AN ADVANCING FRONT DELAUNAY TRIANGULATION ALGORITHM DESIGNED FOR ROBUSTNESS

D. J. Mavriplis

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

ABSTRACT

A new algorithm is described for generating an unstructured mesh about an arbitrary two-dimensional configuration. Mesh points are generated automatically by the algorithm in a manner which ensures a smooth variation of elements, and the resulting triangulation constitutes the Delaunay triangulation of these points. The algorithm combines the mathematical elegance and efficiency of Delaunay triangulation algorithms with the desirable point placement features, boundary integrity, and robustness traditionally associated with advancing-front-type mesh generation strategies. The method offers increased robustness over previous algorithms in that it cannot fail regardless of the initial boundary point distribution and the prescribed cell size distribution throughout the flow-field.

This research was supported under the National Aeronautics and Space Administration under NASA Contract No. NAS1-18605 and No. NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665.

1. INTRODUCTION

One of the main promises of unstructured mesh computational fluid dynamics is the ability to discretize flow-fields about arbitrarily complex geometries in two and three dimensions. With this purpose in mind, a number of algorithms for constructing two-dimensional triangular and three-dimensional tetrahedral unstructured meshes have been developed over the years [1-9]. Of the various methods developed, two types of approaches which have received much attention in the computational fluid dynamics community have been advancing-front-based techniques[1,2] and Delaunay-triangulation-based techniques [3,4,5,6]. These two strategies have most often been perceived as competing approaches to the same problem. However, a Delaunay triangulation merely refers to a particular connectivity associated with a given set of points which possesses certain desirable properties, while an advancing front technique constitutes a point placement strategy while imposing a particular ordering of the element generation process. Thus, the two approaches are in some sense complementary, and several recent attempts to combine the advantages of both methods have appeared in the literature [7,8,9]. This is the approach taken in this work.

One may question the need for yet another unstructured mesh generation strategy, especially in two dimensions. As will be shown, all of the present methodologies offer much room for improvement in terms of either efficiency or robustness, and the present algorithm was designed with such issues in mind. Furthermore, the two-dimensional setting is employed for demonstrating techniques which should eventually be extendable to three dimensions.

1.1. The Advancing Front Approach

In order to understand the development of the present algorithm, it is useful to first examine the advantages and disadvantages of the various existing algorithms. Advancing front techniques begin with a discretization of the geometry boundaries as a set of edges in two dimensions. These edges form the initial front which is to be advanced out into the field. A particular edge of this front is selected, and a new triangle is formed with this edge as its base, by joining the two ends of the current edge either to a newly created point, or to an existing point on the front. The current edge is then removed from the front, since it is now obscured by the new triangle. Similarly, the remaining two edges of the new triangle are either assigned to the front or removed from the front, depending on their visibility, as shown in Figure 1. The front thus constitutes a stack (or priority queue), and edges are continuously added to or removed from the stack. The process terminates when the stack is empty, i.e. when all fronts have merged upon each other and the domain is entirely covered. One of the critical features of such methods is the placement of new points. Upon generating a new triangle, a new point is first placed at a position which is determined to result in an optimal size and shape triangle. The parameters which define this optimum triangle as a function of field position are obtained by a prescribed field function (which may be interpolated from a background grid). The triangle generated with this new point may result in a cross-over with other front edges, and thus may be rejected. This is determined by computing possible intersections with "nearby" front edges. Alternately, an existing point on the front may coincidentally be located very close to the new point, and thus should be employed as the forming point for the new triangle, to avoid the appearance of a triangle with a very small edge at some later stage. Existing candidate points are thus also searched by locating all "nearby" front points.

One of the advantages of such an approach is thus the automatic point placement strategy, which generally results in high-quality elements throughout most of the flow-field due to

the optimum positioning of these new points. Additionally, all real operations performed (such as intersection checking) are of a local nature; i.e. intersection checks are performed with neighboring edges of similar length, thus reducing the chances for round-off error induced failure. Finally, boundary integrity is guaranteed, since the boundary discretization constitutes the initial condition.

The space requirements for such an algorithm are lower than may be expected. Since this is essentially a greedy triangulation [10]; i.e. formed elements are never subsequently modified, all points, edges and triangles which lie behind the front need no longer be considered in the generation process. Thus the only active portion of the data is the front. Since a front is has one lower dimension than the domain to be discretized, the required space for such an algorithm in two dimensions is $O(\sqrt{N})$, where N is the final number of grid points generated. Since N points are added sequentially, the complexity is at most $O(N\sqrt{N})$. However, by employing sophisticated searching techniques such as spatial quad-trees, this complexity is easily lowered to $O(N\log\sqrt{N})$ which is asymptotically equivalent to $O(N\log N)$. Optimal space usage has not in general been achieved, due to the difficulty in continuously dumping out generated elements. However, restart capabilities are easily implemented [11], which can greatly reduce the required working size for a large mesh generation job.

The disadvantages of advancing front techniques mainly relate to their efficiency. The intersection checking phase is a rather brute-force technique for ensuring the acceptability of a new triangle, which is relatively expensive. Additionally, for each generated triangle, the quad-tree data structure must be traversed from top to bottom ($O(\log N)$ steps) in order to locate "nearby" points and edges. Another contributing factor is the fact that advancing front techniques construct the mesh one triangle at a time. Since in two dimensions there are asymptotically twice as many triangles as points, a more efficient strategy would be to construct the mesh one point at a time. Thus, each time a new point is generated, efficiency could be gained by determining all the potential triangles which join this new point to the existing front with a single traversal of the quad-tree data-structure. In three dimensions, the savings are even greater, since there exists on average 5 to 6 times more tetrahedra than vertices.

Finally, even though advancing front techniques rely only on local operations, they may still suffer from robustness problems. Central to the issue of determining acceptable triangles and "best" points, is the determination of a local length scale which defines the region of "nearby" points and edges. This length scale is generally obtained from the field function (which may employ a background grid). Consider the case of two merging fronts. If the field function varies rapidly over the region between the merging fronts, the relative sizes of the edges on one front may be much larger than those on the other front. If a search is initiated from the front with the smaller length scale, the region of "nearby" edges may not contain the appropriate edges and points of the other front, and failure will occur, as shown in Figure 2. Thus, the advancing front-technique can only be guaranteed to produce a valid triangulation if certain non-heuristic constraints are derived and imposed on the variation of the field function.

1.2. Delaunay Triangulation Algorithms

Given a set of points in the plane, there exists many possible triangulations of these points. A Delaunay construction represents a unique triangulation of these points which exhibits a large class of well defined properties. Particular properties can be employed to construct algorithms for generating the Delaunay triangulation of a given set of points.

The empty circumcircle property forms the basis of the Bowyer/Watson algorithm [12,13]. This property states that no triangle in a Delaunay triangulation can contain a point other than its three forming vertices within its circumcircle. Thus, given an initial triangulation, a new point may be inserted into the triangulation by first locating and deleting all existing triangles whose circumcircles contain the newly inserted point. A new triangulation is then formed by joining the new point to all boundary vertices of the cavity created by the previous removal of intersected triangles, as shown in Figure 3.

These algorithms exhibit a worst case complexity of $O(N^2)$ (imagine the case where each newly inserted point intersects all existing triangle circumcircles) and have thus been avoided in the computational geometry literature. However, $O(N^2)$ behavior represents a pathological case and in general mesh generation applications employing this algorithm exhibit close to linear complexity [4]. In fact, it has been shown that $O(N\log N)$ complexity can always be achieved if the order in which the points are inserted is modified [14].

Point insertion algorithms can be employed as the basis for a mesh generation strategy where the mesh points have been predetermined. The mesh points are put in a list, and an initial triangulation is artificially constructed (with auxiliary points) which completely covers the entire domain to be gridded. The mesh points in the list are then inserted sequentially into the existing triangulation using the Bowyer/Watson algorithm. The final mesh is obtained when all points from the list have been inserted. The main problems associated with such an approach relate to the generation of an initial triangulation. While it is not difficult to construct an initial triangulation, the insertion of points can lead to robustness problems due to round-off error. This comes about due to the non-local nature of the real operations required in the insertion process. When an inner boundary point is introduced at the initial stages of the triangulation, a triangle joining this point to the outer boundary will most likely be formed. If the next point introduced is an adjacent boundary point, the distance between these two points may be much smaller than the distance to the outer boundary (i.e. the other dimension of the triangle being intersected), and round-off error alone may cause an improper reconnection.

For non-convex domains, the integrity of the boundaries is not guaranteed by such an approach. This is generally remedied by increasing the boundary point resolution, or by triangulating through the boundary and performing an edge swapping clean-up phase as a post-processing operation to recover the boundary edges [15].

The poor worst case complexity of the above algorithms has lead to the development of a variety of divide and conquer algorithms for the Delaunay triangulation of an existing point set [10,16]. In this approach, the points are recursively divided into two groups, each group is triangulated individually, and the groups are then merged together. Such an approach can be shown to exhibit a worst case complexity of $O(N\log N)$. The merging of two triangulations exhibits certain similarities with the merging of fronts in the advancing front process. However, the algorithms are based on known Delaunay triangulation properties, rather than the assumption of an appropriate length scale, and thus can be proved to yield a correct triangulation under any conditions.

An advancing front type algorithm for constructing a Delaunay triangulation of a given set of points has been demonstrated in the context of mesh generation by Merriam [7]. This approach, which has also been reported in other applications [17,18,19], relies on the empty circumcircle property. An edge on the front is chosen, and a new triangle is tentatively formed by joining the ends of this edge to an arbitrary point of the set of points to be triangulated which lie to the interior of the domain, with regards to the front. If this formed triangle

contains any points within its circumcircle, it cannot be a valid Delaunay triangle, and thus an alternate point is chosen; i.e. the point contained inside the newly formed circumcircle which is closest to its circumcenter. By iterating this procedure, as shown in Figure 4, the appropriate point which produces a triangle containing no points interior to its circumcircle is eventually found. The new triangle is thus accepted, and the front advanced.

The present work makes use of the ideas found in the divide and conquer algorithms and the advancing front Delaunay triangulation algorithm. However, all the algorithms discussed so far assume that the mesh points have been predetermined. What is desired in the mesh generation context is an automatic point placement strategy. There are various Delaunay triangulation algorithms which incorporate automatic point placement strategies. A very simple method [5,6] is based on the Bowyer/Watson algorithm. Starting with an initial coarse triangulation which covers the entire domain, a priority queue is constructed based on some parameter of the individual triangles (circumradius for example). A field value is assumed to exist which determines the local maximum permissible value for the circumradius of the triangles (or other parameter). The first triangle in the queue is examined, and a point is added at its circumcenter if the triangle circumradius is larger than the locally prescribed maximum. This new point is inserted into the triangulation using Bowyer's algorithm, and the newly formed triangles are inserted into the queue if their circumcircles are too large, otherwise they are labeled as acceptable, and do not appear in the queue. The final grid is obtained when the priority queue empties out.

A consequence of this approach is that the final triangulation depends on the order of the insertion of the points. For example, if the queue is ordered by the smallest circumcircle rather than the largest circumcircle, a different triangulation will result. Furthermore, the meshes produced by this strategy do not exhibit the high degree of smoothness and element quality usually produced by the advancing front technique. Modifications to the point placement strategy have been proposed separately by Rebay [8] and Mueller at al [9]. Both methods are quite similar. In [8] for example, the triangles are divided into accepted (small enough) triangles, and waiting (too large) triangles. However, a subset of the waiting triangles is defined as those which border on accepted triangles. These so-called active triangles are the only ones considered as candidates for point insertion. When new points are inserted, they are positioned along the median of the edge separating the active triangle from its neighboring accepted triangle, at a distance which results in the formation of an optimal triangle between the new point and the bounding edge. The optimal size of the triangle is determined from the field function. The initial triangulation is set up by joining all inner boundary points to the outer boundary points, and all triangles adjacent to the boundaries are defined as active. The order in which points are inserted thus resembles the advancing front algorithm. The process begins at the boundaries and marches outwards as new triangles are accepted, and their outer neighbors become candidates for refinement. The produced triangulations exhibit the smooth variations and high quality elements typically associated with advancing front techniques, without the difficulties of merging fronts.

Delaunay techniques involving point placement are much more efficient than advancing front techniques. The absence of a sophisticated spatial data-structure for locating neighboring points, and the lack of an intersection checking routine make these very simple and efficient algorithms. Furthermore, the mesh is generated point by point, rather than one triangle at a time. Each time a point is inserted, all triangles neighboring that point are formed simultaneously, which results in increased efficiency due to the larger number of elements than points in

an unstructured mesh. However, these algorithms still suffer from their inability to guarantee boundary integrity and the use of non-local operations which are prone to round-off error, as can be seen by the large aspect ratio (non-accepted) triangles in Figure 5, (which is taken from [9]). These are precisely the strengths of the advancing front technique. Thus, what is required is an advancing front strategy which automatically positions new points, forms triangles which conform to the Delaunay criterion, and exhibits the efficiency of Delaunay point insertion methods.

2. DESCRIPTION OF PROPOSED ALGORITHM

The proposed algorithm is essentially an advancing front algorithm which adds new points ahead of the front, and triangulates them according to the Delaunay criterion. By making use of certain properties of Delaunay triangulations, one can ensure that only local operations are required and that consistent triangulations are always obtained.

The local property of Delaunay triangulations forms the basis of this algorithm. A field function is defined which determines the maximum permissible circumradius of a triangle as a function of its position in the domain to be discretized. When a new point is added ahead of the front, it is desired to construct all Delaunay triangles which contain this new point but which do not violate the local circumradius bound. Triangles which violate the circumradius bound should not be constructed, even temporarily, for this may require non-local operations and the possibility of round-off induced error. One method of constructing these triangles is simply to join the new point to every possible pair of points in the grid and preserve each potential triangle which does not violate the Delaunay criterion and the circumradius bound. A more efficient technique is to determine a subset of the grid points which is sufficient for locating all acceptable triangles. Such a subset can be formed by considering all points which are less than 2ρ away from the new point, where ρ represents the maximum permissible local circumradius as determined by the field function. Since any resulting triangles will contain an edge joining the new point to one of these candidate points, any points further than a distance 2ρ away from the new point cannot produce a triangle with a circumradius smaller than ρ. Furthermore, it will be shown that we need not consider all such points, but only the points on the front which are within 2ρ of the new point.

When adding a new point, two possibilities exist: either the point is not contained in any existing triangle circumcircle, or there exists a number of triangles whose circumcircles contain this new point. In the former case, we know that all existing triangles will still be valid after the insertion of the new point. Thus any new triangles must be formed by joining the new point to points on the front only. In the latter case, we must determine the set of triangles whose circumcircles are intersected. This set may contain triangles which border on the front as well as triangles which are interior to the mesh. However, the set cannot contain interior triangles without containing at least one front triangle, otherwise the interior triangles would not be visible to the new point after all intersected triangles have been removed, which is required by the properties of a Delaunay triangulation [4]. Thus, in order to locate all intersected triangles, we first locate the intersected front triangles, and then determine the intersected interior triangles by searching the neighbors of these triangles, and the neighbors of any subsequently found intersected triangles. In the traditional point insertion Delaunay algorithms, such situations do not arise; since the triangulation always covers the entire domain, every inserted point must be contained in at least one triangle circumcircle. Furthermore, all intersected triangles can be located using the neighbor search approach, since the grid is fully

connected. In the advancing front version, the neighbor search may be interrupted by the ungridded gap region between fronts. However, the Delaunay visibility property guarantees that all intersected triangles can be located from a neighbor search provided all intersected front triangles are known and used to initiate the search, as shown in Figure 6.

Finally, there is a third situation which must be considered. There may exist a point on the front which, when joined to the two ends of the front edge being considered, forms an acceptable triangle. At first it may appear as if such a situation should not arise. This existing front point should have been linked to the current edge at the time of its insertion. However, due to the variation of the local field function, it is possible that such a triangle would have been rejected at that time, since the field function was not sampled at precisely the same spatial location as when approaching from the other front. In any case, this situation is easily handled. Since it involves the generation of a new triangle without the insertion of a new point, we merely resort to the algorithm reported in [7,17,18,19] for advancing a Delaunay triangulation front on a set of predetermined points.

Thus the algorithm can be summarized as follows.

1. Construct the original front as the set of boundary edges.
2. Chose a particular edge of the front, according to some criterion such as minimum edge length.
3. Determine the maximum permissible circumradius by evaluating the field function at the center of the edge.
4. Locate all front points which are less than 2ρ away from either end point of this edge.
5. Use the algorithm in [7] to determine the Delaunay triangle formed between this edge and the set of candidate points, if such a triangle exists.
6. If this triangle exists and is acceptable (circumradius smaller than ρ), form new triangle, update the front, and proceed to 13. Otherwise create a new point at the appropriate location.
7. Determine all front triangles whose circumcircles are broken by the new point.
8. Using a neighbor search initiated at the intersected front triangles, locate all interior triangles whose circumcircles are intersected by the new point.
9. Remove all such triangles and update the front. Any new front points which result from this operation are added to the list of "close" points.
10. If the circumradius of any of the intersected triangles is larger than the previously determined maximum permissible value ρ, replace the old value by this new maximum, and locate any additional front points which are less then 2ρ away from the new point.
11. Form all possible Delaunay triangles which contain the new point and two other points in the list of "close" points, and which do not violate the local circumradius bound. Such triangles are found using the algorithm in [7].
12. Add these triangles to the mesh and update the front.
13. If front queue is empty, stop, otherwise go to 2.

The searches in steps 4 and 7 must be implemented using quad-tree-type data structures in order to avoid an $O(N\sqrt{N})$ overhead. The actual manner in which new points are positioned in step 6 is taken from [8]. In this work, a triangulation which covers the entire domain always exists, and new points are inserted in the so-called active triangles which border on

previously generated accepted triangles. A new point is positioned along the median of the edge which delimits the active triangle from an accepted triangle at the precise distance which results in a triangle of the desired circumradius when the new point is joined to the end points of this edge. However, the prescribed circumradius may be incompatible with the local triangulation. For example, if the prescribed circumradius is smaller than half the current edge length, there is no point location which yields a triangle of the desired size. In this case, the new point is positioned at the intersection of the edge median and the edge inscribing circle, since this results in the smallest possible triangle circumcircle containing the current edge. On the other hand, if the prescribed circumcircle is much larger than the current edge length, the new point may inadvertently be positioned close to another existing front mesh point, which would result in undesirable triangles away from the current edge. In this case, the point location along the median of the current edge is limited by the circumcenter of the current active triangle, thus guaranteeing that the new point will be at least a distance ρ_{active} away from all other mesh points.

This strategy is mimicked in the current advancing front algorithm. The new point is positioned along the median of the current front edge at a distance which results in a triangle of the desired circumradius. The location of the new point along the median is limited at the lower end by the intersection of the median with the inscribed circle of the current front edge, and at the other extreme by the location of the circumcenter of the Delaunay triangle formed with this edge and existing mesh points, which is found in step 5, as shown in Figure 7. Thus, in step 5, we must ensure that we form any triangle for which the circumradius is up to twice the size of the prescribed circumradius. Any larger triangles will not be useful in limiting the position of the new point. If the circumradius of the formed triangle is smaller than twice the prescribed value, but still larger than the prescribed value itself, the triangle will be employed solely to limit the position of the new point, and then discarded afterwards. On the other hand, if the triangle circumradius is smaller than the prescribed value, the triangle is retained as part of the mesh, and no new point is required.

When new triangles are formed, one must ensure that the integrity of the boundary discretization is not violated. This is accomplished by removing from the list of "close" points all points which are not visible to the new point due to the presence of boundary edges. (In the case of step 5, we remove all points which are not visible to the two end points of the current edge.) In two-dimensions, the existence of constrained Delaunay triangulations [20] guarantees that this is a sufficient condition to obtain a suitable boundary conforming discretization. One method of removing non-visible points is to draw the ray from the new point to the point being tested, and check for intersections with all boundary edges. Since the number of boundary edges is $O(\sqrt{N})$, this can become prohibitively expensive. Hence, a sufficient subset of the boundary edges which are "close enough" to the new point is first determined and then employed to check for intersections. Since the points being tested are all within a distance 2ρ of the new point, we are merely required to test all boundary edges which are within this distance of the new point. These include but are not limited to all boundary edges with an end point which belongs to the current list of "close" points. In order to locate remaining boundary edges whose normal distance to the new point is less than 2ρ but whose end points are further away than 2ρ from the new point, we draw the inscribed circle of the boundary edge, as shown in Figure 8.

We distinguish two cases: the first case when the new point is inside the inscribed circle of the boundary edge, and the second case when the new point lies outside this circle. In the

first case, the boundary edge is added to the list of edges which require searching. In the second case, Figure 8 indicates that the distance from the new point to the end points of the edge can at most be $\sqrt{2}$ times the normal distance from the new point to the edge. Thus, the set of boundary edges required for checking intersections is formed by locating all boundary edges which contain a vertex less than $2\sqrt{2}\rho$ away from the new point, as well as all boundary edges whose inscribed circles are intersected by the new point. The determination of these points and intersected circles can be performed simultaneously with the search for nearby points in step 4 and the search for intersected triangle circumcircles in step 7 respectively.

Using this subset, the number of boundary edges which must be checked for intersections is greatly reduced. In fact, in most cases, typical for the interior regions of the mesh, no "close" boundary edges will be found, and no checking for intersections will be required.

3. RELATIONSHIP WITH PREVIOUS WORK

It is informative to examine the relationship of the present algorithm with those discussed earlier. This work is closely related to that of Rebay [8] and Mueller at el [9]. A similar mesh should be produced by the present method and that of [8], since both use similar point placement strategies, and both produce the Delaunay triangulation of these points. The main difference is that in the previous works, a triangulation which covers the entire domain always exists, whereas in the present work, only the area behind the fronts are covered by a triangulation. In the former case, the existing triangulation is conveniently employed as the basic data-structure (i.e. a linked list) to support the searches for locating intersected triangles and points to which the new point must be connected. In the present work, only the triangles which correspond to "accepted" triangles in [8] are present, and thus more complicated quad-tree type data-structures must be employed to locate neighboring points and intersected triangles on the fronts, while the triangulation can be employed to aid the search in regions behind the fronts. While this adds to the coding complexity and incurs additional overhead, the omission of non-accepted triangles ensures that all real operations are of a local nature, thus minimizing opportuinites for round-off error induced failure. Boundary integrity is also preserved automatically.

The present algorithm also closely resembles the advancing front algorithm of [1,2]. However, explicit intersection checking is not required due to the properties guaranteed by the Delaunay construction. Both approaches rely on the determination of a local characteristic distance which is employed for reducing the number of front edges and points which must be considered in the triangulation process. In the advancing front algorithm of [1,2], this length scale is obtained from the field function (evaluated by interpolating from a background grid). The implicit assumption in this method is that the field function varies slowly with respect to the local cell size, and thus may be considered locally constant when advancing a front or merging two fronts. In cases where this assumption does not hold, the merging of two fronts of widely differing cell sizes may occur, which usually results in a failure of the algorithm. In the present strategy, a local length scale is obtained from the prescribed field function as well. This distance is employed to locate all "nearby" front points. However, an additional search is required to locate front triangles whose circumcircles are intersected by the newly inserted point. If the field function were constant throughout the domain, this second search would not be required, since all vertices of any intersected triangle (which could have a circumcircle no larger than that prescribed by the field function) would be no further from the new point than the constant search distance defined by the field function. Thus, the search for intersected front triangles corresponds to the determination of an alternate characteristic length scale at

neighboring fronts, which is required in order to guarantee a valid triangulation in regions where the field function varies rapidly.

4. COMPLEXITY AND SPACE REQUIREMENTS

The space requirements and computational efficiency of the present algorithm lie in between those of traditional advancing front algorithms and the Bowyer/Watson algorithm for Delaunay triangulation. As opposed to the advancing front algorithms, the present approach does not represent a true greedy algorithm [10]; i.e. triangles behind the front may be subsequently modified. However, the only such triangles which may be modified are those whose circumcircle extends ahead of the front into the ungridded region into which new points are placed. Assuming a relatively smooth distribution of elements behind the front, the number of such non-frozen elements is a constant times the size of the front. Thus, we can expect a space requirement of $O(\sqrt{N})$, although the worst case estimate is more likely $O(N)$. On the other hand, it is a simple matter to create a restart facility which dumps out the generated portion of the grid after a prescribed number of elements have been produced, and reinitializes the generation process using the front of the previous mesh as the initial condition. If no old elements behind the front are considered in the restart process, the resulting mesh may contain regions of locally non-Delaunay triangles along the fronts present at each restart phase. If a true Delaunay triangulation is required, these regions may be converted using the edge-swapping algorithm [21] in a postprocessing phase.

The current algorithm exhibits a worst case complexity of $O(N^2)$, just as the Bowyer/Watson algorithm for Delaunay triangulation. This occurs when the circumcircles of all existing triangles are intersected by each new point, or when all front points must be included in the list of "nearby" points which are candidates for forming a new element. However, for the smooth element and point distributions which are sought in the context of mesh generation, the number of points within the characteristic distance of a newly inserted point and the number of intersected triangles should approach a constant. When the $\log N$ term from the quad-tree structures employed for the search routines on the front is included, a complexity of $O(N \log N)$ can be expected. This is the same complexity exhibited by other advancing front algorithms under the same assumptions. However, the present algorithm can be expected to run significantly faster than other advancing front algorithms since the mesh is generated one point at a time, rather than one triangle at a time. In two dimensions, the differences may be small, especially since two length scales and thus two searches on the front are required for robustness (an additional one for the intersected front triangles). However, in three dimensions where there are on the average 5 to 6 times more tetrahedra than vertices, the $O(logN)$ cost of traversing the octree data-structures may be amortized over all elements generated about each newly inserted mesh point.

On the other hand, the present algorithm will probably not achieve the efficiency exhibited by Delaunay triangulation point insertion methods, due to the need to traverse the quad-tree data structures which are not present in these other methods, and the need to consider a sufficient but not necessary list of candidate points for triangulation at each point insertion process. This cost, as well as the increased coding complexity, is viewed as the price required for additional robustness.

5. EXAMPLES

Figure 9 depicts the process of generating a mesh about a geometry consisting of two thin plates. The boundary discretization of these thin plates is relatively uniform, except for two very large edges on the upper surface of the lower plate. The combination of thin plates and irregular boundary discretization poses a significant challenge to traditional Delaunay triangulation methods, as well as to standard advancing front techniques. In the former case, the boundary integrity is difficult to maintain without adding new boundary points. In the latter case, the merging of two fronts of widely differing length scales is produced. The present algorithm handles this case automatically, as can be seen from the figure. A valid triangulation is observed, even in the region of rapid variation of the characteristic length scale, although the quality of the triangulation degrades in such regions, as would be expected.

For practical problems involving dense meshes, a smooth background field function must be constructed, and sophisticated spatial data-structures must be employed for efficiently performing steps 4 and 7, in section 3.

The background field function is constructed by the method described in [22] with a slight modification. A set of point sources which locally specify element size are placed in the flow field, and a Poisson equation involving these sources is solved on a background mesh. In the present work, the Poisson equation is solved on a mesh formed by constructing a quadtree about the boundary points which define the initial front, as shown in Figure 10. When the field function is sampled at a particular point in the plane, the quadtree element containing this point is located by descending the tree, and the spacing value is taken as a bilinear interpolation of the four values at the corners of the quad element, which have been determined by solving the associated Poisson equation.

The search for "close" points (i.e. step 4 in section 3), is implemented using a standard region quadtree [23]. The search for intersected front triangle circumcircles (i.e. step 7 in section 3) and boundary edge circumcircles is somewhat more involved. This is achieved by first representing each circumcircle by a point in three-dimensional space, with coordinates x, y and r, where x and y are the physical coordinates of the circumcircle center, and r represents the radius of the circumcircle. A region octree containing all front triangle and boundary edge circumcircles is then constructed and maintained dynamically, as the front evolves [23].

In order to determine all circumcircles intersected by a point (x_o, y_o), we draw the cone which has its origin at $(x_o, y_o, 0)$, and a slope angle of 45 degrees, as depicted in Figure 11. We then search all octants of the tree which are contained or intersected by this cone.

In Figure 12, the generation of an unstructured mesh about a multi-element airfoil configuration is depicted. The spacing distribution was determined using 4 source points at the outer boundary, and 6 source points close to the airfoil surfaces. As can be seen, the method yields a smooth variation of elements throughout the flow field, even without any additional mesh smoothing. Figure 13 shows the effect of smoothing the final mesh. Edge swapping is also performed to ensure the mesh remains a Delaunay triangulation, although very few edges require swapping after the smoothing operation.

The mesh contains 41781 triangles and 21232 vertices, which required a total of 100 seconds to generate on a Silicon Graphics 4D35 workstation. In general rates of 350 to 450 triangles per second have been observed on a wide variety of cases. While the quadtree search routines consume less than 5% of the total CPU time, the octree based circumcircle search has been found to consume roughly 35% to 40% of the total time. As expected, the efficiency of

this algorithm appears to fall in between that of the advancing front methods [1,2], and the Delaunay triangulation methods [5,8,9].

It should be noted, however, that algorithms for triangulating a given set of points, such as that described in [7], can be significantly more efficient than the present algorithm. This is partly due to the fact that half the problem has already been solved, i.e. the placement of the grid points. However, it is also largely due to the fact that the data is static, and hence more efficient static data-structures such as fully balanced trees may be used in the search routines. In the present algorithm, the search is executed on the front points which are continuously changing, and thus dynamic data structures which support insertion and deletion of points must be employed.

6. CONCLUSIONS AND FURTHER WORK

These results demonstrate the feasibility of generating unstructured meshes using an advancing front strategy with an automatic point placement facility, while conforming to the rules of Delaunay triangulation. The main advantages of such an approach over traditional advancing front methods are increased robustness through the use of a more theoretically sound approach, while avoiding the boundary integrity and accuracy induced failures of Delaunay point insertion methods.

The octree based search routine for locating intersected front and boundary circumcircles, while providing an order of magnitude increase in efficiency over a brute force type search, still consumes a significant portion of the overall computational time. This indicates that further increases in efficiency of the algorithm may be achieved by re-examining this search operation.

Finally, the implementation of these ideas into the three dimensional setting is also planned.

REFERENCES

1. Peraire, J., Vahdati, M., Morgan, K., and Zienkiewicz, O. C., "Adaptive Remeshing for Compressible Flow Computations", *J. Comp. Phys.*, Vol 72, October, 1987, pp. 449-466

2. Gumbert, C., Lohner, R., Parikh, P., and Pirzadeh, S., "A Package for Unstructured Grid Generation and Finite Element Flow Solvers", *AIAA paper 89-2175* June, 1989.

3. Weatherill, N. P., "The Generation of Unstructured Grids Using Dirichlet Tessalations" *Princeton University Department of Mechanical and Aerospace Engineering Report MAE 1715,* July 1985.

4. Baker, T. J., "Three Dimensional Mesh Generation by Triangulation of Arbitrary Point Sets", *Proc. of the AIAA 8th Comp. Fluid Dyn. Conf.,* AIAA paper 87-1124, June, 1987.

5. Holmes, D. G., and Snyder, D. D., "The Generation of Unstructured Meshes Using Delaunay Triangulation" Numerical Grid Generation in Computational Fluid Mechanics *Proc. of the Second International Conference on Numerical Grid Generation in Computational Fluid Dynamics, Miami, December 1988,* Eds. S. Sengupta, J. Hauser, P. R. Eisman, and J. F. Thompson, Pineridge Press Ltd., 1988.

6. Dey, T. K., Bajaj, C. L., Sugihara, K., "On Good Triangulations in Three Dimensions", *Proceedings of the ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications,* Austin, Texas, June, 1991.

7. Merriam, M. L., "An Efficient Advancing Front Algorithm for Delaunay Triangulation", *AIAA paper 91-0792,* January, 1991

8. Rebay, S., "Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer/Watson Algorithm", *Paper presented at the 3rd Int. Conf. on Numerical Grid Generation in Comp. Fluid Dyn.,* Barcelona, Spain, June 1991.

9. Muller, J. D., Roe, P. L., and Deconinck, H., "A Frontal Approach for Node Generation in Delaunay Triangulations", *Unstructured Grid Methods for Advection Dominated Flows,* VKI Lecture Notes pp. 9-1 9-7, AGARD Publication R-787, March 1992.

10. Preparata, F. P., and Shamos, M. I., *Computational Geometry, An Introduction,* Texts and Monographs in Computer Science, Springer-Verlag, 1985.

11. Pirzadeh, S., "Recent Progress in Unstructured Grid Generation", *AIAA paper 92-0445* January, 1992.

12. Bowyer, A., "Computing Dirichlet Tessalations", *The Computer Journal,* Vol. 24, No. 2, 1981, pp. 162-166

13. Watson, D. F., "Computing the n-dimensional Delaunay Tessalation with Application to Voronoi Polytopes", *The Computer Journal, Vol 24, No. 2,* pp. 167-172, 1981.

14. Guibas, L. J., Knuth, D. E., and Sharir, M., "Randomized Incremental Construction of Delaunay and Voronoi Diagrams", *Stanford University Computer Science Rep. No. STAN-CS-90-1300* January, 1990.

15. George, P. L., Hecht, F., and Saltel, E., "Fully Automatic Mesh Generator for 3D Domains of any Shape", *Impact of Computing in Science and Engineering, Vol 2, No. 3,* pp. 187-218, 1990.

16. Lee, D. T., and Schachter, B., "Two Algorithms for Constructing a Delaunay Triangulation", *International J. Comput. Inform. Sci., Vol 9,* pp. 219-242, 1980.

17. Nelson, J. M., "A Triangulation Algorithm for Arbitrary Planar Domains", *Applied Math Modeling, Vol 2,* September 1978, pp. 151-159.

18. Maus, A., "Delaunay Triangulation and the Convex Hull of n Points in Expected Linear Time", *BIT, Vol 24,* pp. 151-163, 1984

19. Tanemura, M., Ogawa, T., and Ogita, N., "A New Algorithm for Three-Dimensional Voronoi Tessellation", *Journal of Computational Physics,* Vol 51, No. 2, August 1983, pp. 191-207.

20. Chew, L. P., "Constrained Delaunay Triangulations", *Algorithmica, Vol 4,* pp. 97-108, 1989.

21. Lawson, C. L., "Transforming Triangulations", *Discrete Mathematics, Vol 3,* pp. 365-372, 1972.

22. Pirzadeh, S., "Structured Background Grids for Generation of Unstructured Grids by Advancing Front Method", *AIAA Paper 91-3233,* AIAA 9th Applied Aerodynamics Conference, Baltimore, MD, September 1991.

23. Samet, H., "*The Design and Analysis of Spatial Data Structures*", Addison-Wesley, 1990.

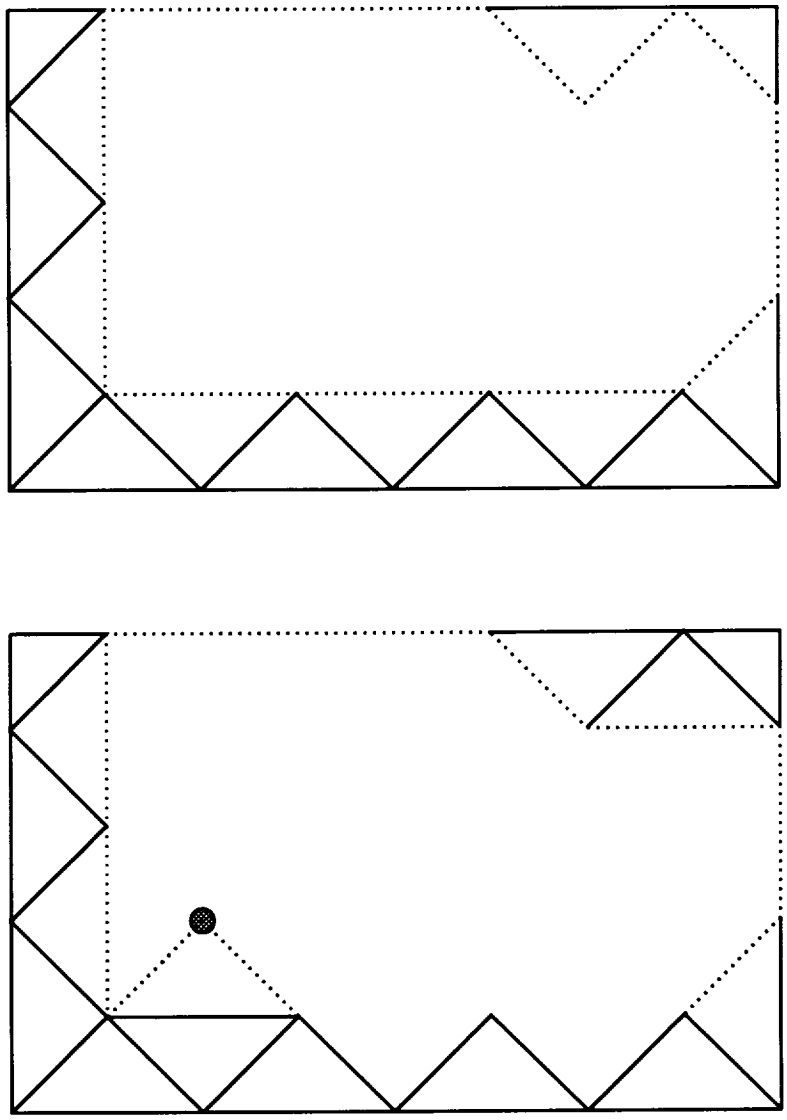

Figure 1
Illustration of Conventional Advancing Front Mesh Generation Concept in Two Dimensions.
Dotted Line Represents Current Front. New Triangles are Generated One at a Time,
by Joining the Two Ends of a Front Edge to Either a Newly Created Point, or an Existing Front Point.

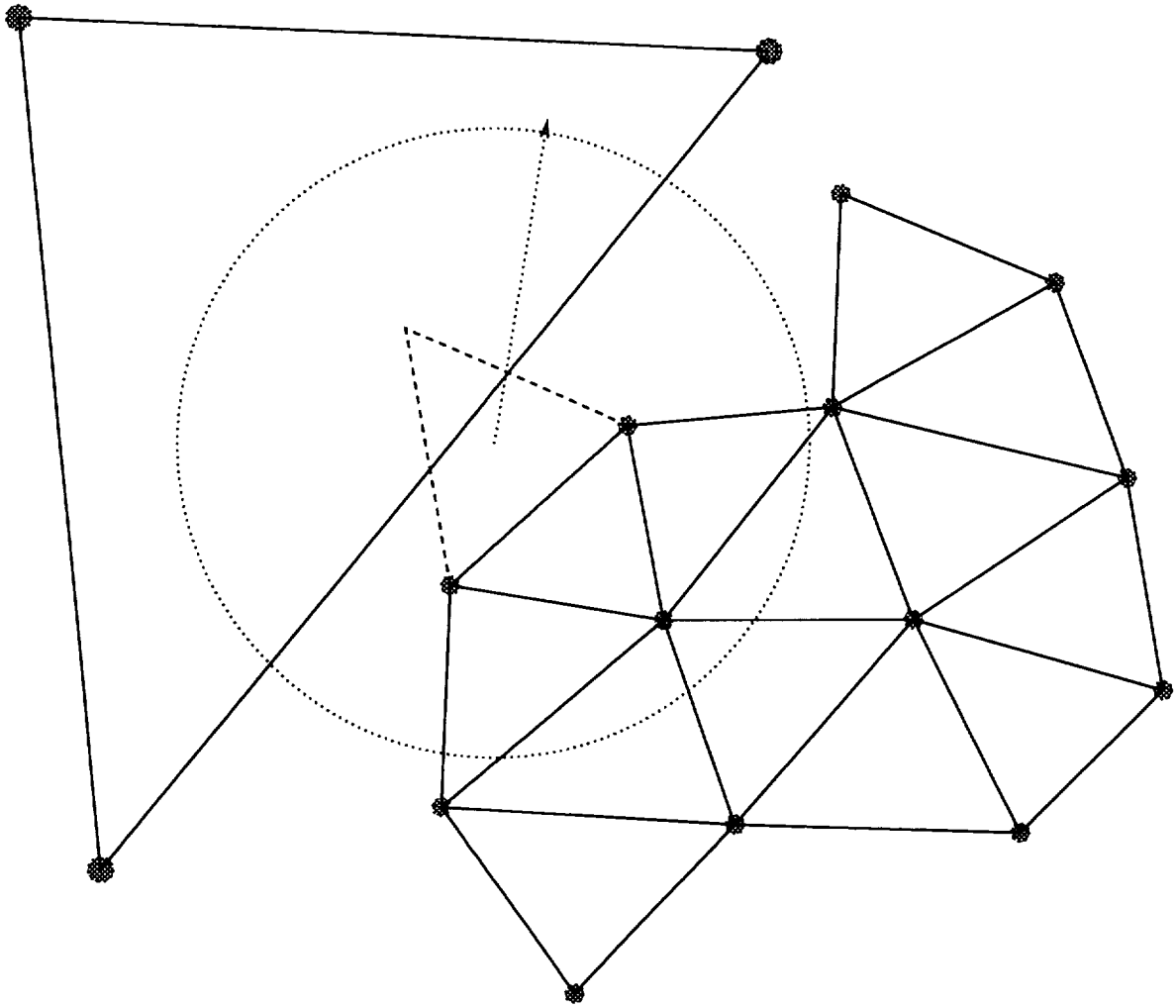

Figure 2
Illustration of a Failure Scenario for the Traditional Advancing Front Algorithm:
The Merging of Two Fronts of Widely Differing Length Scales.
The Advancing Front of Small Triangles May Fail to Locate the End Points of a Large Edge on the Adjacent Front Prior to Cross-Over, Since these Points May Be Outside of the Search Region Defined by the Local Length Scale.

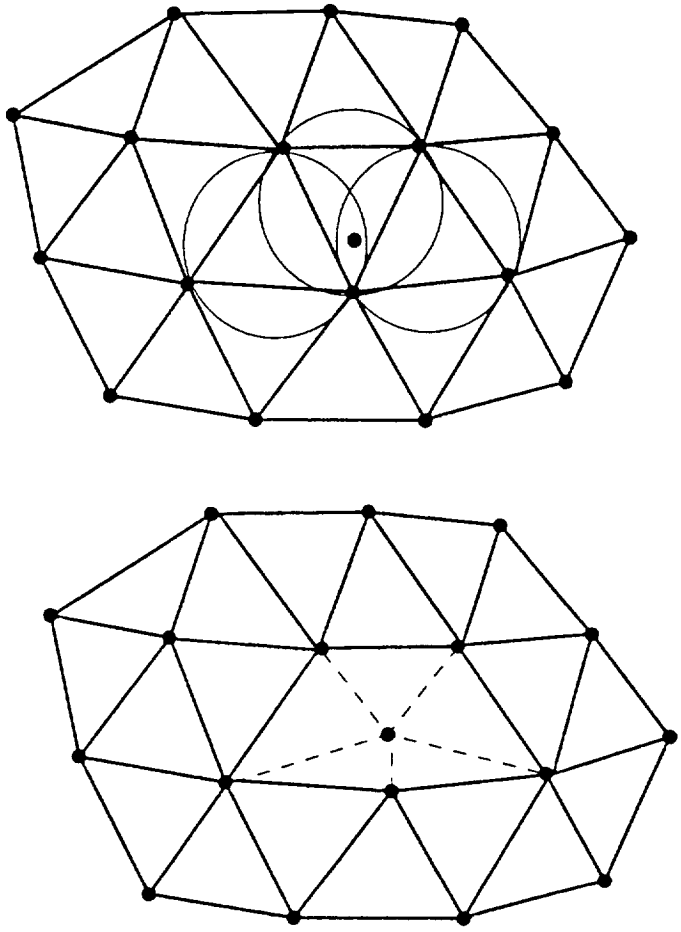

Figure 3
Illustration of Bowyer's Point Insertion Algorithm for Delaunay Triangulation

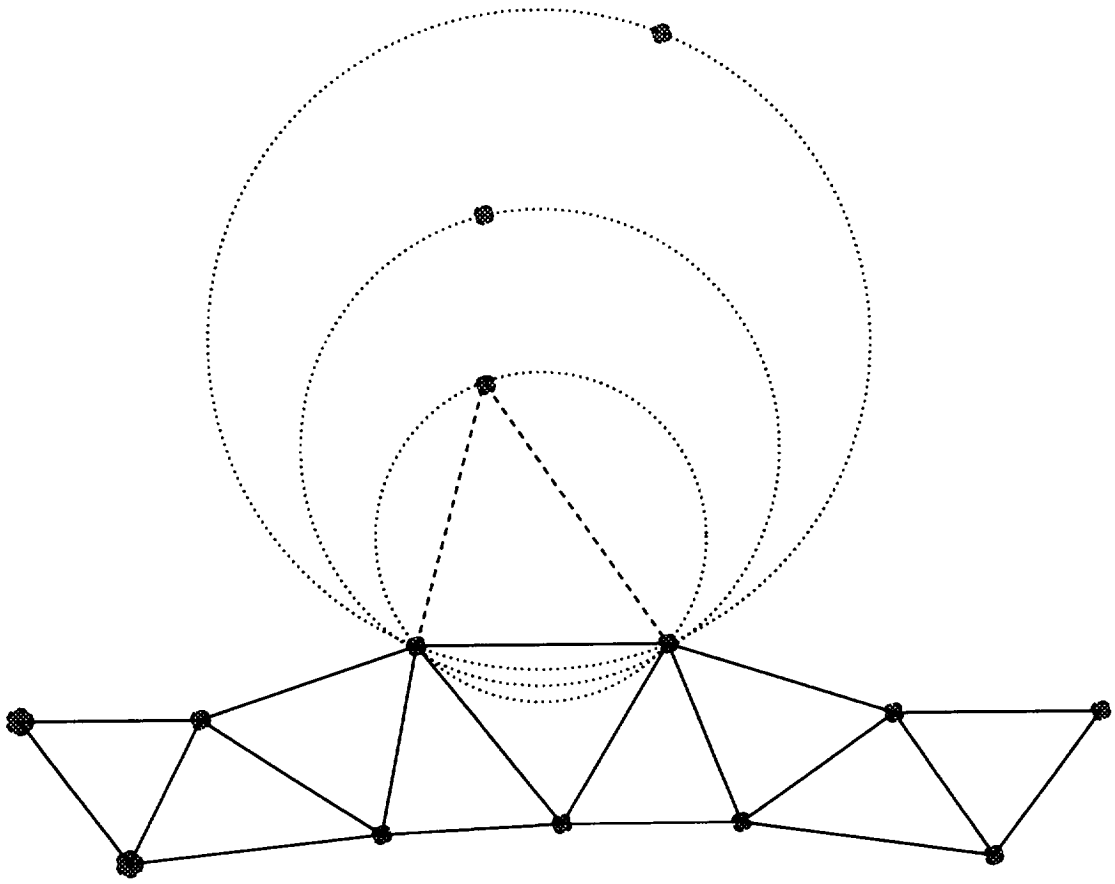

Figure 4
Illustration of the Iteration Sequence Employed by the Advancing Front Delaunay Triangulation Algorithm: A Triangle is Formed by Joining the Front Edge to a Vertex. If the Circumcircle of this Triangle Contains One or More Vertices, the Triangulation is Invalid, and a New Triangulation is Formed Using One of the Vertices Interior to the Previous Circle. The Process Iterates Until a Triangle with a Vertex-Free Circumcircle is Obtained, which Determines Convergence.

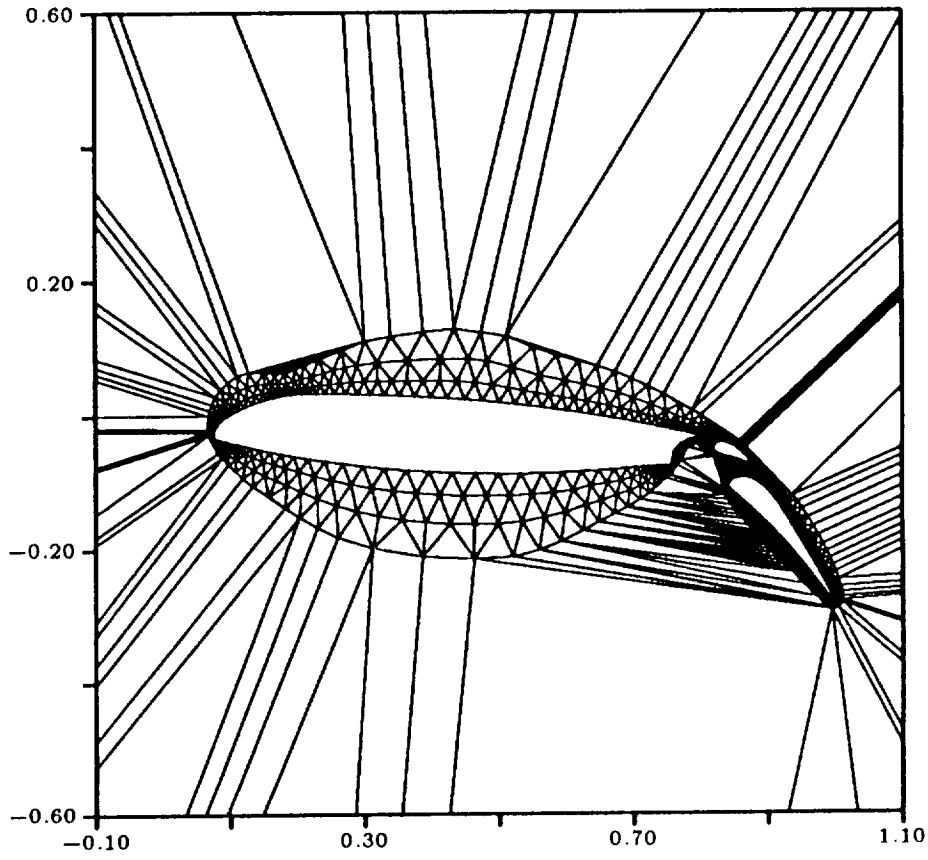

Figure 5
Illustration of the Advancing Front Nature of the Algorithms Described in [8] and [9]. The Illustration is Reproduced from [9] and Depicts the Delaunay Triangulation Obtained at an Intermediate Stage in the Mesh Generation Process.

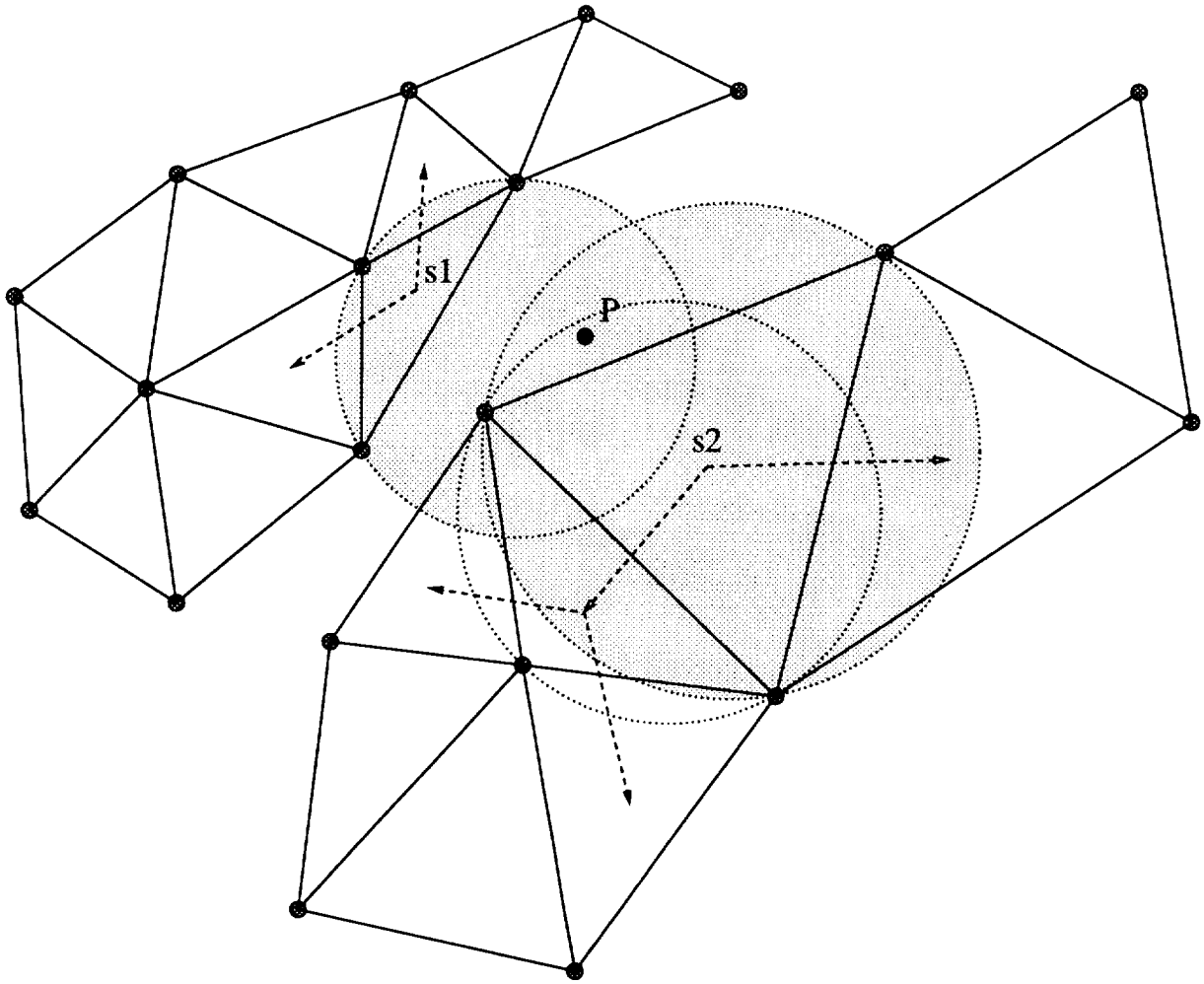

Figure 6

Illustration of the Search for Intersected Circumcircles Employed in The Current Algorithm
Upon Insertion of Point P, the Two Front Triangles S1 and S2 are Flagged as Having their Circumcircles Intersected.
The Two Neighbors of S1 are Searched and Found to be Non-Intersected.
One of the Neighbors of S2 is Flagged as Intersected, which Causes the
Search to Proceed to its Neighbors, Where it Terminates.

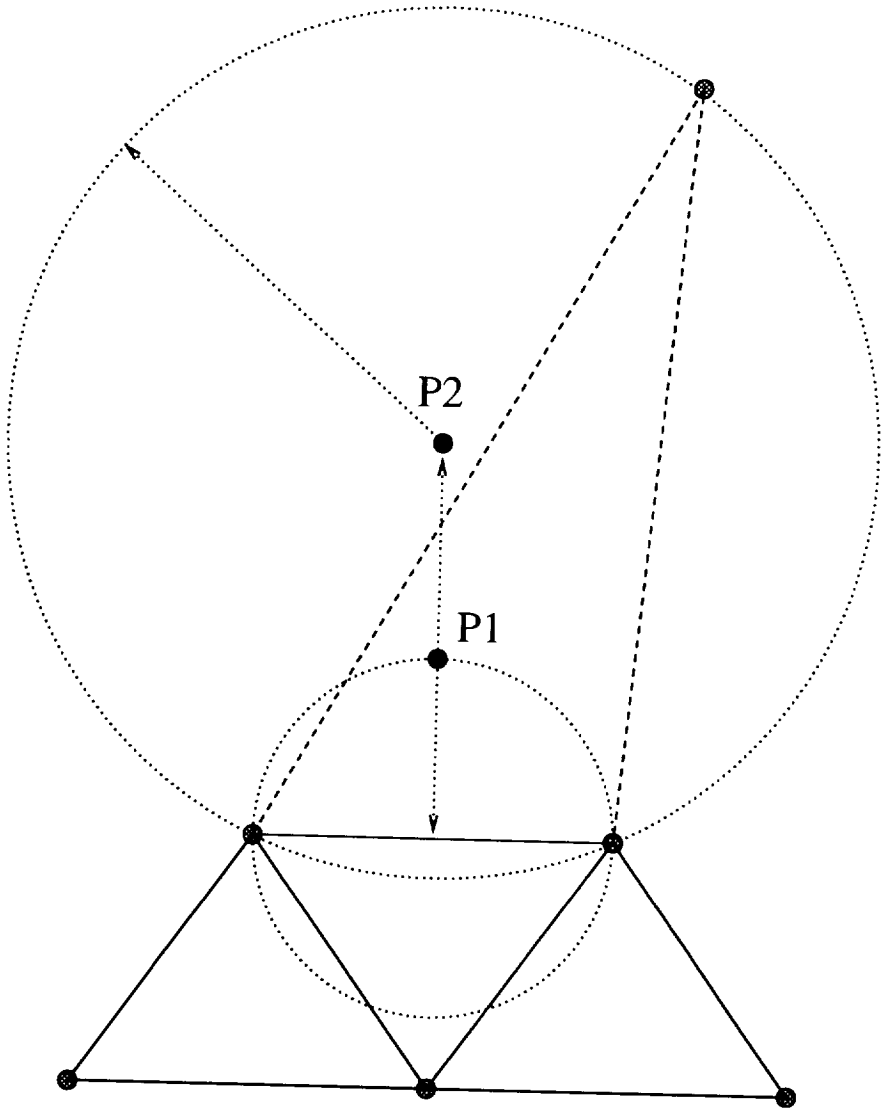

Figure 7

Illustration of the Point Placement Strategy Employed by the Current Algorithm. A New Point is Placed Along the Median of the Front Edge at a Distance Determined by the Prescribed Local Circumcircle Size (Background Function). The Point Position is Limited at the Lower End by the Intersection with the Inscribed Circle of the Front Edge (Point P1) and at the Upper End by the Circumcenter of the Delaunay Triangle Formed Between the Front Edge and Existing Mesh Points (Point P2).

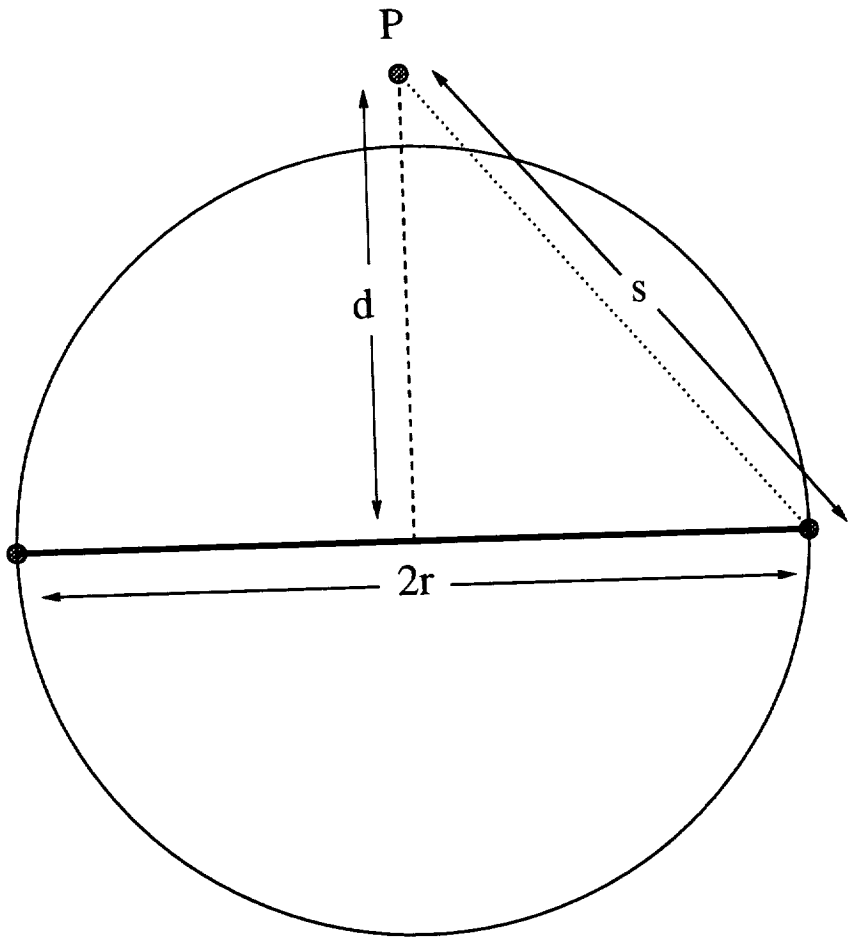

Figure 8
If a Point P is Not Contained in the Inscribed Circle of a Boundary Edge then the Distance from P to the End Points of the Edge s is bounded by $\sqrt{2}d$, Where d Represents the Minimum Distance from P to the Edge. Since $s = \sqrt{d^2+r^2}$, the Bounding Case Occurs when $d = r$, i.e. when P is on the Circle.

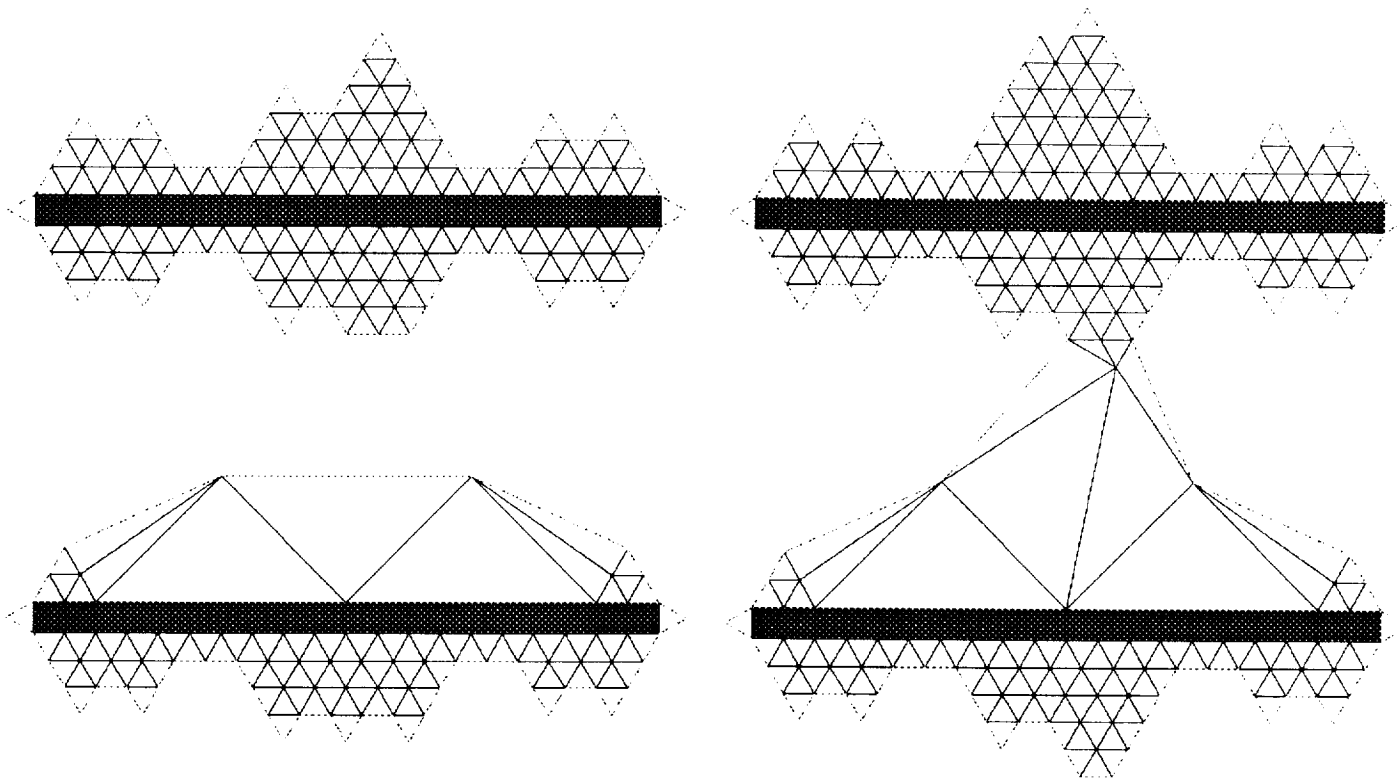

Figure 9 a)
Triangulation of Two Thin Plates With Dissimilar Boundary Point Distributions.
A Situation Involving Two Merging Fronts of Widely Different Length Scales Occurs.
Once a Vertex From the Front of Small Triangles Intersects One of the
Circumcircles of the Large Triangles on the Opposing Front, The Fronts are Merged.

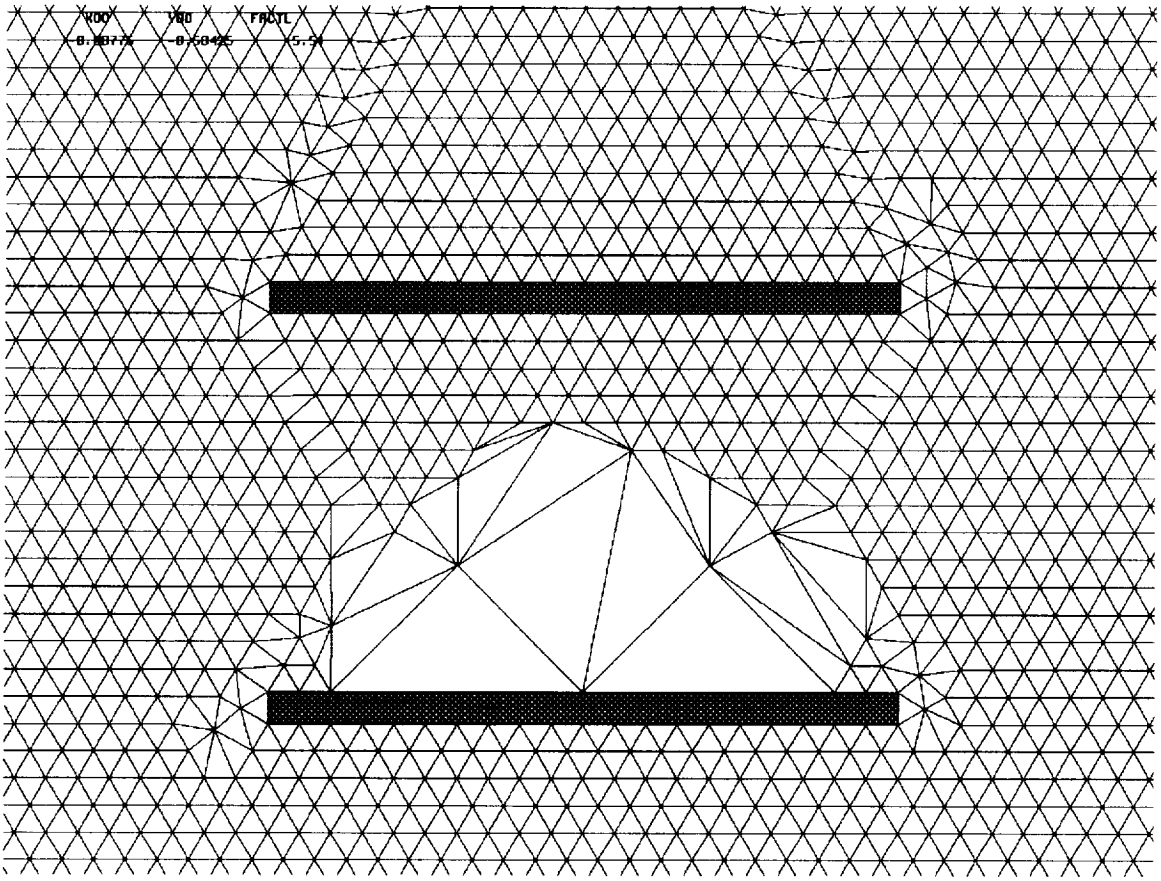

Figure 9 b)
Final Triangulation Produced in Region of Merging Fronts for
the Triangulation of Two Thin Plates

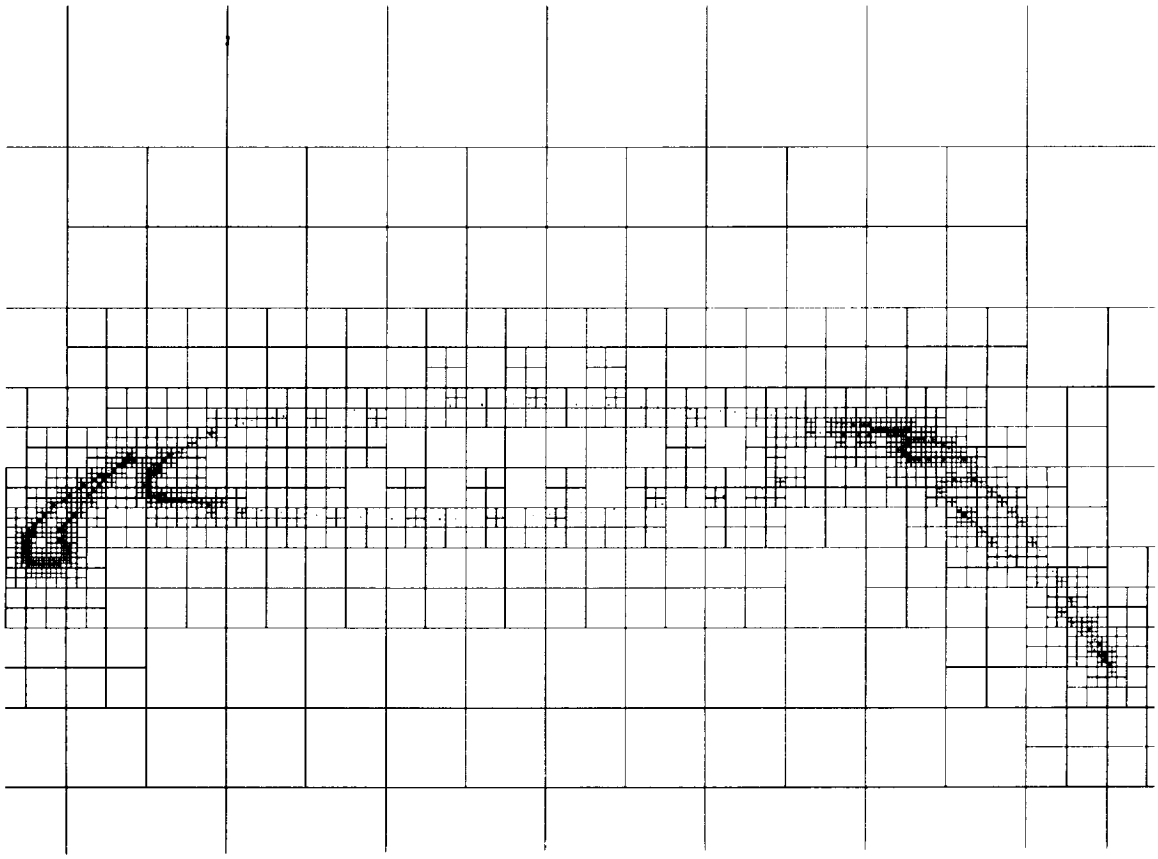

Figure 10
Quadtree Constructed About Initial Boundary Point Distribution
This Quadtree is Employed to Support the Background Spacing Function and
Also Represents the Initial Form of the Quadtree Employed in the Search for "Close" Front Points.

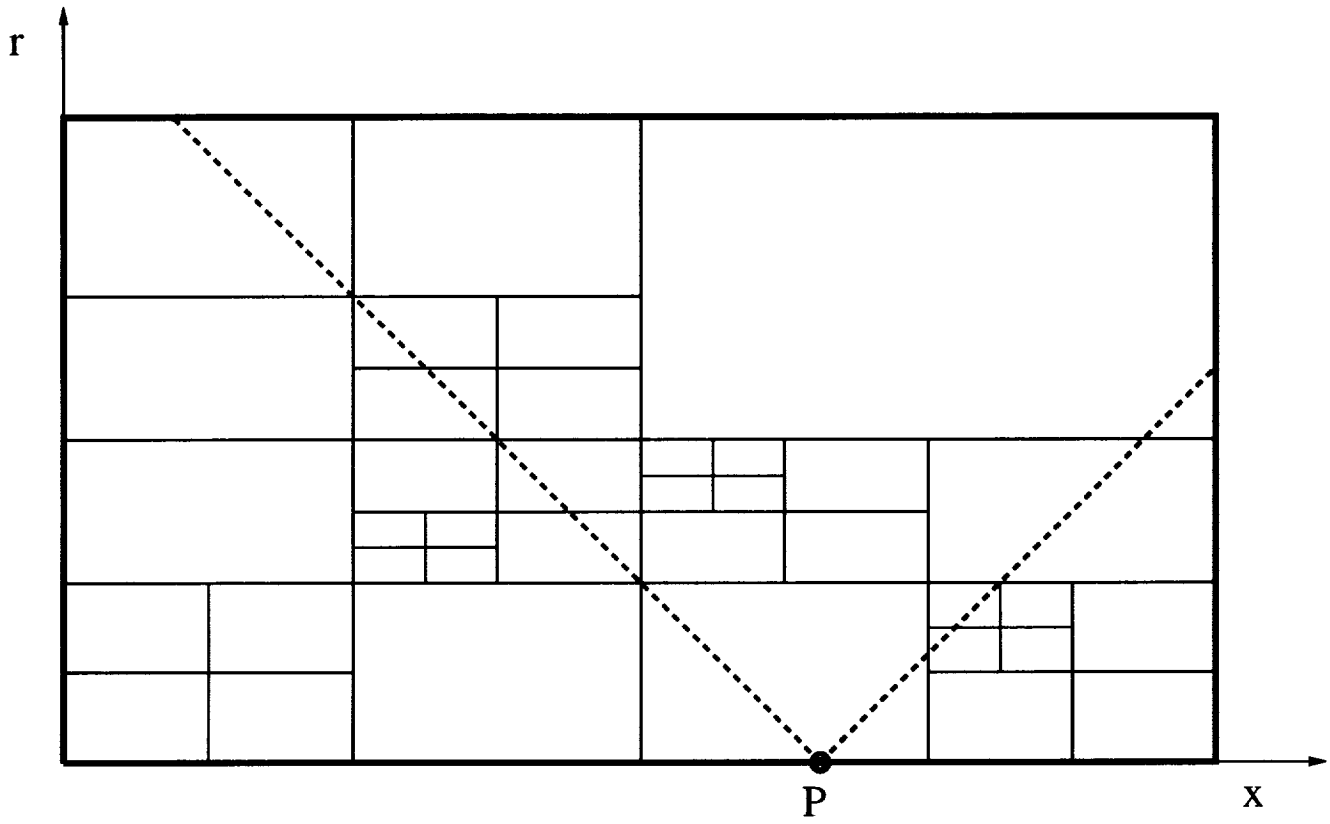

Figure 11

Two-Dimensional Illustration of 3D Octree Employed to Search for Intersected Front Triangle Circumcircles. Circles are Represented as Points in 3D, Determined by their Center (Horizontal Axis X (and Y)) and their Radius (Vertical Axis R). In order to Locate All Circles Intersected By Point P, All Quadrants (Octants) Fully or Partially Contained within the Cone Centered at (P,0) Must be Searched.

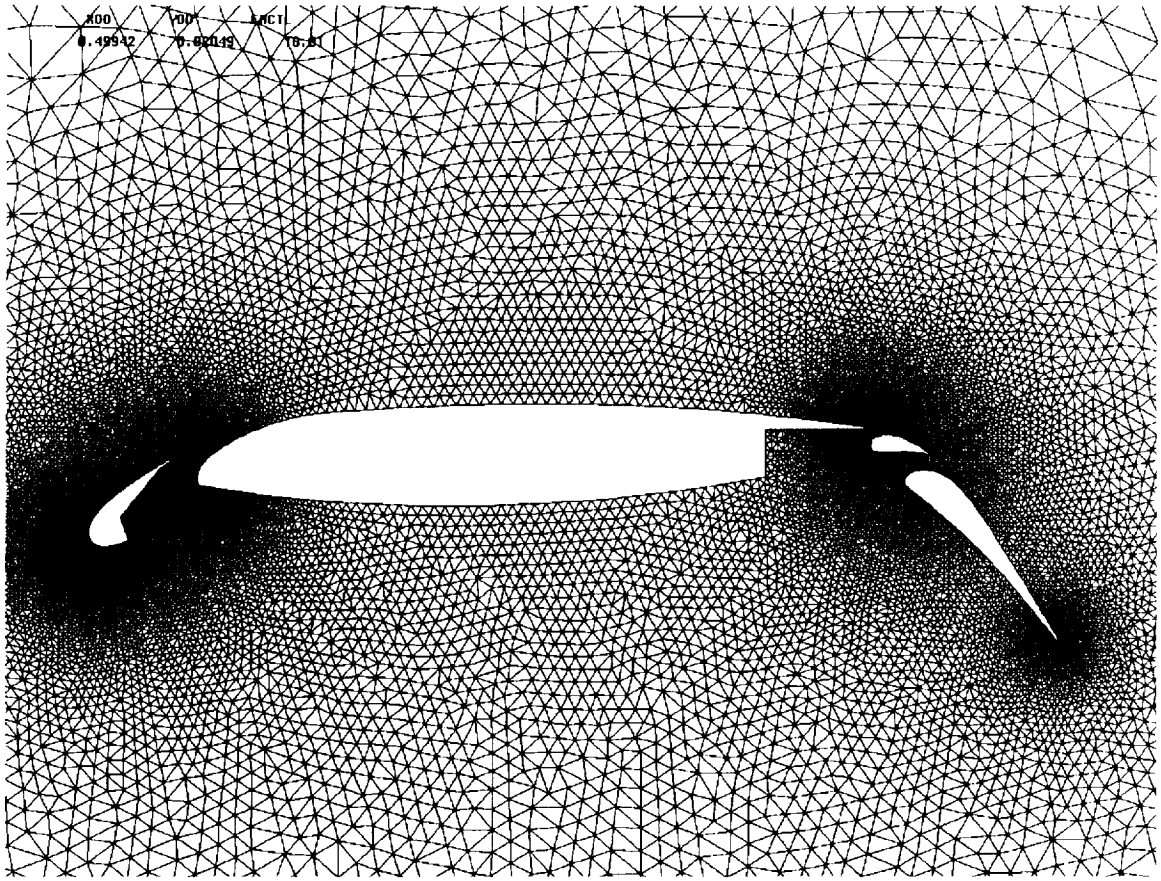

Figure 12 a)
Unstructured Triangular Mesh Generated About a Four Element Airfoil Configuration by the Present Algorithm before the Application of Mesh Smoothing.
(Number of Vertices = 21232, Number of Triangles = 41781)
(Global View)

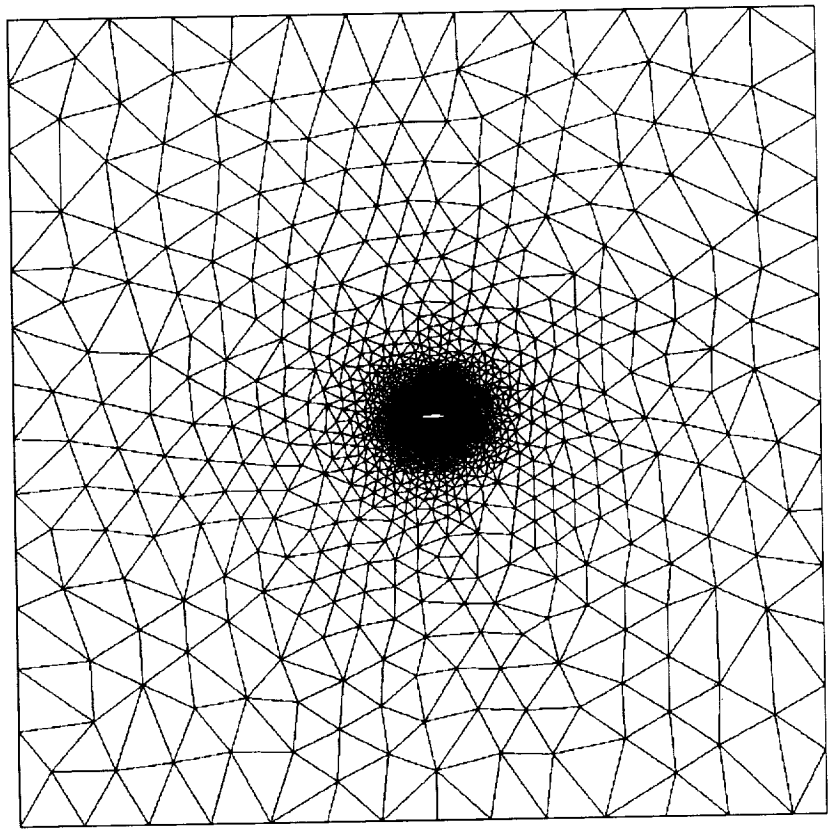

Figure 12 b)
Unstructured Triangular Mesh Generated About a Four Element Airfoil Configuration by the Present Algorithm before the Application of Mesh Smoothing.
(Number of Vertices = 21232, Number of Triangles = 41781)
(Far Field View)

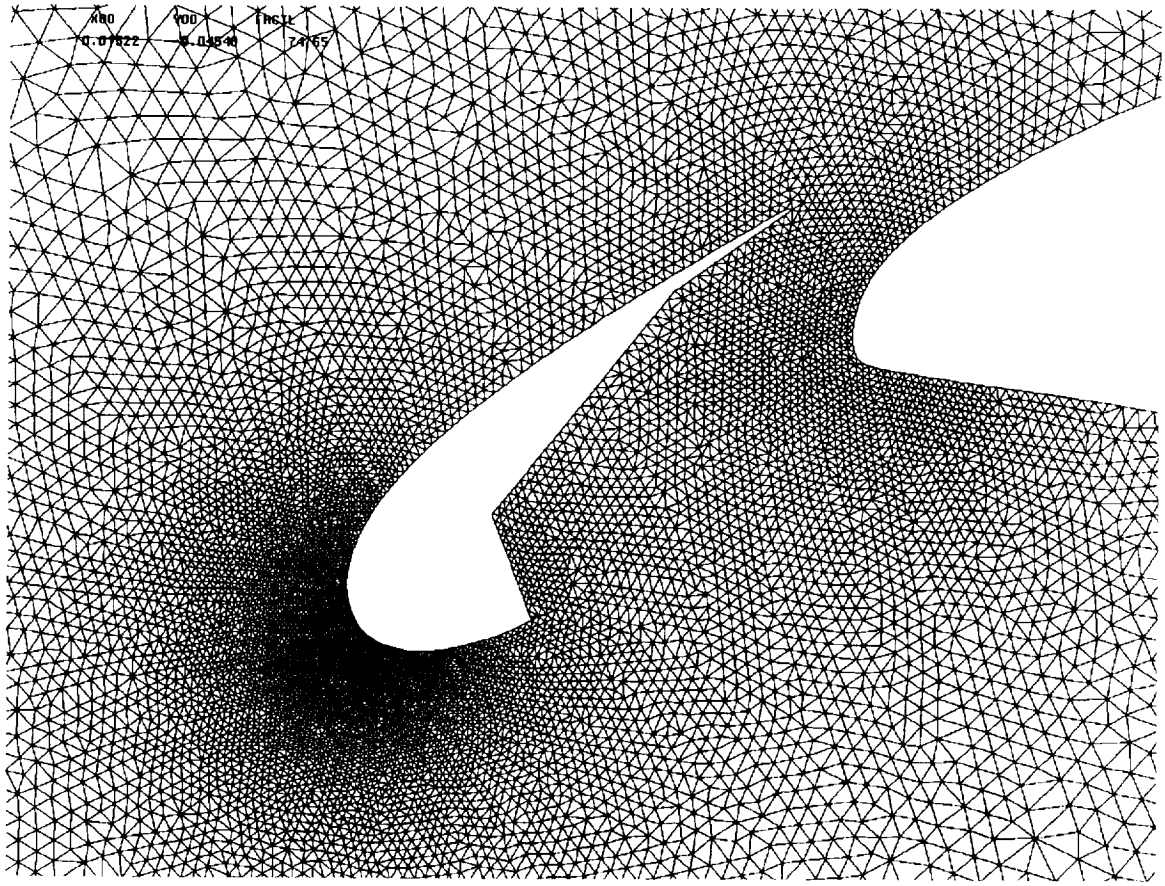

Figure 12 c)
Unstructured Triangular Mesh Generated About a Four Element Airfoil Configuration by the Present Algorithm before the Application of Mesh Smoothing.
(Number of Vertices = 21232, Number of Triangles = 41781)
(Close Up of Leading Edge Slat)

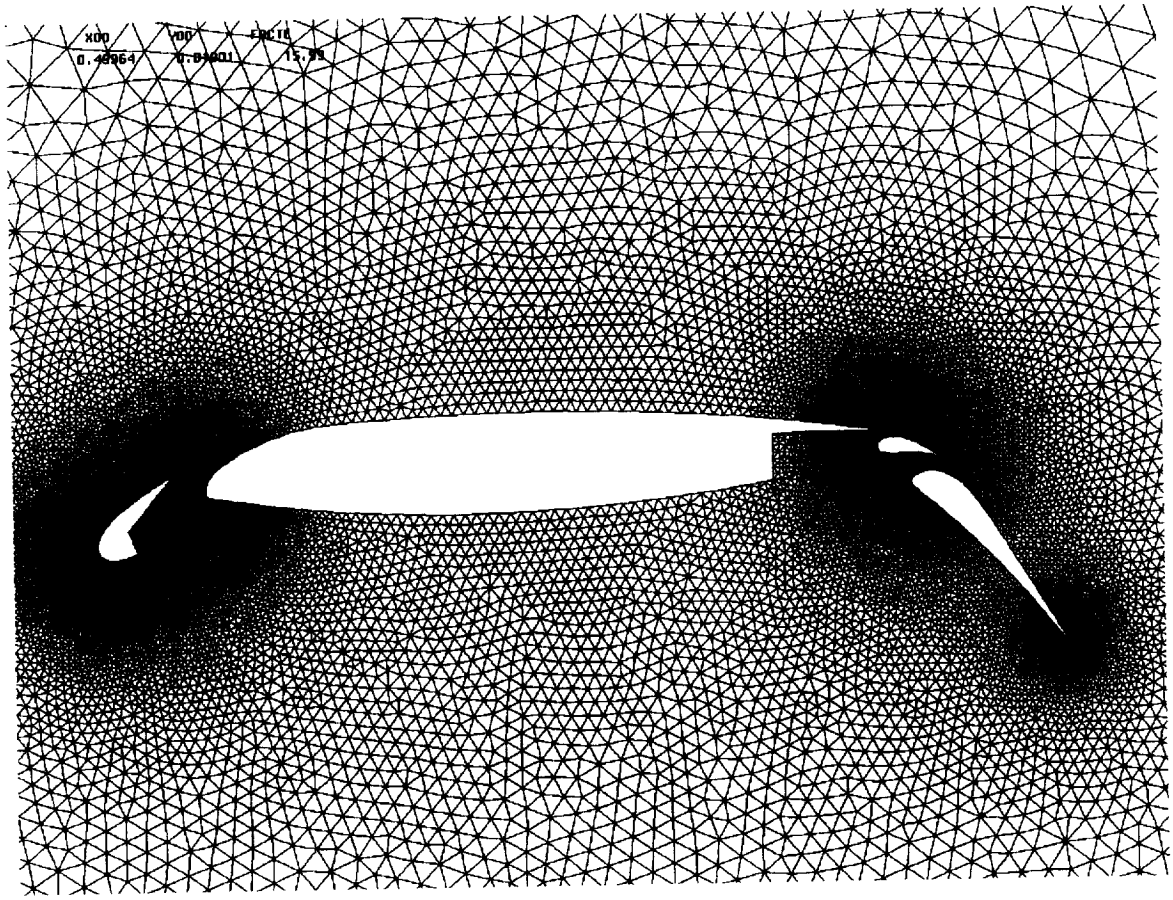

Figure 13 a)
Unstructured Triangular Mesh Generated About a Four Element Airfoil
Configuration by the Present Algorithm After the Application of Mesh Smoothing.
(Number of Vertices = 21232, Number of Triangles = 41781)
(Global View)

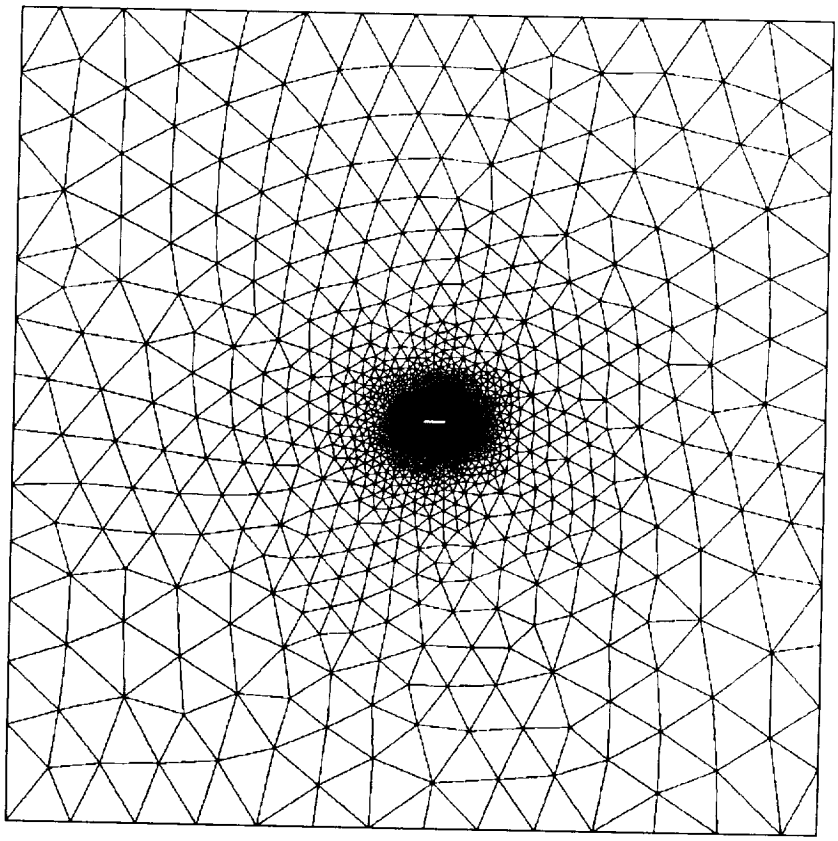

Figure 13 b)
Unstructured Triangular Mesh Generated About a Four Element Airfoil Configuration by the Present Algorithm After the Application of Mesh Smoothing.
(Number of Vertices = 21232, Number of Triangles = 41781)
(Far Field View)

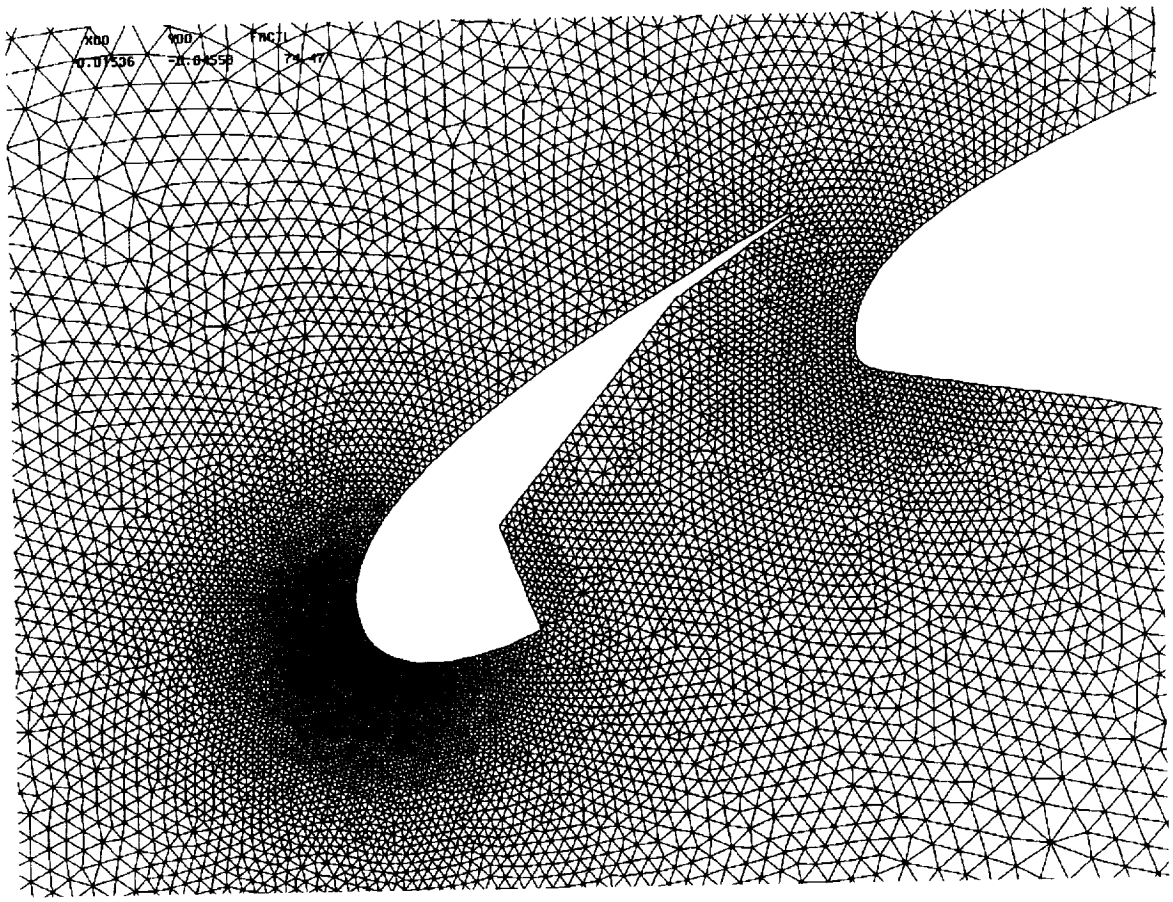

Figure 13 c)
Unstructured Triangular Mesh Generated About a Four Element Airfoil
Configuration by the Present Algorithm After the Application of Mesh Smoothing.
(Number of Vertices = 21232, Number of Triangles = 41781)
(Close Up of Leading Edge Slat)