NASA Contractor Report 195069

ICASE Report No. 95-26

ICASE

UNSTRUCTURED MESH GENERATION AND ADAPTIVITY

D. J. Mavriplis

Lecture notes prepared for 26th Computational Fluid Dynamics Lecture Series Program of the von Karman Institute (VKI) for Fluid Dynamics, Rhode-Saint-Genese, Belgium, 13-17 March 1995.

Contract No. NAS1-19480
April 1995

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA 23681-0001

Operated by Universities Space Research Association

UNSTRUCTURED MESH GENERATION AND ADAPTIVITY

D. J. Mavriplis*

ICASE
Institute for Computer Applications in Science and Engineering
MS 132C, NASA Langley Research Center
Hampton, VA 23681-0001
United States

Abstract

An overview of current unstructured mesh generation and adaptivity techniques is given. Basic building blocks taken from the field of computational geometry are first described. Various practical mesh generation techniques based on these algorithms are then constructed and illustrated with examples. Issues of adaptive meshing and stretched mesh generation for anisotropic problems are treated in subsequent sections. The presentation is organized in an educational manner, for readers familiar with computational fluid dynamics, wishing to learn more about current unstructured mesh techniques.

*This research was supported by the National Aeronautics and Space Administration under NASA Contract 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 23681-0001.

UNSTRUCTURED MESH GENERATION AND ADAPTIVITY

D. J. Mavriplis

ICASE
Institute for Computer Applications in Science and Engineering
MS 132C, NASA Langley Research Center
Hampton, VA 23681-0001
United States

Contents

1 Introduction

The generation of unstructured meshes for computational fluid dynamics problems has evolved rapidly over the last ten years. In fact, it has been stated recently that unstructured mesh generation has reached such a level of maturity that it can be considered a solved problem. Certainly, the generation of unstructured meshes about arbitrarily complex three-dimensional configurations can be routinely performed on present-day workstations, with currently available techniques. In fact, a reader with good programming skills and persistence should be able to create his/her own mesh generation code from the information contained in these notes. However, it would be premature to state that further improvements are not needed in this field. Of the various techniques described in these notes, each has its particular strengths and weaknesses. Further improvements in efficiency, and especially robustness are surely needed. A good definition of an optimal mesh, as this relates to numerical properties of the solution scheme is still lacking, and with it, appropriate techniques for constructing such optimal meshes, particularly for anisotropic or stretched-mesh generation.

Many of the successful algorithms in unstructured mesh generation have found their roots in the field of computational geometry. Computational geometry is the theoretical science concerned with defining or postulating the existence of specific geometrical constructs (*i.e.*, particular triangulations in our case), devising algorithms for generating these constructs, and analyzing the complexity of these algorithms (usually asymptotic worst case complexities). For example, the Delaunay triangulation represents a fundamental computational geometrical construct for which many construction algorithms have been devised and analyzed [1, 2]. Unfortunately, much of the computational geometry work has been confined to two-dimensional triangulations, which constitute planar graphs, and are thus easier to analyze. Also, the concept of an optimal triangulation or algorithm from a computational geometry point of view does not always coincide with the view from a computational-fluid-dynamics point of view, and thus many of the computational geometry results have found little use in the area of unstructured mesh generation. Great progress in unstructured mesh generation has been made by devising heuristic algorithms combined with empirical experience. In some sense, the engineering field of mesh generation has outstriped the more theoretical field of computational geometry, particularly for three-dimensional constructions. However, while heuristic algorithms may work well for most cases encountered in practice, the lack of any sound theoretical justification of these algorithms leaves the door open for possible situations which may result in failure or greatly increased complexity of the algorithm. Hence, the construction of efficient and particularly robust mesh generation techniques can only be achieved through algorithms with solid theoretical justification.

The purpose of these notes is to familiarize the reader with current techniques for unstructured mesh generation, while exposing their individual strengths and weaknesses. A preliminary discussion of selected computational geometry algorithms which are most relevant to mesh generation techniques is first given. Once these essentials have been outlined, a critical description of various mesh generation approaches in both two- and three-dimensions is given. The issue of mesh adaptation is discussed in a following section. In the final section, modifications to the described algorithms as well as alternate strategies for generating anisotropic or stretched meshes are described.

The data-structures required for efficient implementation of many of these algorithms are not discussed in detail. Their description and analysis can be found in appropriate computer science text books. The main data-structures employed in the algorithms of this chapter are the heap-list, and the region quadtree. The heap-list represents a particular implementation of a priority queue, which enables the ordering of elements based on a key [3]. The region quadtree, or octree in three dimensions, is a spatial decomposition data-structure, which enables efficient implementation of spatial search operations such as proximity searches. Quadtrees are discussed in detail in [4]. Both the quadtree and the heap-list are dynamic data-structures, *i.e.*, they enable simple operations

such as insertion and deletion of elements, and are thus particularly well suited for mesh generation and adaptation purposes.

2 Computational Geometry Constructs and Algorithms

2.1 The Delaunay Triangulation

Given a set of points $\{P\}$ in a plane, there exists many ways to join the points together to form a set of non-overlapping triangles which completely covers the domain. The Delaunay triangulation represents a particular construction of this type which has various well defined properties. For example, the Delaunay triangulation is the dual of the Voronoi Tessalation. A Voronoi Tessalation is the graph obtained by drawing the median line-segments which separate the plane into regions which are closer to a given point of $\{P\}$ than to any other point in the set $\{P\}$, as shown in Figure 1.

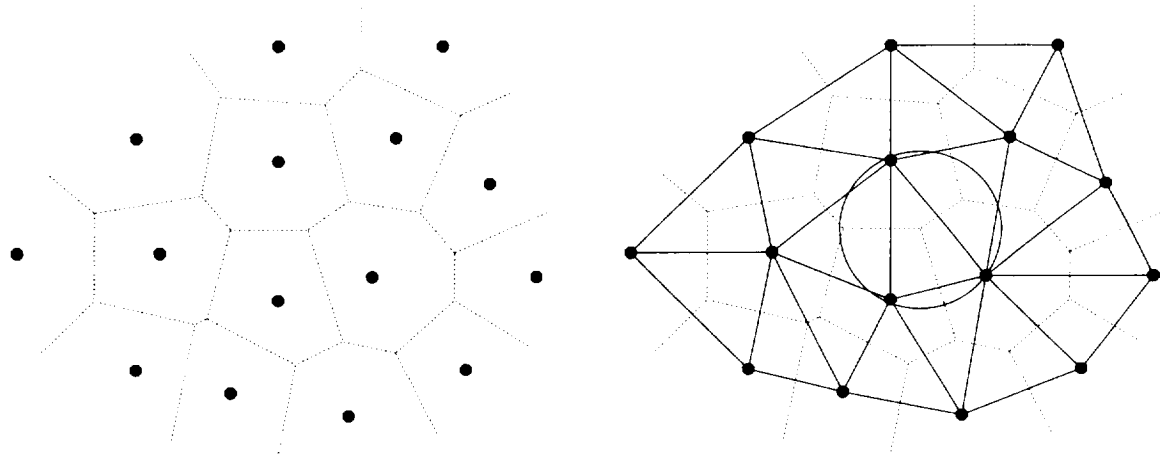

Figure 1: Voronoi tessalation and corresponding Delaunay triangulation of a set of points in the plane, illustrating the empty circumcircle property.

If we draw a line segment between any two points which are neighbors in this Voronoi diagram, we obtain the Delaunay triangulation of these points.

Another property of the Delaunay triangulation is known as the empty circumcircle property. This states that no point of the forming set $\{P\}$ can be contained inside the circumcircle of any triangle, as shown in Figure 1.

It is also well known that the Delaunay triangulation corresponds to a max-min triangulation, *i.e.*, out of all the possible triangulations of a given set of points, it is the triangulation which incurs the largest minimum angle for all triangular elements. Thus, a Delaunay triangulation may be expected to result in "well-shaped" elements, without very small angles.

While there are many other properties associated with the Delaunay triangulation [1, 2], the above are the most useful for mesh generation purposes.

The empty circumcircle property, in particular, is attractive, since it extends readily to three dimensions, by considering the circumsphere associated with each tetrahedron. The empty circumcircle/sphere property forms the basis for several Delaunay triangulation algorithms in both two and three dimensions, which are described below.

2.1.1 Bowyer-Watson Algorithm

This is an incremental algorithm which assumes new points are added sequentially to an existing Delaunay triangulation [5, 6]. When a new point is inserted into the triangulation, the first task

is to identify all existing triangles whose circumcircle is intersected by the new point, as shown in Figure 2. This can be achieved by first locating the triangle which contains the new point. The circumcircle of this triangle must certainly be intersected by the new point. The remaining intersected triangles can be identified by first searching the neighbors of the containing triangle, and then the neighbors of these neighbors, in a breadth-first type search, abandoning the search along any neighbor path where the test result is negative. Properties of the Delaunay triangulation guarantee that such a neighbor search is sufficient to locate all intersected triangles. The union of these intersected triangles is then removed from the current triangulation, thus creating a convex cavity which contains the new point, as shown in Figure 2. A new triangulation is then constructed by joining the new point to all vertices on the boundary of the polygonal cavity. The algorithm extends naturally to three dimensions by considering the circumspheres of tetrahedra and results in the retetrahedralization of a convex polyhedral cavity. Proofs of the convexity of the resulting cavity, and the validity of the retriangulation of this cavity have been formulated in the literature [6, 7].

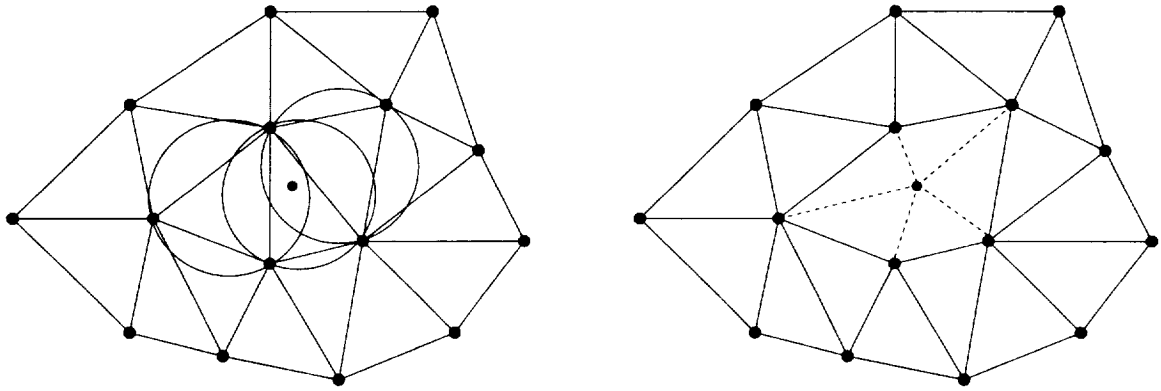

Figure 2: Illustration of the Bowyer/Watson algorithm for constructing Delaunay triangulations.

In order to construct the Delaunay triangulation of a set of points using this algorithm, an initial triangulation is constructed, either by forming the triangulation of the convex hull of the points [2], or more simply by creating a large initial triangulation, using three or four auxiliary points (in two dimensions), which contains all of the points to be triangulated. The points of the set to be triangulated are then put in a list, and inserted sequentially into the evolving triangulation.

The Bowyer-Watson algorithm has proven to be very useful for unstructed mesh generation. It has been shunned by the computational geometry field, mainly due to its poor worst-case complexity of $O(N^2)$ (imagine the case where each newly inserted point intersects all existing triangles). However, for unstructed mesh generation, near linear $O(N)$ performance has been reported for both two- and three-dimensional applications [7, 8, 9]. More recently, it has been shown that the poor worst-case complexity of this algorithm represents a pathological case which can easily be avoided by randomizing the order in which the points are inserted [10].

2.1.2 Green-Sibson Algorithm

This algorithm is similar to the Bowyer-Watson algorithm in that it is based on sequential point-insertion into an existing triangulation, and also relies on the empty circumcircle property [11]. To insert a new point into the triangulation, the triangle which encloses this new point is first located. The point is then inserted into the triangulation simply by joining it to the three vertices of the enclosing triangle, as shown in Figure 3. (In the case where the point falls on a mesh edge, the

edge is split and the point is joined to four mesh vertices; situations in three dimensions involving split faces and edges are treated analogously).

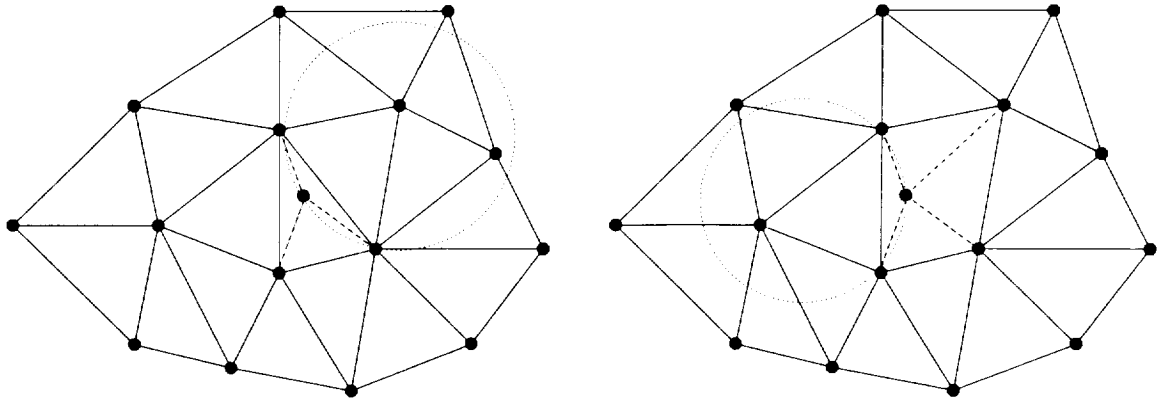

Figure 3: Illustration of the Green-Sibson algorithm for constructing Delaunay triangulations.

The resulting triangulation, although valid, is not necessarily Delaunay, and the remaining task is to transform it into the Delaunay triangulation by rearranging the mesh connectivity in the vicinity of the new point. This is accomplished by examining the three (or four) newly formed triangles. If their circumcircles are all empty, then the triangulation is indeed Delaunay, and no further modifications are required. In the event one of these circumcircles contains a vertex, the edge of the triangle which borders on an "outer" neighbor (*i.e.*, one which does not touch the newly inserted point), is reconfigured or swapped as shown in Figure 3. The reason only these outer edges need be considered is that the initial three (or four) edges which touch the newly inserted point can always be shown to be part of the final Delaunay triangulation (*i.e.*, consider how they may appear in the Bowyer-Watson algorithm). Each time an edge is swapped, two triangles are altered, and these must therefore be checked for the Delaunay criterion. However, each edge that is swapped can be shown to be included in the final Delaunay construction, and thus the only edges which need be considered for swapping are those which border on a modified triangle and an "outer" untouched triangle. The edge swapping procedure begins with the outer edges of the newly formed triangles (which contain the new point as a vertex), and propagates outwards, never reexamining the previously swapped edges, until the procedure terminates when no further edges need be swapped. There are certain practical advantages associated with the Green-Sibson algorithm over the Bowyer-Watson algorithm, for constrained triangulations, as well as for generating triangulations other than Delaunay, as will be demonstrated.

2.1.3 Tanemura-Merriam Algorithm

While the previous algorithms represent a top-down approach, a bottom-up approach to constructing the Delaunay triangulation of a given set of points is afforded by the advancing-front Delaunay algorithm. This technique has apparently been rediscovered several times in various fields throughout the literature [12, 13, 14, 15]. The idea is to construct the triangulation one triangle at a time, beginning at the boundaries of the domain, thus advancing or sweeping a front throughout the domain. The initial front is composed of the set of edge segments which define the convex hull of the point-set to be triangulated (or, for mesh generation purposes, the set of edges which define all physical boundaries of the domain). We begin by choosing an edge of this front. The problem consists of determining the particular point to which the two end points of this edge must be joined in order to construct the unique Delaunay triangle for this edge, which will be present

in the final triangulation. This may be solved in a simple iterative manner. An arbitrary interior point (interior with respect to the orientation of the front) is chosen, and the triangle formed by the two end-points of the front edge and the interior point is constructed. If this triangle contains any other 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.

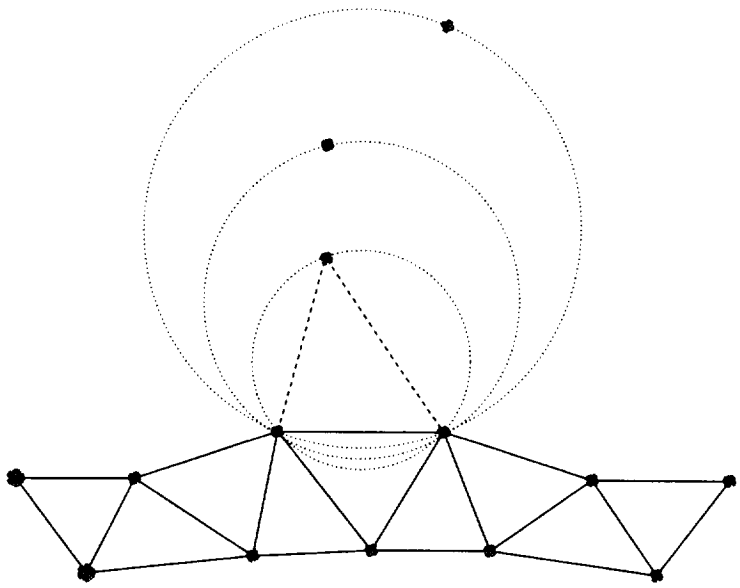

Figure 4: Iterative procedure for determining appropriate point for constructing the next triangle in the advancing-front Delaunay triangulation algorithm.

By iterating on this procedure, as shown in Figure 4, the appropriate point which produces a triangle containing no points interior to its circumcircle is eventually found. This new triangle is therefore accepted, and the front is advanced by removing the current edge from the front, which is now obscured by the new triangle, and adding the new edges to the front. (There may be 0, 1, or 2 new edges depending on whether the chosen point is interior to, or on the front). The algorithm terminates once all edges have been deleted from the front, *i.e.*, when the entire domain has been swept out.

2.1.4 Constrained Delaunay Triangulations

A triangulation of a given set of points which is forced to include as a subset of predetermined edges, is known as a constrained triangulation. Loosely speaking, a constrained triangulation which is as close as possible to a Delaunay triangulation is called a constrained Delaunay triangulation. A more formal definition of a constrained Delaunay triangulation is given by Chew [16]: it is a triangulation which contains a set of prescribed edges, and such that the circumcircle of each triangle contains no other vertex of the mesh which is visible to it. A vertex is visible to a triangle if the line joining the vertex to any point interior to the triangle does not intersect one of the prescribed edges of the mesh. An example of a constrained Delaunay triangulation is given in Figure 5. The existence of constrained Delaunay triangulations ensures the validity of the Tanemura-Merriam algorithm for arbitrary initial (possibly non-convex) fronts. It also guarantees the possibility of modifying an existing Delaunay triangulation to include a set of edges which define the boundaries of the domain to be triangulated.

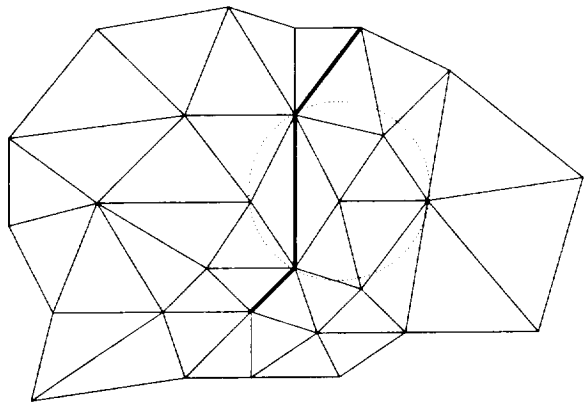

Figure 5: Illustration of constrained Delaunay triangulation.

This is an important consideration for practical mesh generation algorithms, where non-convex and multiply connected domains are often considered. Unfortunately, an equivalent definition for constrained Delaunay triangulations in three dimensions is not available. Thus, the construction and modification of three-dimensional Delaunay triangulations, which conform to a prescribed surface triangulation, has proved to be a non-trivial problem.

2.2 Other Triangulations and Transformation Algorithms

The Delaunay Triangulation is but one of a large number of possible triangulations of a given set of points. Other possible triangulations include the minimum total edge-length triangulation, or minimum weight triangulation, and the minimum maximum-angle triangulation [2]. In fact, any criterion for describing an optimum triangulation may be constructed, even one based on possible variable data-values stored at the vertices of the triangulation, such as in the data-dependent triangulations of [17], and an algorithm for transforming the current triangulation to the so-defined optimum triangulation devised. In practice, an algorithm capable of constructing the global optimum may be difficult to formulate, and local optima, or improvements from an initial mesh are often produced.

2.2.1 Two-Dimensional Edge-Swapping

A method for transforming a 2D triangulation into another 2D triangulation is given by the edge-swapping procedure of Lawson [18]. The algorithm is based on the fact that there are at most two ways of triangulating a set of four points, as shown in Figure 6. For each pair of triangles in the mesh which forms a convex quadrilateral, the original triangulation is compared to the alternative triangulation obtained by swapping the position of the internal diagonal, as shown in Figure 6. If the alternative triangulation is found to better optimize the triangulation criterion, then the diagonal is swapped. By iterative application of this simple diagonal swapping primitive over the entire mesh, the triangulation is eventually transformed into a more optimal triangulation, as defined by the criterion. In [19] it is proven that any given 2D triangulation may be recovered from any initial 2D triangulation through repeated application of the diagonal-swapping primitive, although a general algorithm for achieving this is not given. In general, it is found that a straight-forward application of the edge-swapping procedure gets stuck in local optima, and is often incapable of producing the global optimum mesh.

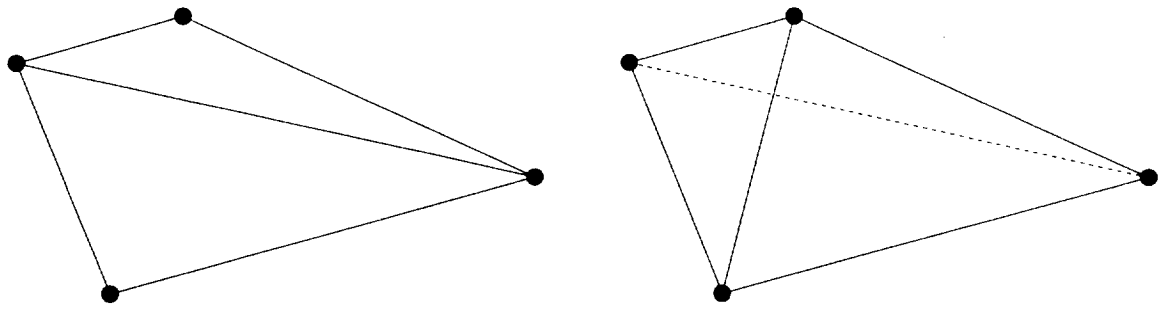

Figure 6: Two possible configurations for the diagonal of a convex pair of triangles in the edge-swapping algorithm.

Furthermore, depending on the order in which the edges are swapped, different local optima may be achieved. A notable exception is the construction of a max-min or Delaunay triangulation. It can be shown that repeated application of the edge-swapping technique based on maximizing the minimum angle always converges to the globally optimum mesh, which is the Delaunay triangulation, regardless of the initial mesh, and the order in which the edges are swapped. The complexity of this procedure is $O(NlogN)$, where N represents the number of vertices.

The globally optimum min-max triangulation, on the other hand, is not generally attainable with this algorithm. A more sophisticated technique, known as the edge-cutting algorithm [20], is capable of transforming any triangulation into the globally optimum min-max triangulation. The algorithm may be interpreted as a generalization of the edge-swapping procedure. The first step consists of locating the maximum angle in the current triangulation. Let prq be the triangle which contains this maximum angle, which is situated at point p, as depicted in Figure 7.

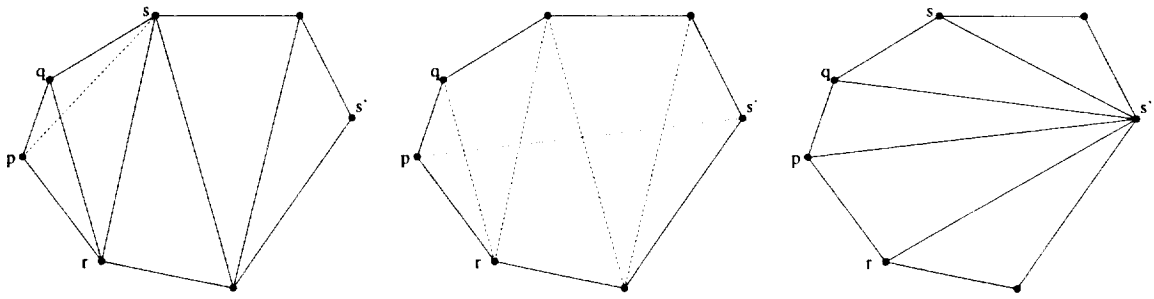

Figure 7: Illustration of the edge-cutting algorithm for constructing min-max triangulations.

In an attempt to reduce this angle, a new point s is chosen, and the edge ps is drawn. The first candidate for the point s is the point of the neighboring triangle rsq. If the two angles at q and r are both smaller than the original angle at p, then the swap is successful, and the procedure may be continued by searching for the next largest angle in the mesh. However, if the angles at q and r are both larger than the original angle at p, no improvements are possible. In the event the angle at q is larger, but the angle at r is smaller than the original angle at p, it can be shown that the optimum new edge must intersect the edge rs. Thus we choose a new point s' in the neighboring triangle of rsq. This procedure may be applied recursively to the neighbors of these neighbors, until a suitable point is found, perhaps several neighbor distances away. The edge ps' is then inserted into the triangulation, and all edges which intersect ps' are removed, as shown in Figure 7. This defines two empty polygons, which are then retriangulated. This is achieved by iteratively defining consecutive vertices of each polygon boundary which form a triangle with all angles smaller than the maximum angle of the original triangulation.

The insertion of edges such as ps', which may intersect many existing edges, involves additional non-local information which enables the algorithm to avoid getting stuck in local optima. However, the complexity of the edge-cutting algorithm is $O(N^2 logN)$, and no three-dimensional extensions have been reported. These are the probable reasons why this algorithm has seldom been implemented for mesh generation applications.

2.2.2 Three-Dimensional Edge-Face Swapping

Three-dimensional tetrahedralization algorithms based on simple edge-face swapping primitives are also possible. These primitives are based on the fact that, in d dimensions, $d+2$ points may be triangulated in at most two ways, as stated by Lawson [21]. Thus, in three dimensions, a set of five points may be triangulated in at most two manners. Depending on the configuration of these five points, either a unique triangulation exists, in which case the configuration is called non-swappable, or two different triangulations are possible, in which case swapping between the set of two constructs is possible. In general, the triangulation of five points in three dimensions may result in 2, 3, or 4 tetrahedra, as shown in Figure 8.

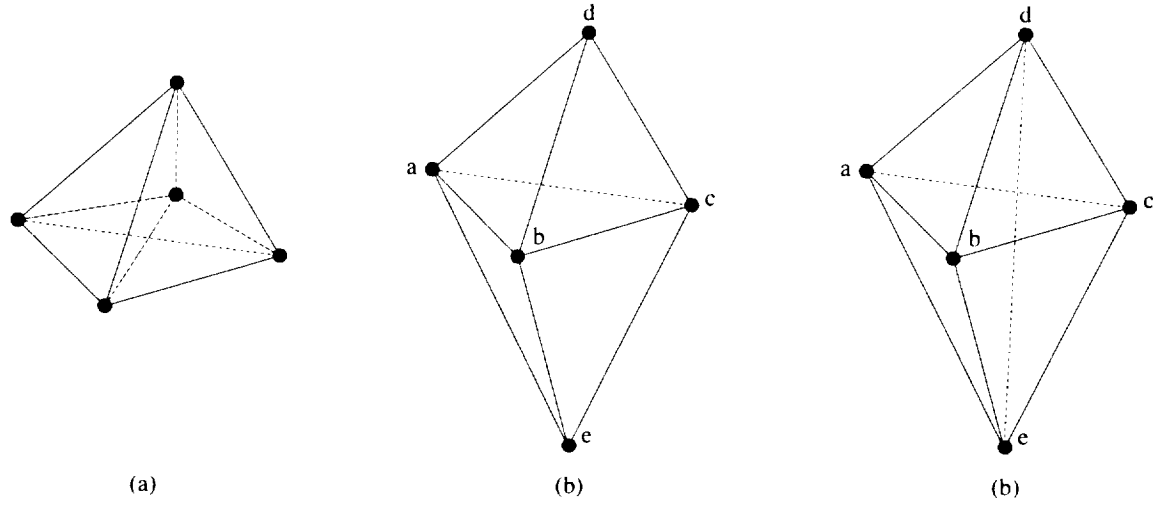

Figure 8: Possible tetrahedralizations of a set of 5 points in three dimensions (neglecting degenerate cases.

This is in contrast to the two-dimensional case, where the number of triangles and edges are identical for all possible triangulations of a given point-set. However, the triangulation of three-dimensional point-sets no longer represents a planar graph, and Euler's formula no longer relates the number of cells to the number of vertices [22]. Thus, various triangulations of the same point-set in 3D can be expected to contain different numbers of cells, faces, and edges, and the face-edge swapping primitives can be expected to modify these numbers.

In the case where five vertices are triangulated with four tetrahedra, (*i.e.*, case (a) of Figure 8), one of the five points is interior to the convex hull of the other four points, and no other triangulation of this configuration is possible. This is a non-swappable configuration. In the case where the five vertices are triangulated with two tetrahedra, (*i.e.*, case (b) of Figure 8), the configuration is swappable provided it is convex. A sufficient condition for convexity is that the line de intersect the face abc in Figure 8 (b). If this is the case, then the two-tetrahedron configuration of Figure 8 (b) may be swapped into the three-tetrahedron configuration of Figure 8 (c) by removing face abc and inserting the edge de (and thus faces ade, bde, cde). If the initial two-tetrahedron configuration is non-convex, then it is non-swappable. (In this case, it corresponds to the configuration in Figure

8 (c), where one of the tetrahedra is missing). Similarly, in the case where the five vertices are triangulated with three tetrahedra, as in Figure 8 (c), the configuration is swappable into the two-tetrahedron configuration of Figure 8 (b), provided it is convex. To determine if the three-tetrahedron configuration is convex, the convexity test may be applied directly to the anticipated two-tetrahedron swapped configuration, which yields the same result. A complete algorithm for transforming three-dimensional triangulations may be given as follows:

Step 1: Tag all mesh faces as candidates for swapping.
Step 2: Choose a pair of neighboring tetrahedra in the mesh which share a candidate face. This defines five vertices, three of which form the face common to both tetrahedra, and two others which form the end points of the two tetrahedra.
Step 3: By searching through neighboring tetrahedra, locate all other elements which contain four of the five vertices.
Step 4: If the number of elements is four, the configuration is non-swappable. Go to step 8.
Step 5: Check for convexity. If the configuration is non-convex, go to step 8.
Step 6: Compare the swapped configuration with the original configuration. Choose the configuration which "improves" the triangulation based on the given criterion.
Step 7: Tag the swapped faces as optimal, and the neighboring faces as future candidates for swapping.
Step 8: If future candidates for swapping exist, go to step 2, else end.

As in the two-dimensional case, these local transformation techniques most often terminate in local optima, with the result that the global optimum triangulation for a given criterion is usually not achieved. While convergence to the globally optimum max-min or Delaunay triangulation is assured in two dimensions, a similar result does not hold in three dimensions. In fact, the max-min and Delaunay triangulation are not equivalent in three dimensions, and three-dimensional Delaunay triangulations must be characterized by the empty-circumsphere property. Using the empty-circumsphere test as a measure of optimality, Joe [23] has shown that local transformations of an arbitrary triangulation are not guaranteed to converge to the Delaunay triangulations. Only under special conditions, such as the addition of a new vertex to an existing Delaunay triangulation following the Green-Sibson algorithm, can the application of local transformations be guaranteed to converge to the Delaunay triangulation [23, 24, 25].

3 Practical Mesh Generation Algorithms

The algorithms described in the previous section can be used to triangulate an existing set of points, or to modify an existing triangulation. As such, they do not constitute mesh generation techniques, but rather form basic building blocks, which can be used in conjunction with other techniques to devise mesh generation strategies. In general, the generation of unstructured meshes involves the creation of both the mesh points and their connectivity. This can be performed either sequentially (pre-generation of mesh points, followed by a triangulation phase), or simultaneously using an advancing-front technique, or a Steiner triangulation technique. (Steiner triangulations refer to the insertion of additional points into an existing triangulation in order to improve the quality of the triangulation.) The problem of mesh generation includes several well defined phases which can be summarized as follows:

- Definition of boundaries of physical domain.
- Definition of element-size distribution as a function of spatial location (element shape distribution is also required in the case of stretched meshes).

- Generation of boundary conforming mesh using a suitable approach.
- Optional mesh post-processing to improve element quality.

The boundaries of the physical domain are usually defined through some CAD-type data-base, which most often involves the use of piecewise spline curves in 2D, and assemblies of trimmed (*i.e.*, possibly intersecting) spline surface patches in 3D. This initial boundary description must be discretized as a set of line segments in 2D or a collection of planar faces in 3D, for mesh generation purposes. This may be achieved prior to, or simultaneously with, the construction of the mesh. The mesh generation procedure must be capable of guaranteeing boundary integrity: in two dimensions this corresponds to generating triangulations which contain the subset of edges which define the domain boundaries (c.f. Figure 9),

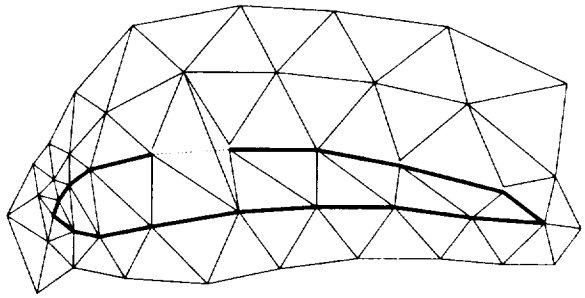

Figure 9: Illustration of non-boundary conforming Delaunay triangulation for a simple airfoil geometry.

while in three dimensions this corresponds to generating tetrahedralizations which contain the subset of triangular faces which define the discretized boundary surfaces. The problem of matching a prescribed surface triangulation in the construction of a three-dimensional volume grid can be considerably involved. A slightly less stringent approach to three dimensional boundary integrity is that of generating a tetrahedralization which contains a possibly arbitrary boundary triangulation which nevertheless does not violate the boundary surface integrity *i.e.* a surface triangulation which does not intersect or cut through the original boundary surface definition. Boundary integrity is extremely important in mesh generation, for if the boundary surfaces cannot be recovered in the mesh, no numerical simulation is possible.

The definition of the element-size distribution may be accomplished either implicitly or explicitly. An implicit definition may involve inferring a value for the element-size in the interior of the mesh from the boundary discretization (*i.e.*, by interpolation from the closest boundaries). Another possibility is the use of an analytic mapping function for generating sets of points with desired spacings (*i.e.*, the use of structured O- or C-meshes for generating point sets) which are subsequently triangulated. Explicit definitions of the element-size distribution involve the construction of a function $s = f(x, y, z)$ valid over the entire physical domain. This function may be constructed analytically, or using a small set of discrete sources such as:

$$s = \sum_{k=1}^{N} s_k \left[1 + \left(\frac{r(\mathbf{x}) - r_k}{\bar{r}_k} \right)^{\gamma_k} \right] \tag{1}$$

where the summation is over all N sources, and $r(\mathbf{x})$ denotes the position vector. The source parameters s_k and r_k represent the prescribed element size at the source, and the location of the source, and $\bar{r}_k$ as well as γ_k define the relative region of influence of the source.

Alternatively, a background grid may be employed, where at each vertex a value of the element size is defined, and values in between vertices are obtained by linear interpolation. These background meshes may be unstructured [26, 27] (triangular, tetrahedral), cartesian [28], or even quadtree-based [29]. In practice, an effective approach is to construct an (elliptic) partial differential equation with the aid of source terms defined in a similar manner to those of equation (1), and to solve this equation on the supporting background grid. The source terms can be prescribed interactively, and correspond to prescribed mesh spacings at these physical locations. This approach is particularly attractive, since the elliptic construction ensures a smooth element-size distribution function, which is desirable for the generation of high quality meshes.

While mesh quality is usually defined somewhat heuristically, this usually entails the notion of well shaped elements with small changes of element size between neighboring elements. Regardless of the triangulation scheme employed (Delaunay, min-max, etc...) it is now well known that acceptable mesh quality cannot be achieved by optimizing mesh connectivity alone, but is necessarily the result of a close matching between the mesh-point distribution and the mesh connectivity employed in the construction of the mesh. (This is even more important in the case of stretched or anisotropic meshes). Post-processing techniques may be employed for improving mesh quality. These often include minor redistribution of the mesh points through Laplacian-type smoothing techniques, often followed by reconfiguration of the mesh connectivity. While these techniques are often successful in improving grid quality by removing minor irregularities, they seldom adequately correct for serious deficiencies in grid quality. Thus the ability to generate a high quality mesh from the outset is of the utmost importance. Finally, mesh generation procedures must be robust. This not only means that they must use proven constructs and exact algorithms, but they must be designed to avoid and/or deal with ambiguous situations which typically arise in computational geometry problems. The use of exact arithmetic [30] is perhaps the best approach to resolving such issues.

3.1 Quad/Octree Based Mesh Generation

One of the earliest and simplest methods for generating unstructured meshes involves the use of quad and octrees in two and three dimensions respectively. Considering a two-dimensional example for simplicity, an initial quad is formed which is large enough to cover the entire domain. Assuming a mesh element-size distribution function exists, the quadtree is recursively subdivided until all leaf quads are no larger than the local value of the element-size distribution function. If this function is only defined on the domain boundaries, the quadtree may be initially subdivided along the boundaries. Subdivision in the domain interior can then proceed by ensuring that jumps in the sizes of neighboring leaf elements never exceeds 2:1. Triangular elements can be generated by forced subdivision of the quad elements [31], or by using the quadtree vertices in a Delaunay point-insertion algorithm [32]. At the physical boundaries, the quadtree must be made boundary conforming. This is usually accomplished by warping the mesh, *i.e.*, displacing the closest quadtree vertices to coincide with the boundary curve. Quad/octree methods are relatively simple and efficient. Their main deficiencies relate to the quality of the mesh near boundaries due to the warping procedure.

3.2 Advancing-Front Methods

Advancing-front methods involve the simultaneous generation of mesh points and their connectivity. The idea is to build the mesh element by element, adding new elements to previously generated elements, thus sweeping out a front across the entire domain. They usually rely on an explicitly defined element-size distribution function, which is most often constructed using a background grid [26, 27, 33].

Advancing-front techniques begin with a discretization of the domain boundaries as a set of edges in two dimensions. This is accomplished by placing points along the boundary curves governed by the local values of the element-size distribution function, as shown in Figure 10. 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 11.

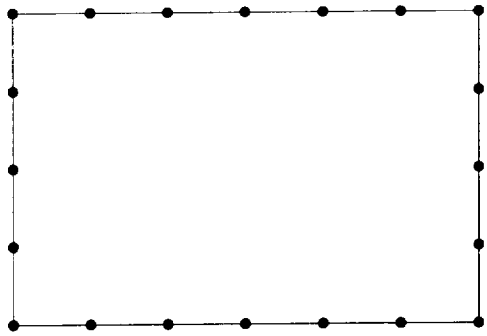

Figure 10: Initial boundary discretization for the advancing-front method.

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.

The selection of the next edge in the front may be based on various criteria. A good strategy is to always choose the smallest edge in the front, thus ensuring the front grows from regions of small cells towards regions of large cells. This has been found to yield smooth high-quality element distributions. Such a strategy can easily be implemented by encoding the front as a heap-list of edges, with the edge-length as the heap ordering key [3].

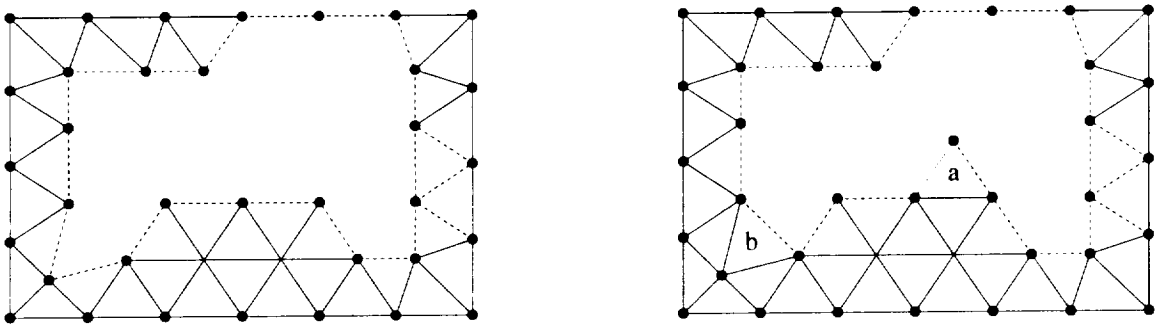

Figure 11: Generation of new triangle using new point (a), or existing front point (b), in advancing-front method.

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 *i.e.*, along the median of the front edge, at a distance defined by the local value of the element-size distribution function, as shown in Figure 12. 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. Thus, all front points within a certain distance of the new point must be located. This involves the determination of an appropriate length scale for defining the search region. A typical example of a length scale and search region often employed is shown in Figure 12. The final accepted triangle will be the triangle formed from the current front edge and the new point, or one of the "nearby" front points, which does not intersect any front edges, and best conforms to the local element-size distribution function.

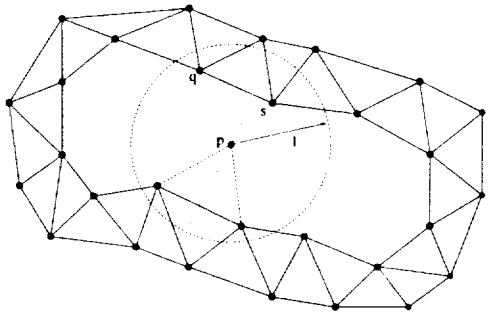

Figure 12: Placement of new point in advancing-front method.

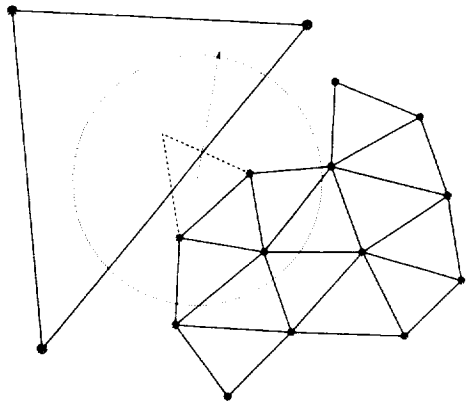

Figure 13: Possible failure of advancing-front method for merging fronts of dissimilar sizes.

The determination of the set of "nearby" front points involves a proximity search which is usually implemented using a quadtree data-structure [4]. Thus, a quadtree based on the front points must be maintained dynamically, with points inserted and deleted as the front advances. The front is thus simultaneously represented as a heap and a quadtree.

The space requirements for such an algorithm are lower than may be expected. Since this is essentially a greedy triangulation [2], *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 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 $O(Nlog\sqrt{N})$ which is asymptotically equivalent to $O(NlogN)$. 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 [34, 29], which can greatly reduce the required working size for a large mesh generation job.

One of the advantages of such an approach is 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 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 ($O(logN)$ 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). In 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 13. Thus, the success of the advancing-front technique relies on the existence of a smoothly varying field function.

3.3 Delaunay Point-Insertion Methods

Delaunay-based methods offer the possibility of constructing mesh generation strategies based on proven computational-geometry algorithms (although this does not in itself guarantee a more efficient/robust overall approach).

Some of the earliest Delaunay-based mesh generation strategies relied on predetermined mesh-point sets [8, 7, 35, 36]. For multi-component geometries, each component represents a simple configuration which can be fitted locally with a simple structured mesh (*i.e.*, for example using a structured O-mesh about each element of a multi-element airfoil. By constructing a set of overlapping structured meshes in this manner, and discarding the connectivity of these meshes, as well as the points which fall outside of the physical domain, a set of points is obtained which may be used as the basis for a Delaunay triangulation mesh generation strategy, using the Bowyer/Watson or Green-Sibson algorithms.

In one particular approach [8, 35], an initial Delaunay mesh is constructed by joining one of the inner boundary points to all of the outer boundary points in a hub and spoke type arrangement. The inner boundary points are then inserted into the triangulation using the Bowyer/Watson algorithm. After all inner boundary points are inserted, a check for boundary integrity is performed. If boundary violations are encountered, additional boundary mesh-points can be inserted to remedy the situation. Once boundary integrity has been recovered, triangles exterior to the domain are removed, thus effectively freezing all boundary defining edges, and all interior points are inserted and triangulated using the Bowyer/Watson algorithm. Due to the overlapping mesh construction of the point-set, coincidentally close mesh points may be produced, and mesh smoothing is employed as a post-processing operation in order to relieve effects caused by local irregularities in the final mesh spacing, as seen in Figure 14. Mesh-point filtering techniques may also been employed to restore smooth mesh-point distributions [37]. These techniques have also been employed for three-dimensional mesh generation about aircraft-type configurations by Baker [7, 38]. Although predetermined mesh-point set techniques have been very successful for certain classes of geometries, the simultaneous mesh-point generation and triangulation techniques embodied in Steiner triangulations offer improved flexibility and automation for arbitrary geometries. A Steiner triangulation is a triangulation whose additional points are inserted in order to improve the quality

of the triangulation. Assuming an initial triangulation and an element-size distribution function exist, the triangulation can be incrementally modified by continually inserting new points until the final mesh closely matches the prescribed element-size distribution function in all regions of the domain. A particularly effective strategy is to insert the new mesh points at the circumcenters of the triangles which are flagged for improvement [39, 40, 41, 9, 25].

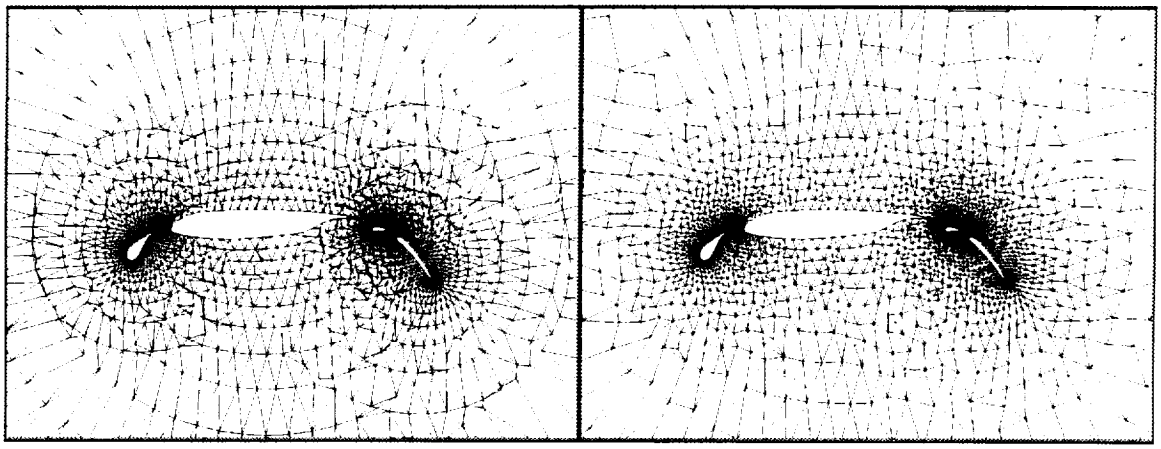

Figure 14: Delaunay triangulation of mesh-point distribution generated from overlapping structured meshes, before and after application of smoothing.

This strategy can be proved to result in triangulations where the angles are all bounded between 30° and 120°, neglecting boundary effects [16, 42]. This result is a consequence of the fact that the circumcenter, where the new point is placed, is equidistant from the three vertices of its forming triangle, which represent the closest points to the new point, since the circle is necessarily empty by the Delaunay criterion. An effective mesh generation strategy using this approach can be formulated as follows:

Step 1: Construct an element-size distribution function $s = f(x, y, z)$.
Step 2: Discretize all boundary curves based on the above function.
Step 3: Construct an initial triangulation which covers the entire domain (usually by forced triangulation of a large quadrilateral in 2D or hexahedron in 3D).
Step 4: Insert all the boundary points into the triangulation using the Bowyer/Watson or Green-Sibson algorithm.
Step 5: Construct a heap-list of all triangles in the mesh which are larger than the local value specified by the element-size distribution function (usually all existing triangles at this initial stage); the heap-list is ordered by some measure of triangle size (*i.e.*, circumradius) or quality (maximum angle).
Step 6: Pull the first triangle off the top of the heap-list, insert a new point at its circumcenter and retriangulate using the Bowyer/Watson or Green-Sibson algorithm. For each of the newly formed triangles, which are larger than the local value specified by the element-size distribution function, insert them into the heap-list.
Step 7: If the heap-list is empty, stop, else go to step 6.
Step 8: Recover the boundary integrity.

The heap-list empties out when the triangulation converges to the element-size distribution prescribed by the function $s = f(x, y, z)$. This method results in a very efficient mesh generation

technique. Since the identity of the originating triangle for the circumcenter point-insertion operation is initially known, the search for the containing triangle in the Delaunay algorithm is initiated with this triangle. In most cases, this corresponds to the containing triangle, and in all other cases, the containing triangle is only several neighbors away. Thus, the union of intersected triangles in Watson's algorithm can usually be found very rapidly, in constant time. Since the heap-list operations (insert/delete) can all be performed in $0(logN)$ time, the space and time requirements of this algorithm are $0(NlogN)$. In practice, this method has produced some of the most efficient mesh generation codes available today, capable of generating 1 million three-dimensional elements per hour on present-day workstations [43].

In the above description of the algorithm, the boundary recovery procedure is performed as a final step, after all mesh points have been inserted. In practice, this procedure may be performed either in the final stages of the algorithm, or earlier on, just after the boundary points have been inserted, *i.e.*, between steps 4 and 5. Recovering the boundary integrity in the final stages of the algorithm enables the boundaries to be neglected and the domain to be treated as convex, during the point-insertion phase. This permits a straight-forward application of the Delaunay insertion algorithm at this stage. Furthermore, mesh quality is much higher at the end of the point-insertion procedure, which may result in a more robust boundary recovery phase. On the other hand, new points may be introduced arbitrarily close to the eventual boundary surfaces, during the point-insertion process, which may result in a poor quality mesh after the boundary recovery operation.

If the boundary recovery is performed just after the insertion of the boundary points, interior mesh points near the boundary may be positioned more optimally (see for example section 3.5). However, this requires boundary integrity be maintained subsequently throughout the remainder of the grid generation process, and implies the application of Delaunay point-insertion algorithms in non-convex domains for step 6. In two dimensions, the existence of a constrained Delaunay triangulation guarantees that the point-insertion algorithms can be extended by terminating the search for intersected triangles in the given direction, when a boundary is encountered. In three dimensions, although similar techniques have been employed, there is no guarantee that a valid triangulation will result, since no three-dimensional definition of a constrained Delaunay triangulation is available.

One way of recovering the boundary integrity, either at the end of the mesh generation process, or after the insertion and triangulation of the prescribed boundary points, is to insert a sufficient number of additional boundary points until the Delaunay triangulation conforms to the boundary. A more elegant approach consists of modifying the existing Delaunay triangulation using the edge swapping (in 2D) and face-edge swapping (in 3D) primitives discussed in section 2.2 to produce a locally modified triangulation which conforms to the boundary. This approach also offers the possibility of matching exactly a predetermined boundary discretization. In two dimensions, each edge of the prescribed boundary discretization is searched for, in the Delaunay mesh. If one of these edges is not found, but the two endpoints of the edge are located in neighboring triangles, then a simple edge swap, as depicted in Figure 6, is sufficient to recover the missing edge. In the event these endpoints are located in non-neighboring triangles, then all triangles which the missing edge intersects are first located. These triangles are all removed, and the missing edge is inserted, as shown in Figure 15. This creates two polygons, on either side of the new edge, which are then triangulated, using, for example, the Tanemura-Merriam algorithm.

The situation in three dimensions is somewhat more complex. The procedure is usually divided into two phases for a prescribed boundary surface triangulation: firstly the recovery of the boundary edges, and secondly the recovery of the boundary faces. The face-edge swapping primitives of section 2.2 are utilized to transform the Delaunay triangulation into one which contains the boundary edges and faces. However, contrary to the two-dimensional case, the insertion of additional points is often required to enable the completion of the boundary recovery procedure in three dimensions.

Weatherill [40] has categorized all possible boundary edge and face intersection types, and lists rules for inserting new points to enable recovery of these boundary elements. Generally speaking, he advocates the creation of new points on the surface, at the point of intersection with the existing grid element. Once all boundary edges and faces have been recovered, the new surface points may be removed in an attempt to recover the original surface discretization. However, in some cases, the removal of these point may result in cavities which cannot be re-tetrahedralized.

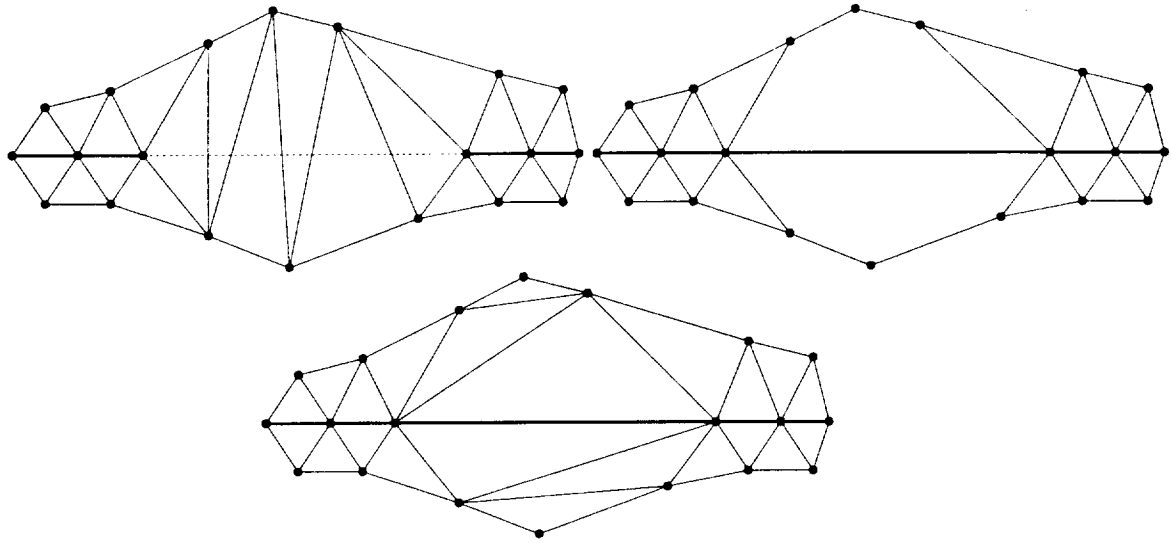

Figure 15: Illustration of the insertion of a missing boundary edge into an existing Delaunay triangulation, and the resulting empty polygons which must be retriangulated.

In [9], a similar method which introduces new points off the surface is developed, thus permitting the recovery of the original surface discretization.

The mesh-point distribution produced by the circumcenter point-placement strategies, while certainly acceptable, are somewhat irregular and lack the high degree of smoothness produced by the advancing-front method. This is perhaps due to the fact that Steiner triangulations are top-down approaches, which seek to improve existing triangulations through refinement. A more serious difficulty with Delaunay point-insertion methods relates to robustness problems due to round-off error. Particularly in the initial phases, when the boundary points are inserted, highly distorted triangular/tetrahedral elements are formed. The finite precision real arithmetic utilized to compute all quantities (*i.e.*, cell volume, circumcenter, etc.) may lead to failure of the algorithm simply due to insufficient accuracy (even in 64 bit arithmetic). One remedy is to alter the order in which the points are inserted, or to employ auxiliary (possibly temporary) points to avoid the creation of highly skewed elements. However, this may also have the effect of modifying the resulting triangulation. Ultimately, the best technique for ensuring robustness is to resort to infinite precision arithmetic using integer programming techniques [30], perhaps in combination with some of the above strategies.

3.4 Advancing-Front Delaunay Triangulation

In an effort to alleviate some of the drawbacks of point-insertion Delaunay triangulation methods, an advancing-front Delaunay triangulation algorithm was recently developed by the author [29]. This method enjoys the smooth point distribution and guaranteed boundary integrity of the traditional advancing-front methods, while using the well founded principles of Delaunay triangulation to replace the heuristics present in the reconnection phase of traditional advancing-front methods.

Robustness without the use of high accuracy arithmetic is assured by only constructing triangular elements which are acceptable in size and shape, as determined by the background spacing function, *i.e.*, the highly skewed triangles which appear temporarily in the point-insertion methods are never formed.

A background spacing function $s = f(x, y)$ is used to determine the maximum permissible circumradius ρ of a triangle as a function of spatial location. Each time 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 local 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 front points which are less than 2ρ away from the new point. The advancing-front Delaunay triangulation algorithm for fixed point sets (Tanemura-Merriam) described in section 2.1.3 [14, 15] can then be employed to construct the Delaunay triangles between these front points and the new point. As in the traditional advancing-front algorithm, it is possible that a new triangle may be formed from the current front edge with an existing front point. Such cases are automatically detected by the Tanemura-Merriam algorithm, and the new point is rejected. Finally, when a new point is introduced, it is possible that it intersects circumcircles of existing triangles. If this is the case, these triangles must be deleted prior to the formation of new triangles, since they are in violation of the Delaunay criterion. The method is set up similarly to the traditional advancing-front procedure. A background spacing function is defined, and the boundaries are discretized and constitute the initial front. The basic algorithm can be summarized as follows:

Step 1: Construct the original front as a set of boundary edges.
Step 2: Choose a particular edge from the front, according to some criteria such as minimum edge length.
Step 3: Locate all front points which are less than 2ρ away from either end point of this edge, where ρ is the local circumradius bound determined from the field function.
Step 4: Use the Tanemura-Merriam algorithm to determine the Delaunay triangle formed between this edge and the set of candidate points, if such a triangle exists.
Step 5: If this triangle exists and is acceptable (circumradius smaller than ρ), form a new triangle, update the front, and proceed to step 12. Otherwise create a new point at the appropriate location.
Step 6: Determine all front triangles whose circumcircles are intersected by the new point.
Step 7: Using a neighbor search initiated at the intersected front triangles, locate all interior triangles whose circumcircles are intersected by the new point.
Step 8: 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.
Step 9: If the circumradius of any of the new 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 than 2ρ away from the new point.
Step 10: 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 Tanemura-Merriam algorithm.
Step 11: Add these triangles to the mesh and update the front.
Step 12: If the front queue is empty, stop, otherwise go to step 2.

As in previous advancing-front algorithms, the front is stored as a heap-list in order to facilitate the choice in step 2. A front-based quadtree is also used to perform the search for "close" points in step 3. A second quadtree is also employed in the search for intersected front triangle circumcircles. (The construction of quadtrees for geometric entities other than points, such as circles, is described in [4]).

In most cases when a new point is inserted ahead of the front, no intersected circumcircles are detected, and few if any other front points are within the critical 2ρ distance of the new point, thus the construction of new triangles is quite simple. When fronts merge upon each other, the situation becomes more complex. Triangle circumcircles from the opposing front are usually intersected. This alerts the front being advanced to the proximity of the neighboring front and provides an extra length scale (the circumradius of the intersected triangle) which is utilized effectively in mending the two fronts as shown in Figure 16.

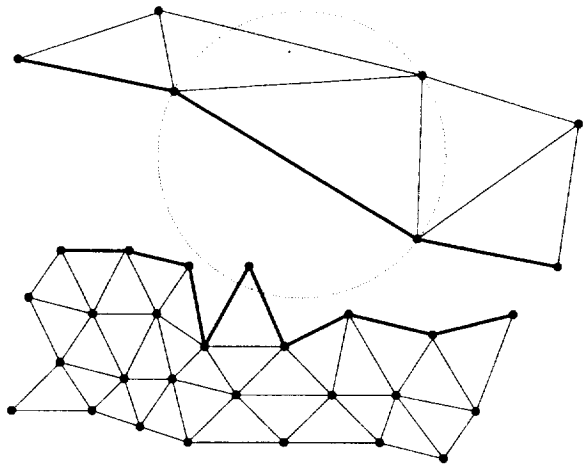

Figure 16: Illustration of the use of the front triangle circumcircle to detect the proximity of a front. (compare with the situation depicted in Figure 13).

The Delaunay construction naturally results in additional information which is essential for merging two fronts of dissimilar length scales. This is precisely the information which is missing in the traditional advancing-front algorithms, which are prone to failure when fronts of dissimilar scales are encountered. Although a smoothly varying background function can reduce the likelihood of having to merge dissimilar fronts, this cannot be guaranteed. The extra information inherent in the Delaunay-based algorithm permits the merging of arbitrarily dissimilar fronts, thus increasing robustness.

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 [2], *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 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 $logN$ term from the quad-tree data-structures employed for the search routines on the front is included, a complexity of $O(NlogN)$ 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 triangle circumcircles). 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.

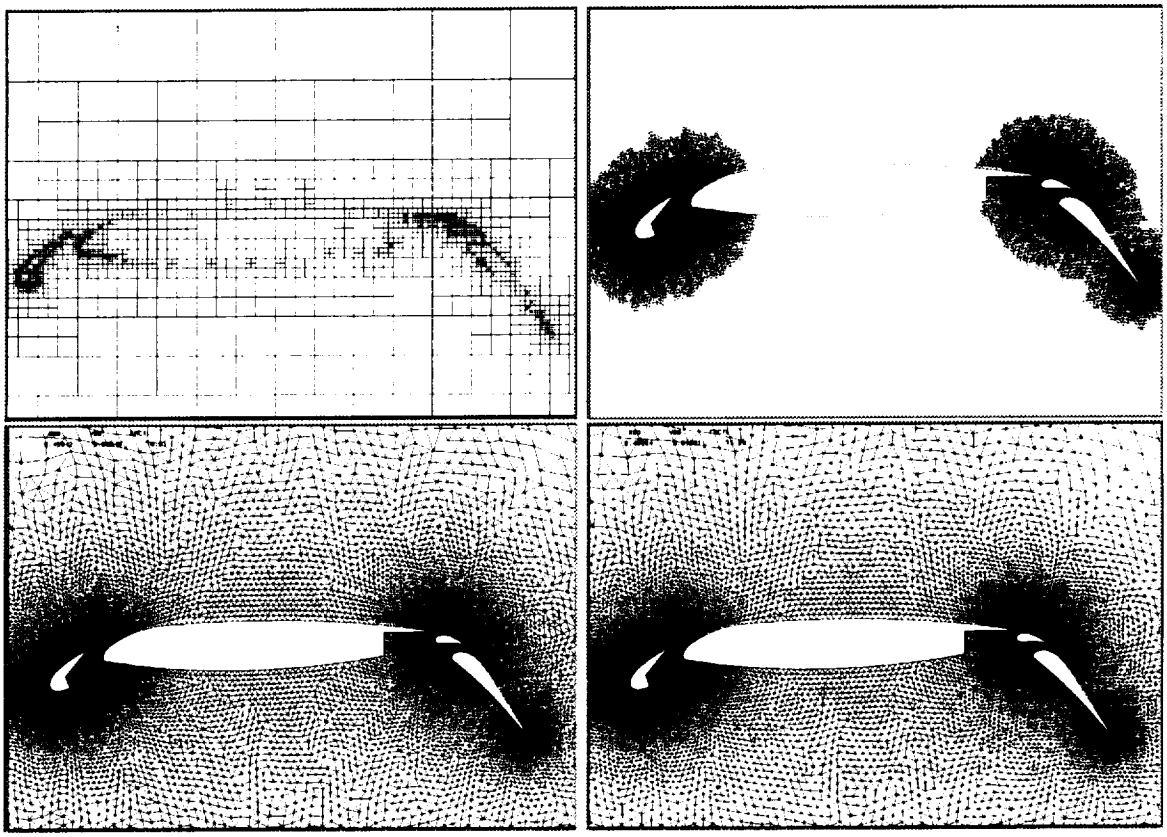

Figure 17: Illustration of advancing-front Delaunay triangulation procedure. (i) Quadtree of initial boundary discretization. (ii) Grid at intermediate stage of generation. (iii) Final generated mesh prior to the application of smoothing. (iii) Final smoothed mesh.

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. An example of the mesh generation process using the advancing-front Delaunay triangulation algorithm is depicted in Figure 17. The initial front quadtree is shown in the first part of the figure. Two copies of this quadtree are kept. The first is static and forms the supporting structure for the background spacing function. In order to evaluate this function at a given spatial location $s = f(x, y)$, the quadtree is descended to the leaf quad which contains the particular (x, y) location, and the function value is obtained by bilinear interpolation of the values at the four quad vertices. The second copy of this quadtree is dynamic, and is advanced along with the front throughout the mesh generation process. This quadtree is employed for the spatial searches required by the algorithm. The second part of the figure illustrates an intermediate stage of the mesh generation process, while the final mesh is depicted in the third part of the figure, and the smoothed mesh in the fourth part of the figure. It is noteworthy that the final mesh generated in this fashion exhibits a very regular distribution of elements, even before the application of post-processing techniques. (Compare this with the first part of Figure 14, for example).

3.5 Advancing-Front Point-Insertion Methods

Another option for combining the advantages of Delaunay triangulation with those of advancing-front techniques is to employ the point-insertion algorithms (e.g. Bowyer/Watson or Green-Sibson) in conjunction with an advancing-front type point-placement strategy. The advantages of such methods are the smooth point distributions associated with advancing-front methods, coupled with the efficiency and ease of implementation of the Delaunay point-insertion methods. The disadvantages are the same as those described for the circumcircle point-insertion methods of section 3.3, *i.e.*, the application of boundary recovery techniques, and possible robustness problems associated with the use of finite-precision arithmetic, due to the presence of highly skewed elements at intermediate stages of the mesh generation process. However, these methods are considerably simpler to implement than the algorithm of the previous section. They extend readily to three dimensions, and can also be used to generate triangulations other than the Delaunay triangulation, as will be shown. Such methods have been described by Rebay [44], and Mueller et al. [45], and later taken up by Marcum et al. [43, 46]. Assuming an element-size distribution function has been defined, and is used to control the maximum permissible circumradius of a triangle as a function of spatial location, Rebay [44] begins by classifying all triangles of the initial triangulation as either accepted, active, or waiting. An accepted triangle is one that satisfies the circumradius-bound defined by the field function, while a waiting triangle is one that does not. Active triangles are waiting triangles which are neighbors of accepted triangles. Active triangles are the only triangles considered for refinement. Thus, the set of edges which delimits the border between accepted and waiting triangles constitutes the front, and active triangles are simply triangles which have an edge on the front. Mueller et al. [45] and Marcum [46] have devised point placement strategies which mimic hyperbolic structured mesh generation techniques. At each stage, the entire front is considered, and a new set of points is created ahead of the front, similarly to a new layer of points in a structured hyperbolic mesh scheme. These points are then filtered, in order to remove points which may be too close together, such as in concave regions of the front, and the filtered point-set is inserted into the mesh sequentially, using the Delaunay point-insertion methods. A more traditional technique for advancing the front by using a heap-list, which always chooses the smallest front-edge, has also been found to work well. A particular implementation of these algorithms, as performed by the author is given below:

Step 1: Define a field-spacing function and discretize the boundaries accordingly.
Step 2: Create an initial triangulation which covers the entire domain (using auxiliary points)
Step 3: Insert all boundary points into the triangulation using a Delaunay point-insertion algorithm (Bowyer/Watson or Green-Sibson).
Step 4: Recover the boundary integrity using edge-swapping techniques (or edge-face swapping in 3D)
Step 5: Flag all triangles outside of the physical domain as accepted, and all triangles inside the domain which violate the local circumradius-bound as waiting (usually all initial triangles will be in violation).
Step 6: The set of edges which delimits accepted and waiting triangles forms the current front (which coincides with the boundaries at this stage). Construct a heap-list of these edges ordered by smallest edge-length.
Step 7: Pick the smallest front edge from the top of the heap-list, and determine the position of the new point ahead of this edge, based on the local value of the field function.
Step 8: Insert this point into the triangulation using the Delaunay point-insertion algorithm.
Step 9: Reclassify newly formed triangles as accepted and waiting, based on the local circumradius bound, and update the front accordingly.
Step 10: If the front is not empty, go to step 7, else stop.

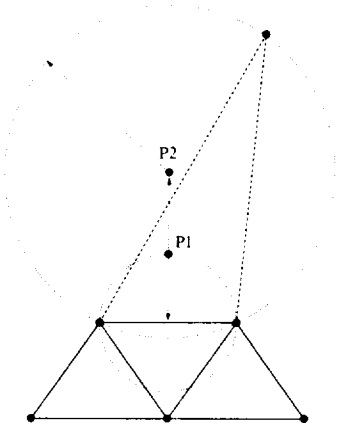

Figure 18: Illustration of point-placement strategy for advancing-front Delaunay point-insertion method.

The point-placement strategy is illustrated in Figure 18. The new point is placed along the median of the current front-edge, at a distance which results in a triangle of circumradius ρ, as prescribed by the field function. 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, (since this is the smallest possible circumcircle which can be formed with a triangle consisting of this edge and a third point), and at the other extreme by the location of the circumcenter of the current waiting triangle formed with this edge. This extreme corresponds to a circumcenter point-insertion strategy, as described in section 3.3, which ensures the new point will not fall coincidentally close to an existing mesh point. Figure 19 illustrates the mesh generation process at an intermediate stage for the same configuration shown in the previous section. The front is clearly visible, and the identity of the accepted and waiting triangles is fairly evident. This algorithm uses the same point-placement strategy as the algorithm of the previous section, and thus should yield the identical mesh as the advancing-front Delaunay procedure of the previous section, provided the same background

spacing function is specified (not the case in this example). The main difference between these two algorithms is that, in the former case, the waiting triangles which are most often highly skewed, are never formed, thus leaving an ungridded gap region between the fronts. This results in the need to employ quadtree data-structures to perform the search for nearby front points and circumcircles. In the present method, the domain is always entirely covered by a triangulation, which is utilized in an efficient manner for supporting all spatial searching operations (implemented as neighbor walks).

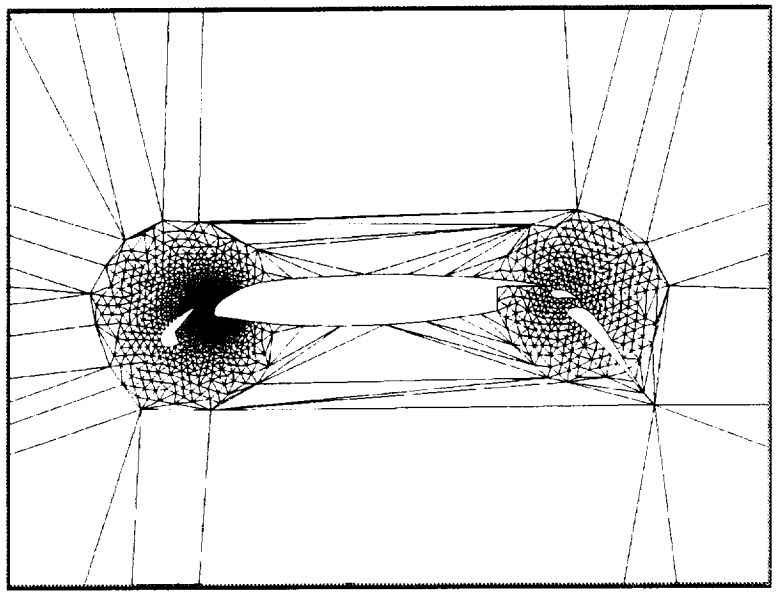

Figure 19: Partially completed advancing-front Delaunay point-insertion mesh.

A notable feature of the present methodology is that the boundary recovery procedure must be performed immediately after the insertion of the boundary points, and cannot be performed at the end of the mesh generation operation, after all interior points have been added, as discussed in section 3.3. This is due to the fact that the boundary discretization constitutes the initial front, the location of which is required to guide the placement of new points. Once the boundary discretization has been recovered, it must be maintained throughout the insertion of all interior mesh points. While in two dimensions this is easily achieved by modifying the point-insertion algorithms according to the principles of constrained Delaunay triangulations, the lack of any such notion in three dimensions makes the situation more difficult. Indeed, the initial mesh with recovered boundaries in general will not represent a Delaunay triangulation, and may lead to failure of the Delaunay-based point-insertion algorithms.

An alternative technique [25, 46] circumvents this issue. The approach is based on relaxing the constraint that the final mesh be as close as possible to Delaunay, and relies on variants of the Green-Sibson point-insertion algorithm. The basic mesh generation procedure is similar to that described above, except that in step 8, a truncated Green-Sibson algorithm is employed to insert new field points. The new point is inserted by joining it to the four vertices of its enclosing tetrahedron (or five vertices of two neighboring tetrahedra in the event the point coincides with a face, with a similar extension for a point coinciding with an edge), and then swapping faces and edges of the newly formed and subsequently modified elements according to the in-circle criterion, until a Delaunay mesh is recovered, or the process can no longer "improve" the mesh. Boundary integrity is easily enforced in this manner, since face-edge swaps which alter the boundary discretization are simply prohibited. While the Bowyer/Watson algorithm relies on the Delaunay property of the mesh in order to guarantee the convexity of the reconstructed region and the validity of the

new triangulation, the Green-Sibson algorithm can still be employed with initial constructions other than Delaunay triangulations. In such cases it reduces to a local optimization technique which attempts to approximate the Delaunay triangulation. By modifying the face-edge swapping criterion, the algorithm can be employed to construct other types of triangulations, such as min-max triangulations. (However, forward as well as backward propagation of the swapping tests must be implemented in such cases as described in section 2.1.2). In fact, Delaunay triangulations may not be the optimal construction for three-dimensional mesh generation problems, in spite of all the elegant properties associated with such triangulations. Since it is possible to form a sliver tetrahedron with vanishingly small volume for which the circumsphere is not excessively large (*i.e.*, imagine four coplanar points on the sphere), Delaunay triangulations in three dimensions often admit such ill-behaved elements. This suggests that min-max triangulations may be better suited for three-dimensional grid generation purposes.

On the other hand, the use of Green-Sibson type point-insertion local-reconnection algorithms for generating min-max triangulations suffers from the inability to recover the globally optimum min-max triangulation. In fact, the algorithm typically gets caught very quickly in a local optimum and results in poor quality meshes. A key feature which enables a closer approximation of the global optimum and yields higher quality meshes, is the use of an intermediate Delaunay in-circle swapping test, as demonstrated in [43, 46]. In this approach, each time a new point is inserted, the surrounding faces and edges are first swapped according to the traditional Delaunay in-circle test, and then re-swapped according to the min-max criterion. The use of this intermediate pseudo-Delaunay construction serves to broaden the range of influence of the new point, thus providing a more global effect, and avoiding the local optima which typically plague the straight application of the min-max criterion.

4 Adaptive Meshing

Aside from the treatment of complex geometries, the second main advantage of unstructured meshes is the ease with which solution-adaptive meshing may be implemented. Since no inherent structure is assumed in the representation of unstructured meshes, mesh refinement and coarsening may be performed arbitrarily in any region of the mesh. The principle idea of adaptive meshing is, of course, to enable a higher accuracy solution at lower cost, through a more optimal distribution of grid points for each computed solution. The entire procedure is akin to a control problem, where the output (the solution) drives the machinery which generates the solution itself. The basic steps in an adaptive meshing solution strategy are:

- Computation of initial solution.
- Estimation of local error in solution.
- Modification of mesh according to estimated error values.
- Initialization of solution on adapted mesh.
- Resumption of numerical solution procedure.

A complete adaptive solution package must therefore include a flow solution module, an error estimation technique, and a grid adaptation (refinement-derefinement) module, as well as an accurate representation of the actual configuration geometry. Thus, the construction of an adaptive solution methodology represents a large investment in software development, particularly in three dimensions.

The need to store geometry data is dictated by the requirements of placing new adaptively introduced boundary points on the curves or surfaces which define the original geometry, rather than simply on the coarse grid approximation to this geometry, as shown in Figure 20.

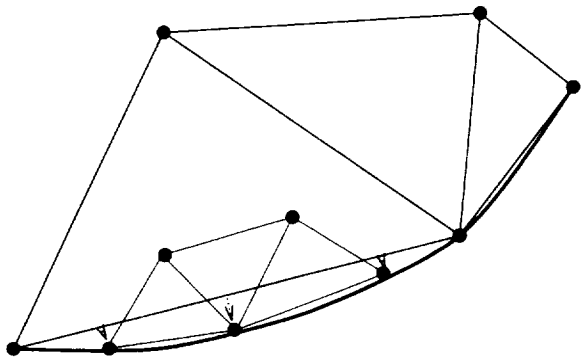

Figure 20: Illustration of fine and coarse grid discretizations of curved boundary.

The extraction and reduction of this data from its defining source (usually a CAD data-package) may prove to be a difficult task, particularly for complex three-dimensional configurations. Another possibility is to encode the geometry as a very fine set of points (finer than the highest expected mesh resolution). This seemingly primitive technique may be making a comeback, particularly with the availability of rapid high-resolution geometry scanning devices.

The actual mechanics for modifying the mesh, *i.e.*, adding, removing, and displacing mesh points, as well as reconnecting them, is usually the topic which receives the most elaboration in adaptive meshing papers. It is also perhaps the best understood of all the required elements. Most adaptive meshing techniques can be implemented as extensions or enhancements to known mesh generation strategies. The formulation of an effective mesh refinement criterion, on the other hand, is a difficult task which has not been adequately resolved to date. The main problem is that an exact characterization of the error requires a knowledge of the solution itself, which is obviously impractical. While finite-element error estimates have been developed for simple elliptic problems, the difficulty is compounded for fluid dynamics problems by the fact that the governing equations represent a coupled system of non-linear hyperbolic partial-differential equations. A true characterization of the error would necessarily require information from all flow variables. Estimates which rely on the smoothness properties of the solution break down in regions near discontinuities such as at shocks. Furthermore, all error estimates rely on the fact that the computed solution is asymptotically close to the exact solution. The non-linear and hyperbolic character of the governing equations may, however, result in situations where this assumption does not hold, at least locally. Consider for example a separation bubble which only appears when certain upstream flow features have been adequately resolved, or the diffusion of a wake profile due to inadequate upstream resolution. In such cases, the downstream solution is in no way asymptotically close to the exact solution in these regions.

The inadequacy of current error estimators, coupled with the fact that the computed solution may be far removed from the exact solution, are the main reasons why adaptive meshing cannot be utilized in lieu of mesh generation itself. In other words, the idea of initiating the calculation with an extremely coarse grid (possibly just fine enough to resolve the essential geometry topology) and relying on adaptive meshing to generate the final mesh, is impractical since many flow features will never be captured on such a mesh. The generation of a suitable initial mesh, with good resolution in regions of expected solution activity, is essential for good overall adaptive solution performance.

4.1 Refinement Criteria

The most popular refinement criteria for fluid-flow problems are essentially heuristically derived gradient-based criteria, which involve a single or multiple physical flow variables. As an example, the gradient of pressure can be used to identify inviscid flow features, while the gradient of velocity can be used to track shear layers. In actual fact, it is the undivided gradient which is employed:

$$\epsilon = \frac{\partial u}{\partial x} \cdot h \tag{2}$$

or, discretely:

$$\epsilon = \Delta u \tag{3}$$

The use of an undivided gradient ensures that the value of ϵ, which should approximate the error, decreases as the mesh size is reduced. Since second-order methods, which are typically employed, can represent linear solutions exactly, an undivided second difference may be expected to yield a better refinement criteria:

$$\epsilon = \frac{\partial^2 u}{\partial x^2} \cdot h \tag{4}$$

Both first and second difference based refinement criteria have been tested and employed in the literature. However, the use of first differences is more conservative (*i.e.*, produces more refinement) and is often found to work better in practice. Lohner [47] advocates the use of a non-dimensional criterion which is designed to assign equal weight to weak as well as strong flow features. This is achieved by forming the ratio of second and first derivatives. The refinement indicator can be written as:

$$\epsilon = \frac{|\frac{\partial^2 u}{\partial x^2}|}{h|\frac{\partial u}{\partial x}| + \alpha \overline{u}} \tag{5}$$

where $\overline{u}$ represents some neighborhood average of the flow variable u, and α is a small parameter. This extra term in the denominator acts as a "noise" filter which avoids triggering refinement in regions of small solution oscillations. This type of refinement criterion has been used extensively for two- and three-dimensional transient flow solutions involving shock waves.

An interesting study of the effectiveness of various refinement criteria for steady-state problems can be found in the paper by Warren et al. [48]. The error levels in various adapted mesh solutions using different refinement criteria were assessed by comparing the adapted mesh solutions to the "exact" solution (*i.e.*, a solution computed on a highly resolved, globally refined mesh). Their experiments indicate that the modification of the undivided differences in equation (3) to include a local mesh length-scale such as:

$$\epsilon = \Delta u \cdot \Delta x \tag{6}$$

produces a more effective refinement criterion. This is partially due to the fact that while the simple undivided difference form of equation (3) decreases in magnitude as the mesh is refined in smooth regions of flow, it remains approximately constant in the vicinity of shock waves, since the shock wave profile steepens as the mesh is refined, and the jumps remain almost constant. However, even

in regions of smooth flow, the additional length scale weights large cells more heavily than small cells, and drives the adaptation process closer towards global refinement. The introduction of an extra length scale thus produces a more conservative criterion (*i.e.*, results in more refinement) and thus greater reduction of the error can be expected. This asymptotic tendency towards global refinement is actually the behavior which is desired for steady-state problems. While adaptive meshing is typically thought of as a process that refines only local regions of the mesh, this is characteristic of the initial stages of an adaptive mesh procedure, where the appropriate distribution of mesh resolution is set up to match the solution. Once this has been achieved, and the error is presumably equipartitioned throughout the domain, any increase in solution accuracy can only come about from global refinement. Refinement criteria must therefore be able to reproduce this behavior.

Although gradient-based refinement criteria have been employed successfully in computational fluid dynamics, such techniques are not well founded and are far from optimal. The effectiveness of a refinement criterion is measured in terms of accuracy delivered for the number of mesh points required. In the above discussion, the more conservative gradient-based criteria are obviously the most successful at reducing the solution error, but also result in excessive refinement, and are not necessarily the most efficient. The construction of more optimal refinement criteria will require the development of better error estimation techniques most likely through the use of some type of extrapolation method [49].

4.2 Mesh Adaptation Techniques

Mesh adaptation may involve the addition of extra vertices, the removal of vertices, the redistribution of existing vertices and the reconnection of mesh vertices. The character of the problem to be solved dictates the requirements of the mesh adaptation strategy. For example, steady-state problems usually involve a small number of adaptation phases as part of a lengthy solution process. Therefore, relatively sophisticated (*i.e.*, more optimal) adaptation strategies can be employed, such as, in the extreme case, complete mesh regeneration. Mesh refinement procedures are most important here, while de-refinement has only a minor effect and can often be omitted for steady-state cases. For transient problems, mesh adaptation must be performed every several time-steps, and thus efficiency is much more important than optimality. Mesh refinement and de-refinement are both essential for transient cases. Furthermore, the accuracy of interpolation from the original mesh to the refined mesh affects the solution accuracy (unlike the steady-state case), and thus accurate transfer schemes are required. These requirements have often lead to the use of simple element subdivision schemes for transient flows.

For steady-state as well as transient problems, mesh adaptation strategies are generally based on one of the previously discussed mesh generation procedures, and can be implemented as an extension of the originating procedure.

4.2.1 Delaunay Point-Insertion Based on Solution Gradients

The Bowyer/Watson and Green-Sibson algorithms described in section 2.1 enable the introduction of new points in any regions of an existing mesh, and are thus natural candidates for mesh adaptation algorithms. A particular implementation of an adaptive mesh solution strategy using the Bowyer/Watson algorithm performed by the author [50] is described below. Assuming an initial Delaunay triangulation mesh has been constructed, the flow solution is computed on this mesh. Using the undivided gradient of density as a refinement criterion (*c.f.* equation (3)), each edge of the mesh is then examined and flagged for refinement if the difference of density along the edge is larger than some threshold value, which is set proportional to the average of all density differences taken over all mesh edges. This simple edge-based refinement strategy tends to produce non-isotropic point distributions which triangulate poorly. Therefore, a more isotropic refinement

strategy is desired. This is achieved by effecting a single loop over the mesh triangles and flagging the remaining edges for refinement of any triangle which has a single or two refined edges. Note that this operation requires the use of a cell-to-edge data-structure, in addition to the cell-to-node data-structure employed by the Delaunay triangulation algorithm, and the edge-to-node data-structure employed by the flow solution algorithm.

For each flagged edge, a new point is created at the midpoint of the edge. For boundary edges, these points are repositioned onto the spline curve which defines the original geometry, as depicted in Figure 20. The new points are put in a list, and then inserted and triangulated sequentially into the mesh using the Bowyer/Watson algorithm. The new refined mesh is smoothed by repositioning the points according to a Laplacian filtering technique. The solution is then interpolated in a piecewise linear manner from the original mesh to the new finer mesh (using the interpolation operators constructed for the multigrid algorithm as described in the next chapter) and the flow is solved on the new mesh. The entire procedure is repeated several times, until the desired accuracy (or grid size) is attained.

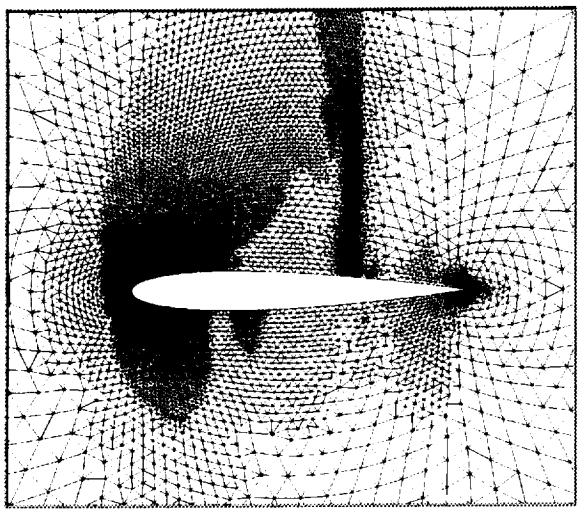

Figure 21: Adaptively generated mesh for NACA 0012 airfoil.

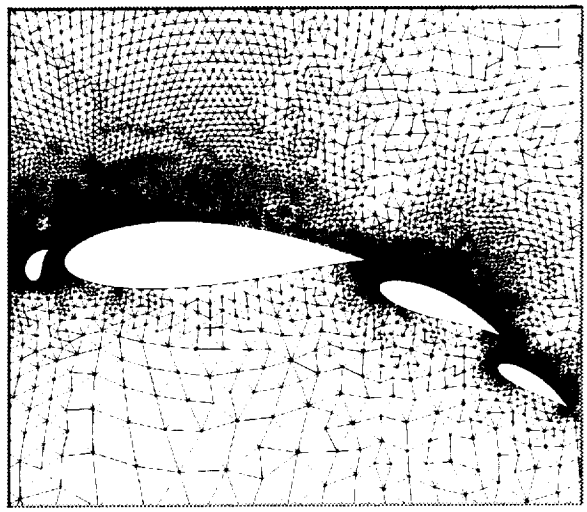

Figure 22: Adaptively generated mesh for idealized four element airfoil geometry.

Figure 21 illustrates the final mesh obtained by this procedure, after three adaptive refinement passes, for the computation of transonic flow over a NACA 0012 airfoil. The flow features are rather well defined in this case, and the various levels of refinement are evident in the final mesh. Figure 22 illustrates an adaptive mesh obtained for the computation of inviscid subsonic flow over a multi-element airfoil. Here, the flow is smooth and features are not well defined. The various levels used to construct the adapted mesh are not as evident, and the procedure results in a smooth variation of the mesh in most regions of the domain. This same procedure extends readily to three dimensions. Figure 23 depicts an adapted tetrahedral mesh employed to compute the transonic flow over an ONERA M6 wing. This mesh contains a total of 173,412 points, and was obtained through two applications of adaptive refinement. The double shock structure in the flow solution is clearly reflected in the mesh refinement pattern.

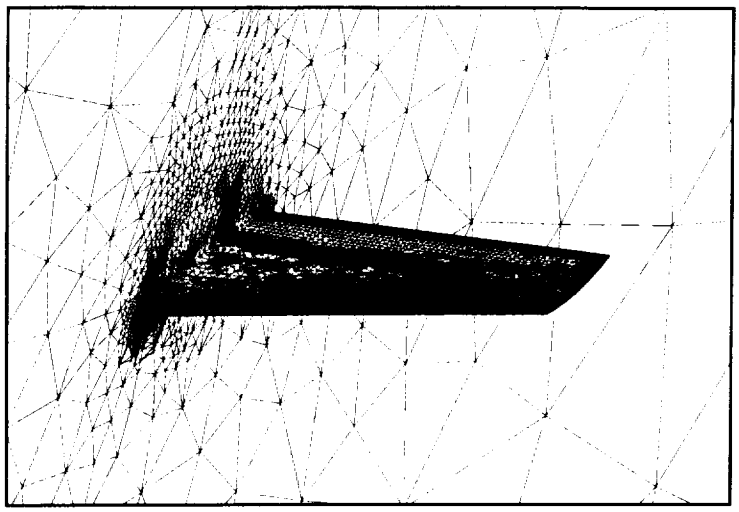

Figure 23: Adaptively generated mesh in three dimensions for ONERA M6 wing.

4.2.2 Extensions to Steiner Triangulation

Any mesh generation technique which relies on the use of a background element-size distribution function can be extended in a natural manner for adaptive meshing problems. Consider the Steiner triangulation methods described in section 3.3, where new points are continually added to the mesh, until elements which satisfy the background size-function are obtained. Once the flow solution has been obtained on the initial mesh, the background function may be modified to reflect the result of the application of a refinement criterion to this solution, (*i.e.*, the background function can be modified to specify small element sizes in regions of high flow gradients or solution error). A new adapted mesh may then be constructed by simply restarting the Steiner triangulation algorithm, and resuming triangulation until convergence to the new background function is achieved. This strategy, of course, only produces additional refinement. In principle, mesh de-refinement may also be achieved by removing points sequentially, and retriangulating the resulting convex cavity for each deleted point, in regions where the mesh element-size is smaller than that prescribed by the background function. This may be achieved, for example, by applying the Tanemura-Merriam Delaunay triangulation algorithm to the vertices of the cavity which is created upon the removal of a point from the current mesh. In three dimensions, however, it is possible to encounter cavities which cannot be tetrahedralized, and a more sophisticated approach may be required. An alternative is to maintain a data-structure which encodes a hierarchical history of the triangulation construction, as suggested by Barth [51].

4.2.3 Adaptive Remeshing

The background element-size distribution function of traditional advancing-front methods may also be modified in a solution adaptive manner. For background functions defined on a (coarse) unstructured grid, one approach is to define the new function on the current mesh slated for refinement [26]. The advancing-front mesh generation software can then be utilized to regenerate a new mesh from scratch, using the newly defined background function to determine the solution-adaptive mesh distribution. A less expensive alternative is to use local remeshing [34]. In this approach, the current mesh is removed in various regions of the domain where the discrepancy between current and desired mesh resolution is large. New local meshes are then generated in

these void regions using the advancing-front method, where the initial front is determined by the boundary between meshed and void regions.

4.2.4 h-Refinement or Subdivision Techniques

h-refinement techniques, which rely on the subdivision of mesh elements according to a predetermined set of primitive operations, are both simple and efficient. These methods result in fine grid elements which are fully nested with their forming coarse grid elements, as well as fine grids which contain, as a subset of their vertices, all the coarse grid vertices. These properties enable a very accurate and efficient transfer of variables from one grid to another, as well as a simple framework for encoding history effects to determine parent-child relationships between cells. This, in turn, enables efficient use of de-refinement techniques. Efficiency, de-refinement, and accuracy of interpolation make these methods ideal for transient problems. The fully nested property of these techniques has also been exploited for constructing unstructured multigrid algorithms based on such meshes (see chapter on multigrid methods).

The main drawback of such methods is the possibility of generating ill-shaped elements and meshes with arbitrarily high connectivity at isolated vertices. In order to avoid such situations, strict rules on the permitted element subdivision types must be enforced. A set of primitives for the subdivision and de-refinement of three-dimensional tetrahedra are given by Lohner [47], and reproduced in Figure 24.

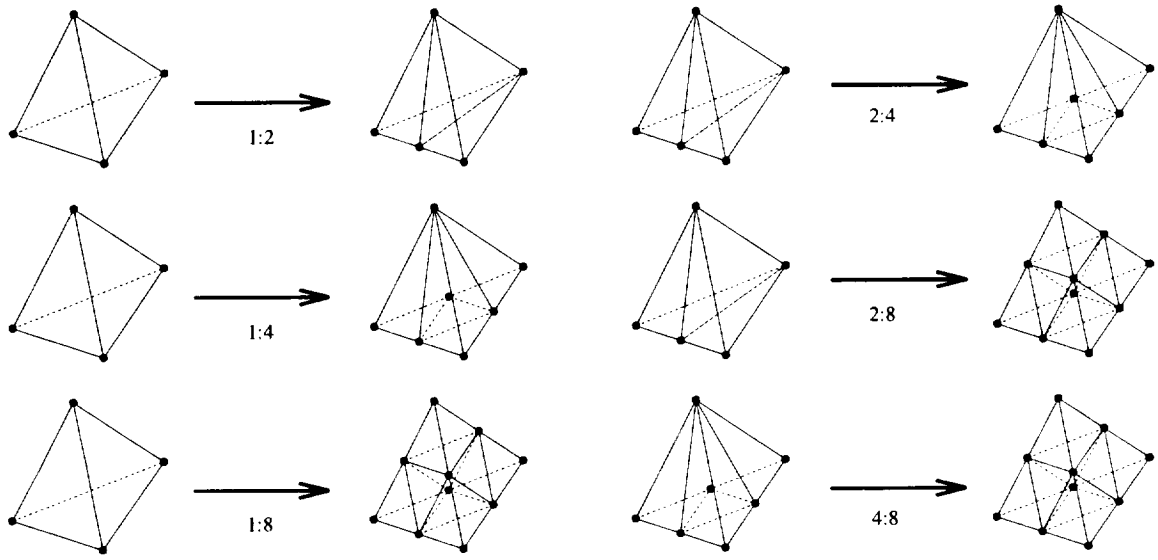

Figure 24: Rules for subdivision refinement of three dimensional tetrahedral meshes.

1:2, 1:4, and 1:8 refinements are permitted. Subsequent refinements of 2:4, 2:8, and 4:8 are also permitted. However, if a 1:2, 1:4, or 2:4 configuration is to be refined further, it must first be transformed to a 1:8 refinement, and then its child elements refined recursively. When applying these rules to a set of elements in a mesh, care must be taken to ensure compatible refinement patterns are obtained on neighboring elements, and to avoid jumps larger than 8:1 between neighboring elements. This is achieved by iteratively modifying the refinement types on groups of mesh elements until a compatible pattern is achieved.

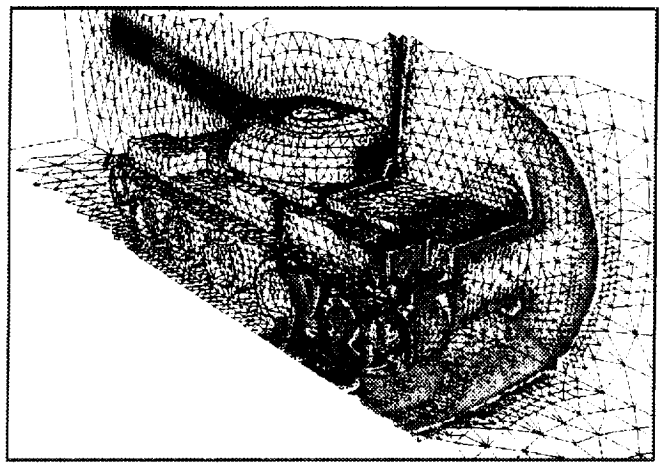

Figure 25: Example of three-dimensional mesh adaptation by subdivision for transient shock-wave problem over an unfriendly configuration. (Reproduced from [52] with permission).

Figure 25 illustrates an example of a transient shock wave problem computed with an adaptive h-refinement technique, taken from [52], using the technique described above. Another method for avoiding ill-shaped elements and poor mesh connectivity is to perform an edge-swapping or face-edge swapping operation in two dimensions or three dimensions, respectively, after the element subdivision operations, in order to improve the mesh topology. While substantial improvement in mesh quality can be obtained, particularly if the strict refinement rules have been relaxed, the nested property of the meshes is lost, and the parent-child history essential for de-refinement becomes more complicated to recover.

4.2.5 Mesh Movement Techniques

Mesh movement techniques involve the movement of mesh vertices in order to improve the discretization accuracy, by drawing points towards regions of high solution error, and away from regions of excessive resolution. These are often performed without altering the mesh connectivity, which makes then particularly simple to implement. While they have proved useful for structured meshes (where the mesh structure must be preserved), they are rarely used in this form for unstructured meshes, since mesh refinement is easily implemented. They are most often employed in conjunction with other adaptive meshing techniques as a smoother, in order to improve local grid quality. A simple construction of Laplacian-type smoothing operations for two-dimensional meshes is given by updating the coordinates of each mesh point as:

$$x^{new} = x^{old} + \frac{1}{N}\sum_{k=1}^{N}(x^{old} - x_k)$$
$$y^{new} = y^{old} + \frac{1}{N}\sum_{k=1}^{N}(y^{old} - y_k) \qquad (7)$$

where the summation is over all the neighboring vertices of the considered point. This procedure mimics a Jacobi iteration for a Laplacian operator on the coordinates of the grid points, and can be applied iteratively for all grid points. This particular formulation does not exclude the possibility of forming negative area elements. Although formulations have been developed for excluding such possibilities, they are often expensive and difficult to solve. A simpler approach is to unsmooth grid points in regions where negative elements are created. This approach, combined with the use

of face-edge swapping for improving grid connectivity, can substantially improve mesh quality (*c.f.* Figure 14).

While mesh movement methods have seldom been employed as a solution-adaptive meshing procedure for unstructured meshes, a notable exception is the use of these techniques for shock fitting purposes [53, 54]. If the mesh points of a triangular mesh are displaced in such a manner as to align the mesh edges with the shocks in the flow, then it is possible to capture the shock over a single mesh cell interface using, for example, a cell-centered scheme with a Rieman interface solver.

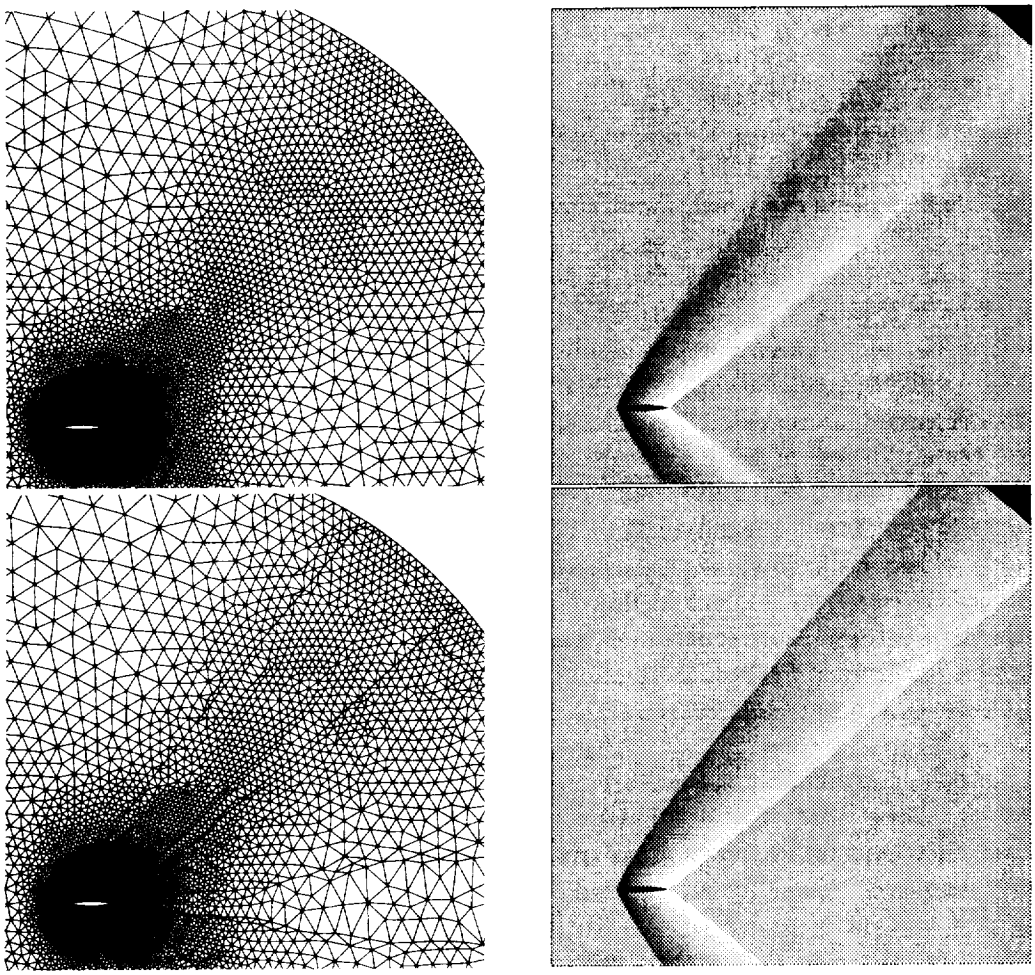

Figure 26: Original mesh, adapted mesh, and corresponding solutions, using mesh-movement shock fitting approach. (Reproduced from [54] with permission).

Figure 26 illustrates the use of this technique for capturing a bow and fishtail shock about a supersonic airfoil. Not only can very weak shocks be well captured, but the required number of grid points and computational effort is much lower than for mesh refinement techniques.

5 Stretched-Mesh Generation

The drive towards full Navier-Stokes solvers has necessitated the development of stretched grid generation techniques in order to resolve the thin boundary-layer and wake regions which are characteristic of high-Reynolds-number viscous flows. Proper boundary-layer and wake resolution usually requires mesh spacing several orders of magnitude smaller in the direction normal to the boundaries than in the streamwise direction, resulting in large cell aspect-ratios in these regions.

The application of the triangulation methods described in the preceding sections to these types of mesh-point distributions results in awkward triangulations, which are ill-suited for numerical computations. This is due to the fact that most of these methods have been conceived for the generation of isotropic meshes, and are ill-suited for the generation of high-aspect-ratio elements. Consider, for example, the Delaunay triangulation of such a point-set. In two dimensions, Delaunay triangulations are equivalent to max-min triangulations, *i.e.*, they correspond to the triangulation which maximizes the smallest angles of all elements. Since high aspect-ratio triangles necessarily contain at least one very small angle, Delaunay triangulations actually attempt to avoid the generation of such elements.

There are two possible approaches in dealing with these difficulties. The first is to make use of alternative element types in regions of high mesh stretching, such as quadrilaterals in two dimensions and prisms or hexahedra in three dimensions [55, 56]. The second approach is to define the types of triangle/tetrahedral elements which are desirable for stretched mesh generation, and to modify existing methods or devise new techniques for generating meshes which contain such elements.

In practice, both approaches are viable. In fact, the two approaches are less distinct than may be expected, since it is always possible to subdivide a mixed-element mesh into a fully triangular or tetrahedral mesh, and similarly most highly stretched triangular/tetrahedral meshes can be transformed into mixed quadrilateral/triangular or prismatic-tetrahedral meshes by identifying appropriate mesh edges for removal [15, 43]. In fact, the definition of a stretching direction in itself implies a certain degree of local structure in the mesh, which enables the simple extraction of quadrilaterals/prisms from a triangular/tetrahedral mesh. Similarly, no loss of flexibility is implied by the use of semi-structured quadrilateral or prismatic meshes in highly-stretched regions. The main drawbacks of using mixed-element meshes is the lack of homogeneity in the grid structure, which may complicate procedures such as flow solution and mesh adaptation. The advantages are reduced overheads for these same procedures, due to the lower overall connectivity of these mesh elements and the resulting reduction in the number of mesh edges. In three dimensions, the use of prismatic elements are particularly attractive, since this permits the use of triangular surface meshes.

If fully triangular/tetrahedral meshes are to be employed, a characterization of the optimal stretched triangle shape needs to be defined. In [57], it is shown how the accuracy of a two-dimensional finite-element approximation on triangular elements degrades as the maximum angle of the elements increases.

Figure 27: Illustration of two types of high aspect ratio triangles: non-obtuse (a) and obtuse (b).

Thus, as shown in Figure 27, stretched obtuse triangles which contain one large angle and two small angles are to be avoided, while stretched nearly right-angle triangles, with one small angle and two nearly right angles are to be preferred. (Similarly, one may infer that the types of tetrahedra which result from the subdivision of a thin prism in 3D are desirable). Thus, a triangulation procedure which avoids obtuse triangles in favor of right-angle triangles would be desirable. It should be

noted , however, that even if a "perfect" triangulation scheme exists, acceptable mesh elements will only be obtained provided the vertices are positioned appropriately. Thus, most stretched-mesh generation techniques involve careful point-placement strategies as well as modified triangulation schemes.

5.1 Stretched Delaunay Point-Insertion

One of the earliest techniques for generating highly stretched triangular meshes [36, 37, 58] makes use of a locally-mapped Delaunay triangulation point-insertion method. The basic idea is to define a local stretching vector (direction and magnitude of stretching) at each vertex. New points are inserted into the triangulation using the Bowyer/Watson Delaunay triangulation algorithm. However, this triangulation is carried out in a transformed space, which is obtained by locally remapping the physical space according to the values of the stretching vector in the vicinity of the new point. This results in a nearly isotropic Delaunay triangulation in the transformed space, but in a stretched Delaunay triangulation in physical space, where triangle circumcircles become circumellipses, as shown in Figure 28.

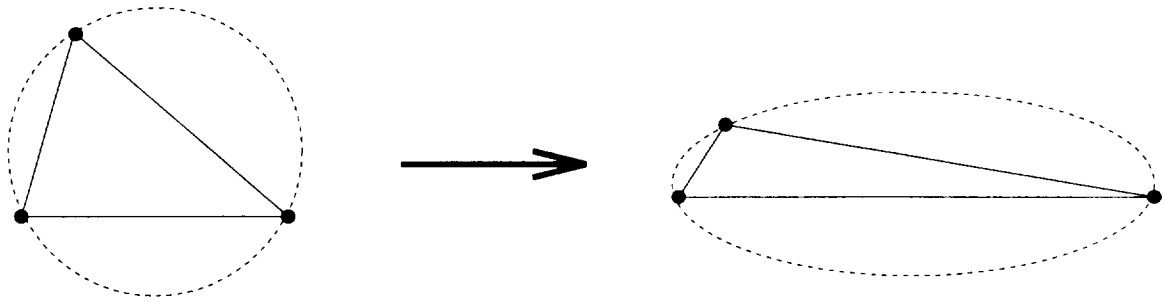

Figure 28: Circumellipse of a stretched Delaunay triangulation.

The problem of constructing a global mapping for complex domains with variable distributions of stretching represents a formidable task. However, since the the Delaunay triangulation is a local construction, the problem can be formulated in a simpler manner, using local mappings. Each time a new point is to be inserted, a local mapping may be constructed, using the local values of the stretching vectors, which may be averaged and taken as constant throughout the region to be restructured. This enables the use of a variable stretching distribution, but requires the distribution of stretching to be smooth, and to vary slowly with respect to the point-distribution. In fact, a close coupling between the stretching and mesh-point distributions is required in order to ensure the generation of appropriately shaped triangular elements. In the present approach, this is achieved by generating both mesh-point and stretching distributions from a set of overlapping stretched structured meshes. The original method [37] can be summarized as follows:

Step 1: Generate a highly stretched structured mesh about each geometry component.
Step 2: Filter out far-field and downstream wake points in these structured meshes.
Step 3: Define a stretching vector at each remaining point, based on the local structured grid-cell orientation, stretching direction, and aspect-ratio (stretching magnitude).
Step 4: Using the Bowyer/Watson algorithm, construct the (unstretched) Delaunay triangulation of this set of points.
Step 5: Smooth the distribution of stretching vectors by performing several passes of averaging stretchings with their neighboring values.

Step 6: Swap the edges of the mesh according to the Delaunay in circle criterion, measured in the locally mapped space, as defined by the average local stretching vector (*i.e.*, a stretched Delaunay in ellipse criterion).
Step 7: Smooth the mesh-point distribution and reswap the edges (this step may be repeated several times).

The use of point-sets derived from structured meshes ensures that the points will be distributed in a configuration which favors the formation of nearly right-angle triangles in the highly stretched regions of the mesh, such as near the walls and in the wake regions. Step 3 ensures the compatibility between the stretching distribution (and thus triangulation criterion) and point distribution, while step 5 is necessary to guarantee a smooth distribution of stretching in regions where the structured meshes overlap. It is noteworthy that an initial triangulation is required in order to accomplish this smoothing. Figure 29 illustrates a stretched unstructured mesh produced by this technique for a simple airfoil configuration.

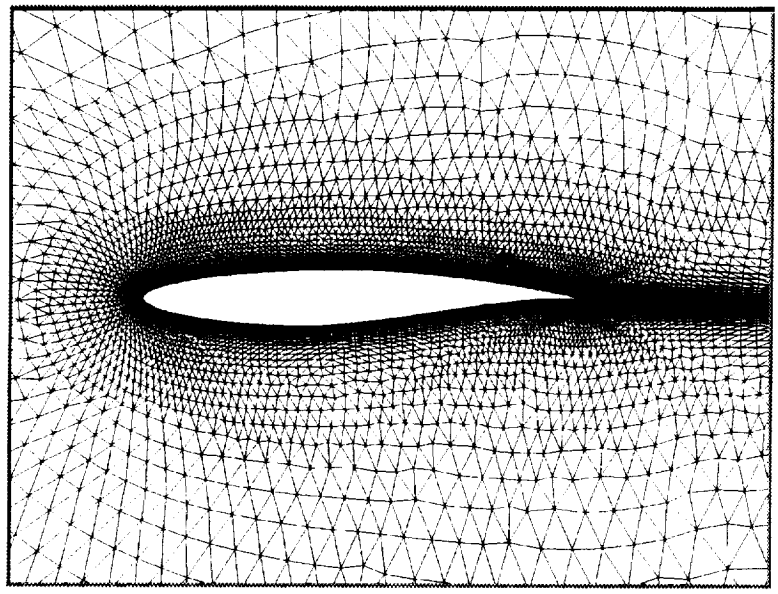

Figure 29: Stretched unstructured mesh produced by mapped Delaunay triangulation procedure for RAE 2822 airfoil.

The present method permits a smooth transition between stretched and unstretched regions, as well as between two neighboring stretched regions, and enables a straight-forward implementation of adaptive meshing techniques. New mesh points may be inserted into the existing mesh by first assigning them a stretching vector taken as the average of the neighboring stretching vectors, and then using the Bowyer/Watson algorithm in the locally stretched space to triangulate the new point. The final mesh may be post-processed with several passes of smoothing and edge-swapping. An example of an adaptively generated stretched unstructured mesh about a four-element airfoil is illustrated in Figure 30.

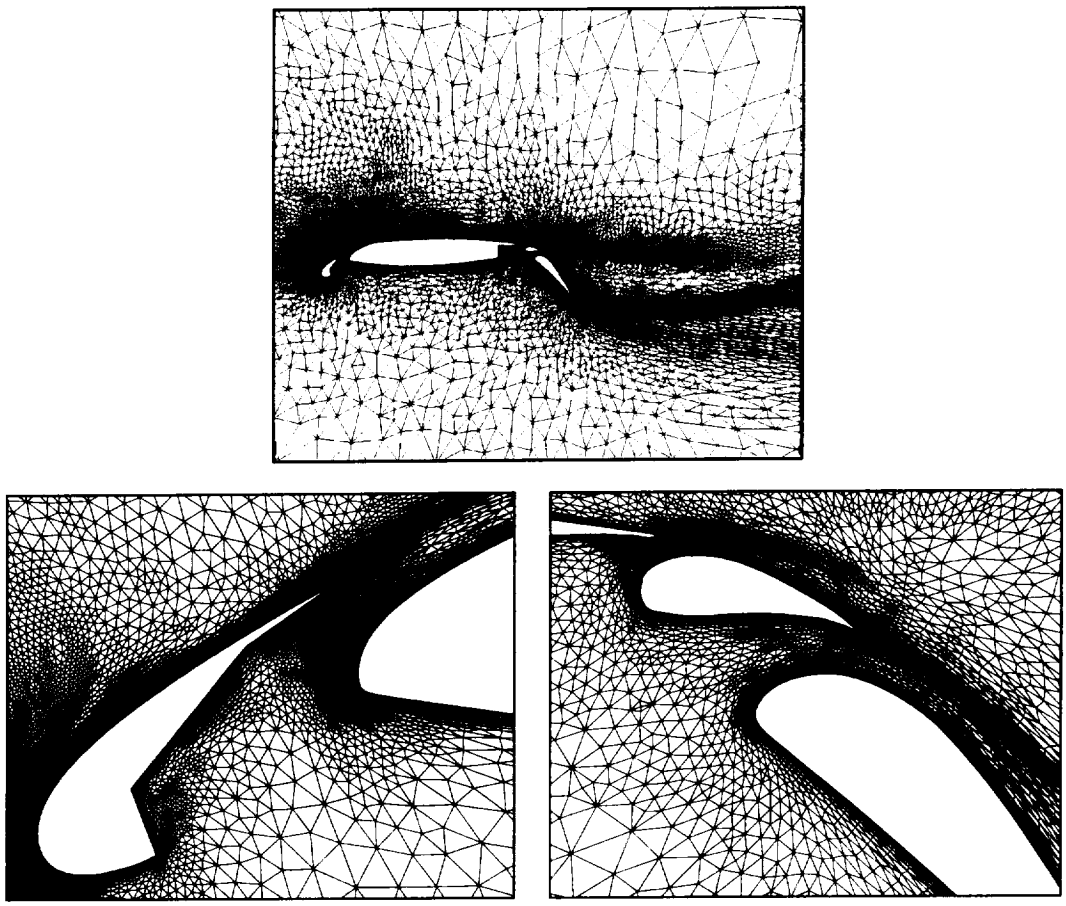

Figure 30: Adapted stretched mesh for computing viscous flow over four-element airfoil.

In retrospect, this mesh generation approach can be viewed as the construction of an initial (Delaunay) triangulation, followed by an edge-swapping reconnection phase governed by a criterion determined by a distribution of stretching. In this respect, it is similar to some of the more recent methods for constructing stretched triangulations.

5.2 Hybrid Methods

One of the most evident techniques for constructing stretched meshes for viscous flows is to use a hybrid technique which makes use of a structured mesh in the thin boundary-layer and wake regions, which require high degrees of stretching, and an unstructured mesh in the regions of inviscid flow, where isotropic constructions are desirable. In two dimensions, the structured portion of the mesh consists of quadrilaterals, while in three dimensions either hexahedra or prismatic (semi-structured) elements may be used. These elements may be later subdivided into triangles or tetrahedra to produce a homogeneous mesh, or alternatively they may be retained, and the flow solver may be modified to capitalize on the local grid structure.

An example of a hybrid approach to stretched mesh generation consists of generating a local structured mesh using a hyperbolic mesh generation technique, up to a prespecified distance away from the boundary surfaces, and an advancing-front unstructured mesh generation technique to complete the mesh by filling in the remainder of the domain with an isotropic unstructured mesh [56]. The problems associated with hybrid mesh constructions involve the adequate definition of the thickness of the structured mesh layer, as well as difficulties involved in concave regions or regions where neighboring boundaries are in close proximity, which may result in overlapping of

the structured meshes. Another difficulty consists of producing a mesh with a smooth transition between stretched structured regions and isotropic unstructured regions. Many of these difficulties can be overcome by employing a structured grid layer of variable thickness.

5.3 Semi-Structured-Unstructured Hybrid Meshes

The hybrid structured-unstructured approach may be made more flexible by employing so-called semi-structured meshes in the stretched mesh regions [59]. Semi-structured meshes are essentially structured meshes with a variable thickness or normal resolution. A semi-structured mesh may be constructed by first generating a structured mesh about each geometry component, over a distance which covers a significant portion of the domain (*i.e.*, a distance much greater than the boundary-layer thickness), and then removing the regions of the mesh which overlap with neighboring structured meshes and/or geometry components, as well as regions of the mesh where stretching is not required, *i.e.*, where the structured mesh cell aspect-ratios are less than unity. The remaining portion of the domain is then gridded with an isotropic triangulation scheme. This results is an automatic determination of the inner-mesh thickness, and a more smooth transition between the inner and outer meshes.

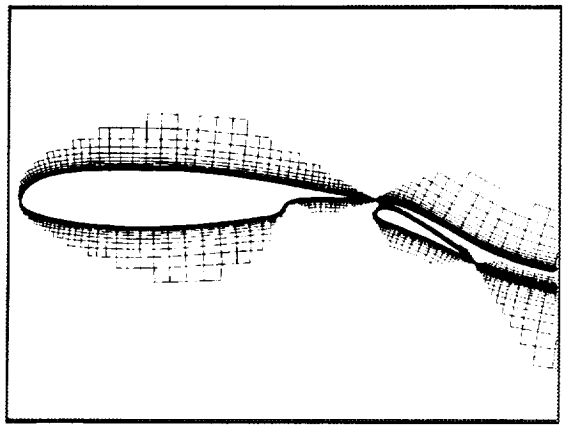

Figure 31: Semi-structured mesh generated prior to completion of isotropic region.

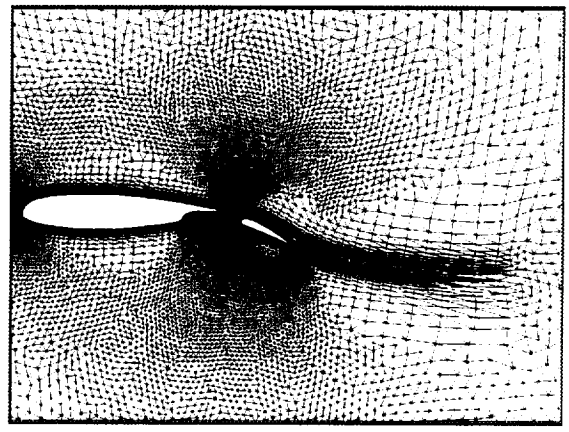

Figure 32: Final Unstructured mesh generated using semi-structured approach.

An example of a semi-structured mesh generated by the author is depicted in Figures 31 and 32. The outer boundary is somewhat ragged, but this presents no difficulty for the unstructured mesh approach which is used to complete the mesh. In this case, the advancing-front Delaunay triangulation scheme was employed in the isotropic regions of the domain, and elements of the semi-structured mesh have been subdivided into triangles. A noteworthy point concerns the structure of the mesh in the wake regions. While highly stretched meshes are required in wake regions, unless these are to be carried to the outer boundary of the domain, they must be blended with the isotropic portion of the mesh in a smooth manner. In the present implementation, this is accomplished by coarsening the wake-point distribution in the normal direction increasingly in the downstream direction, and readjusting the connectivity. This operation is performed prior to the generation of the isotropic portion of the mesh, since it results in a modification of the initial boundary discretization of the remaining area to be gridded isotropically.

The drawbacks of this method are the requirement of being able to generate structured meshes about arbitrary geometry components, the task of locating overlapping structured mesh regions, and the ability of the method to merge two neighboring semi-structured meshes.

5.4 Advancing-Layers Method

The advancing-layers method [60, 61, 62, 63, 64] represents a generalization of the concept of semi-structured meshes. Rather than construct an inner semi-structured mesh by truncating a structured mesh, the advancing-layers technique uses structured hyperbolic mesh generation principles locally at each boundary edge or face to advance a new layer in the normal direction, thus creating a new cell. Advancing-layers can also be viewed as a modification to the unstructured advancing-front algorithm to create stretched quadrilateral (or prismatic) elements using different point-placement and reconnection strategies. This is achieved by capitalizing on the similarities between hyperbolic mesh generation and advancing-front techniques.

The point placement strategy of the advancing-layers method relies on the surface normals of the boundary discretization. A line segment normal to the boundary surface is first associated with each boundary point. These normals are often smoothed by averaging their orientations with those of their neighbors. Progressive smoothing, which vanishes near the boundary and increases away from the boundary is typically employed. Quadrilateral elements (or prismatic elements in three dimensions) are created by placing new points along the boundary normals and connecting them to neighboring points in a specified pattern.

The layers are advanced out from the original discretized boundaries with an increasing step-size which results in progressively decreased stretching. For each normal stack of cells associated with a boundary edge or face, the advancing process is terminated either when an opposing front is encountered, or when the cell aspect-ratio becomes close to unity. Once the advancing-layers procedure terminates, the remainder of the domain is filled in with an isotropic unstructured mesh, as in the previous methods. While the rate of growth of the advancing layers is typically prescribed, a background function is used to determine the resolution of the initial boundary discretization as well as to control that of the unstructured portion of the mesh. When the advancing-layers phase terminates, the existing mesh looks much like the semi-structured or truncated hyperbolic structured mesh of Figure 31. A three-dimensional example of the advancing layers method, taken from [64], is depicted in Figure 33, for a four-element wing configuration, showing the surface mesh, and the partially completed mesh of stretched layers. As in the semi-structured mesh procedure, the layers may be conserved as quadrilaterals or prisms in two or three dimensions respectively, or they may be subdivided into triangles or tetrahedra.

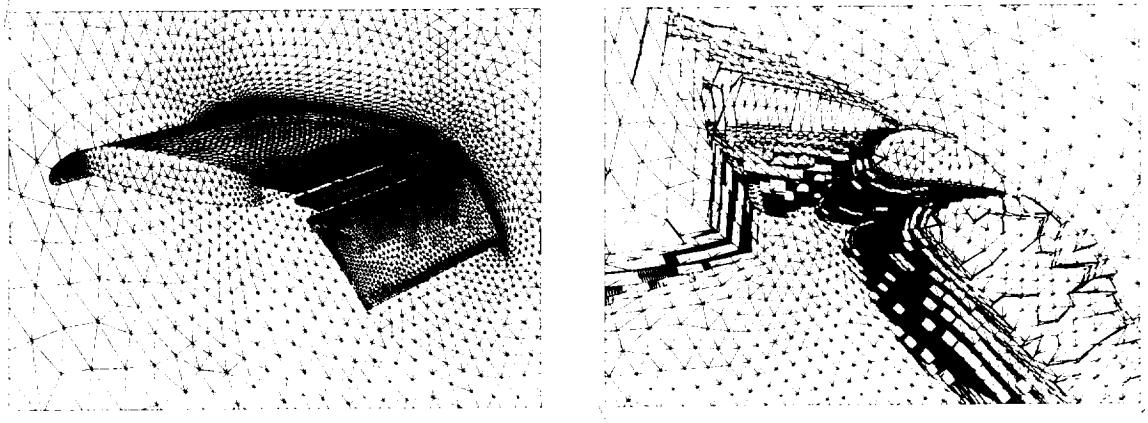

Figure 33: Illustration of advancing-layers method for three-dimensional unstructured mesh generation about segmented wing geometry. (Reproduced from [64] with permission).

5.5 Advancing-Front Min-Max Triangulations

A somewhat more unified approach to generating highly stretched triangulations involves the use of min-max triangulations, *i.e.*, triangulations which minimize the maximum angles produced in the elements. While Delaunay triangulations were written-off as unsuitable for stretched mesh generation, the desire to avoid obtuse or large angle triangles in such meshes implies the feasibility of using min-max triangulations for generating stretched meshes.

However, connectivity strategies alone cannot guarantee high quality meshes, and thus compatible point-placement strategies must be developed. The advancing-front min-max triangulation algorithm of Marcum [43] makes use of two distinct point placement strategies, one for highly stretched regions of the mesh, and another for isotropic regions. In the stretched mesh regions, the point placement strategy resembles that described in the advancing-layers method, *i.e.*, points are placed along smoothed boundary normal segments. In the isotropic regions, points are placed according to the same rules described for the advancing-front Delaunay triangulation algorithm of section 3.5, making use of a background element-size distribution function. The mesh generation process begins by generating a coarse triangulation which covers the entire domain. The domain boundaries are then discretized according to the background function, and these boundary points are inserted into the existing triangulation. Once all boundary points have been inserted, the boundary integrity is recovered. The advancing-front min-max procedure is then initiated by creating new points using the appropriate point-placement strategy and inserting each point into the existing triangulation using the min-max variant of the Green-Sibson algorithm (*i.e.*, forced triangulation followed by edge-swapping). The unifying feature of this strategy is that the insertion and triangulation of new points is identical in stretched and unstretched regions. The point placement methods are however different in these two regions, and the method retains a somewhat hybrid character. An example of a stretched unstructured mesh generated by this technique is shown in Figure 34.

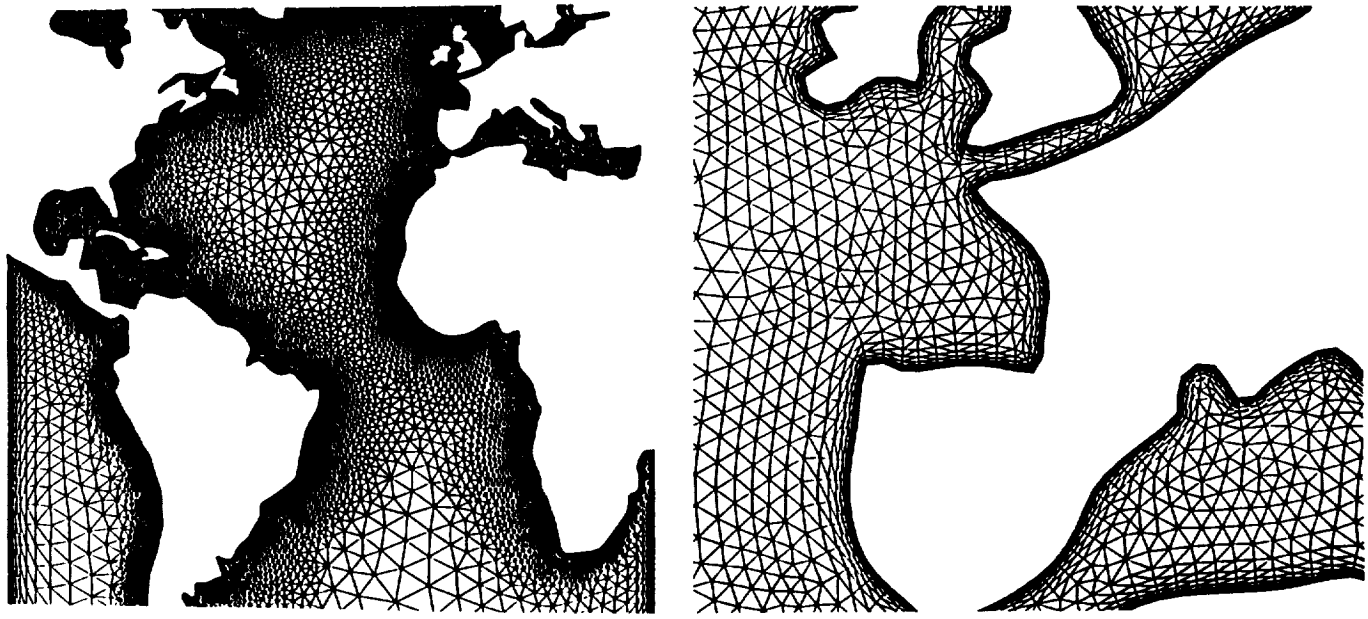

Figure 34: Stretched unstructured mesh produced by advancing-front min-max triangulation procedure for familiar two-dimensional geometry, with illustration of stretched triangulation over the coast of Belgium. (Reproduced from [43] with permission).

The method has also been demonstrated in three dimensions, as shown in Figure 35. In three dimensions, each time a new point is inserted, the mesh must first be swapped to a local Delaunay

configuration, and then to the min-max configuration, in order to attain a more globally optimum final min-max configuration, as described in section 3.5.

An alternate point-placement strategy for min-max triangulations which does not rely on an advancing-front is given by Barth [65]. One of the drawbacks of the reliance on min-max triangulations is the possibility of the min-max criterion resulting in undesirable connectivities for seemingly well distributed vertices. For highly stretched meshes, the min-max triangulation may be very sensitive to the placement of vertices, with small displacements leading to substantially different connectivities. The precise placement of grid points in the highly stretched regions is thus much more critical than in "forced" triangulation methods such as the advancing-layers method.

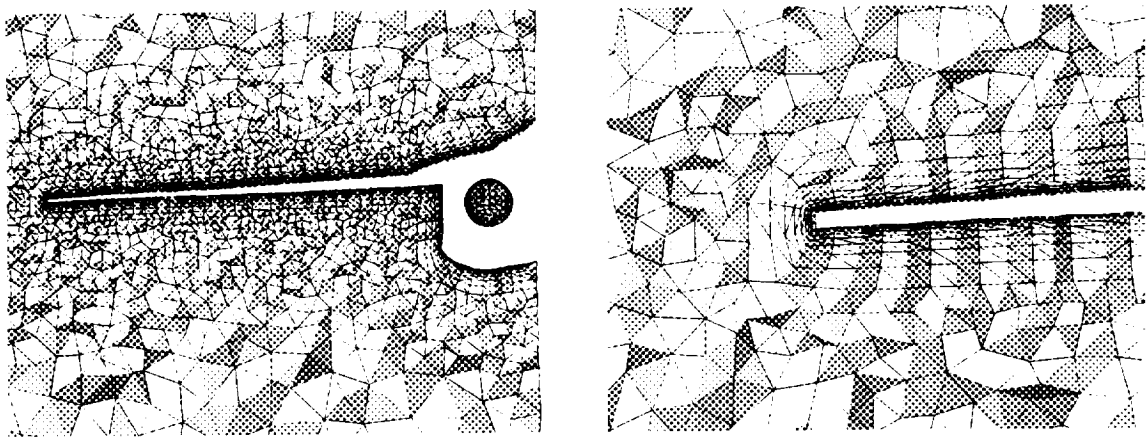

Figure 35: Section of an advancing-front min-max triangulation grid in three dimensions about an F-18 aircraft configuration. (Reproduced from [43] with permission).

References

[1] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data-structure. *ACM Computing Surveys*, 23:345–405, 1991.

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

[3] R. Sedgewick. *Algorithms.* Addison-Wesley, 1988. second edition.

[4] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[5] A. Bowyer. Computing Dirichlet tessalations. *The Computer Journal*, 24(2):162–166, 1981.

[6] D. F. Watson. Computing the n-dimensional Delaunay tessalation with application to Voronoi polytopes. *The Computer Journal*, 24(2):167–171, 1981.

[7] T. J. Baker. Three dimensional mesh generation by triangulation of arbitrary point sets. AIAA paper 87-1124, June 1987.

[8] N. P. Weatherill. The generation of unstructured grids using Dirichlet tessalations. Princeton University Department of Mechanical and Aerospace Engineering Report MAE 1715, July 1985.

[9] P. L. George F. Hecht and E. Saltel. Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering*, 33:975–995, 1991.

[10] L. J. Guibas D. E. Knuth and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. Stanford University Computer science Rep. no. STAN-CS-90-1300, 1990.

[11] P. J. Green and R. Sibson. Computing the Dirichlet tessalation in the plane. *The Computer Journal*, 21(2):168–173, 1977.

[12] J. M. Nelson. A triangulation algorithm for arbitrary planar domains. *Applied Math Modeling*, 2:151–159, 1978.

[13] A. Maus. Delaunay triangulation and the convex hull of N points in expected linear time. *BIT*, 24:151–163, 1984.

[14] M. Tanemura T. Ogawa and N. Ogita. A new algorithm for three-dimensional Voronoi tessellation. *Journal of Computational Physics*, 51(2):191–207, 1983.

[15] M. L. Merriam. An efficient advancing-front algorithm for Delaunay triangulation. AIAA paper 91-0792, January 1991.

[16] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[17] D. Nira D. Levin and S. Rippa. Data dependent triangulations for piecewise linear interpolation. *Journal of Numerical Analysis*, 10(1):137–154, 1990.

[18] C. L. Lawson. Software for C^1 surface interpolation. In *Mathematical Software III, ed. John D. Rice*, New York, 1977. Academic Press.

[19] C. L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3:365–372, 1972.

[20] H. Edelsbrunner T. S. Tan and R. Waupotitsch. An $O(N^2 log N)$ time algorithm for the minmax angle triangulation. *SIAM Journal on Scientific and Statistical Computing*, 13:994–1008, 1992.

[21] C. L. Lawson. Properties of n-dimensional triangulations. *CAGD*, 3:231–246, April 1986.

[22] M. E. Mortenson. *Geometric Modeling*. John Wiley and Sons, 1985.

[23] B. Joe. Three-dimensional triangulations from local transformations. *SIAM Journal on Scientific and Statistical Computing*, 10:718–741, 1989.

[24] V. T. Rajan. Optimality of the Delaunay triangulation in R^d. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 357–372, 1991.

[25] T. J. Barth. Aspects of unstructured grids and finite-element volume solvers for the Euler and Navier-Stokes equations. In *von Karman Institute Lecture Series*, AGARD Pub. R-787, 1992.

[26] J. Peraire M. Vahdati K. Morgan and O. C. Zienkiewicz. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics*, 72:449–466, October 1987.

[27] C. Gumbert R. Lohner P. Parikh and S. Pirzadeh. A package for unstructured grid generation and finite element flow solvers. AIAA paper 89-2175, June 1989.

[28] S. Pirzadeh. Structured background grids for generation of unstructured grids by advancing-front method. *AIAA Journal*, 31(2):257–265, February 1993.

[29] D. J. Mavriplis. An advancing-front Delaunay triangulation algorithm designed for robustness. AIAA paper 93-0671, January 1993.

[30] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, second edition, 10 January 1981.

[31] M. A. Yerry and M. S. Shephard. Automatic three dimensional mesh generation by the modified-octree technique. *International Journal for Numerical Methods in Engineering*, 20:1965–1990, 1984.

[32] J. H. Cheng P. M. Finnigan A. F. Hathaway A. Kela and W. J. Schroeder. Quadtree/octree meshing with adaptive analysis. In *Numerical Grid Generation in Computational Fluid Mechanics*. Pineridge Press, December 1988. Proc. of the Second International Conference on Numerical Grid Generation in Computational Fluid Dynamics, Miami, eds. S. Sengupta, J. Hauser, P. R. Eisman, and J. F. Thompson.

[33] L. Formaggia. An unstructured mesh generation algorithm for three dimensional aeronautical configurations. In *Proc. of the Third International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Barcelona, Spain*. New York, 1991. North-Holland. eds. A. S. Arcilla, J. Hauser, P. R. Eisman, and J. F. Thompson.

[34] S. Pirzadeh. Recent progrss in unstructured grid generation. AIAA paper 92-0445, January 1992.

[35] D. J. Mavriplis. *Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes*. PhD thesis, Princeton University, MAE Department, 1987.

[36] D. J. Mavriplis. Adaptive mesh generation for viscous flows using Delaunay triangulation. *Journal of Computational Physics*, 90(2):271–291, 1990.

[37] D. J. Mavriplis. Unstructured and adaptive mesh generation for high-Reynolds number viscous flows. In *Proc. of the Third International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Barcelona, Spain*, pages 79–92, New York, 1991. North-Holland. eds. A. S. Arcilla, J. Hauser, P. R. Eisman, and J. F. Thompson.

[38] T. J. Baker. Unstructured meshes and surface fidelity for complex shapes. In *Proc. of 10th AIAA CFD Conference, Honolulu, Hawaii*, pages 714–725, 1991. AIAA paper 91-1591-CP.

[39] D. G. Holmes D. D. Snyder. The generation of unstructured meshes using Delaunay triangulation. In *Numerical Grid Generation in Computational Fluid Mechanics*. Pineridge Press, December 1988. Proc. of the Second International Conference on Numerical Grid Generation in Computational Fluid Dynamics, Miami, eds. S. Sengupta, J. Hauser, P. R. Eisman, and J. F. Thompson.

[40] N. P. Weatherill O. Hassan and D. L. Marcum. Calculation of steady compressible flowfields with the finite-element method. AIAA paper 93-0341, January 1993.

[41] W. K. Anderson. A grid generation and flow solution method for the Euler equations on unstructured grids. *J. Comp. Phys.*, 110(1):23–38, 1994.

[42] L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the 9th ACM Symposium on Computational Geometry*, 1993.

[43] D. L. Marcum. Generation of unstructured grids for viscous flow applications. AIAA paper 95-0212, January 1995.

[44] S. Rebay. Efficient unstructured mesh generation by means of Delaunay triangulation and Bowyer/Watson algorithm. *Journal of Computational Physics*, 106(1):125–138, 1993.

[45] J. D. Mueller P. L. Roe and H. Deconinck. A frontal approach for node generation in Delaunay triangulations. In *von Karman Institute Lecture Series*, AGARD Pub. R-787, 1992.

[46] D. L. Marcum and N. P. Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. AIAA paper 94-1926, June 1994.

[47] R. Lohner. Finite-element methods in CFD: Grid generation, adaptivity and parallelization. In *von Karman Institute Lecture Series*, AGARD Pub. R-787, 1992.

[48] G. Warren W. K. Anderson J. L. Thomas and S. L. Krist. Grid convergence for adaptive methods. AIAA paper 91-1592, June 1991.

[49] G. F. Carey and J. T. Oden. *Finite Elements.* Prentice-Hall, New Jersey, 1983.

[50] D. J. Mavriplis. Accurate multigrid solution of the Euler equations on unstructured and adaptive meshes. *AIAA J.*, 28(2):213–221, 1990.

[51] T. J. Barth. Randomized multigrid. AIAA paper 95-0207, January 1995.

[52] R. Lohner and J. D. Baum. Adaptive h-refinement on 3D unstructured grids for transient problems. *International Journal for Numerical Methods in Fuids*, 14:1407–1419, 1992.

[53] M. Paraschivoiu, J. Y. Trepanier, M. Reggio, and R. Camarero. A conservative dynamic discontinuity tracking algorithm for the Euler equations. AIAA Paper 94-0081, January 1994.

[54] J. vanRosendale. Floating shock fitting via Lagrangian adaptive meshes. ICASE report no.94-89, NASA CR 194997, November 1994.

[55] K. Nakahashi. FDM-FEM approach for viscous flow computations over multiple bodies. AIAA paper 87-0604, January 1987.

[56] S. Ward and Y. Kallinderis. Hybrid prismatic/tetrahedral grid generation for complex 3D geometries. AIAA paper 93-0669, January 1993.

[57] I. Babushka and A. K. Aziz. On the angle condition in the finite-element method. *SIAM Journal of Numerical Analysis*, 13(6), 1976.

[58] M. G. Vallet F. Hecht and B. Mantel. Anisotropic control of mesh generation based upon a Voronoi type method. In *Proc. of the Third International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Barcelona, Spain*, pages 93–103, New York, 1991. North-Holland. eds. A. S. Arcilla, J. Hauser, P. R. Eisman, and J. F. Thompson.

[59] R. Lohner. Matching semi-structured and unstructured grids for Navier-Stokes calculations. AIAA paper 93-3348, July 1993.

[60] J. D. Mueller P. L. Roe and H. Deconinck. Delaunay based triangulations for the Naiver-Stokes equations with minimum user input. In *Proc. of the 13th International Conference on Numerical Methods in Fluid Dynamics, Rome, Italy*, pages 125–129. Springer-Verlag, July 1992. Lecture Notes in Physics, No. 414, eds. M. Napolitano and F. Sabetta.

[61] M. J. Marchant and N. P. Weatherill. Unstructured grid generation for viscous flow simulations. In *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, pages 151–162. Pineridge Press, April 1994. Proc. of the Fourth International Conference on Numerical Grid Generation in Computational Fluid Dynamics, Swansea, Wales, UK eds. N. P. Weatherill, P. R. Eiseman, J. Hauser, and J. F. Thompson.

[62] O. Hassan E. J. Probert N. P. Weatherill M. Marchant K. Morgan and D. L. Marcum. The numerical simulation of viscous transonic flow using unstructured grids. AIAA paper 94-2346, June 1994.

[63] S. Pirzadeh. Unstructured viscous grid generation by the advancing-layers method. *AIAA Journal*, 32(8):1735–1737, August 1994.

[64] S. Pirzadeh. Viscous unstructured three-dimensional grids by the advancing- layers method. AIAA paper 94-0417, January 1994.

[65] T. J. Barth. Steiner triangulation for isotropic and stretched elements. AIAA paper 95-0213, January 1995.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1995	Contractor Report

4. TITLE AND SUBTITLE
UNSTRUCTURED MESH GENERATION AND ADAPTIVITY

5. FUNDING NUMBERS
C NAS1-19480
WU 505-90-52-01

6. AUTHOR(S)
D. J. Mavriplis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Computer Applications in Science
and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER
ICASE Report No. 95-26

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR-195069
ICASE Report No. 95-26

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Dennis M. Bushnell
Final Report
Lecture notes for 26th CFD Lecture Series of von Karman Institute; AGARD publication

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited

Subject Category 64

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

An overview of current unstructured mesh generation and adaptivity techniques is given. Basic building blocks taken from the field of computational geometry are first described. Various practical mesh generation techniques based on these algorithms are then constructed and illustrated with examples. Issues of adaptive meshing and stretched mesh generation for anisotropic problems are treated in subsequent sections. The presentation is organized in an educational manner, for readers familiar with computational fluid dynamics, wishing to learn more about current unstructured mesh techniques.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Unstructured; Mesh; Adaptive	47
	16. PRICE CODE: A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		