associated value of NEIBR is less than zero, is chosen as a basic front and its neighbour must be inside the current domain.

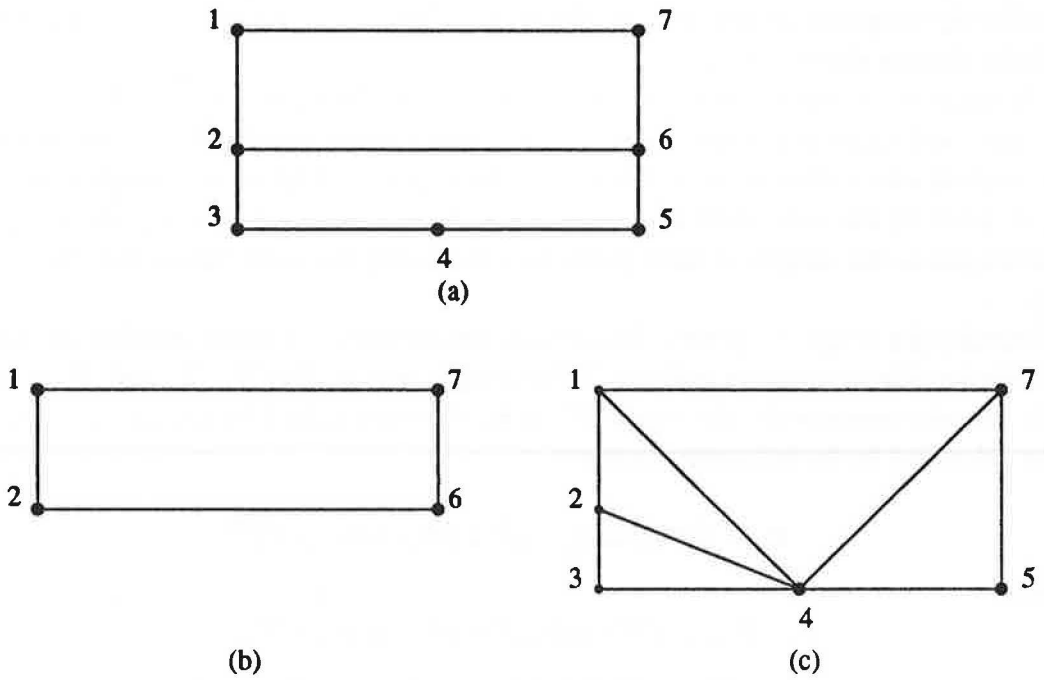

Figure 6: The interface appears differently

3.6 Mesh Refinement

At this stage a mesh with geometrical points already exists but it is not suitable for the finite element analysis. The mesh requires refinement, i.e. additional mesh points need to be introduced. These points should be created automatically and their distribution should be sensible. There are two main approaches to refinement: *a posteriori* refinement and *a priori* refinement. In the former, the solution to the problem on the existing mesh is used to guide where new points are introduced. For examples of this approach we refer the reader to [7, 8, 9, 13]. In the latter, Bryant [1] presents a method for two-dimensional mesh refinement which makes use of distance function defined at each existing mesh point to control the position of the newly produced points. The functions are concerned with the length of the element sides. Weight functions are introduced for three-dimensional mesh refinement in [3, 4], which are concerned with the volume of the elements.

In MSH3D, weight functions in three directions at each point are adopted which control the distribution of the points more efficiently and which ensure that the size of the elements generated varies smoothly in the domain.

3.6.1 Weight Functions

Each point is given three values as weight functions to indicate separately the average length of the finally produced element sides in the three axis directions of the coordinate system. The smaller the value of the weight at a point, the smaller the size of the elements around that point.

The weight function at the geometrical points are given by the user. In order to simplify the input for the weights, three integers are required, rather than three real values, as the weights in MSH3D. The integers only indicate the rate of variation in the size of elements in the domain, i.e. a weight of 10 defined at point A and 50 at point B is equivalent to a weight of 1 at A and 5 at B.

The easy way to think about the integer weights is: consider the average element size to be 100% and define the weights at normal points as 100; define different weights at some special points about which the element size is to vary.

The integers are transferred into real values by a scaling factor, which ensures that the average of the three real values at a certain normal point is approximately equal to the average of the length of the element sides connected to the point. This certain point and the average length of the element size are given by the user. MSH3D requires the weights to be given only at some geometrically important points; the weights at other points are obtained by the interpolation from the important points.

Generally, the weight at a point is different in each direction, and varies smoothly following two perpendicular ellipses as shown in Figure 7. For point O with weights W_x, W_y and W_z in X, Y and Z axis direction respectively, the weight W_o in the direction defined by $(x_1, y_1, z_1) - (x_2, y_2, z_2)$ can be calculated by the following equations:

$$W_o = [(W_{xy} cos Q_{z-xy})^2 + (W_z sin Q_{z-xy})^2]^{1/2} \tag{1}$$

where,

$$W_{xy} = [(W_x cos Q_{xy})^2 + (W_y sin Q_{xy})^2]^{1/2} \tag{2}$$

$$Q_{xy} = arctan(|(y_1 - y_2)/(x_1 - x_2)|) \tag{3}$$

$$Q_{z-xy} = arctan(|z_1 - z_2|/[(x_1 - x_2)^2 + (y_1 - y_2)^2]^{1/2}. \tag{4}$$

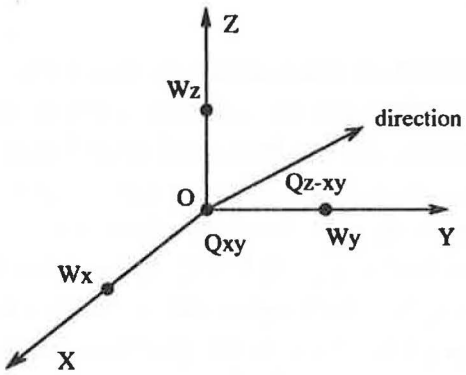

Figure 7: The calculation of the weight in any direction

3.6.2 Additional Point Generation

The procedure to generate an additional point is:

1. Choose an element.
2. Test whether the sides of that element are acceptable or not.
3. Bisect an unacceptable side to produce a new point.

An unacceptable side is one which satisfies,

$$L > (W_1 + W_2)/1.5, \tag{5}$$

where L is the length of the side, W_1 and W_2 the weights at the two end points (X1 and X2) in the direction defined by that side.

The three coordinates and the three weights of the new point are,

$$X_i = (W_1 * X2_i + W_2 * X1_i)/(W_1 + W_2) \tag{6}$$

$$W_{xi} = 2 * W_{1X_i} * W_{2X_i}/(W_{1X_i} + W_{2X_i}) \tag{7}$$

The Delaunay criterion is then applied to insert the new point into the mesh. The mesh refinement is finished when all the existing element sides are deemed acceptable.

The mesh refinement is carried out in each domain separately. The new points produced on interfaces are inserted to the neighbouring domains after the mesh refinement is completed in the current domain.

4 Test Examples

In order to illustrate the functionality of the mesh generator MSH3D, we present three test cases. The geometrical description of each device is provided in a geometry neutral file [5] and the nodal weights for the refinement process are provided in a separate file.

The generator can display the geometrical models, the triangular elements on the surfaces both in two dimensions and three dimensions and the tetrahedral mesh. The generator can be easily run interactively.

The first example is a block $p - n$ junction [16] originally specified by GEC Research in order to provide a simple test structure in which one-dimensional, two-dimensional and three-dimensional physical behaviour is exhibited. The structure, shown in Figure 8, is non-convex and the upper part of the structure is made of a different type of Silicon to the base. Therefore the algorithm has to ensure the integrity of the material interface.

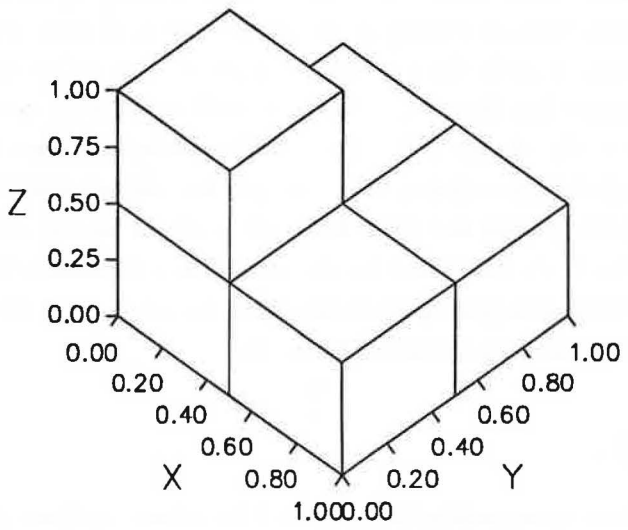

Figure 8: Block $p - n$ Junction: The Geometry

In Figure 9 we see the final mesh which contains 204 nodes and comprises 814 tetrahedra. This was genereated from the initial geometric specification of the device which contains 22 nodes.

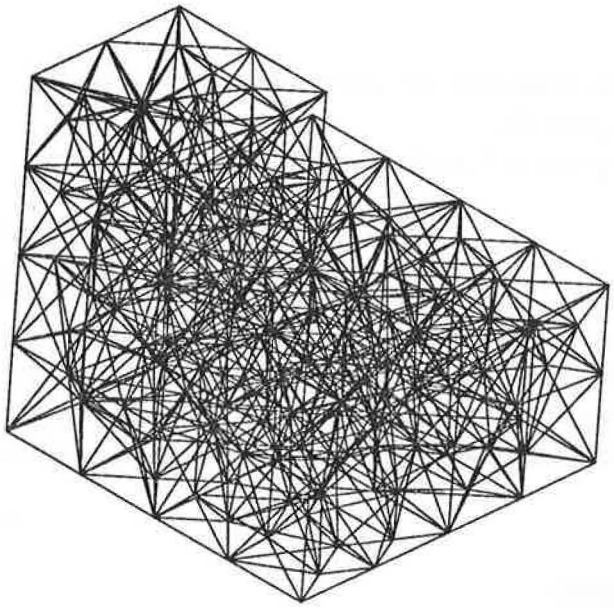

Figure 9: Block $p - n$ Junction: The Mesh

Figure 10 shows the surface mesh corresponding to the mesh shown in Figure 9. The view in this figure is from the other side of the device to the viewpoint used for Figures 8 and 9.

The second example is a corner diode with a thin oxide overlay [16]. This device is not only non-convex but has three distinct material regions: the oxide layer and two types of silicon. The geometry is shown in Figure 11.

In Figure 12 we see the final mesh which contains 593 nodes and comprises 2424 tetrahedra. The initial geometric specification of this device contained 59 points. Figure 13 shows the surface mesh corresponding to the mesh shown in Figure 12.

The third example is a metal-oxide-silicon field transistor (MOSFET). This is structurally more complicated than the first two examples. It comprises four distinct material regions, each of whose integrity must be preserved in the meshing process. In addition, the thin layer of oxide material on top of the base means that there is a subdomain with high aspect ration elements. (The thin oxide layer at the front of the device is 3×10^{-6}cm thick while the base region is 2×10^{-4}cm thick.) Furthermore, the geometry contains some surfaces not aligned with the major axes, as we can see in the transition between thin and thick layers of oxide on the top surface of the domain.

In Figure 15 we se the final mesh for this device which contians 529 nodes and comprises 2040 tetrahedra. This mesh was generated starign from an initial mesh of 44 nodes. The surface mesh corresponding to this mesh is shown in figure 16.

5 Summary

In this report we have surveyed the Delaunay and the advancing-front algorithm for three-dimensional mesh generation. We have presented a new implementation of the Delaunay algorithm which is valid for convex and non-convex domains, and for domains whose subdomains have high aspect-ratios. We have addressed the problems which sometimes arise in computing the Delaunay triangulation of a non-convex domain. We have described MSH3D, a computer program embodying the implementation of the Delaunay algorithm described in this report. The data structures used are discussed in

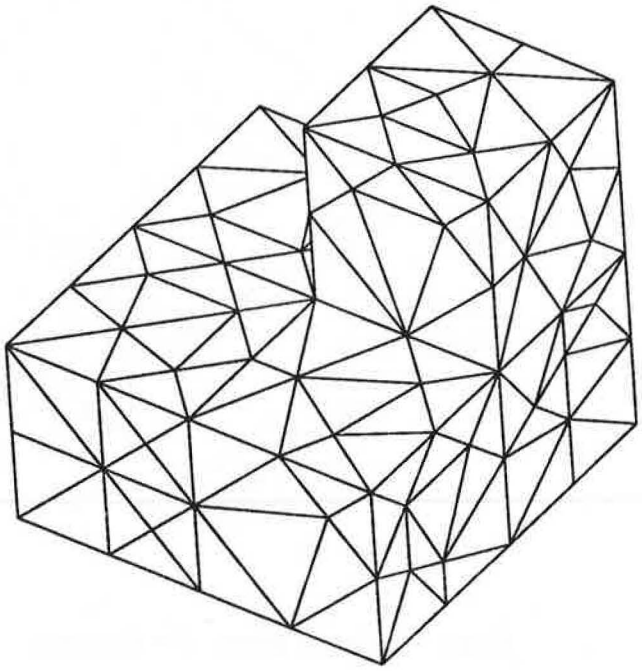

Figure 10: Block $p - n$ Junction: The Surface Mesh

detail and some examples arising from semiconductor device modeling are presented.

References

[1] **C.F. Bryant** "Two-Dimensional Automatic Triangular Mesh Generation" *IEEE Trans. on Mag.* **MAG–21** No.6, 2547–2550 (1985).

[2] **James C. Cavendish, David A. Field and William H. Frey** "An Approach to Automatic Three-Dimensional Finite Element Mesh Generation" *Int. J. Num. Meth. Eng.* **21** 329–347 (1985).

[3] **J.L. Coulomb, Y. Du Terrail, G. Meunier** "Two 3D Parametered Mesh Generators for the Magnetic Field Computation" *IEEE Trans. on Mag.* **MAG–20** No. 5, 1900–1902 (1984).

[4] **J.L. Coulomb, Y. Du Terrail, G. Meunier** "A Finite Element Package for Magnetic Computation" *IEEE Trans. on Mag.* **MAG–21** No.6, 2499–2502 (1985).

[5] **K.P. Duffey, C.R.I. Emson** "RALBIC — A Simple Neutral File for Finite Element Data" Rutherford Appleton Laboratory, RAL–87–103 (1987).

[6] **N. Ferguson** *Aspects of Delaunay Triangulation and Mesh Generation in Two and Three Dimensions*, Ph.D. Thesis, University of Dublin, (1990).

[7] **P. Fernandes, P. Girdinio, P. Molfino, M. Repetto** "An Enhanced Error Estimator Procedure for Finite Element Field Computation with Adaptive Mesh Refinement" *IEEE Trans. Magn.* **26** No. 5, 2187–2189 (1990)

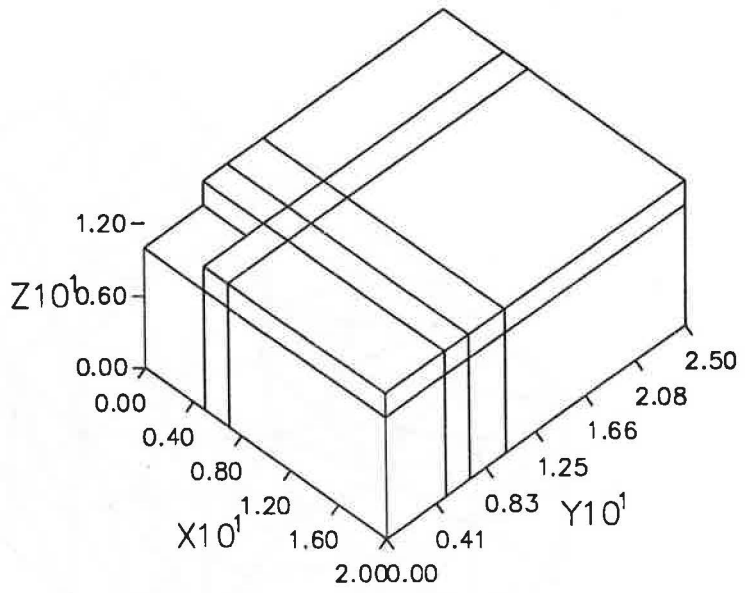

Figure 11: Corner Diode: The Geometry

[8] **P. Fernandes, P. Girdinio, G. Molinari, M. Repetto** "Local Error Estimation Procedures As Refinement Indicators in Adaptive Meshing" *IEEE Trans. Magn.* **27** No. 5, 4189–4192 (1991)

[9] **P. Fernandes, P. Girdinio, M. Repetto, G. Secondo** "Refinement Strategies in Adaptive Meshing" *IEEE Trans. Magn.* **28** No. 2, 1739–1742 (1992)

[10] **P.L. George, F. Hecht, E. Saltel** "Constraint of The Boundary And Automatic Mesh Generation" from *Numerical Grid Generation In Computational Fluid Mechanics'88* (Ed. S.Sengupta et al) 589–597 (1988).

[11] **P.L. George, F. Hecht, E. Saltel** "Automatic 3D Mesh Generation With Prescribed Meshed Boundaries" *IEEE Trans. on Mag.* **MAG–26** No.2, 771–774 (1990).

[12] **P.L. George, F. Hecht, E. Saltel** "Fully Automatic Mesh Generation for 3D Domains of Any Shape" *Impact of Computing in Science and Engineering* (1990).

[13] **N.A. Golias, T.D. Tsiboukis** "Three-Dimensional Automatic Adaptive Mesh Generation" *IEEE Trans. Magn.* **28** No. 2, 1700–1703 (1992)

[14] **C. Greenough, D. Gunasekera, C.J. Fitzsimons, P.A. Mawby, M.S. Towers** "Modelling Semiconductor Devices in Three Dimensions" from *Proceedings of the Sixth International Conference on the Numerical Analysis of Semiconductor Devices: NASECODE VI* (ed. J.J.H. Miller) Boole Press, Dublin (1989)

[15] **C. Greenough, C.J. Fitzsimons, R.F. Fowler** "Software for Modelling Semiconductor Devices in Three Dimensions" *RAL–91–042* (1991)

[16] **D. Gunasekera, C. Greenough** "Comprehensive Testing of the Solver Module" EVEREST ESPRIT Project 962E, Workpackage 5 Report (1988)

[17] **S.H. Lo** "A New Mesh Generation Scheme for Arbitrary Planar Domains" *Int. J. Num. Meth. Eng.* **21** 1403–1426 (1985).

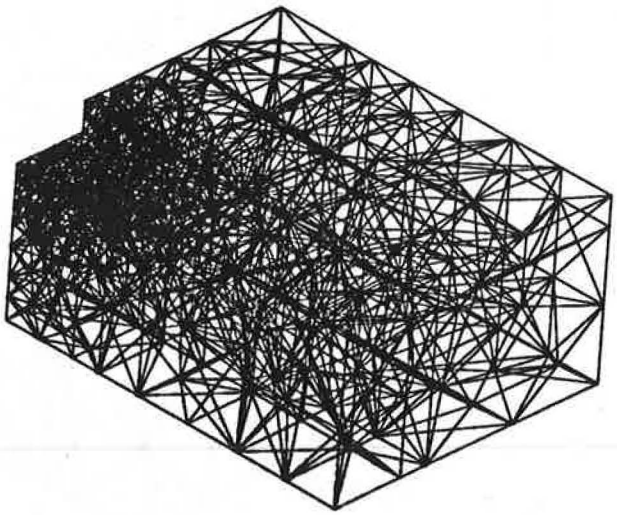

Figure 12: Corner Diode: The Mesh

[18] **R. Lohner, P. Parikh** "Generation of Three-Dimensional Unstructured Grids by The Advancing Front Method" *Proceedings AIAA 26th Aerospace Sciences Meeting* Reno, NV, AIAA Paper 88-0515 (1988).

[19] **Nguyen-van-phai** "Automatic Mesh Generation with Tetrahedron Elements" *Int. J. Num. Meth. Eng* **18** 273–289 (1982).

[20] **Arne Maus** "Delaunay Triangulation and The Convex Hull of n Points in Expected Linear Time" *B.I.T.* **24** 151–163 (1984).

[21] **J.M. Nelson** "A Triangulation Algorithm For Arbitrary Planar Domains *Appl. Math. Modeling* **2** 151–159 (1978).

[22] **J. Peraire, J. Peiro, L. Formaggia, K. Morgan, and O.C. Zienkiewicz** "Finite Element Euler Computations in Three Dimensions" *Proceedings AIAA 26th Aerospace Sciences Meeting* Reno, NV, (1988).

[23] **A. Perronnet** "A Generator of Tetrahetral Finite Elements for Multimaterial Objects or Fluids" from *Numerical Grid Generation in Computational Fluid Mechanics'88* (Ed. S.Sengupta et al) 719–728 (1988).

[24] **W.J. Schroeder** "Geometry-Based Fully Automatic Mesh Generation And The Delaunay Triangulation" *Int. J. Num. Meth. Eng.* **26** 2503–2515 (1988).

[25] **Michael Sever** "Delaunay Partitioning in Three Dimensions and Semiconductor Models" *COMPEL* **5** No. 2, 75–93, (1986).

[26] **D.N. Shenton, Z.J. Cendes** "Three-Dimensional Finite Element Mesh Generation Using Delaunay Tessellation" *IEEE Trans. on Mag.* **MAG-21** No.6, 2535–2538 (1985).

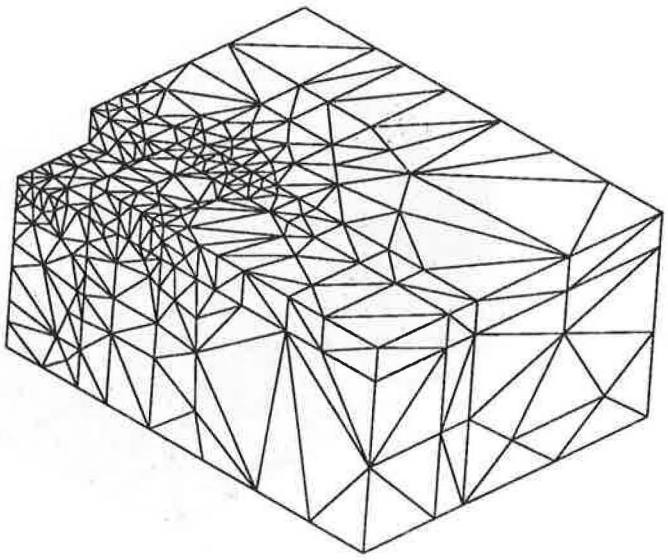

Figure 13: Corner Diode: The Surface Mesh

[27] **Yuan Jiansheng, Shao Hanhuang** "Automatic Three-Dimensional Boundary Element Mesh Generation" from *Electromagnetic Fields in Electrical Engineering (Proceedings of BISEF'88)* (Ed. Ding Shunnian) I. Academic Publishers, Oxford 365–367, (1989).

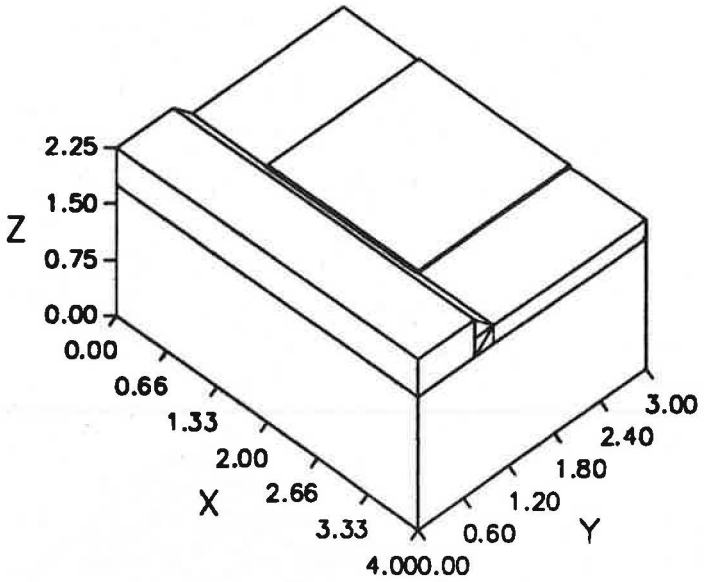

Figure 14: MOSFET: The Geometry

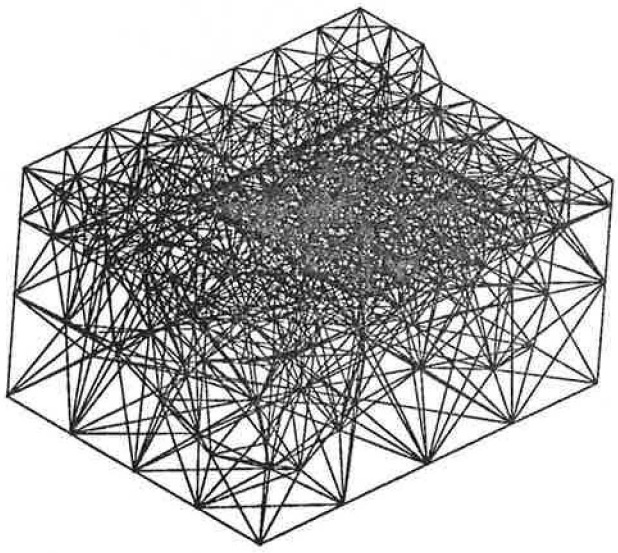

Figure 15: MOSFET: The Mesh

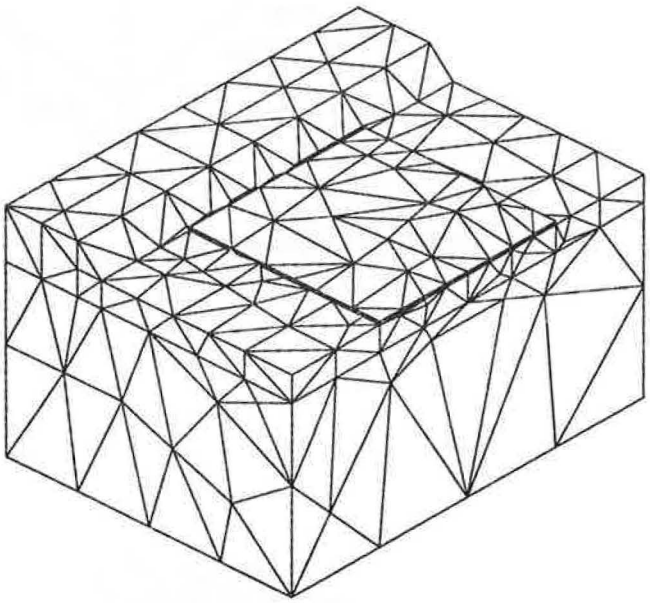

Figure 16: MOSFET: The Surface Mesh