Pergamon

Mathl. Comput. Modelling Vol. 24, No. 10, pp. 97–112, 1996
Copyright©1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0895-7177/96 $15.00 + 0.00

PII: S0895-7177(96)00168-9

A Simple Unstructured Tetrahedral Mesh Generation Algorithm for Complex Geometries

B. K. KARAMETE
Engineering Sciences Department, Middle East Technical University
06531, Ankara, Turkey

H. U. AKAY
CFD Laboratory, Department of Mechanical Engineering
Purdue School of Engineering and Technology, IUPUI
Indianapolis, IN 46202, U.S.A.

T. TOKDEMIR AND M. GER
Engineering Sciences Department, Middle East Technical University
06531, Ankara, Turkey

(Received and accepted October 1995)

Abstract—This paper describes the logic of a dynamic algorithm for a general 3D tetrahedrization of an arbitrarily prescribed geometry. The algorithm requires the minimal surface information. The solid modelling of the 3D objects needs not to be given in full explicit connectivity definitions of the surface triangles. The generated tetrahedrons have empty circumspheres which are the indication of the Delaunay property. A new automatic node replacement scheme reflecting the initial surface nodal spacings is developed. The successive refinement scheme results in such a point distribution that the algorithm does not require any surface conforming checks to avoid penetrated surface boundaries and overlapped tetrahedrons. The surface triangles become a direct consequence of interior tetrahedrization. The rules of the generation algorithm are simple to understand and to program. Some of the existing methods in the literature and the effectiveness of the geometric searching strategies are discussed in the context.

Keywords—Unstructured grids, Tetrahedral elements, Computational grid, Mesh generation, CFD.

1. INTRODUCTION

The use of finite elements and finite volumes, irrespective of the nature of the application, requires discretization of the domain over which a set of governing equations usually in the form of a PDE is to be solved, subject to a set of boundary conditions (BC). Both the geometry of the domains and the PDE's and BC's being problem specific show great variation from one application to another. Therefore, a general purpose mesh generation algorithm (GPMGA) applicable to arbitrary domains has been and still is in great demand.

One of the first attempts in developing a GPMGA is the work of Green and Sibson [1]. It was based on an optimal triangulation algorithm of Pitteway [2]. The first implementations of triangulation algorithms were in 2D interpolation of random data and used for contouring and/or surface generation by Mclain [3] and Sibson [4]. The advances in FEM and the convenience of triangulation made the triangulation algorithm very much in scrutinization; in the early 1980's, the

Part of this research was conducted while the first author was visiting IUPUI's CFD Laboratory through a NATO fellowship provided by the Natural Science and Technology Institute of Turkey.

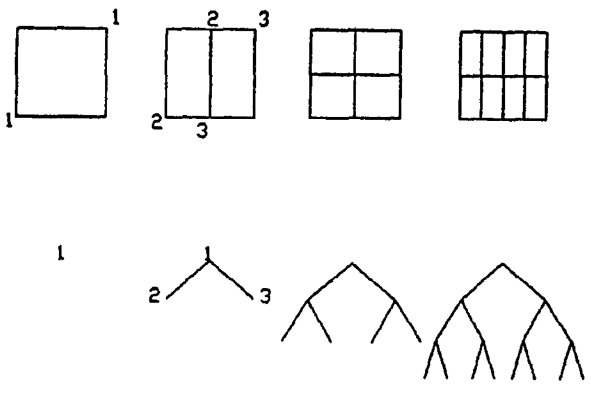
principles of the computation of the Dirichlet tesselations were discussed in detail by Watson [5] and Bowyer [6]. They proposed algorithms resulting in triangles with Delaunay properties. These algorithms which can be used with convex and simple domains minimized the computation time for the completion of triangulation. The resulting meshes of triangular cells with no interior and circumferential nodes thus obtained are unstructured. Therefore, in such an unstructured mesh, the number of cells including a node as a vertex is not necessarily the same for all nodes within the domain. Although the resulting unstructured meshes fail to comply with the directionality concept, they provide a very powerful tool in discretization of complex shapes. Furthermore, unstructured meshes provide ease and speed to the solution of most PDE's discretized based on integral procedures such as FE and Finite Volume, particularly over complex shapes.

2. PREVIOUS METHODS

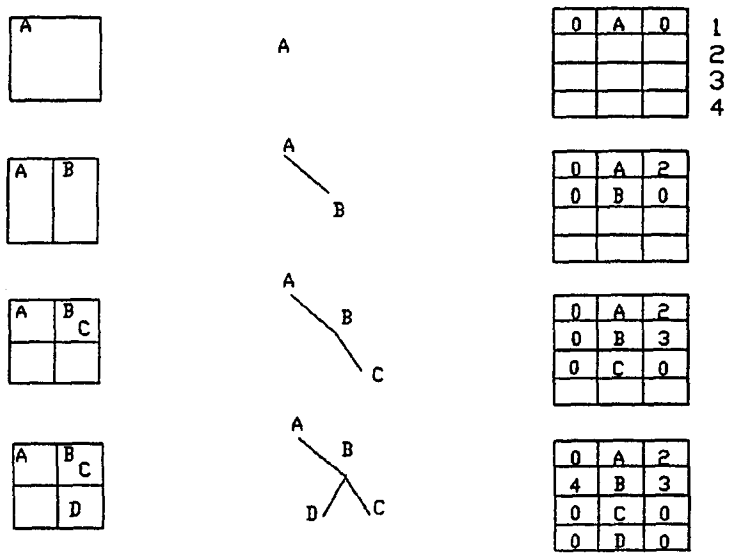
The unstructured mesh generation methods for CFD applications and adaptive grid design methods have been summarized and discussed by Perarire [7] and Kikuchi [8], respectively. The task to be undertaken in forming up a triangular mesh out of a set of arbitrary points is a very trivial one for the human brain. Yet, the translation of this process into a robust, efficient computational technique is difficult as pointed out by Weatherill [9]. This time can be minimized if the points are first ordered such that contiguous points in the list are neighbors in physical space. An intuitive way of achieving this is to subdivide the domain. The algorithm revisited by Bonet [10] is to store the points in a binary tree in such a way that the structure of the tree reflects the positions of the points in space. The extension of binary tree searching to N-dimensional problems is known as a digital tree algorithm. Broadly speaking, an alternating digital tree (ADT) can be defined as a binary tree in which a set of n points are stored following certain geometrical criteria. These criteria are based on the similarities arising between the hierarchical and parental structure of a binary tree and a recursive bisection process: each node in the tree has two sons; likewise a bisection process divides a given region into two smaller subregions. Consequently, it is possible to establish an association between tree nodes and subregions of the domain as shown in Figure 1a. The construction of the ADT structure is depicted in Figure 1b. It should be noted here that the region related to a given node k contains all the subregions related to nodes descending from k, and all the points stored in these nodes must also lie inside the region represented by node k. This feature can be effectively used to reduce the cost of a geometric searching process by checking, at any node k, the intersection between the searching range (a, b) and the region represented by node k, namely, (c, d). If these two regions fail to overlap, then the complete set of points stored in the subtree rooted at k can be disregarded from the search, thus avoiding the need to examine the coordinates of every single point.

There exists a considerable cost of the insertion process of the ADT algorithm if compared with the cost of storing the points in a sequential list, but justifiable in view of the reduction in searching costs that ADT structures will provide. Bowyer's point insertion algorithm for the Delaunay triangulation requires a considerable amount of work to be implemented efficiently in a distributed machine, thus considerably long computational time. It is further costly to check for geometric intersection when new triangles are created. Furthermore, the Delaunay triangulation is designed for convex domains. Therefore a boundary conforming routine should be adopted for the algorithm in order to cope with nonconvex multiply-connected planar domains.

Lo [11] has designed an efficient algorithm for the construction of Delaunay triangulations over nonconvex planar domains. He proposed a method for interior node generation by bombarding the region with constant y lines. The spacing between any two imaginary horizontal lines is exactly equal to the average element size. Each horizontal line must cut the domain in an even number of points, and the intersection points are arranged in ascending magnitude of x. Nodes are generated on this horizontal line between the cuts according to the prescribed spacing. This only suggests a series of potential positions where nodes can be generated; however, whether a



(a) The relation between the binary nodes and the subregions of the domain.



(b) The construction of ADT structure and its memory management.

Figure 1.

node is finally generated depends on how close it is from the domain boundary. Therefore, a quality check is a must as if the point were a vertex of a boundary triangle. If the assumed triangle is nearly equilateral, then the point is accepted as an interior node. The triangulation is advanced by updating the moving front of newly created triangles in a cyclic manner till no point lies unconnected on the domain. The main difference of Lo's algorithm from Bonet's ADT structure is its geometrical modelling. The points can be so arranged in a list that the contiguity in the physical space is achieved since the positions of horizontal lines can address the location of interior nodes.

Fukuda and Suhara [12] and Cavendish [13] generated interior nodes using random numbers and some geometrical restrictions in determining the location of the nodes. These algorithms have certain drawbacks as discussed in detail in Lo's previous paper [14].

One of the latest studies on unstructured grid generation is performed by Suzuki [15]. In the moving front technique which has been discussed by Löhner and Parikh [16], it is required that the candidate nodes must be situated to the right of the generating front line segment to prevent overlapped triangles. This is a sign test of a cross product between the line segment $|AB|$ and the line connecting one of the generating nodes and the candidate node $|AC|$. However, in contrast to this region exclusion method via the cross product of $\overrightarrow{AB} \times \overrightarrow{AC}$, Suzuki translated and rotated the coordinates so that the nodes located to the right of the generating line segment are moved into the positive side of the v axis which is the transformed y axis. The point C giving the smallest v coordinate of the circumcenter of the triangle $\overset{\triangle}{ACB}$, is selected among the candidate nodes.

Since the triangulation algorithms consist of general methods to link the given point distribution by triangles, the quality of the triangles should be provided by an efficient internal node generation scheme. The locations of internal nodes can be estimated by either a functional relationship or a rule based algorithm. The example to the functional generator is the algebraic serendipity mapping method automated by Zienkiewicz and Phillips [17]. In this method, the selected concave, say, eight nodes in physical coordinates are enough to interpolate the interior nodes explicitly by $x(\xi,\eta) = \sum_{i=1}^{8} x_i N_i(\xi,\eta)$ and $y(\xi,\eta) = \sum_{i=1}^{8} y_i N_i(\xi,\eta)$, where N_i's are the shape function and ξ and η change between $[-1,1]$. Unfortunately, this kind of point generation is difficult since many serendipity blocks are needed to cover a complex geometry. Lo's region bombarding algorithm is a rule based approach. It checks the quality of the triangle by considering the minimum value of the norm $|\overrightarrow{AC}|^2 + |\overrightarrow{BC}|^2$ for a generating line $|AB|$.

The present study which is discussed below is somewhat similar to the approach depicted by Suzuki. However, it does not involve any axis transformation, and its generation algorithm is quite different and simpler.

3. METHOD

3.1. General

One of the advantages of Delaunay triangulations, as opposed to triangulations constructed heuristically, or rule based, is that they automatically avoid forming triangles with small included angles as shown by Lawson [18] and Sloan [19]. Because of this property, Delaunay triangulations are particularly suited to grid generation for finite element analysis and contouring algorithms. A number of algorithms for constructing Delaunay triangulations have been proposed [20–26].

The arguments that have been stated for tetrahedrons are almost exactly the same for triangle formations. In fact, the 3D algorithm is basically nothing but the extension of the algorithm first applied to 2D geometries. Although the following discussions are focused on 3D mesh generation, the examples, the differences, and the matching parts with the 2D case are expressed when necessary.

One of the main differences of this study from those in and the ones discussed in Section 2 is its simplicity in generating the Delaunay property tetrahedrons without the necessity of an extra check for geometric intersection to avoid overlapped tetrahedrons and surface boundary penetration. By means of a few basic rules, boundary integrity and conformity have been established. In this method, the direction of generation of the tetrahedrons cannot be determined *a priori* as opposed to advancing-front techniques. This further imposes the method to be implemented regarding the dynamic programming strategies involving dynamically sized array definitions.

Moreover, the algorithm requires minimal surface information. The mesh should be created from the data of a mesh of its boundary. The surface triangulation representing the boundary of the domain should be given explicitly. The mesh of the boundary in terms of explicit definitions of triangles are actually popular as implemented in the study of George [27]. However, this requires an additional task of surface mesh generation. In this study, the generation task of the surface triangles is not needed at all; in fact the surface triangulation comes out to be a direct result of tetrahedrization of the interior domain. Hence, the geometric modelling time is considerably minimized by dividing the 3D geometry into patches over which the surface normals are assumed to be the same. The surface patch numbers are encapsulated in the nodal information, i.e., each node has the information of which patch(es) it belongs to.

Since the Delaunay triangulation methodology is a tool for convex domains, the nonconvex and/or multiply-connected domains should be handled by other means. In other words, the mesh generator engine should be modified to produce the boundary-conforming grids. The most popular method is the sweepline concept as discussed by Fang and Kennon [28]. The basic idea behind the method is to exclude or sweep the unnecessary triangles which are not inside the actual boundary of the domain after the convex mesh generation has been completed without paying attention to the concavity of the boundary. In this study, however, the problem has been solved by imposing a simple rule, saying that the triangles or the tetrahedrons in the 3D case cannot be generated from boundary line segments or boundary surface triangles in 3D. The rule is too simple but efficient. Furthermore, it can easily be plugged into an existing convex mesh generator without destroying the code.

The geometry information can be fed into the algorithm either by means of a simple format text file which can be an output of another program or the output of a CAD system which has already been rearranged for the desired input of the algorithm.

3.2. Data Structure

In most branches of computational mechanics, the choice of a suitable data structure with which to manipulate data and apply the algorithmic operations is of key importance. The data structure has an important influence on the efficiency of an algorithm, both readable and robust. In the work to be discussed, it has been designed to use five lists of simple dynamically sized array structures: *point*, *tetrahedron*, *point-tetrahedron*, *search*, and *adt lists.*

The 3D world is represented by means of the surface patches. The *point-list* holds the nodal information. Each surface node is affiliated with its three (x, y, z) coordinates and the number of patches it belongs to and the corresponding patch numbers. For instance, a node at the corner of a cube possesses three patch numbers since it is in the neighborhood of three faces having different surface normals. Similarly a node on the face has only one surface patch number. The second list which is assigned to store the output data of the tetrahedrization is termed as the *tetrahedron-list.* Each of its items encapsulates a four element integer array holding the vertex node numbers of a tetrahedron. Finally, to speed up the searching processes, the third list is designed to hold the cross information of how a node is linked with its surroundings. The *point-tetrahedron-list* has one integer indicating the number of tetrahedrons and an integer array having the corresponding numbers of tetrahedrons that the node is belonging to. The items of the *point-tetrahedron-list* would be definite through tetrahedron generations. Initially, the tetrahedron lists of the points are empty since no tetrahedrons are being generated yet. The *search-list* item is a dynamic variable storing only an integer value pointing to the number of the nodes in the *point-list.* The *adt-list* item has the data fields of *Level* showing the bisection coordinate, the longinteger valued *Right* and *Left* sons, and the rectangular range *RangeRecord*, where the point resides. The data structure of the algorithm for a unit cube has been depicted in Figure 2.

As soon as the algorithm starts to generate tetrahedrons, a new *tetrahedron-list* is created and updated at the insertion of each new tetrahedron having four node numbers indicating the vertices. Contrary to the growing *tetrahedron-list*, the *search-list* shrinks by disposing its items as soon as the possible candidate nodes start diminishing through tetrahedron generation. At the end of the algorithm, there remains no *search-list* item since every node is a member of a tetrahedron and the domain is completely discretized by the tetrahedrons.

3.3. The Main Generation Algorithm

Regarding a few basic rules itemized and explained below, the tetrahedrization engine can easily be explained.

- The generating triangular surface is a triangle which has been given the ability to seek the appropriate node among the candidate nodes in order to generate a tetrahedron.
- Each tetrahedron has three generating triangular surfaces since the fourth one has already been used to generate the tetrahedron itself as shown in Figure 3a.

Point-List

Point no.	x	y	z	Total Patches	Patch no.
0	0.0	0.0	0.0	3	1 4 5
1	1.0	0.0	0.0	3	1 2 5
2	0.0	1.0	0.0	3	0 1 4
3	1.0	1.0	0.0	3	0 1 2
4	0.0	0.0	1.0	3	3 4 5
5	1.0	0.0	1.0	3	2 3 5
6	0.0	1.0	1.0	3	3 4 0
7	1.0	1.0	1.0	3	2 3 0
8	0.5	1.0	1.0	2	0 3

Point-Tetrahedron-List

Point no.	Total Tetras	Tetrahedron no.
0	5	0 2 3 5 7
1	5	0 1 3 4 6
2	2	5 7
3	3	3 6 7
4	3	0 1 2
5	2	1 4
6	2	2 5
7	2	4 6
8	8	0 1 2 3 4 5 6 7

Tetrahedron-List

Tetra no.	Node list
0	0 4 1 8
1	4 1 8 5
2	4 8 0 6
3	0 8 1 3
4	1 8 5 7
5	8 0 6 2
6	8 1 3 7
7	8 3 0 2

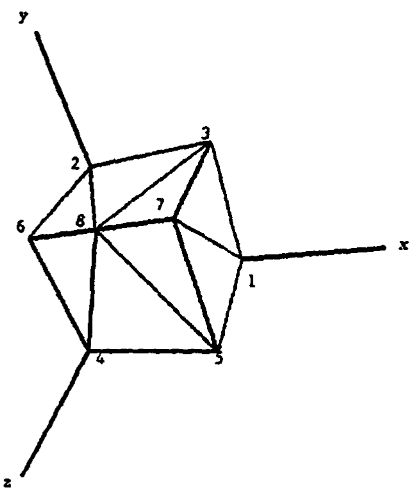
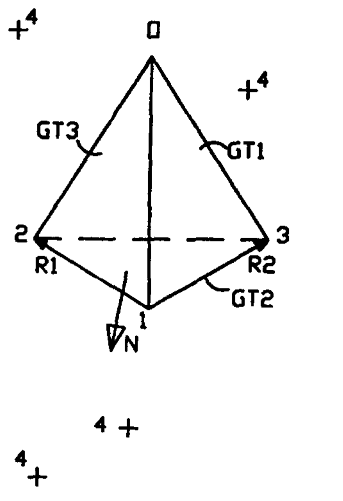


Figure 2. Database structure for a unit cube.

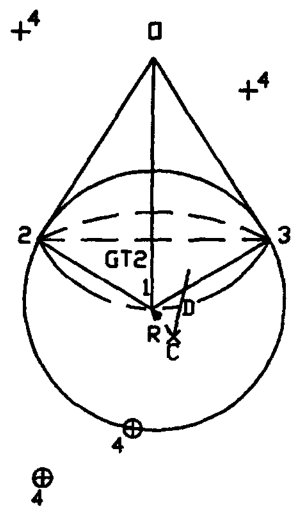
- Each generating triangle (GT) is to be processed only once to prevent the idle tetrahedrizations.
- The surface triangles are said to be illegal generating surfaces. The algorithm immediately skips to another GT in those cases. This is the sufficient condition to cope with nonconvex complex geometries.

The above rules of mesh generation are exactly the same in the 2D case except for the terminology. In the 2D case, the generating geometry is a line segment called the generating line (GL). Each triangle has two GL's, and the boundary line segments are said to be illegal GL segments. Similarly, each GL is processed only once to avoid idle triangulations.

Under the assumption of the existence of the first tetrahedron, its three GT's are tried one after another for further tetrahedrizations. The nodes at the other side of a GT plane with respect to the fourth node of the tetrahedron are said to be candidate nodes of that GT as depicted in Figure 3b. The outward normal vector $\overrightarrow{N}$ of a GT can be found by the cross product of two line segment vectors $\overrightarrow{R1}$ and $\overrightarrow{R2}$. The $1-2-3$ node numbering of a GT is so adjusted that the result of $\overrightarrow{R1} \times \overrightarrow{R2}$ always gives outward normal with respect to the generating tetrahedron. Defining



(a) The generating triangles of a tetrahedron.



(b) The nomenclature and the sketch of the generation algorithm.

GT1, GT2, GT3: Generating Triangles		R1, R2: Pos. Vectors
GT1: 013	Current GT: GT2	N: Normal Vector
GT2: 123	D: Min. Distance	R: Radius Vector
GT3: 021	⊕ Candidate Nodes for GT2	
	× Center of 1234 Tetra's Sphere	

Figure 3.

a reference position tensor $d_{ij} = x_{ij} - x_{1j}$ whose elements are vectors referenced from the GT's first node such as $d_{02} = y_0 - y_1$, the candidate nodes and the circumspheres that pass through each node and the GT can be computed. The candidate nodes are found by checking the sign of the dot product of $\overrightarrow{d_{4j}} \cdot \overrightarrow{N}$. If the sign of the result is positive, then the node can be assigned as a candidate node for the current tetrahedrization.

Secondly, the center coordinates (a, b, c) and the radius R of the circumsphere that passes through each candidate node and GT pair are calculated. In order to find out the circumsphere data for each candidate node, the equation of the sphere is written for each vertex (x_i, y_i, z_i) of each tetrahedron, and the system of equations are tried to be linearized by means of the devised procedure indicated below:

$$(x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2 = R^2, \tag{1}$$

$$x_i^2 + y_i^2 + z_i^2 - 2(x_i a + y_i b + z_i c) + a^2 + b^2 + c^2 = R^2, \qquad \text{for } i = 1, 2, 3, 4. \tag{2}$$

Let

$$e_i^2 = x_i^2 + y_i^2 + z_i^2 \tag{3}$$

$$A_i^2 = R^2 - e_i^2; \tag{4}$$

then

$$a^2 + b^2 + c^2 = A_i^2 + 2(x_i a + y_i b + z_i c), \tag{5}$$

$$a^2 + b^2 + c^2 = A_1^2 + 2(x_1 a + y_1 b + z_1 c), \qquad \text{for } i = 1. \tag{6}$$

Subtracting (6) from (5) and rearranging for $i = 2, 3, 4$:

$$(x_i - x_1)a + (y_i - y_1)b + (z_i - z_1)c = -\frac{1}{2}(A_i^2 - A_1^2) \tag{7}$$

$$\Delta_i = -\frac{1}{2}(A_i^2 - A_1^2) = \frac{1}{2}(e_i^2 - e_1^2), \tag{8}$$

we obtain the resulting linear system as follows:

$$d_{ij}a_j = \Delta_i, \tag{9}$$

where $d_{ij} = x_{ij} - x_{1j}$, e.g., $d_{22} = x_{22} - x_{12} = y_2 - y_1$.

The above system (9) is computed for $\vec{a} = [a, b, c]^{\mathrm{T}}$, and the radius vector is defined from the center to the second node of GT. Finally the perpendicular distance D from the center of the circumsphere to the GT for each candidate node is computed by simply taking the dot product of the radius vector $\vec{R}$ and unit normal vector $\vec{n}$ as formulated below:

$$\vec{R} = (x_2 - a)\,\vec{i} + (y_2 - b)\,\vec{j} + (z_2 - c)\,\vec{k}, \tag{10}$$

$$\overrightarrow{R1} = d_{21}\,\vec{i} + d_{22}\,\vec{j} + d_{23}\,\vec{k}, \tag{11}$$

$$\overrightarrow{R2} = d_{31}\,\vec{i} + d_{32}\,\vec{j} + d_{33}\,\vec{k}, \tag{12}$$

$$\vec{N} = \overrightarrow{R1} \times \overrightarrow{R2}, \tag{13}$$

$$\vec{n} = \frac{\vec{N}}{|\vec{N}|}, \tag{14}$$

$$D = \vec{R} \cdot \vec{n}. \tag{15}$$

Among the candidate nodes, the node giving the minimum D value and the GT are used to form the tetrahedron. Having generated the tetrahedron, the algorithm skips to the other GT which is not a boundary triangle and is not processed previously; the same procedure exactly applies until the whole domain is completely discretized with tetrahedrons. Similarly, the principle of generating the triangles is almost the same. In the 2D case, the node giving the minimum distance between the center of the circumcircle and the current GL is used as the main criterion to generate triangles. This concept has been called as minimum-distance-to-center criterion by Mclain [3].

3.4. Searching Process

The insertion of points and the searching process might be in sequential or binary manner. The latter approach provides computational saving. In binary tree (ADT) insertion, the points are ordered such that contiguous points in the list are neighbors in physical space. The contribution of these two insertion techniques to the tetrahedrization algorithm is sensed in the searching process. In order to satisfy a criterion, an appropriate point is searched for from the whole list of points in sequential insertion, whereas in binary tree insertion, the neighboring points are accessible and it is needless to search for the overall points in the list. The dynamic *search-list* array consists of the node numbers no matter how the searching process is carried out, sequential or binary ADT. Therefore, the general procedural layout of the triangulation algorithm can easily be switched between the sequential and binary searching processes.

4. ADAPTIVE REFINEMENT

4.1. General

The algorithm described above gives no indication as to how nodes should be generated. Hence, a procedure must be constructed to perform this task. The problem of how to place nodes in a domain in order to obtain a satisfactory sequence of meshes has been approached in a wide variety of ways. A number of methods use the set of boundary nodes to guide node placement [29–32]. The merit of the proposed algorithm enables the generation of boundary conforming meshes and, hence, eliminates the need for the geometric intersection check. This feature further decreases the computational burden as the other algorithms in the literature suffer. The algorithm is therefore coupled with the refinement strategy devised below.

4.2. Refinement Strategy

The grid generated within the domain should reflect the initial boundary point spacing. The point generation for arbitrary domains is achieved using the following automatic node placement scheme.

1. *Boundary Mesh Generation.* Given an ordered set of boundary nodes, construct an initial mesh generation. *Point-List* now contains only the boundary nodes.
2. *Interior Node Generation.* Interior node generation is done by simply inserting a point at the geometric center of the triangle if that triangle satisfies certain criteria. The criterion is different for boundary triangles having one side being a boundary line segment and the interior triangles or interior tetrahedrons.
   (a) *The criterion for boundary triangles.* In order not to check any geometric intersection to avoid boundary penetration, and hence to save computation, it is not permitted to accept a node insertion nearer to the boundary than the length of the boundary line segment. In 2D, the minimum length side of the boundary triangle ABC is computed, say, line AB. Second, to accept the midpoint of the triangle, say point P, the mean side length, d_m, of the subtriangle ABP is computed. Finally, if d_m is greater than $\|AB\|$, then the node P is accepted and inserted to *Point, Point-Triangle* and *Search Lists.* Similarly, in the 3D case, if the patch numbers of only two nodes of a boundary triangle matches with each other, the insertion of additional nodes at the geometric centers of four faces of the tetrahedron is checked to satisfy the same criterion stated for planar meshes.
   (b) *The criterion for interior triangles.* In boundary triangulation, it is seen that some triangles have greater areas than the others. To accelerate the convergence of mesh refinement, it is found convenient to compute the mean, μ, and the standard deviation, σ, of the average incircle radia of the triangles. In the 2D case, the triangles which have average incircle radia greater than the $\mu +$ (iteration number) $* \, \sigma$ are accepted for their midnode insertions. The uncertainty region for the shapes of the triangles shrinks, and the triangles tend to gather around the mean value of their incircle radia. Since the band width of the node insertion filter is increased by the iterations, the number of triangles that pass the filter decreases during the refinement process. If the percent difference of the number of nodes between the consecutive iterations is less than ten percent, the adaptation is automatically ceased. Consequently, in 3D geometries, the concept is exactly the same as in the planar case, except that the statistics are now calculated for the insphere radia of the tetrahedrons.

4.3. Summary of Rules

The rules of the mesh generation algorithm can be summarized below.

- There should be five dynamic lists holding the necessary information as discussed in data structure section.
- The insertion of points into the domain may be in sequential or binary manner.
- Each formed tetrahedron has three GT's.
- The surface triangles are illegal GT's.
- Each GT is processed only once.
- The check for geometric intersection is relaxed by not permitting a node insertion nearer to the boundary than the length of the boundary line segment.
- The node giving the minimum distance from the center of the circumsphere to the GT is used as the criterion to generate the tetrahedron.
- Iteration continues until no node numbers in the *search-list* are left.

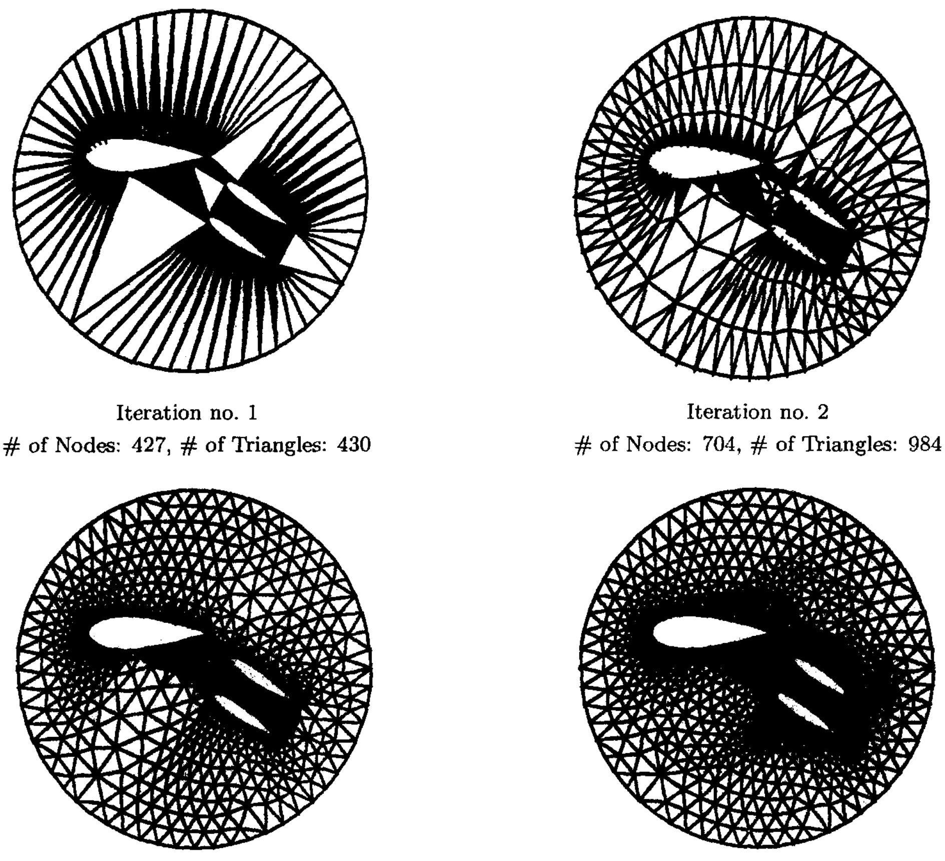
The computer implementation of the mesh generation method can be followed through algorithmic statements listed below.

```
Begin(Tetrahedrization)
 Read geometrical data from a disk file or directly from the graphics.
 Load Point_List with this data.
 Construct an imaginary tetrahedron {npqr} having {pqr} surface on a patch.
   Set {pqr} as the generating triangle (GT).
   Search for the nodes in the Search-List.
   Find node {m} giving minimum-distance-to-center.
   Load tetrahedron {pqrm} to Tetrahedron_List as the first item.
   Load tetrahedron number to Each four nodes' Point-Tetrahedron_List.
   Set Active tetrahedron number to -1.
   Repeat
    Increase Active tetrahedron number by one.
     For i:=1 to 3 do
     Begin
       Extract active tetrahedron from Tetrahedron_List.
       Case i of 1: {npqr} corresponds to Active Tetrahedron's node
                     indices of {0-1-2-3} (GT1) so that outward normal of
                     GT points away from the Active Tetrahedron.
                 2: npqr->{2-1-3-0} (GT2)
                 3: npqr->{1-0-3-2} (GT3)
       End of Case.
       The face {pqr} is the current GT.
       Set boolean {Skip} to False.
       Check to see {pqr} to be a surface triangle.
       If yes Then {Skip}=True
       Else
       Begin
        Extract tetrahedron numbers of {p,q,r} from Point_Tetrahedron_List.
        Check to see one of the {p,q,r}'s tetrahedron numbers other than
           the active tetrahedron to coincide.
        If yes Then {Skip}=True and Break if loop.
       End
         If Not {Skip} Then
          Begin
           ( Apply the main generation procedure discussed in Section 3.3)
           Search for the points opposite to point {n} from the Search_List.
           Find Point {m} satisfying minimum-distance-to-center criterion.
           Advance Tetrahedron_List size by one.
           Load new tetrahedron {pqrm} into Tetrahedron_List.
         End(Skip)
     End(i)
  Until Active Tetrahedron number >= Tetrahedron_List size.
End(Tetrahedrization).
```

The above algorithm requires the pregeneration of only one surface triangle which is achieved by the 2D version of the algorithm. The construction of the imaginary tetrahedron is provided by the recursive call to the same versatile main generation procedure discussed in Section 3.3.

5. RESULTS

The sample sets of iterative sequence of unstructured meshes generated by the convenient and the simple method proposed for the triangulation and tetrahedrization of complex geometries are given in Figures 4–6. The final grid quality is directly influenced by the distribution of the initial grid control nodes. The nodes of the final grid are clustered around the denser grid distribution of the initial control nodes. To illustrate this effect, we have added more control nodes at the leading and the trailing edges of the airfoil of the Figure 5 as shown in Figure 7. As may be observed, a better distribution of grid nodes is achieved when additional control nodes are introduced as depicted in Figure 8. These illustrate that the control of geometric adaptive schemes can easily be placed in front of the solution adaptive methods with the effective use of control nodes. To help in the success of the first trial solution, the problem specific information, i.e., the potential places of the nodes along the boundaries and inside the domain, are arranged and plugged into the above algorithm via a simple format disk file. In order to define an interior line source, e.g., it is enough to supply the physical coordinates of the nodes along the line with a special mark meaning that smoothing would not be applied after consecutive mesh generations. The smoothing process in which the nodes are relocated by averaging the coordinates of the midpoints of the neighboring tetrahedrons with the same surface patch provides 'good-shaped' triangles.



Iteration no. 1
# of Nodes: 427, # of Triangles: 430

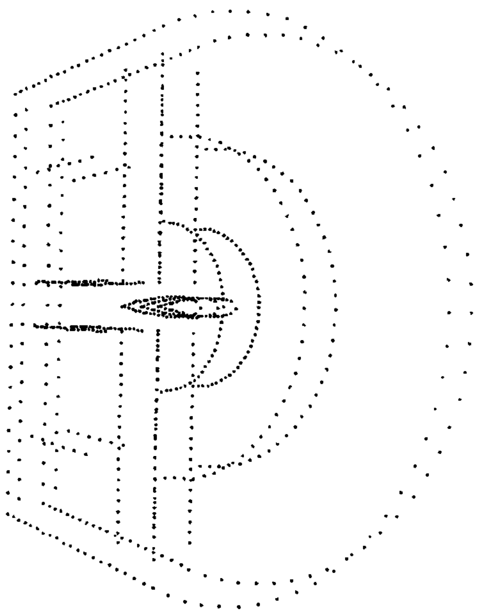
Iteration no. 2
# of Nodes: 704, # of Triangles: 984

Iteration no. 3
# of Nodes: 1428, # of Triangles: 2432

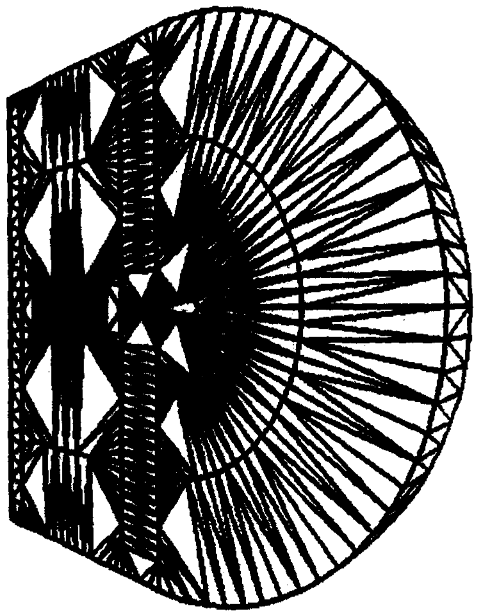
Iteration no. 4
# of Nodes: 2368, # of Triangles: 4312

Figure 4. The set of sample unstructured planar adaptive refinements applied to cascaded airfoils geometry.

The method incorporating the rules outlined in Section 4.3 results in a computational effort showing a linear relationship between the CPU time and the total number of nodes. The performance result shown in Figure 9 is obtained on an IBM/RS6000 Model 320H running under AIX 3.2. The slope of the linear line depends on the hardware. The complexity of the geometry further imposes the computational effort to increase since the algorithm checks for the GT whether it is a member of the the same surface patch or not. Therefore, the number of surface patches is an important parameter, but the effect of it on the computation is just one extra loop

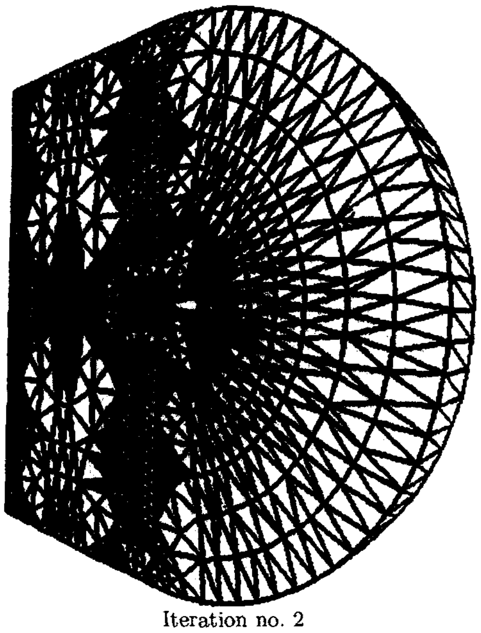


Number of Grid Control Points: 794



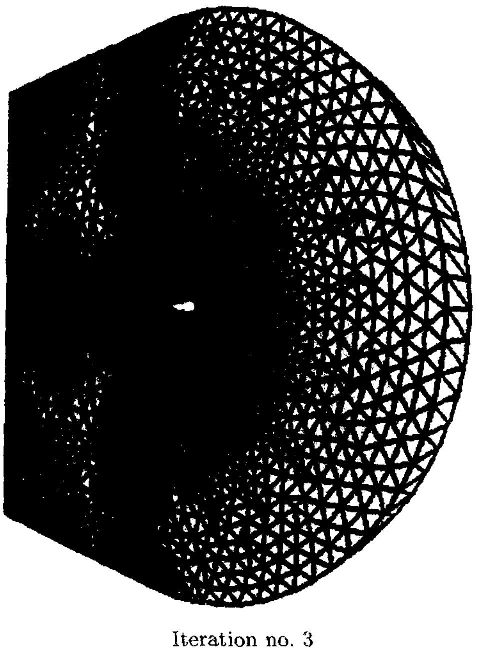
Iteration no. 1

Number of Nodes: 794, Number of Tetrahedrons: 2181



Iteration no. 2

Number of Nodes: 2898, Number of Tetrahedrons: 8462



Iteration no. 3

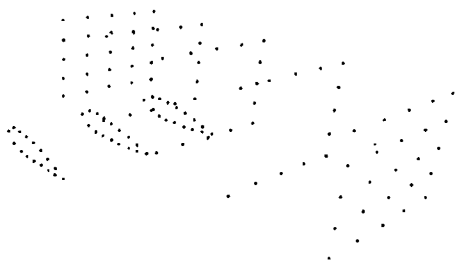
Number of Nodes: 10086, Number of Tetrahedrons: 28598

Figure 5. The geometrical refinement set of three-dimensional tetrahedral mesh generations for an NACA0012 airfoil geometry.

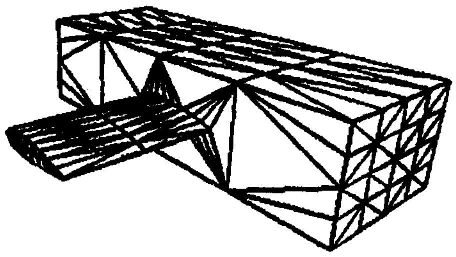
for each surface patch. Hence, the linearity again is preserved irrelevant of the surface patch numbers.

6. DISCUSSION

The final result of the method coupled with the node generator scheme is to provide a mesh which is no longer a coarse one for the success of the first trial solution of the PDE(s) and not

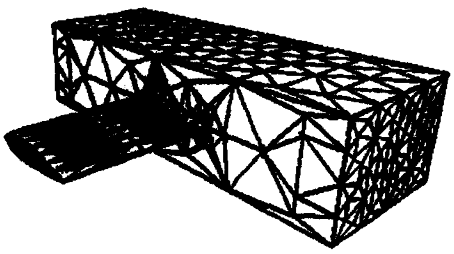


Number of Grid Control Points: 153



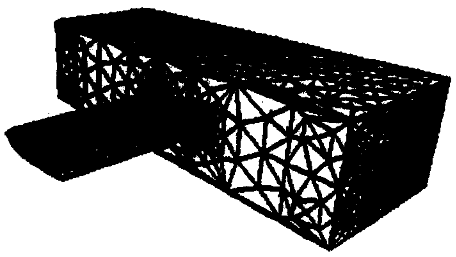
Iteration no. 1

Number of Nodes: 153, Number of Tetrahedrons: 352



Iteration no. 2

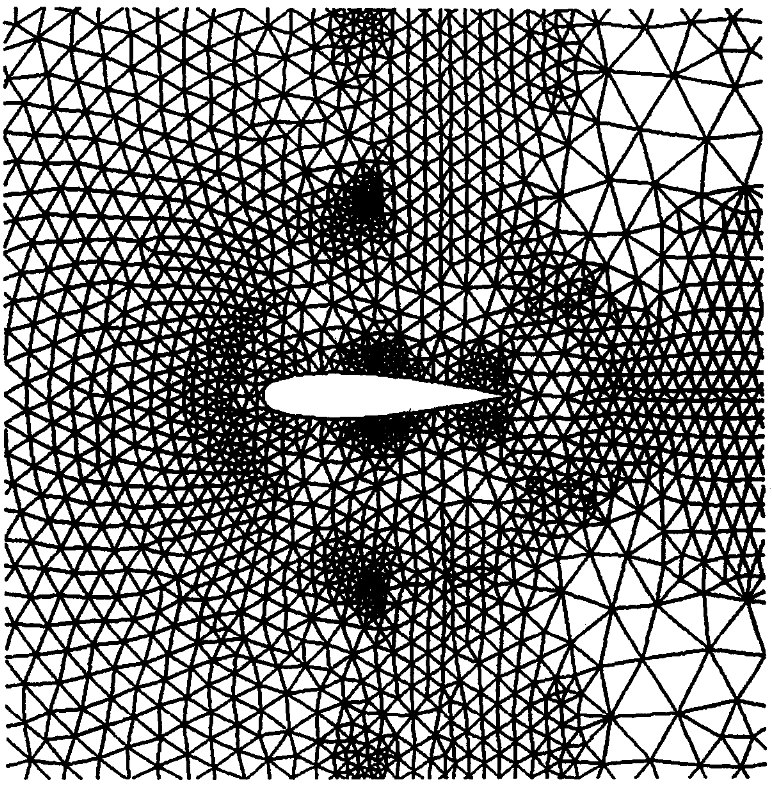
Number of Nodes: 1784, Number of Tetrahedrons: 5316



Iteration no. 3

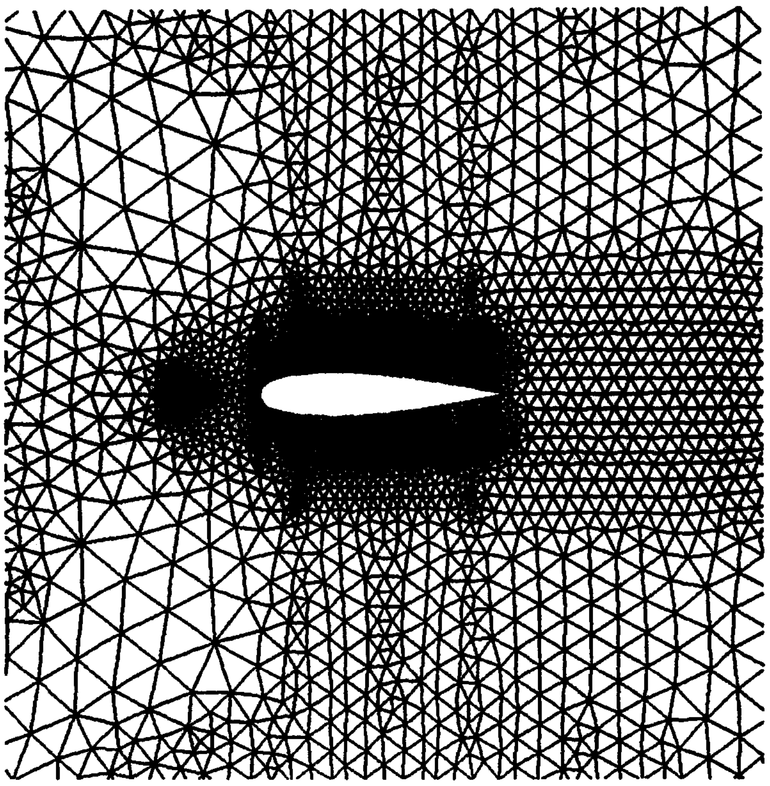
Number of Nodes: 16980, Number of Tetrahedrons: 51036

Figure 6. The adaptive sequences of three-dimensional tetrahedral mesh generation for a solid wing geometry.



(a) Before.

Figure 7. The effect of initial control points on grid quality.



(b) After.

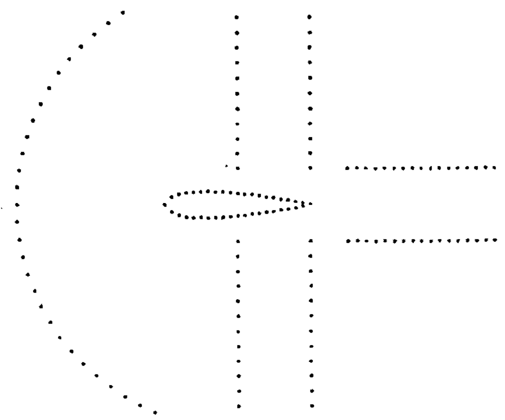
Figure 7. (cont.)

yet a fine one to be over-deterministic, even for the final solution dependent adaptation, though the grid resolutions provided at the final refinement should be at a certain level of lamination that an attempt to solve the physical problem could converge with an acceptable discretization error margin which will then be swept by the solution adapted algorithms.

The computational aspect of the algorithm actually favors the ADT binary searching process. On the other hand, the computational success of the binary searching process is closely related to the definition of the neighboring boundary of a GL or GT. The broader selection of the neighboring boundary may result in detecting too many nodes that not all of them are valuable for being a candidate node to be checked by minimum-distance-to-center criterion and may retard computations. On the other hand, the narrower selection of the neighboring box may cause nonoptimal triangles which do not possess empty circumdisk feature. The strategy followed in this study, which is a conservative one, developed in view of the computational trials on various complex geometries, is thus in favor of the former selection. It results in optimal triangles with empty circumdisks. However, the strategy might need further work to be implemented to accelerate the searching process in determining the optimum neighborhoods.

As soon as the node insertion nearer to the boundary than the length of the boundary line segment is not permitted, any node generator methodology can be incorporated in place of the one proposed in this study. The linear computational efficiency proves the method to be a fast one and simple to interact with by the user. The rules of the proposed method are simple to understand and apply. At the end of the code, the information regarding the connectivity of the triangles is written on disk for further manipulation such as finite element analysis or contouring purposes.

The possible future attack would be to adjust the code to execute under parallel and solution adaptive computing environments and methodologies, respectively.



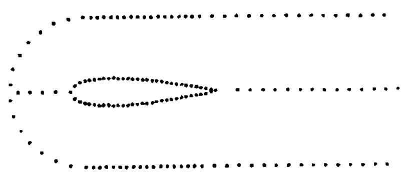


Figure 8. The initial control point selections of Figures 7a and 7b, respectively.

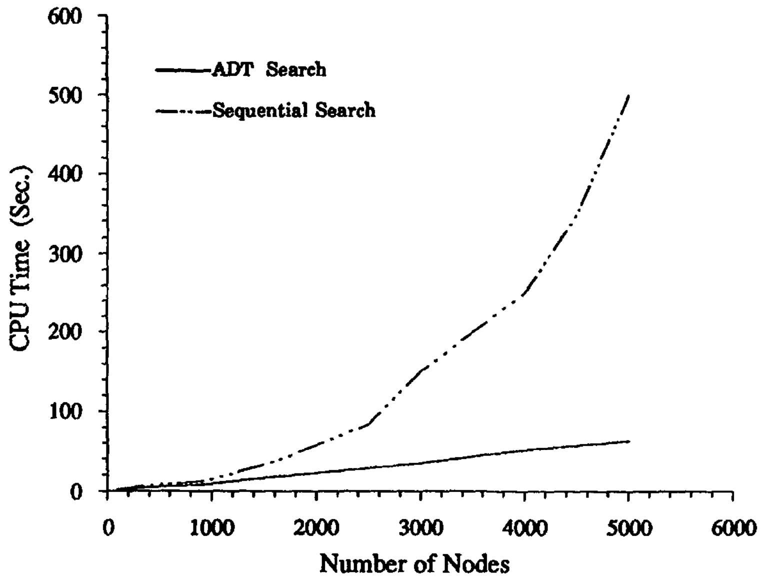


Figure 9. The computational efforts between the CPU time and total number of nodes for binary and sequential searching processes. (Test is performed on an IBM/RS6000 Model 320H running under AIX 3.2.)

REFERENCES

1. N. Pitteway, Computer graphics research in an academic environment, *Datafair 73 Conference Proceedings* **2**, 471–478 (1973).
2. P. Green and R. Sibson, Computing Dirichlet tesselations in the plane, *The Computer Journal* **21** (2), 168–173 (1978).
3. D.H. Mclain, Two dimensional interpolation from random data, *The Computer Journal* **19** (2), 178–181 (1976).

4. R. Sibson, Locally equiangular triangulations, *The Computer Journal* **21** (3), 243–245 (1978).
5. D.F. Watson, Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes, *The Computer Journal* **24** (2), 167–172 (1981).
6. A. Bowyer, Computing Dirichlet tesselations, *The Computer Journal* **24** (2), 162–166 (1981).
7. J. Peraire, Unstructured mesh generation methods for CFD, *Von Karman Ins. Lecture Series 1990-06*, (1990).
8. N. Kikuchi, Adaptive grid design methods for FEA, *Comput. Meths. Appl. Mech. Engrg.* **55**, 129–160 (1986).
9. N.P. Weatherill, Grid generation, *Von Karman Ins. Lecture Series 1990-06*, (1990).
10. J. Bonet and J. Peraire, An alternating digital tree (ADT) algorithm for geometric searching and intersection problem, *Int. J. Numer. Meth. Eng.* **31**, 1–17 (1991).
11. S.H. Lo, Delaunay triangulation of non-convex planar domains, *Int. J. Numer. Meth. Eng.* **28**, 2695–2707 (1989).
12. J. Fukuda and J. Suhara, Automatic mesh generation for FEA, *Proc. Int. Conf. on FEM*, Beijing, China, pp. 931–937, (1982).
13. J.C. Cavendish, Automatic triangulation of arbitrary planar domains for the FEM, *Int. J. Numer. Meth. Eng.* **8**, 679–696 (1974).
14. S.H. Lo, A new mesh generation scheme for arbitrary planar domains, *Int. J. Numer. Meth. Eng.* **21**, 1403–1426 (1985).
15. M. Suzuki, Surface grid generation, *Int. J. Numer. Meth. in Fluids* **17**, 163–176 (1993).
16. R. Löhner and P. Parikh, Generation of three dimensional unstructured grids by the advancing-front method, *Int. J. Numer. Meth. Eng.* **8**, 1135–1149 (1988).
17. O.C. Zienkiewicz and D.V. Phillips, An automatic mesh generation scheme for plane and curved surfaces by isoparametric co-ordinates, *Int. J. Numer. Meth. Eng.* **3**, 519–528 (1971).
18. C.L. Lawson, *Software for C^1 Interpolation: Mathematical Software III*, Academic Press, New York, (1977).
19. S.W. Sloan, A fast algorithm for constructing Delaunay triangulations in plane, *Adv. Eng. Software* **9**, 34–55 (1987).
20. D.T. Lee and B.J. Schachter, Two algorithms for constructing a Delaunay triangulation, *Int. J. Comp. and Inf. Sciences* **9**, 219–242 (1980).
21. S.W. Sloan and G.T. Houlsby, An implementation of Watson's algorithm for computing 2-dimensional Delaunay triangulations, *Adv. Eng. Software* **6**, 192–197 (1984).
22. T.P. Fang and L.A. Piegl, Algorithm for Delaunay triangulation and convex-hull computation using a sparse matrix, *Comp. Aided Design* **24**, 425–436 (1992).
23. N.P. Weatherill, A method for generating irregular computational grids in multiply connected planar domains, *Int. J. Numer. Meth. in Fluids* **8**, 181–197 (1988).
24. N.P. Weatherill, O.Hassan and D.L. Marcum, Compressible flowfield solutions with unstructured grids generated by Delaunay triangulation, *AIAA Journal* **33**, 1196–1204 (1995).
25. P.L. George and F. Hermeline, Delaunay's mesh of a convex polyhedron in dimension d. Application to arbitrary polyhedra, *Int. J. Numer. Meth. Eng.* **33**, 975–995 (1992).
26. P.L. George, Automatic mesh generator using the Delaunay Voronoi principle, *Surv. Math. Ind.* **4**, 239–247 (1995).
27. P.L. George, F. Hecht and E. Saltel, Fully automatic mesh generator for 3D domains of any shape, *Impact of Computing Science and Eng.* **2**, 187–218 (1990).
28. J. Fang and S.R. Kennon, Unstructured grid generation for non-convex domains, Paper 89-1983, pp. 512–524, AIAA, Reno, NV, (1989).
29. W.H. Frey, Selective refinement: A new strategy for automatic node placement in graded triangular meshes, *Int. J. Numer. Meth. Eng.* **24**, 2183–2200 (1987).
30. S.Ø. Wille, A structured tri-tree search method for generation of optimal unstructured finite element grids in two and three dimensions, *Int. J. Numer. Meth. in Fluids* **14**, 861–881 (1992).
31. G. Subramanian, V.V.S. Raveendra and M.G. Kamath, Robust boundary triangulation and Delaunay triangulation of arbitrary planar domains, *Int. J. Numer. Meth. Eng.* **37**, 1779–1789 (1994).
32. N.P. Weatherill and O. Hassan, Efficient three dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints, *Int. J. Numer. Meth. Eng.* **37**, 2005–2039 (1994).