R.787

ASPECTS OF UNSTRUCTURED GRIDS AND FINITE-VOLUME SOLVERS FOR THE EULER AND NAVIER-STOKES EQUATIONS

by

Timothy J. Barth
CFD Branch
NASA Ames Research Center
Moffett Field, CA 94035
United States

Contents

1.0 Preliminaries
- 1.1 Graphs and Meshes
- 1.2 Duality
- 1.3 Data Structures

2.0 Some Basic Graph Operations Important in CFD
- 2.1 Planar Graphs with Minimum Out-Degree
- 2.2 Graph Ordering Techiques
- 2.3 Graph Partitioning for Parallel Computing

3.0 Triangulation Methods
- 3.1 Voronoi Diagram and Delaunay Triangulation
- 3.2 Properties of a 2-D Delaunay Triangulation
- 3.3 Algorithms for 2-D Delaunay Triangulation
- 3.4 Other 2-D Triangulation Algorithms
- 3.5 2-D Steiner Triangulations
- 3.6 Three-Dimensional Triangulations

4.0 Some Theory Related to Finite-Volume Solvers.
- 4.1 Scalar Conservation Law Equations
- 4.2 Comparison of Finite-Volume and Galerkin Finite-Element Methods
- 4.3 Edge Formulas
- 4.4 Godunov Finite-Volume Schemes
- 4.5 k-exact Reconstruction
- 4.6 Upwind Advection Scheme with $k = 0$ Reconstruction
- 4.7 Upwind Advection Scheme with $k = 1$ Linear Reconstruction
- 4.8 Maximum Principles and Delaunay Triangulation

5.0 Finite-Volume Solvers for the Euler Equations
- 5.1 Euler Equations in Integral Form
- 5.2 Extension of Scalar Advection Schemes to Systems of Equations
- 5.3 Implicit Linearizations

6.0 Numerical Solution of the Navier-Stokes Equations with Turbulence
- 6.1 Turbulence Modeling for Unstructured grids
- 6.2 A One-Equation Turbulence Transport Model

Introduction

One of the major achievements in engineering science has been the development of computer algorithms for solving nonlinear differential equations such as the Navier-Stokes equations. These algorithms are now used in the practical engineering design of devices such as cars and airplanes as well as theoretical studies of complex phenomena such as fluid turbulence. In past years, limited computer resources have motivated the development of efficient numerical methods in computational fluid dynamics (CFD) utilizing *structured* meshes. These meshes are comprised of systematic arrays of quadrilateral or hexahedral cells. The use of structured meshes greatly simplifies the implementation of CFD algorithms on conventional computers. Structured meshes also permit the use of highly efficient solution techniques such as alternating direction implicit (ADI) iteration schemes or multigrid. Following the dramatic improvement in computing speed in recent years, emphasis has shifted towards the design of algorithms capable of treating complex geometries. The automatic generation of structured grids about complex geometries is problematic. Unstructured grids offer one promising alternative technique for treating these general geometries. Unstructured meshes have irregular connectivity and usually contain triangles and/or quadrilaterals in two dimensions and tetrahedra and/or hexahedra in three dimensions. The generation and use of unstructured grids poses new challenges in computational fluid dynamics. This is true for both grid generation as well as for the design of algorithms for flow solution. The purpose of these notes is to present recent developments in the unstructured grid generation and flow solution technology.

1.0 Preliminaries

1.1 Graphs and Meshes

Graph theory offers many valuable theoretical results which directly impact the design of efficient algorithms using unstructured grids. For purposes of the present discussion, only *simple* graphs which do not contain self loops or parallel edges will be considered. Results concerning simple graphs usually translate directly into results relevant to unstructured grids. The most famous graph theoretic result is Euler's formula which relates the number of edges $n(e)$, vertices $n(v)$, and faces $n(f)$ of a polyhedron (see figure 1.0(a)):

$$n(f) = n(e) - n(v) + 2 \qquad \text{(Euler's formula)} \qquad (1.0)$$

This polyhedron can be embedded in a plane by mapping one face to infinity. This makes the graph formula (1.0) applicable to 2-D unstructured meshes. In the example below, the face 1-2-3-4 has mapped to infinity to form the exterior (infinite) face. If all faces are numbered including the exterior face, then Euler's formula (1.0) remains valid.

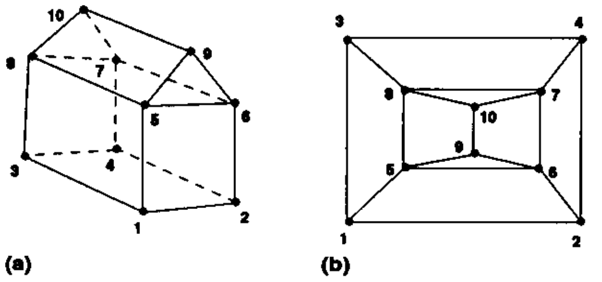


Figure 1.0 (a) 3-D Polyhedron, (b) 2-D Planar embedding

The infinite face can be eliminated by describing the outer boundary in terms of boundary edges which share exactly one interior face (interior edges share two). We also consider boundary edges which form simple closed curves in the interior of the mesh. These curves serve to describe possible objects embedded in the mesh (in this case, the polygon which they form is not counted as a face of the mesh). The number of these polygons is denoted by $n(h)$. The modified Euler's formula now reads

$$n(f) + n(v) = n(e) + 1 - n(h) \qquad (1.1)$$

Since interior edges share two faces and boundary edges share one face, the number of interior and boundary edges can be related to the number of faces by the following formula:

$$2n(e)_{interior} + n(e)_{bound} = \sum_{i=3}^{max\ d(f)} i\ n(f)_i \quad (1.2)$$

where $n(f)_i$ denotes the number of faces of a particular edge degree, $d(f) = i$. Note that for pure triangulations T, these formulas can be used to determine, *independent of the method of triangulation*, the number of triangles or edges given the number of vertices $n(v)$, boundary edges $n(e)_{bound}$, and interior holes $n(h)$

$$n(f)_3 = 2n(v) - n(e)_{bound} - 2 + 2n(h) \quad (1.3)$$

or

$$n(e) = 3n(v) - n(e)_{bound} - 3 + 3n(h). \quad (1.4)$$

This is a well known result for planar triangulations. (For brevity, we will sometimes use N to denote $n(v)$ in the remainder of these notes.) In many cases boundary edges are not explicitly given and the boundary is taken to be the convex hull of the vertices, i.e. the smallest convex polygon surrounding the vertices. (To obtain the convex hull in two dimensions, envision placing an elastic rubber band around the cloud of points. The final shape of this rubber band will be the convex hull.) ***A key observation for planar meshes is the asymptotic linear storage requirement with respect to the number of vertices for arbitrary mesh arrangements.***

The Euler formula extends naturally to an arbitrary number of space dimensions. In this general setting, Euler's formula relates basic components (vertices, edges, faces, etc.) of higher dimensional polytopes. In computational geometry jargon, vertices, edges, and faces are all specific examples of "k-faces". A 0-face is simply a vertex, a 1-face corresponds to a edge, a 2-face corresponds to a *facet* (or simply a face), etc. A polytope $\mathcal{P}$ in $\mathbf{R}^d$ contains k-faces $\forall\ k \in \{-1,0,1,...,d\}$. The -1-face denotes the ***null set*** face by standard convention. Let the number of k-faces contained in the polytope $\mathcal{P}$ be denoted by $N_k(\mathcal{P})$. For example, $N_0(\mathcal{P})$ would denote the number of vertices. Using this notation, we have the following relationships:

$$N_0 \equiv n(v) \ \text{(vertices)}, \quad N_1 \equiv n(e) \ \text{(edges)}$$
$$N_2 \equiv n(f) \ \text{(faces)}, \quad N_3 \equiv n(\phi) \ \text{(volumes)}$$

By convention, there is exactly one null set contained in any polytope, i.e. $N_{-1}(\mathcal{P}) = 1$ and by definition $N_d(\mathcal{P}) = 1$. Using these results, we can succinctly state the general Euler formula for an arbitrary polytope in $\mathbf{R}^d$

$$\sum_{k=-1}^{d} (-1)^k N_k(\mathcal{P}) = 0 \quad \text{(Euler's Formula in } \mathbf{R}^d\text{)} \tag{1.5}$$

On the surface of a polyhedron in 3-space, the standard Euler formula (1.0) is recovered since

$$-1 + N_0 - N_1 + N_2 - 1 = 0$$

or

$$n(f) + n(v) = n(e) + 2.$$

To obtain results relevant to three-dimensional unstructured grids, the Euler formula (1.5) is applied to a four-dimensional polytope.

$$n(f) + n(v) = n(e) + n(\phi) \tag{1.6}$$

This formula relates the number of vertices, edges, faces, and volumes $n(\phi)$ of a three-dimensional mesh. As in the two-dimensional case, this formula does not account for boundary effects because it is derived by looking at a single four-dimensional polytope. The example below demonstrates how to derive exact formulas including boundary terms for a tetrahedral mesh. Derivations valid for more general meshes in three dimensions are also possible.

<u>Example:</u> Derivation of Exact Euler Formula for 3-D Tetrahedral Mesh.

Consider the collection of volumes incident to a single vertex v_i and the polyhedron which describes the shell formed by these vertices. Let $F_b(v_i)$ and $N_b(v_i)$ denote the number of faces and vertices respectively of this polyhedron which actually lie on the boundary of the entire mesh. Also let $E(v_i)$ denote the total number of edges on the polyhedron surrounding v_i. Finally, let $d_\phi(v_i)$ and $d_e(v_i)$ denote the number of tetrahedral volumes and edges respectively that are incident to v_i. On this polyhedron, we have exact satisfaction of Euler's formula (1.0), i.e.

$$\overbrace{d_\phi(v_i) + F_b(v_i)}^{\text{polyhedral faces}} + \overbrace{d_e(v_i) + N_b(v_i)}^{\textit{vertices on polyhedron}} = E(v_i) + 2 \tag{1.7}$$

Note that this step assumes that the polyhedron is homeomorphic to a sphere (otherwise Euler's formula fails). In reality, this is not a severe assumption. (It would preclude a mesh consisting of two tetrahedra which touch at a single vertex.) On the polyhedron we also have that

$$2E(v_i) = 3\overbrace{(d_\phi(v_i) + F_b(v_i))}^{\text{polyhedral faces}}. \tag{1.8}$$

Combining (1.7) and (1.8) yields

$$d_e(v_i) = \frac{1}{2}d_\phi(v_i) + \frac{1}{2}F_b(v_i) - N_b(v_i) + 2. \tag{1.9}$$

Summing this equation over <u>all</u> vertices produces

$$\bar{d}_e n(v) = \frac{1}{2}\left(\bar{d}_\phi n(v) + \sum_i F_b(v_i)\right) - \sum_i N_b(v_i) \tag{1.10}$$

where $\bar{d}_e$ and $\bar{d}_\phi$ are the average vertex degrees with respect to edges and volumes. Since globally we have that

$$\bar{d}_e n(v) = 2n(e), \quad \bar{d}_\phi n(v) = 4n(\phi), \tag{1.11}$$

substitution of (1.11) into (1.10) reveals that

$$n(e) = n(\phi) + n(v) + \frac{1}{4}\sum F_b(v_i) - \frac{1}{2}\overbrace{\sum N_b(v_i)}^{n(v)_{bound}}. \tag{1.12}$$

Finally, note that $\sum F_b(v_i) = 3n(f)_{bound}$. Inserting this relationship into (1.12) yields

$$n(e) = n(\phi) + n(v) + \frac{3}{4}n(f)_{bound} - \frac{1}{2}n(v)_{bound} \tag{1.13}$$

Other equivalent formulas are easily obtained by combining this equation with the formula relating volume, faces, and boundary faces, i.e.

$$n(f) = 2n(\phi) + \frac{1}{2}n(f)_{bound} \tag{1.14}$$

An exact formula, similar to (1.6), is obtained by combining (1.14) and (1.13)

$$n(e)+n(\phi)=n(f)+n(v)+\frac{1}{4}n(f)_{bound}-\frac{1}{2}n(v)_{bound} \tag{1.15}$$

1.2 Duality

Given a planar graph G, we informally define a dual graph G_{Dual} to be any graph with the following three properties: each vertex of G_{Dual} is associated with a face of G; each edge of G is associated with an edge of G_{Dual}; if an edge separates two faces, f_i and f_j of G then the associated dual edge connects two vertices of G_{Dual} associated with f_i and f_j. This duality plays an important role in CFD algorithms.

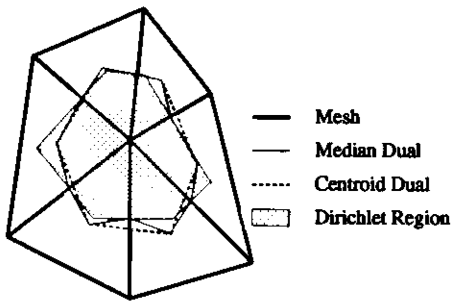


Figure 1.1 Several triangulation duals.

In figure 1.1, edges and faces about the central vertex are shown for duals formed from median segments, centroid segments, and by Dirichlet tessellation. (The Dirichlet tessellation of a set of points is a pattern of convex regions in the plane, each region being the portion of the plane closer to some given point P of the set of points than to any other point.) Two-dimensional finite-volume schemes for the Euler and Navier-Stokes equations are frequently developed which form control volumes from either faces (cells) of the mesh or faces of the mesh dual. Schemes which use the cells of the mesh as control volumes are often called "cell-centered" schemes. Other "vertex" schemes use mesh duals constructed from median segments, Dirichlet regions, or centroid segments. In all of these schemes, the primary computational effort is associated with the calculation of the flux of mass, momenta, and energy through an edge associated with the control volume. The one-to-one correspondence of edges of a mesh and mesh dual (ignoring boundaries) means that there is very little difference in computational effort in schemes based on mesh faces or duals. This observation is not true in three dimensions! Consider a three dimensional tetrahedral mesh. The duality for this nonplanar arrangement is between *edges* of the tetrahedral mesh and *faces* of the dual. In other words, for each edge of the mesh there is a one-to-one correspondence with a face of the dual (ignoring boundaries). Again, the main computational effort associated with finite-volume schemes for solving the Euler and Navier-Stokes equations is the calculation of the flux through each face of the control volume. If the control volumes are the tetrahedra themselves (cell-centered scheme), then a flux must be calculated for each tetrahedral face. This means that the work is proportional to the number of faces of the tetrahedral mesh. From eqn. (1.14), the number of faces of a tetrahedral mesh is related to the number of tetrahedra and boundary faces by

$$work_{c-c\ scheme} \propto n(f) = 2n(\phi) + \frac{1}{2}n(f)_{bound}.$$

If the control volumes of the finite-volume scheme are formed from a mesh dual (vertex scheme), then the number of flux calculations is proportional to the number of faces of the mesh dual which is roughly equal to the number of edges of the original tetrahedral mesh. From eqn. (1.13) we have that

$$work_{vert\ scheme} \propto n(e) = n(\phi) + n(v) + \frac{3}{4}n(f)_{bound} - \frac{1}{2}n(v)_{bound}$$

To better understand the work estimates, define β such that $n(\phi) = \beta n(v)$. Practically speaking, β usually ranges from 5-7 for tetrahedral meshes. Taking the ratio of the work estimates for the cell-centered and vertex scheme, ignoring boundary terms and assuming an identical constant of proportionality, we obtain

$$\frac{work_{c-c\ scheme}}{work_{vert\ scheme}} = \frac{2\beta n(v)}{(1+\beta)n(v)} = \frac{2\beta}{1+\beta} \quad (1.16)$$

The work for the cell-centered scheme approaches twice that of the vertex scheme. The reader should not automatically infer that the vertex scheme is preferred. The question of solution accuracy of the two approaches needs to be factored into the equation. The answer to the question of which is "better" is still a subject for debate.

1.3 Data Structures

The choice of data structures used in representing unstructured grids varies considerably depending on the type of algorithmic operations to be performed. In this section, a few of the most common data structures will be discussed. The mesh is assumed to have a numbering of vertices, edges, faces, etc. In most cases, the physical coordinates are simply listed by vertex number. The "standard" finite element (FE) data structure lists connectivity of each element. For example in figure 1.2(a), a list of the three vertices of each triangle would be given.

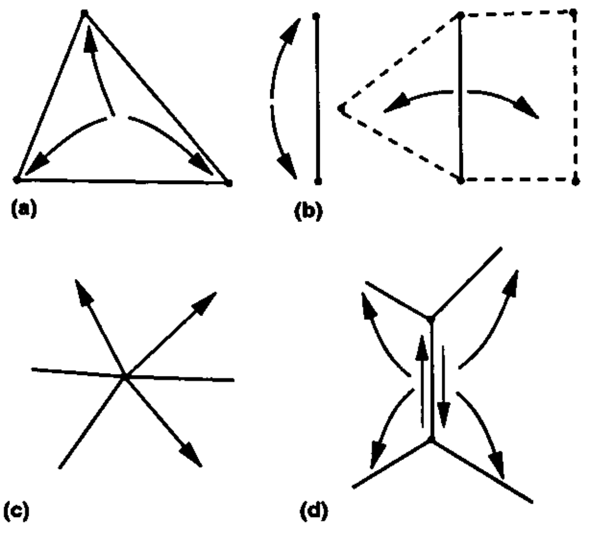


Figure 1.2 Data structures for planar graphs. (a) FE data structure, (b) Edge structure, (c) Out-degree structure, (d) Quad-edge structure.

The FE structure extends naturally to three dimensions. The FE structure is used extensive in finite element solvers for solids and fluids.

For planar meshes, another typical structure is the *edge structure* (figure 1.2(b)) which lists connectivity of vertices and adjacent faces by storing a quadruple for each edge consisting of the origin and destination of the each edge as well as the two faces (cells) that share that edge. This structure allows easy traversal through a mesh which can be useful in certain grid generation algorithms. (This traversal is not easily done using the FE structure.) The extension to three dimensions is facewise (vertices of a face are given as well as the two neighboring volumes) and requires distinction between different face types.

A third data structure provides connectivity via *vertex lists* as shown in figure 1.2(c). The brute force approach is to list all adjacent neighbors for each vertex (usually as a linked-list). Many sparse matrix solver packages specify nonzeros of a matrix using row or column storage schemes which list all nonzero entries of a given row or column. For discretizations involving only adjacent neighbors of a mesh, this would be identical to specifying a vertex list. An alternative to specifying all adjacent neighbors is to direct edges of the mesh. In this case only those edges which point outward from a vertex are listed. In the next section, it will be shown that an *out-degree list* can be constructed for planar meshes by directing a graph such that no vertex has more than three outgoing edges. This is asymptotically optimal. The extension of the out-degree structure to three dimensions is not straightforward and algorithms for obtaining optimal edge direction for nonplanar graphs are still under development.

The last structure considered here is the *quad-edge structure* proposed by Guibas and Stolfi [1], see figure 1.2(d). Each edge is stored as pair of directed edges. Each of the directed edges stores its origin and pointers to the next and previous directed edge of the region to its left. The quad-edge structure is extremely useful in grid generation where distinctions between topological and geometrical aspects are sometimes crucial. The structure has been extended to three dimensional arrangements by Dobkin and Laslo [2] and Brisson [3].

2.0 Some Basic Graph Operations Important in CFD

Implementation of unstructured grid methods on differing computer architectures has stimulated research in exploiting properties and characterizations of planar and nonplanar graphs. For example in Hammond and Barth [4], we exploited a recent theorem by Chrobak and Eppstein [5] concerning directed graphs with minimum out-degree. In this section, we review this result as well as presenting other basic graph operations that are particularly useful in CFD. Some of these algorithms are specialized to planar graphs (2-D meshes) while others are very general and apply in any number of space dimensions.

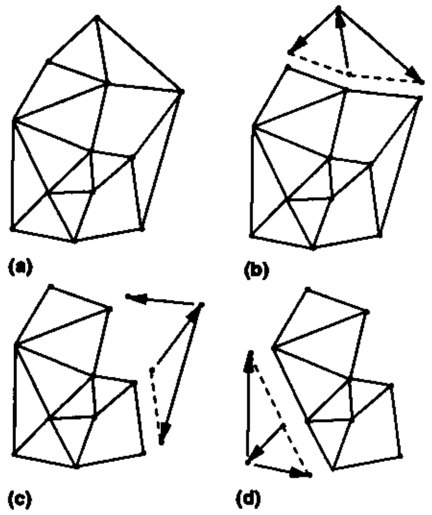
2.1 Planar Graphs with Minimum Out-Degree

Theorem: *Every planar graph has a 3-bounded orientation*, [5].

In other words, each edge of a planar graph can be assigned a direction such that the maximum number of edges pointing outward from any vertex is less than or equal to three. A constructive proof is given in ref.[5] consisting of the following steps. The first step is to find a *reduceable* boundary vertex. A reduceable boundary vertex is any vertex on the boundary with incident exterior (boundary) edges that connect to at most two other boundary vertices and any number of interior edges. Chrobak and Eppstein prove that reduceable vertices can always be found for arbitrary planar graphs. (In fact, two reduceable vertices can always be found!) Once a reduceable vertex is found then the two edges connecting to the other boundary vertices are directed *outward* and the remaining edges are always directed *inward*. These directed edges are then removed from the graph, see Figures 2.0(a-j). The process is then repeated on the remaining graph until no more edges remain. The algorithm shown pictorially in figure 2.0 is summarized in the following steps:

Algorithm: Orient a Graph with maximum out-degree ≤ 3.

Step 1. Find reduceable boundary vertex.
Step 2. Direct exterior edges outward and interior edges inward.
Step 3. Remove directed edges from graph.
Step 4. If undirected edges remain, go to step 1



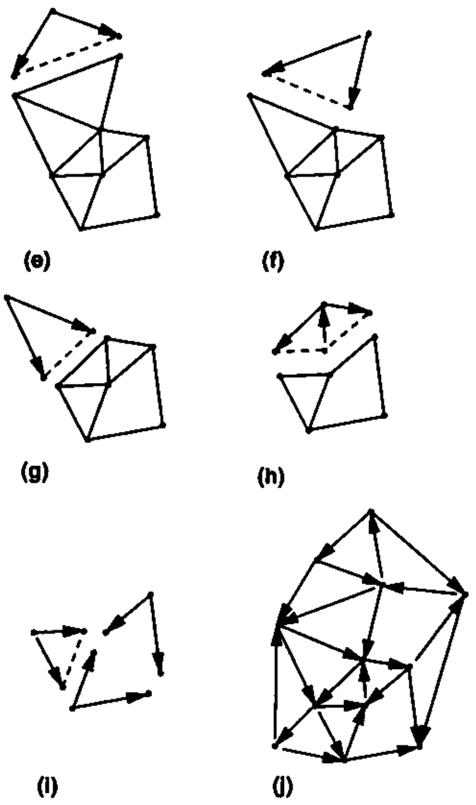


Figure 2.0 (a-i) Mesh orientation procedure with out-degree 3, (j) final oriented triangulation.

Linear time algorithms are given in [5]. In the paper by Hammond and Barth, we exploit the out-degree property to provide optimal load balancing on a massively parallel computer. Details are given in a later section.

2.2 Graph Ordering Techniques

The particular ordering of solution unknowns can have a marked influence on the amount of computational effort and memory storage required to solve large sparse linear systems and eigenvalue problems. In many algorithms, the band width and/or profile of the matrix determines the amount of computation and memory required. Most meshes obtained from grid generation codes have very poor natural orderings. Figures 2.1 and 2.2 show a typical mesh generated about a multi-component airfoil and the nonzero entries associated with the "Laplacian" of the graph. The Laplacian of a graph would represent the nonzero entries due to a discretization which involves only adjacent neighbors of the mesh. Figure 2.2 indicates that the band width of the natural ordering is almost equal to the dimension of the matrix! In parallel computation, the ordering algorithms can be used as means for *partitioning* a mesh among processors of the computer. This will be addressed in the next section.

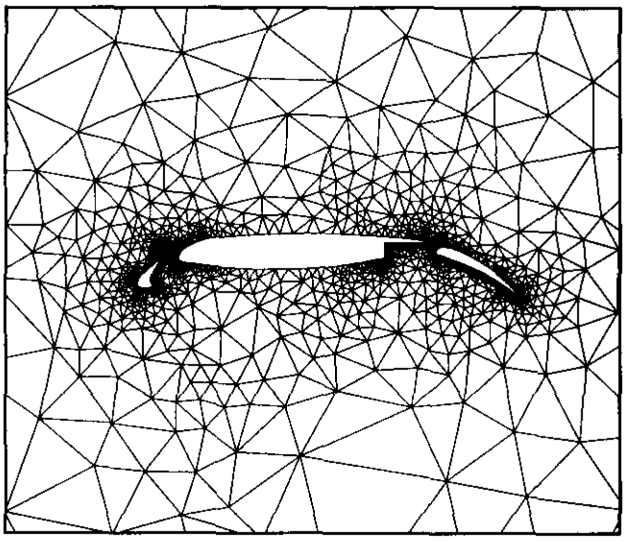


Figure 2.1 Typical Steiner triangulation about multi-component airfoil.

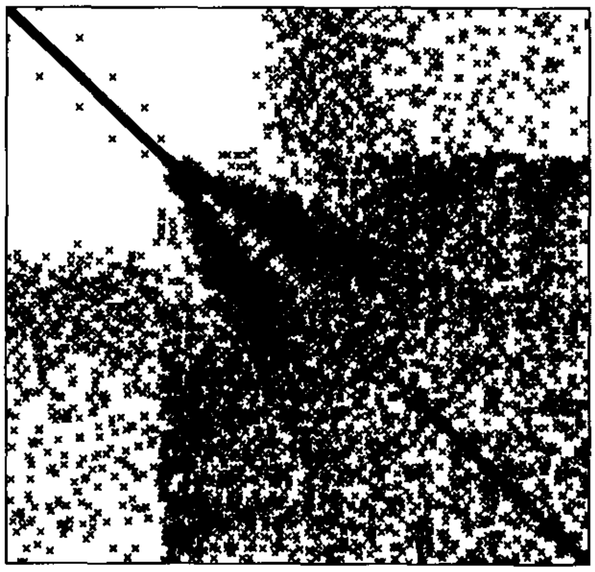


Figure 2.2 Nonzero entries of Laplacian matrix produced from natural ordering.

Several algorithms exist which construct new orderings by attempting to minimize the band width of a matrix or attempting to minimize the fill that occurs in the process matrix factorization. These algorithms usually rely on heuristics to obtain high efficiency, and do not usually obtain an optimum ordering. One example would be Rosen's algorithm [6] which iterates on the ordering to minimize the maximum band width.

Algorithm: Graph ordering, Rosen.

Step 1. Determine band width and the defining index pair (i, j) with $(i < j)$

Step 2. Does their exist an exchange which increases i or decreases j so that the band width is reduced? If so, exchange and go to step 1

Step 3. Does their exist an exchange which increases i or decreases j so that the band width remains the same? If so, exchange and go to step 1

This algorithm produces very good orderings but can be very expensive for large matrices. A popular method which is much less expensive for large matrices is the Cuthill-McKee [7] algorithm.

Algorithm: Graph ordering, Cuthill-McKee.

Step 1. Find vertex with lowest degree. This is the *root* vertex.

Step 2. Find all neighboring vertices connecting to the root by incident edges. Order them by increasing vertex degree. This forms level 1.

Step 3. Form level k by finding all neighboring vertices of level $k-1$ which have not been previously ordered. Order these new vertices by increasing vertex degree.

Step 4. If vertices remain, go to step 3

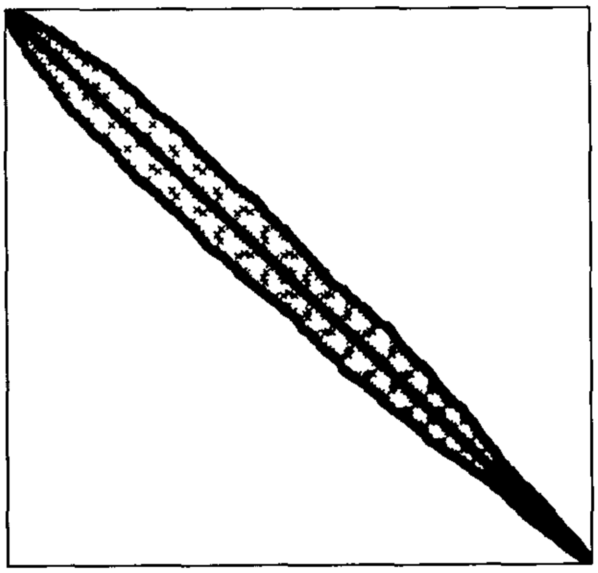


Figure 2.3 Nonzero entries of Laplacian matrix after Cuthill-McKee ordering.

The heuristics behind the Cuthill-McKee algorithm are very simple. In the graph of the mesh, neighboring vertices must have numberings which are near by, otherwise they will produce entries in the matrix with large band width. The idea of sorting elements among a given level is based on the heuristic that vertices with high degree should be given indices as large as possible so that they will be as close as possible to vertices of the *next* level generated. Figure 2.3 shows the dramatic improvement of the Cuthill-McKee ordering on the matrix shown in figure 2.2.

Studies of the Cuthill-McKee algorithm have shown that the profile of a matrix can be greatly reduced simply by reversing the ordering of the Cuthill-McKee algorithm, see George [8]. This amounts to a renumbering given by

$$k \rightarrow n - k + 1 \tag{2.1}$$

where n is the size of the matrix. While this does not change the bandwidth of the matrix, it can dramatically reduce the fill that occurs in Cholesky or L-U matrix factorization when compared to the original Cuthill-McKee algorithm.

2.3 Graph Partitioning for Parallel Computing

An efficient partitioning of a mesh for distributed memory machines is one that ensures an even distribution of computational workload among the processors and minimizes the amount of time spent in interprocessor communications. The former requirement is termed *load balancing*. For if the load were not evenly distributed, some processors will have to sit idle at synchronization points waiting for other processors to catch up. The second requirement comes from the fact that communication between processors takes time and it is not always possible to hide this latency in data transfer. In our parallel implementation of a finite-volume flow solver on unstructured grids, data for the nodes that lie on the boundary between two processors is exchanged, hence requiring a bidirectional data-transfer. On many systems, a synchronous exchange of data can yield a higher performance than when done asynchronously. To exploit this fact, edges of the communication graph are colored such that no vertex has more than one edge of a certain color incident upon it. A communication graph is a graph in which the vertices are the processors and an edge connects two vertices if the two corresponding processors share an interprocessor boundary. The colors in the graph represent separate communication cycles. For instance, the mesh partitioned amongst four processors as shown in figure 2.4(a), would produce the communication graph shown in figure 2.4(b).

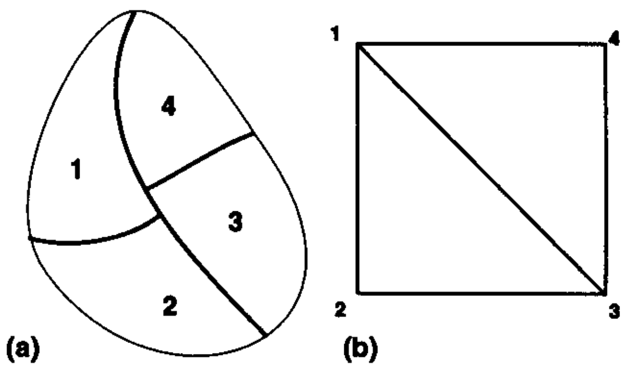


Figure 2.4 (a) Four partition mesh, (b) Communication graph.

The graph shown in figure 2.4(b) can be colored edgewise using three colors. For example, in the first communication cycle, processors $(1,4)$ could perform a synchronous data exchange as would processors $(2,3)$. In the second communication cycle, processors $(1,2)$ and $(3,4)$ would exchange and in the third cycle, processors $(1,3)$ would exchange while processors 2 and 4 sit idle. Vizing's theorem proves that any graph of maximum vertex degree Δ (number of edges incident upon a vertex) can be colored using n colors such that $\Delta \leq n \leq \Delta + 1$. Hence, any operation that calls for every processor to exchange data with its neighbors will require n communication cycles.

The actual cost of communication can often be accurately modeled by the linear relationship:

$$Cost = \alpha + \beta m \tag{2.2}$$

where α is the time required to initiate a message, β is the rate of data-transfer between two processors and m is the message length. For n messages, the cost would be

$$Cost = \sum_n (\alpha + \beta m_n). \tag{2.3}$$

This cost can be reduced in two ways. One way is to reduce Δ thereby reducing n. The alternative is to reduce the individual message lengths. The bounds on n are $2 \leq N \leq P-1$ for $P \geq 3$ where P is the total number of processors. Figure 2.5(a) shows the partitioning of a mesh which reduces Δ, and 2.5(b) shows a partitioning which minimizes the message lengths. For the mesh in figure 2.5(a), $\Delta = 2$ while in figure 2.5(b), $\Delta = 3$. However, the average message length for the partitions shown in figure 2.5(b) is about half as much as that in figure 2.5(a).

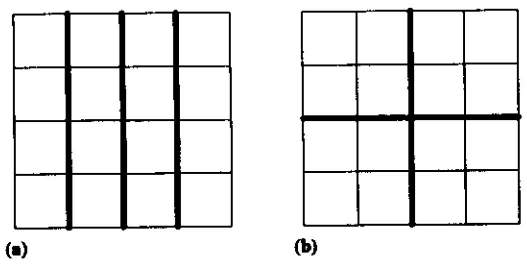


Figure 2.5 (a) Mesh partitioning with minimized Δ, (b) Mesh with minimizes message length.

In practice, it is difficult to partition an unstructured mesh while simultaneously minimizing the number and length of messages. In the following paragraphs, a few of the most popular partitioning algorithms which approximately accomplish this task will be discussed. All the algorithms discussed below: coordinate bisection, Cuthill-McKee, and spectral partitioning are evaluated in the paper by Venkatakrishnan, Simon, and Barth [9]. This paper evaluates the partitioning techniques within the confines of an explicit, unstructured finite-volume Euler solver. Spectral partitioning has been extensively studied by Simon [10]. The algorithms have also been implemented in three dimensions by A. Gandhi working in the CFD branch at NASA Ames Research Center.

Note that for the particular applications that we have in mind (a finite-volume scheme with solution unknowns at vertices of the mesh), it makes sense to partition the domain such that the separators correspond to edges of the mesh. Also note that the partitioning algorithms all can be implemented recursively. The mesh is first divided into two sub-meshes of nearly equal size. Each of these sub-meshes is subdivided into two more sub-meshes and the process in repeated until the desired number of partitions P is obtained (P is a integer power of 2). Since we desire the separator of the partitions to coincide with edges of the mesh, the division of a sub-mesh into two pieces can be viewed as a 2-coloring of *faces* of the sub-mesh. For the Cuthill-McKee and spectral partitioning techniques, this amounts to supplying these algorithms with the *dual* of the graph for purposes of the 2-coloring. The balancing of each partition is usually done cellwise; although an edgewise balancing is more appropriate in the present applications. Due to the recursive nature of partitioning, the algorithms outlined below represent only a single step of the process.

Coordinate Bisection

In the coordinate bisection algorithm, face centroids are sorted either horizontally or vertically depending of the current level of the recursion. A separator is chosen which balances the number of faces. Faces are colored depending on which side of the separator they are located. The actual edges of the mesh corresponding to the separator are characterized as those edges which have adjacent faces of different color, see figure 2.6. This partitioning is very efficient to create but gives sub-optimal performance on parallel computations owing to the long message lengths than can routinely occur.

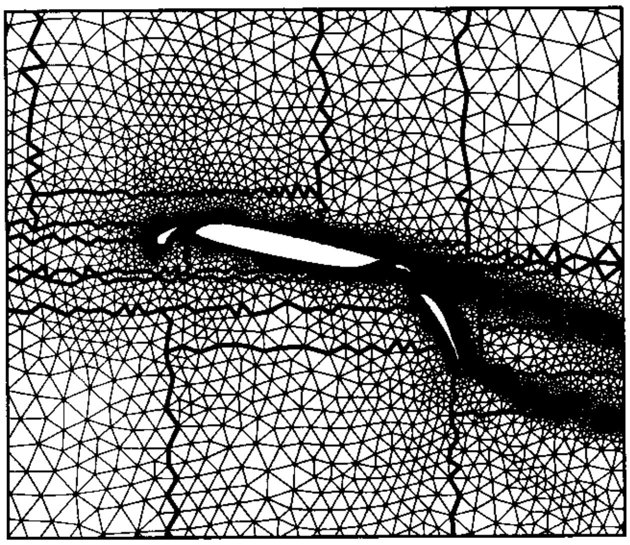


Figure 2.6 Coordinate bisection (16 partitions).

Cuthill-McKee

The Cuthill-McKee (CM) algorithm described earlier can also be used for recursive mesh partitioning. In this case, the Cuthill-McKee ordering is performed on the *dual* of the mesh graph. A separator is chosen either at the median of the ordering (which would balance the coloring of faces of the original mesh) or the separator is chosen at the level set boundary *closest* to the median as possible. This latter technique has the desired effect of reducing the number of disconnected sub-graphs that occur during the course of the partitioning. Figure 2.7 shows a Cuthill-McKee partitioning for the multi-component airfoil mesh. The Cuthill-McKee ordering tends to produce long boundaries because of the way that the ordering is propagated through a mesh. The maximum degree of the communication graph also tends to be higher using the Cuthill-McKee algorithm. The results shown in ref. [9] for multi-component airfoil grids indicate a performance on parallel computations which is slightly worse than the coordinate bisection technique.

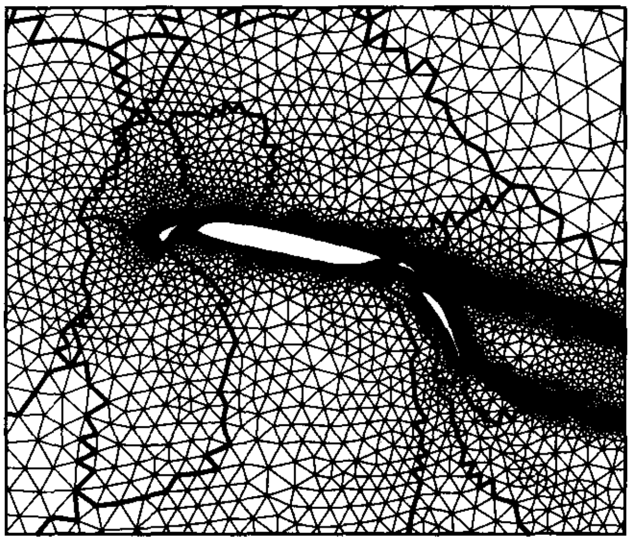


Figure 2.7 Cuthill-McKee partitioning of mesh.

Spectral Partitioning

The last partitioning considered is the spectral partitioning which exploits properties of the Laplacian $\mathcal{L}$ of a graph (defined below). The algorithm consists of the following steps:

Algorithm: Spectral Partitioning.

Step 1. Calculate the matrix $\mathcal{L}$ associated with the Laplacian of the graph (dual graph in the present case).

Step 2. Calculate the eigenvalues and eigenvectors of $\mathcal{L}$.

Step 3. Order the eigenvalues by magnitude, $\lambda_1 \leq \lambda_2 \leq \lambda_3 ... \lambda_N$.

Step 4. Determine the smallest nonzero eigenvalue, λ_f and its associated eigenvector $\mathbf{x}_f$ (the Fiedler vector).

Step 5. Sort elements of the Fiedler vector.

Step 6. Choose a divisor at the median of the sorted list and 2-color vertices of the graph (or dual) which correspond to elements of the Fielder vector less than or greater than the median value.

The spectral partitioning of the multi-component airfoil is shown in figure 2.8. In reference [9], we found that parallel computations performed slightly better on the spectral partitioning than on the coordinate bisection or Cuthill-McKee. The cost of the spectral partitioning is very high (even using a Lanczos algorithm to compute the eigenvalue problem). It has yet to be determined if the spectral partitioning will have practical merit.

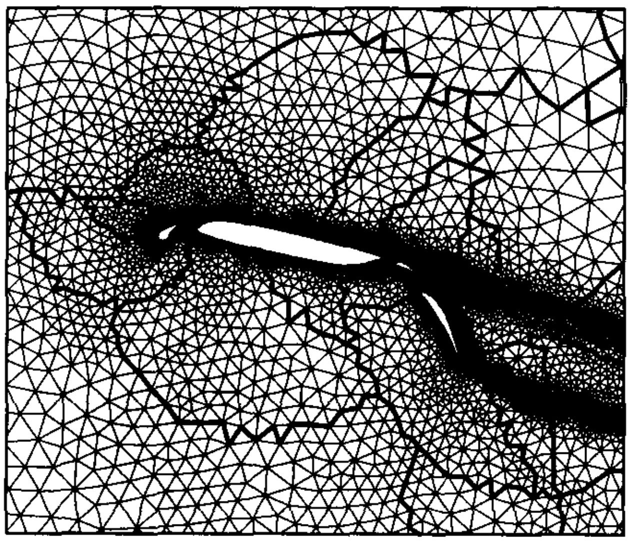


Figure 2.8 Spectral partitioning of multi- component airfoil.

The spectral partitioning exploits a peculiar property of the "second" eigenvalue of the Laplacian matrix associated with a graph. The Laplacian matrix of a graph is simply

$$\mathcal{L} = -\mathcal{D} + \mathcal{A}. \tag{2.4}$$

where $\mathcal{A}$ is the standard adjacency matrix

$$\mathcal{A}_{ij} = \begin{cases} 1 & e(v_i, v_j) \in G \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

and $\mathcal{D}$ is a diagonal matrix with entries equal to the degree of each vertex, $\mathcal{D}_i = d(v_i)$. From this definition, it should be clear that rows of $\mathcal{L}$ each sum to zero. Define an N-vector, $\mathbf{s} = [1, 1, 1, ...]^T$. By construction we have that

$$\mathcal{L}\mathbf{s} = 0. \tag{2.6}$$

This means that at least one eigenvalue is zero with $\mathbf{s}$ as an eigenvector.

The objective of the spectral partitioning is to divide the mesh into two partitions of equal size such that the number of edges cut by the partition boundary is approximately minimized.

Technically speaking, the smallest nonzero eigenvalue need not be the second. Graphs with disconnected regions will have more that one zero eigenvalue depending on the number of disconnected regions. For purposes of discussion, we assume that disconnected regions are not present, i.e. that λ_2 is the relevant eigenmode.

Elements of the proof:

Define a partitioning vector which 2-colors the vertices

$$\mathbf{p} = [+1, -1, -1, +1, +1, ..., +1, -1]^T \quad (2.7)$$

depending on the sign of elements of $\mathbf{p}$ and the one-to-one correspondence with vertices of the graph, see for example figure 2.9. Balancing the number of vertices of each color amounts to the requirement that

$$\mathbf{s} \perp \mathbf{p} \quad (2.8)$$

where we have assumed an even number of vertices.

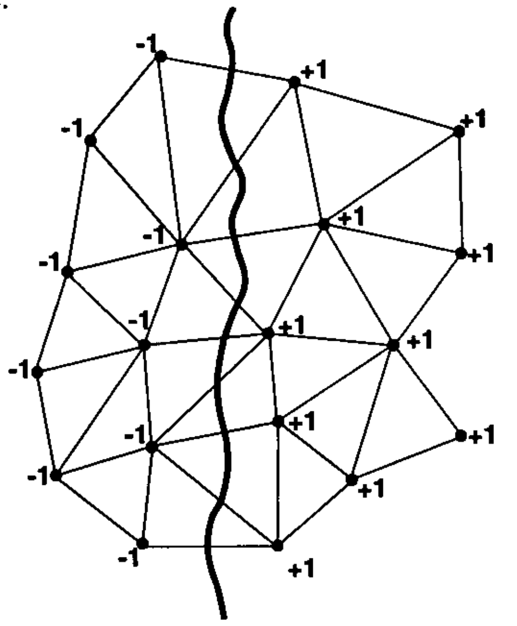


Figure 2.9 Arbitrary graph with 2-coloring showing separator and cut edges.

The key observation is that the number of cut edges, E_c, is precisely related to the L_1 norm of the Laplacian matrix multiplying the partitioning vector, i.e.

$$4E_c = \|\mathcal{L}\mathbf{p}\|_1 \quad (2.9)$$

which can be easily verified. The goal is to minimize cut edges. That is to find $\mathbf{p}$ which minimizes $\|\mathcal{L}\mathbf{p}\|_1$ subject to the constraints that $\|\mathbf{p}\|_1 = N$ and $\mathbf{s} \perp \mathbf{p}$. Since $\mathcal{L}$ is a real symmetric (positive semi-definite) matrix, it has a complete set of real eigenvectors which can be orthogonalized with each other. The next step of the proof would be to extend the domain of $\mathbf{p}$ to include real numbers (this introduces an inequality) and expand $\mathbf{p}$ in terms of the orthogonal eigenvectors.

$$\mathbf{p} = \sum_{i=1}^{n} c_i \mathbf{x}_i \quad (2.10)$$

By virtue of (2.6) we have that $\mathbf{x}_1 = \mathbf{s}$. It remains to be shown that $\|\mathcal{L}\mathbf{p}\|_1$ is minimized when $\mathbf{p} = \mathbf{p}' = n\mathbf{x}_2/\|\mathbf{x}_2\|_1$,i.e. when the Fiedler vector is used. Inserting this expression for $\mathbf{p}$ we have that

$$\|\mathcal{L}\mathbf{p}'\|_1 = n\lambda_2 \quad (2.11)$$

It is a simple matter to show that adding any other eigenvector component to $\mathbf{p}'$ while insisting that $\|\mathbf{p}\|_1 = N$ can only increase the L_1 norm. This would complete the proof. Figure 2.10 plots contours (level sets) of the Fiedler vector for the multi-component airfoil problem.

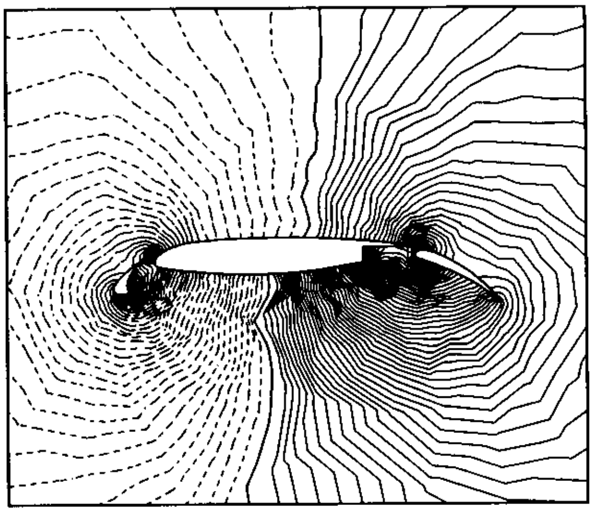


Figure 2.10. Contours of Fiedler Vector for Spectral Partitioning. Dashed lines are less than the median value.

3.0 Triangulation Methods

Although many algorithms exist for triangulating sites (points) in an arbitrary number of space dimensions, only a few have been used on practical problems. In particular, Delaunay triangulation has proven to be a very useful triangulation technique. This section will present some of the basic concepts surrounding Delaunay and related triangulations as well as discussing some of the most popular algorithms for constructing these triangulations. The discussion of the advancing front method of grid generation will be deferred to Professors Morgan and Löhner.

3.1 Voronoi Diagram and Delaunay Triangulation

Recall the definition of the Dirichlet tessellation in a plane. The Dirichlet tessellation of a point set is the pattern of convex regions, each being closer to some point P in the point set than to any other point in the set. These Dirichlet regions are also called Voronoi regions. The edges of Voronoi polygons comprise the Voronoi diagram, see figure 3.1. The idea extends naturally to higher dimensions.

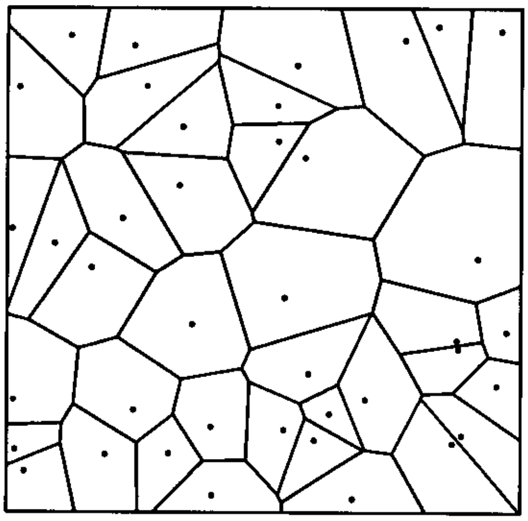


Figure 3.1 Voronoi diagram of 40 random sites.

Voronoi diagrams have a rich mathematical theory. ***The Voronoi diagram is believed to be one of the most fundamental constructs defined by discrete data.*** Voronoi diagrams have been independently discovered in a wide variety of disciplines. Computational geometricians have a keen interest in Voronoi diagrams. It is well known that Voronoi diagrams are related to convex hulls via stereographic projection. Point location in a Voronoi diagram can be performed in $O(\log(n))$ time with $O(n)$ storage for n regions. This is useful in solving post-office or related problems in optimal time. Another example of the Voronoi diagram which occurs in the natural sciences can be visualized by placing crystal "seeds" at random sites in 3-space. Let the crystals grow at the same rate in all directions. When two crystals collide simply stop their growth. The crystal formed for each site would represent that volume of space which is closer to that site than to any other site. This would effectively construct a Voronoi diagram. We now consider the role of Voronoi diagrams in Delaunay triangulation.

Definition: The Delaunay triangulation of a point set is defined as the dual of the Voronoi diagram of the set.

The Delaunay triangulation in two space dimensions is formed by connecting two points if and only if their Voronoi regions have a common border segment. If no four or more points are cocircular, then we have that ***vertices of the Voronoi are circumcenters of the triangles.*** This is true because vertices of the Voronoi represent locations that are equidistant to three (or more) sites. Also note that from the definition of duality, edges of the Voronoi are in one-to-one correspondence to edges of the Delaunay triangulation (ignoring boundaries). Because edges of the Voronoi diagram are the locus of points equidistant to two sites, each edge of the Voronoi diagram is perpendicular to the corresponding edge of the Delaunay triangulation. This duality extends to three dimensions in a straightforward way. The Delaunay triangulation possesses several alternate characterizations and many properties of importance. Unfortunately, not all of the two dimensional characterizations have three-dimensional extensions. To avoid confusion, properties and algorithms for construction of two dimensional Delaunay triangulations will be considered first. The remainer of this section will then discuss the three-dimensional Delaunay triangulation.

3.2 Properties of a 2-D Delaunay Triangulation

(1) *Uniqueness.* The Delaunay triangulation is unique. This assumes that no four sites are cocircular. The uniqueness follows from the uniqueness of the Dirichlet tessellation.

(2) *The circumcircle criteria.* A triangulation of $N \geq 2$ sites is Delaunay if and only if the circumcircle of every interior triangle is point-free. For if this was not true, the Voronoi regions associated with the dual would not be convex and the Dirichlet tessellation would be invalid. Related to the circumcircle criteria is the *incircle* test for four points as shown in figures 3.2-3.3.

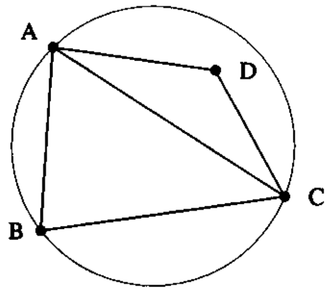


Figure 3.2 Incircle test for ΔABC and D (true).

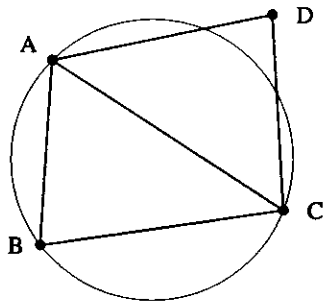


Figure 3.3 Incircle test for ΔABC and D (false).

This test is true if point D lies interior to the circumcircle of ΔABC which is equivalent to testing whether $\angle ABC + \angle CDA$ is less than or greater than $\angle BCD + \angle BAD$. More precisely we have that

$$\angle ABC + \angle CDA = \begin{cases} < 180^\circ & \text{incircle false} \\ 180^\circ & \text{A,B,C,D cocircular} \\ > 180^\circ & \text{incircle true} \end{cases} \tag{3.1}$$

Since interior angles of the quadrilateral sum to 360°, if the circumcircle of ΔABC contains D then swapping the diagonal edge from position $A-C$ into $B-D$ guarantees that the new triangle pair satisfies the circumcircle criteria. Furthermore, this process of diagonal swapping is local, i.e. it does not disrupt the Delaunayhood of any triangles adjacent to the quadrilateral.

(3)*Edge circle property.* A triangulation of sites is Delaunay if and only if there exists *some* circle passing through the endpoints of each and every edge which is point-free. This characterization is very useful because it also provides a mechanism for defining a *constrained* Delaunay triangulation where certain edges are prescribed *apriori*. A triangulation of sites is a constrained Delaunay triangulation if for each and every edge of the mesh there exists some circle passing through its endpoints containing no other site in the triangulation which is *visible* to the edge. In figure 3.4, site d is not visible to the segment a-c because of the constrained edge a-b.

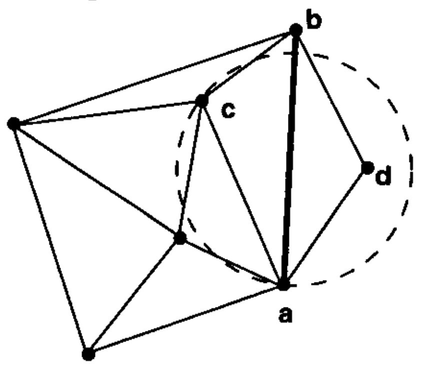


Figure 3.4 Constrained Delaunay triangulation. Site d is not visible to a-c due to constrained segment a-b.

(4)*Equiangularity property.* Delaunay triangulation maximizes the minimum angle of the triangulation. For this reason Delaunay triangulation often called the MaxMin triangulation. This property is also locally true for all adjacent triangle pairs which form a convex quadrilateral. This is the basis for the local edge swapping algorithm of Lawson [11] described below.

(5)*Minimum Containment Circle.* A recent result by Rajan [12] shows that the Delaunay triangulation minimizes the maximum containment circle over the entire triangulation. The containment circle is defined as the smallest circle enclosing the three vertices of a triangle. This is identical to the circumcircle for acute triangles and a circle with diameter equal to the longest side of the triangle for obtuse triangles (see figure 3.5).

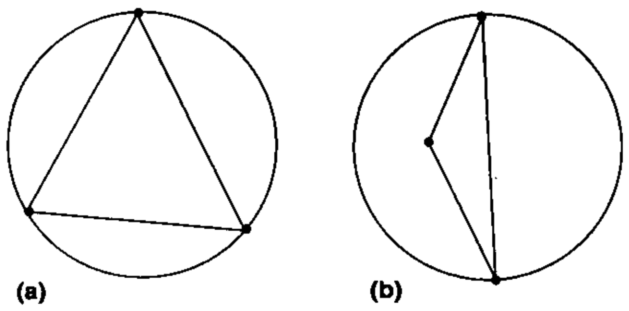


Figure 3.5 Containment circles for acute and obtuse triangles.

This property extends to n dimensions. Unfortunately, the result does not hold lexicographically.

(6)*Nearest neighbor property.* An edge formed by joining a vertex to its nearest neighbor is an edge of the Delaunay triangulation. This property makes Delaunay triangulation a powerful tool in solving the closest proximity problem. Note that the nearest neighbor edges do not describe all edges of the Delaunay triangulation.

(7)*Minimal roughness.* The Delaunay triangulation is a minimal roughness triangulation for arbitrary sets of scattered data, Rippa [13]. Given arbitrary data f_i at all vertices of the mesh and a triangulation of these points, a unique piecewise linear interpolating surface can be constructed. The Delaunay triangulation has the property that of all triangulations it minimizes the roughness of this surface as measured by the following Sobolev semi-norm:

$$\int_T \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right] dx\, dy \tag{3.2}$$

This is a interesting result as it does not depend on the actual form of the data. This also indicates that Delaunay triangulation approximates well those functions which minimize this Sobolev norm. One example would be the harmonic functions satisfying Laplace's equation with suitable boundary conditions which minimize exactly this norm. In a later section, we will prove that a Delaunay triangulation guarantees a maximum principle for the discrete Laplacian approximation (with linear elements).

3.3 Algorithms for 2-D Delaunay Triangulation

We now consider several techniques for Delaunay triangulation in two dimensions. These methods were chosen because they perform optimally in rather different situations. The discussion of the 2-D algorithms is organized as follows:

(a) Incremental Insertion Algorithms
- (i) Bowyer algorithm
- (ii) Watson algorithm
- (iii) Green and Sibson algorithm

(b) Divide and Conquer Algorithm

(c) Tanemura/Merriam Algorithm

(d) Global Edge Swapping (Lawson)

It should be pointed out that there appears to be some confusion in the CFD literature concerning the Bowyer[14] and Watson[15] algorithms. What is sometimes described as Bowyer's algorithm is actually Watson's algorithm. This is surprising since both the Bowyer and Watson algorithms appeared as back-to-back articles in the same journal! The fundamental difference (as we will see) is that the Bowyer algorithm is implemented in the *Voronoi* plane and the Watson algorithm is implemented in the *triangulation* plane.

3.3a Incremental Insertion Algorithms

For simplicity, assume that the site to be added lies within a bounding polygon of the existing triangulation. If we desire a triangulation from a new set of sites, three initial phantom points can always be added which define a triangle large enough to enclose all points to be inserted. In addition, interior boundaries are usually temporarily ignored for purposes of the Delaunay triangulation. After completing the triangulation, spurious edges are then deleted as a postprocessing step. Incremental insertion algorithms begin by inserting a new site into an existing Delaunay triangulation. This introduces the task of point location in the triangulation. Some incremental algorithms require knowing which triangle the new site falls within. Other algorithms require knowing *any* triangle whose circumcircle contains the new site. In either case, two extremes arise in this reguard. Typical mesh adaptation and refinement algorithms determine the particular cell for site insertion as part of the mesh adaptation algorithm, thereby reducing the burden of point location. In the other extreme, initial triangulations of randomly distributed sites usually require advanced searching techniques for point location to achieve asymptotically optimal complexity $O(N \log N)$. Search algorithms based on quad-tree and split-tree data structures work extremely well in this case. Alternatively, search techniques based on "walking" algorithms are frequently used because of their simplicity. These methods work extremely well when successively added points are close together. The basic idea is start at the location in the mesh of the previously inserted point and move one edge (or cell) at a time in the general direction of the newly added point. In the worst case, each point insertion requires $O(N)$ walks. This would result in a worst case overall complexity $O(N^2)$. For randomly distributed points, the average point insertion requires $O(N^{\frac{1}{2}})$ walks which gives an overall complexity $O(N^{\frac{3}{2}})$. For many applications where successive points tend to be close together, the number of walks is roughly constant and these simple algorithms can be very competitive. Using any of these techniques, we can proceed with the insertion algorithms.

Bowyer's algorithm

The basic idea in Bowyer's algorithm is to insert a new site into an existing Voronoi diagram (for example site Q in figure 3.6), determine its territory (dashed line in figure 3.6), delete any edges completely contained in the territory, then add new edges and reconfigure existing edges in the diagram. The following is Bowyer's algorithm essentially as presented by Bowyer (see reference [14] for complete details):

Algorithm: Incremental Delaunay triangulation, Bowyer [14].

Step 1. Insert new point (site) Q into the Voronoi diagram.
Step 2. Find any existing vertex in the Voronoi diagram closer to the new point than to its forming points. This vertex will be deleted in the new Voronoi diagram.
Step 3. Perform tree search to find remaining set of deletable vertices $\mathcal{V}$ that are closer to the new point than to their forming points. (In figure 3.6 this would be the set $\{v_3, v_4, v_5\}$)
Step 4. Find the set $\mathcal{P}$ of forming points corresponding to the deletable vertices. In figure 3.6, this would be the set $\{p_2, p_3, p_4, p_5, p_7\}$.
Step 5. Delete edges of the Voronoi which can be described by pairs of vertices in the set $\mathcal{V}$ if both forming points of the edges to be deleted are contained in $\mathcal{P}$

Step 6. Calculate the new vertices of the Voronoi, compute their forming points and neighboring vertices, and update the Voronoi data structure.

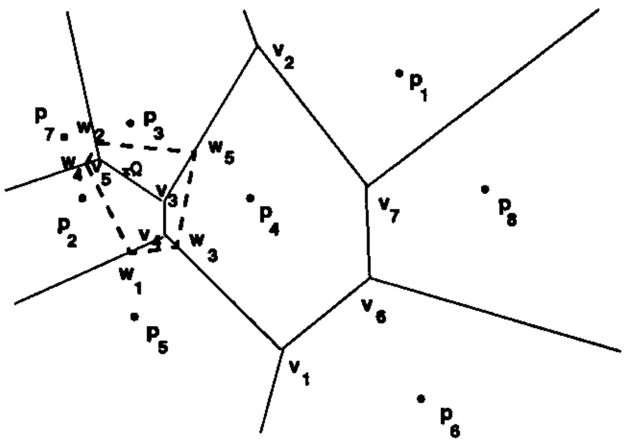


Figure 3.6 Voronoi diagram modified by Bowyer.

Implementational details and suggested data structures are given in the paper by Bowyer.

Watson's algorithm

Implementation of the Watson [15] algorithm is relatively straightforward. The first step is to insert a new site into an existing Delaunay triangulation and to find *any* triangle (the root) such that the new site lies interior to that triangles circumcircle. Starting at the root, a tree search is performed to find all triangles with circumcircle containing the new site. This is accomplished by recursively checking triangle neighbors. (The resulting set of deletable triangles violating the circumcircle criteria is independent of the starting root.) Removal of the deletable triangles exposes a polygonal cavity surrounding site Q with all the vertices of the polygon visible to site Q. The interior of the cavity is then retriangulated by connecting the vertices of the polygon to site Q, see figure 3.7(b). This completes the algorithm. A thorough account of Watson's algorithm is given by Baker [16] where he considers issues associated with constrained triangulations.

Algorithm: Incremental Delaunay triangulation, Watson [15].

Step 1. Insert new site Q into existing Delaunay triangulation.
Step 2. Find any triangle with circumcircle containing site Q.
Step 3. Perform tree search to find remaining set of deletable triangles with circumcircle containing site Q.
Step 4. Construct list of edges associated with deletable triangles. Delete all edges from the list that appear more that once.
Step 5. Connect remaining edges to site Q and update Delaunay data structure.

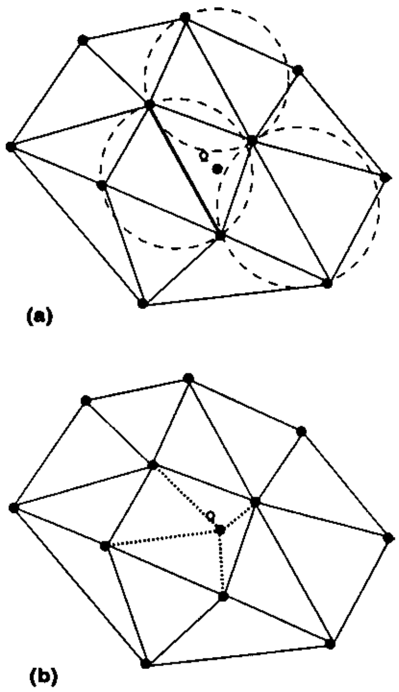
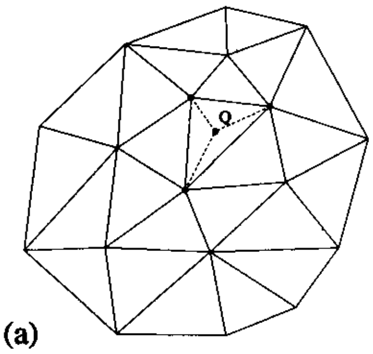


Figure 3.7 (a) Delaunay triangulation with site Q added. (b) Triangulation after deletion of invalid edges and reconnection.

Green and Sibson algorithm

The algorithm due to Green and Sibson [17] is very similar to the Watson algorithm. The primary difference is the use of local edge swapping to reconfigure the triangulation. The first step is location, i.e. find the triangle containing point Q. Once this is done, three edges are then created connecting Q to the vertices of this triangle as shown in figure 3.8(a). If the point falls on an edge, then the edge is deleted and four edges are created connecting to vertices of the newly created quadrilateral. Using the circumcircle criteria it can be shown that the newly created edges (3 or 4) are automatically Delaunay. Unfortunately, some of the original edges are now incorrect. We need to somehow find all "suspect" edges which could possibly fail the circle test. Given that this can be done (described below), each suspect edge is viewed as a diagonal of the quadrilateral formed from the two adjacent triangles. The circumcircle test is applied to either one of the two adjacent triangles of the quadrilateral. If the fourth point of the quadrilateral is interior to this circumcircle, the suspect edge is then swapped as shown in figure 3.8(b), two more edges then become suspect. At any given time we can immediately identify all suspect edges. To do this, first consider the subset of all triangles which share Q as a vertex. One can guarantee at all times that all initial edges incident to Q are Delaunay and any edge made incident to Q by swapping must be Delaunay. Therefore, we need only consider the remaining edges of this subset which form a polygon about Q as suspect and subject to the incircle test. The process terminates when all suspect edges pass the circumcircle test.



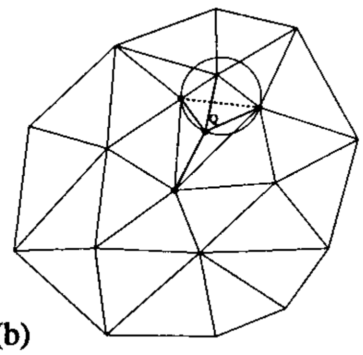


Figure 3.8 (a) Inserting of new vertex, (b) Swapping of suspect edge.

The algorithm can be summarized as follows:

Algorithm: Incremental Delaunay Triangulation, Green and Sibson [17]
Step 1. Locate existing cell enclosing point Q.
Step 2. Insert site and connect to 3 or 4 surrounding vertices.
Step 3. Identify suspect edges.
Step 4. Perform edge swapping of all suspect edges failing the incircle test.
Step 5. Identify new suspect edges.
Step 6. If new suspect edges have been created, go to step 3.

The Green and Sibson algorithm can be implemented using standard recursive programming techniques. The heart of the algorithm is the recursive procedure which would take the following form for the configuration shown in figure 3.9:

```
procedure swap[ $v_q$, $v_1$, $v_2$, $v_3$, $edges$]
if($incircle[v_q,v_1,v_2,v_3]$ = TRUE)then
    call reconfig_edges[$v_q$, $v_1$, $v_2$, $v_3$, $edges$]
    call swap[$v_q$, $v_1$, $v_4$, $v_2$, $edges$]
    call swap[$v_q$, $v_2$, $v_5$, $v_3$, $edges$]
endif
endprocedure
```

This example illustrates an important point. The nature of Delaunay triangulation guarantees that any edges swapped incident to Q will be final edges of the Delaunay triangulation. This means that we need only consider *forward propagation* in the recursive procedure. In a later section, we will consider incremental insertion and edge swapping for generating non-Delaunay triangulations based on other swapping criteria. This algorithm can also be programmed recursively but requires *backward propagation* in the recursive implementation. For the Delaunay triangulation algorithm, the insertion algorithm would simplify to the following three steps:

Recursive Algorithm: Incremental Delaunay Triangulation, Green and Sibson
Step 1. Locate existing cell enclosing point Q.
Step 2. Insert site and connect to surrounding vertices.
Step 3. Perform recursive edge swapping on newly formed cells (3 or 4).

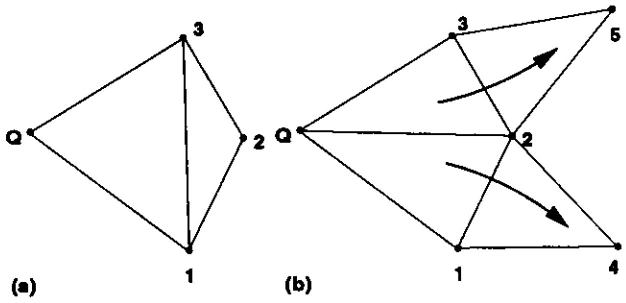


Figure 3.9 Edge swapping with forward propagation.

3.3b Divide-and-Conquer Algorithm

In this algorithm, the sites are assumed to be *prespecified.* The idea is to partition the cloud of points T (sorted along a convenient axis) into left (L) and right (R) half planes. Each half plane is then recursively Delaunay triangulated. The two halves must then be merged together to form a single Delaunay triangulation. Note that we assume that the points have been sorted along the x-axis for purposes of the following discussion (this can be done with $O(N \log N)$ complexity).

Algorithm: Delaunay Triangulation via Divide-and-Conquer
Step 1. Partition T into two subsets T_L and T_R of nearly equal size.
Step 2. Delaunay triangulate T_L and T_R recursively.
Step 3. Merge T_L and T_R into a single Delaunay triangulation.

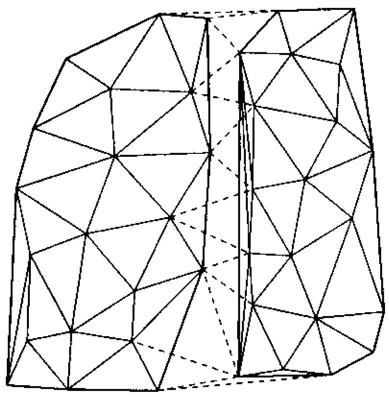


Figure 3.10 Triangulated subdivisions.

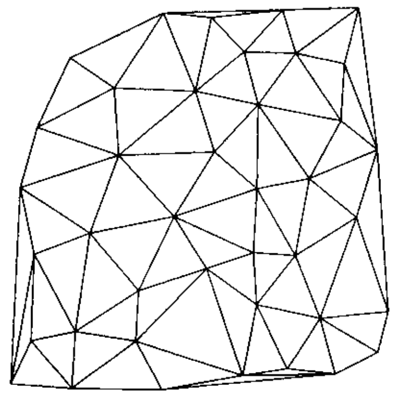


Figure 3.11 Triangulation after merge.

The only difficult step in the divide-and-conquer algorithm is the merging of the left and right triangulations. The process is simplified by noting two properties of the merge:

(1) *Only cross edges (L-R or R-L) are created in the merging process.* Since vertices are neither added or deleted in the merge process, the need for a new R-R or L-L edge indicates that the original right or left triangulation was defective. (Note that the merging process will require the deletion of edges L-L and/or R-R.)

(2) *Vertices with minimum (maximum) y value in the left and right triangulations always connect as cross edges.* This is obvious given that the Delaunay triangulation produces the convex hull of the point cloud.

Given these properties we now outline the "rising bubble" [1] merge algorithm. This algorithm produces cross edges in ascending y-order. The algorithm begins by forming a cross edge by connecting vertices of the left and right triangulations with minimum y value (property 2). This forms the initial cross edge for the rising bubble algorithm. More generally consider the situation in which we have a cross edge between A and B and all edges incident to the points A and B with endpoints above the half plane formed by a line passing through $A - B$, see figure 3.12.

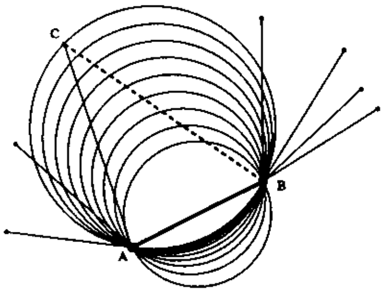


Figure 3.12 Circle of increasing radius in rising bubble algorithm.

This figure depicts a continuously transformed circle of increasing radius passing through the points A and B. Eventually the circle increases in size and encounters a point C from the left or right triangulation (in this case, point C is in the left triangulation). A new cross edge (dashed line in figure 3.12) is then formed by connecting this point to a vertex of $A - B$ in the other half triangulation. Given the new cross edge, the process can then be repeated and terminates when the top of the two meshes is reached. The deletion of $L - L$ or $R - R$ edges can take place during or after the addition of the cross edges. Properly implemented, the merge can be carried out in linear time, $O(N)$. Denoting $T(N)$ as the total running time, step 2 is completed in approximately $2T(N/2)$. Thus the total running time is described by the recurrence $T(N) = 2T(N/2) + O(N) = O(N \log N)$.

3.3c Tanemura/Merriam Algorithm

Another algorithm for performing Delaunay triangulation is the advancing front method developed by Tanemura, Ogawa, and Ogita [18] and later rediscovered by Merriam [19]. Here the idea is to start with a known boundary edge and form a new triangle by joining both endpoints to one of the interior points. This may generate up to two additional edges, providing they aren't already part of another triangle. After all the boundary edges have been incorporated into triangles, the new edges will appear to be a (somewhat ragged) boundary. This moving boundary is often called an advancing front. The process continues until the front vanishes. The problem here is to make the triangulation Delaunay. This can be done by taking advantage of the *incircle* property; the circumcircle of a Delaunay triangle contains no other points. This allows the appropriate point to be selected iteratively as shown in Fig. 3.13.

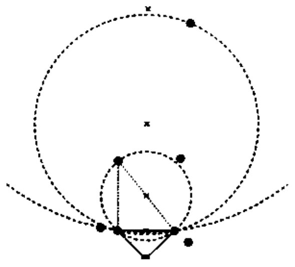


Figure 3.13 A straightforward iteration procedure selects the node which generates the smallest circumcircle for a given edge. The absence of nodes inside the circumcircle establishes convergence.

The iteration begins by selecting any node which is on the desired side of the given edge. If there are no such nodes, the given edge is part of a convex hull. Next, the circumcircle is constructed which passes through the edge endpoints and the selected node. Finally, check for nodes inside this circle. If there are any, replace the selected node with the node closest to the circumcenter and repeat the process. When the circumcircle is empty of nodes, connect the edge endpoints to the selected node.

3.3d Delaunay Triangulation Via Edge Swapping

This algorithm due to Lawson [11] assumes that a triangulation exists (not Delaunay) then makes it Delaunay through application of edge swapping such that the equiangularity of the triangulation increases. The equiangularity of a triangulation, $A(T)$, is defined as the ordering of angles $A(T) = [\alpha_1, \alpha_2, \alpha_3, ..., \alpha_{3n(c)_3}]$ such that $\alpha_i \leq \alpha_j$ if $i < j$. We write $A(T^*) < A(T)$ if $\alpha_j^* \leq \alpha_j$ and $\alpha_i^* = \alpha_i$ for $1 \leq i < j$. A triangulation T is globally equiangular if $A(T^*) \leq A(T)$ for all triangulations T^* of the point set. Lawson's algorithm examines all interior edges of the mesh. Each of these edges represents the diagonal of the quadrilateral formed by the union of the two adjacent triangles. In general one must first check if the quadrilateral is convex so that a potential diagonal swapping can place without edge crossing. If the quadrilateral is convex then the diagonal position is chosen which optimizes a local criterion (in this case the local equiangularity). This amounts to maximizing the minimum angle of the two adjacent triangles. Lawson's algorithm continues until the mesh is locally optimized and locally equiangular everywhere. It is easily shown that the condition of local equiangularity is equivalent to satisfaction of the incircle test described earlier. Therefore a mesh which is locally equiangular everywhere is a Delaunay triangulation. Note that each new edge swapping (triangulation T^*) insures that the global equiangularity increases $A(T^*) > A(T)$. Because the triangulation is of finite dimension, this guarantees that the process will terminate in a finite number of steps.

Iterative Algorithm: Triangulation via Lawson's Algorithm

```
swapedge = true
While(swapedge)do
  swapedge = false
  Do (*all interior edges*)
    If (*adjacent triangles form convex quad*)then
      *Swap diagonal to form* $T^*$.
      If (*optimization criteria satisfied*)then
        $T = T^*$
        swapedge = true
      EndIf
    EndIf
  EndDo
EndWhile
```

When Lawson's algorithm is used for constructing Delaunay triangulations, the test for quadrilateral convexity is not needed. It can be shown that nonconvex quadrilaterals formed from triangle pairs *never* violate the circumcircle test. When more general optimization criteria is used (discussed later), the convex check must be performed.

3.4 Other 2-D Triangulation Algorithms

In this section, other algorithms which do not necessarily produce Delaunay triangulations are explored.

The MinMax Triangulation

As Babuška and Aziz [22] point out, from the point of view of finite elements the MaxMin (Delaunay) triangulation is not essential. What is essential is that no angle be too close to 180°. In other words, triangulations which minimize the maximum angle are more desirable. These triangulations are referred to as MinMax triangulations. One way to generate a 2-D MinMax triangulations is via Lawson's edge swapping algorithm. In the case, the diagonal position for convex pairs of triangles is chosen which minimizes the maximum interior angle for both triangles. The algorithm is guaranteed to converge in a finite number of steps using arguments similar to Delaunay triangulation. Figures 3.14 and 3.15 present a Delaunay (MaxMin) and MinMax triangulation for 100 random points.

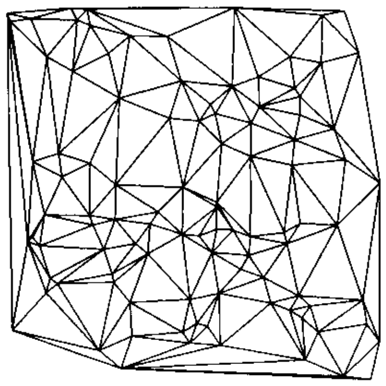


Figure 3.14 Delaunay Triangulation.

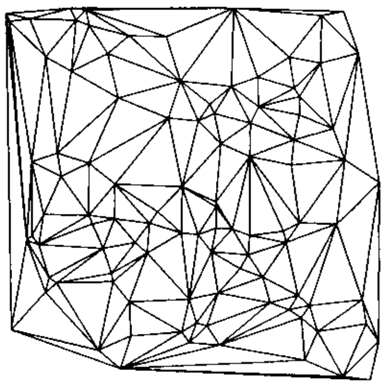


Figure 3.15 MinMax Triangulation.

Note that application of local MinMax optimization via Lawson's algorithm may only result in a mesh which is *locally* optimal and not necessarily at a global minimum. Attaining a globally optimal MinMax triangulation is a much more difficult task. The best algorithm to present date (Edelsbrunner, Tan, and Waupotitsch [23]) has a high complexity of $O(n^2 \log n)$. Wiltberger [24] has implemented a version of the Green and Sibson algorithm [17] which has been modified to produce locally optimal MinMax triangulations using incremental insertion and local edge swapping. The algorithm is implemented using recursive programming with complete forward and backward propagation (contrast figures 3.16 and 3.9). This is a necessary step to produce locally optimized meshes.

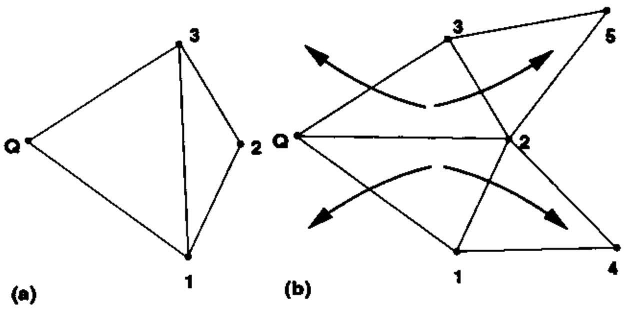


Figure 3.16 Edge swapping with forward and backward propagation in Wiltberger algorithm.

The MinMax triangulation has proven to be very useful in CFD. Figure 3.17 shows the Delaunay triangulation near the trailing edge region of an airfoil with extreme point clustering.

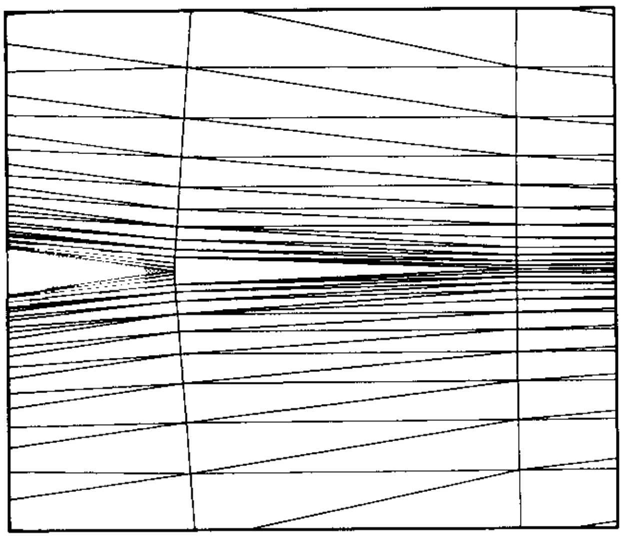


Figure 3.17 Delaunay triangulation near trailing edge of airfoil.

Upon first inspection, the mesh appears flawed near the trailing edge of the airfoil. Further inspection and extreme magnification near the trail edge of the airfoil (figure 3.18) indicates that the grid is a mathematically correct Delaunay triangulation. Because the Delaunay triangulation does not control the maximum angle, the cells near the trailing edge have angles approaching 180°. The presence of nearly collapsed triangles leaves considerable doubt as to the accuracy of any numerical solutions computed in the trailing edge region.

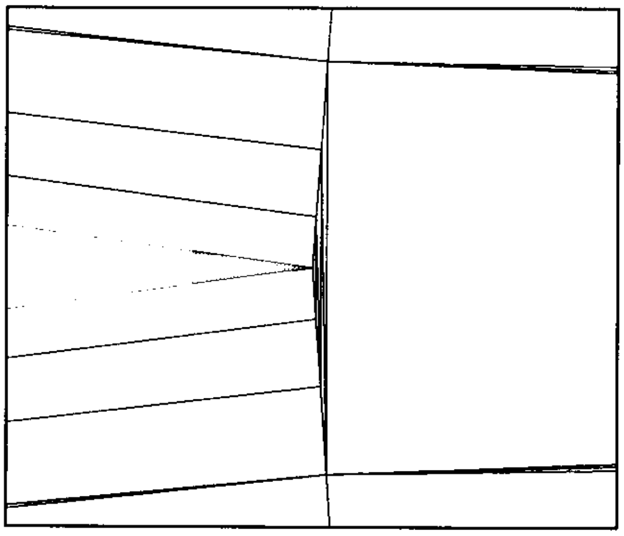


Figure 3.18 Extreme closeup of Delaunay triangulation near trailing edge of airfoil.

Edge swapping based on the MinMax criteria via Lawson's algorithm or incremental insertion using the Wiltberger algorithm produce the desired result as shown in figure 3.19.

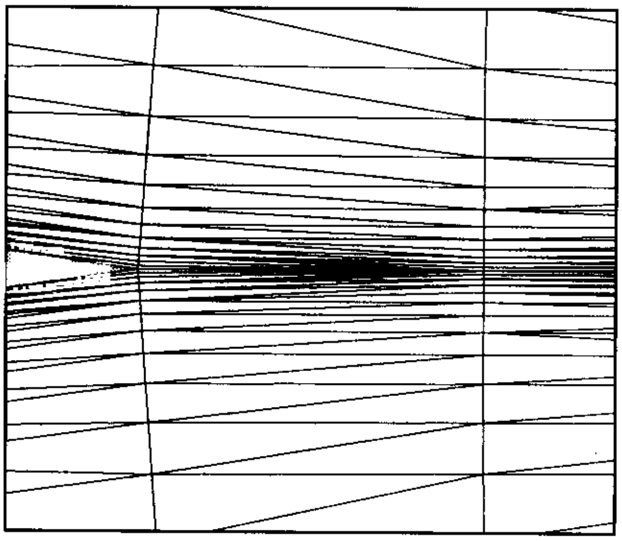


Figure 3.19 MinMax triangulation near trailing edge of airfoil.

The Greedy Triangulation

A greedy method is one that never undoes what it did earlier. The greedy triangulation continually adds edges compatible with the current triangulation (edge crossing not allowed) until the triangulation is complete, i.e. Euler's formula is satisfied. One objective of a triangulation might be to choose a set of edges with shortest total length. The best that the greedy algorithm can do is adopt a local criterion whereby only the shortest edge available at that moment is considered for addition to the current triangulation. (This does not lead to a triangulation with shortest total length.) Note that greedy triangulation easily accommodates constrained triangulations containing interior boundaries and a nonconvex outer boundary. In this case the boundary edges are simply listed first in the ordering of candidate edges. The entire algorithm is outlined below.

```
Algorithm: Greedy Triangulation
Step 1. Initialize triangulation $T$ as empty.
Step 2. Compute $\binom{n}{2}$ candidate edges.
Step 3. Order pool of candidate edges.
Step 4. Remove current edge $e_s$ from ordered pool.
Step 5. If( $e_s$ does not intersect edges of $T$ ) add $e_s$ to $T$
Step 6. If(Euler's formula not satisfied) go to Step 4.
```

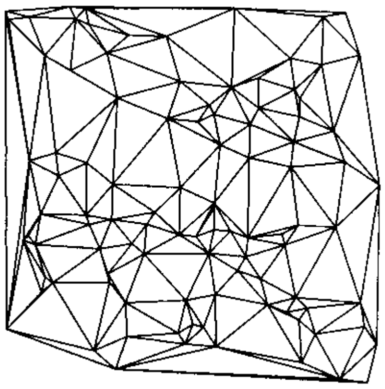


Figure 3.20 Greedy Triangulation.

Figures 3.14 and 3.20 contrast the Delaunay and greedy algorithm. The lack of angle control is easily seen in the greedy triangulation. The greedy algorithm suffers from both high running time as well as storage. In fact a naive implementation of Step 5. leads to an algorithm with $O(N^3)$ complexity. Efficient implementation techniques are given in Gilbert [25] with the result that the complexity can be reduced to $O(N^2 \log N)$ with $O(N^2)$ storage.

Data Dependent Triangulation

Unlike mesh adaptation, a data dependent triangulation assumes that the number and position of vertices is fixed and unchanging. Of all possible triangulations of these vertices, the goal is to find the best triangulation under data dependent constraints. In Nira, Levin, and Rippa [26], they consider several data dependent constraints together with piecewise linear interpolation. In order to determine if a new mesh is "better" than a previous one, a local cost function is defined for each interior edge. Two choices which prove to be particularly effective are the JND (Jump in Normal Derivatives) and the ABN (Angle Between Normals). Using their notation, consider an interior edge with adjacent triangles T_1 and T_2. Let $P(x,y)_1$ and $P(x,y)_2$ be the linear interpolation polynomials in T_1 and T_2 respectively:

$$P_1(x,y) = a_1x + b_1y + c_1$$

$$P_2(x,y) = a_2x + b_2y + c_2$$

The JND cost function measures the jump in normal derivatives of P_1 and P_2 across a common edge with normal components n_x and n_y.

$$s(f_T,e) = |n_x(a_1 - a_2) + n_y(b_1 - b_2)|, \quad \text{(JND cost function)}$$

The ABN measures the acute angle between the two normals formed from the two planes P_1 and P_2. Again using the notation of [26]:

$$s(f_T,e) = \theta = \cos^{-1}(A)$$

$$A = \frac{a_1a_2 + b_1b_2 + 1}{\sqrt{(a_1^2 + b_1^2 + 1)(a_2^2 + b_2^2 + 1)}}, \quad \text{(ABN cost function)}$$

The next step is to construct a global measure of these cost functions. This measure is required to decrease for each legal edge swapping. This insures that the edge swapping process terminates. The simplest measures are the l_1 and l_2 norms:

$$R_1(f_T) = \sum_{edges} |s(f_T,e)|$$

$$R_2(f_T) = \sum_{edges} s(f_T,e)^2$$

Recall that a Delaunay triangulation would result if the cost function is chosen which maximizes the minimum angle between adjacent triangles (Lawson's algorithm). Although it would be desirable to obtain a global optimum for all cost functions, this could be very costly in many cases. An alternate strategy is to abandon the pursuit of a globally optimal triangulation in favor of a locally optimal triangulation. Once again Lawson's algorithm is used. Note that in using Lawson's algorithm, we require that the global measure decrease at each edge swap. This is not as simple as before since each edge swap can have an effect on other surrounding edge cost functions. Nevertheless, this domain of influence is very small and easily found.

Iterative Algorithm: Data Dependent Triangulation via Modified Lawson's Algorithm

```
swapedge = true
While(swapedge)do
  swapedge = false
  Do (all interior edges)
    If (adjacent triangles
        form convex quadrilateral)then
       Swap diagonal to form $T^*$.
       If ($R(f_{T^*}) < R(f_T)$)then
         $T = T^*$
         swapedge = true
       EndIf
    EndIf
  EndDo
EndWhile
```

Edge swapping only occurs when $R(f_{T^*}) < R(f_T)$ which guarantees that the method terminates in a finite number of steps. Figures 3.14 and 3.21 plot the Delaunay triangulation of 100 random vertices in a unit square and piecewise linear contours of $(1 + \tanh(9y - 9x))/9$ on this mesh. The exact solution consists of straight line contours with unit slope.

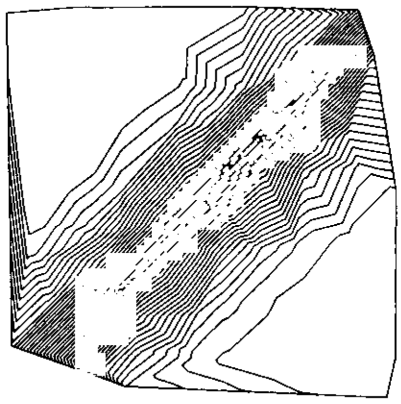


Figure 3.21 Piecewise Linear Interpolation of $(1 + \tanh(9y - 9x))/9$.

In figures 3.22 and 3.23 the data dependent triangulation and solution contours using the JND criteria and l_1 measure suggested in [26] are plotted.

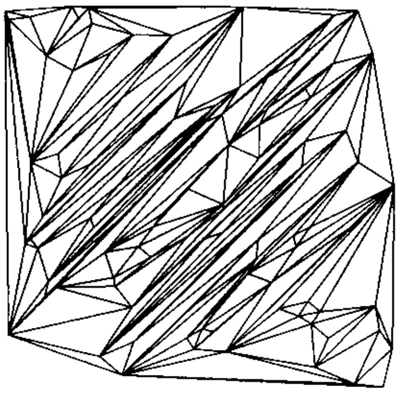


Figure 3.22. Data Dependent Triangulation.

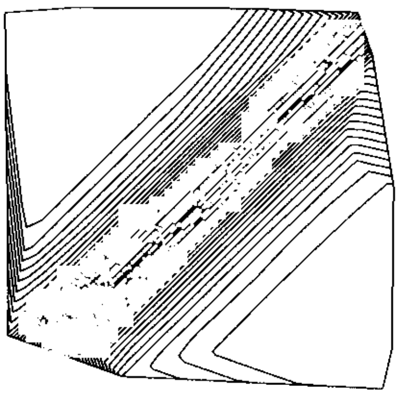


Figure 3.23 Piecewise Linear Interpolation of $(1 + \tanh(9y - 9x))/9$.

Note that the triangulations obtained from this method are not globally optimal and highly dependent on the order in which edges are accessed. Several possible ordering strategies are mentioned in [27].

3.5 2-D Steiner Triangulations

Definition: A Steiner triangulation is any triangulation that adds additional sites to an existing triangulation to improve some measure of grid quality.

Technically speaking, the method of advancing front grid generation discussed by Professors Morgan and Löhner in these notes would be a special type of Steiner triangulation. The insertion algorithms described earlier also provide a simple mechanism for generating Steiner triangulations. Holmes [28] demonstrated the feasibility of inserting sites at circumcenters of Delaunay triangles into an existing 2-D triangulation to improve measures of grid quality. This has the desired effect of placing the new site in a position that guarantees that no other site in the triangulation can lie closer that the radius of the circumcircle, see figure 3.24. In a loose sense, the new site is placed as far away from other nearby sites as conservatively possible.

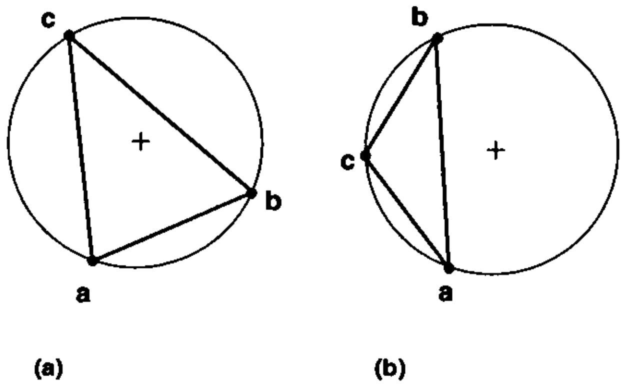


Figure 3.24 Inserting site at circumcenter of acute and obtuse triangles.

Warren *et al* [29] and Anderson [30] further demonstrated the utility of this type of Steiner triangulation in the generation and adaptive refinement of 2-D meshes. The algorithm developed by Wiltberger [24] also permits Steiner triangulations based on either MinMax or MaxMin (Delaunay) insertion. Only in the latter case is the insertion at triangle circumcenters truly justifiable. The paragraphs below give an expanded discussion of 2-D Steiner triangulation.

Steiner Grid Generation

The 2-D Steiner point grid generation algorithm described in [28,29,30] consists of the following steps. The first step is the Delaunay triangulation of the boundary data. Usually three or four points are placed in the far field with convex hull enclosing all the boundary points. Starting with a triangulation of these points, sites corresponding to boundary curves are incrementally inserted using Watson's algorithm in [28,29,30] and Green and Sibson's algorithm in [17] as shown in figure 3.25. The initial triangulation does not guarantee that all boundary edges are members of the triangulation. This can be remedied in a variety of ways. One technique adds additional points to the triangulation so as to guarantee that the resulting Delaunay triangulation contains all the desired boundary edges, see reference [16]. Another approach performs local edge swapping so as to produce a *constrained* Delaunay triangulation which guarantees that all boundary edges are actual edges of the triangulation.

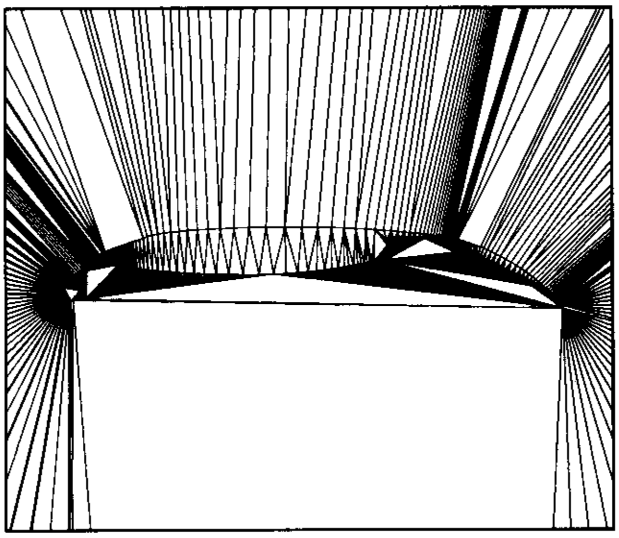


Figure 3.25 Initial triangulation of boundary points.

In either event, the boundary edges are marked so that they cannot be removed as the triangulation is refined. The algorithms described in [28,29,30] interrogate triangles in an arbitrary order (this makes the triangulation nonunique). The user must specify some measure of quality for triangle refinement (aspect ratio, area, containment circle radius, for example) and a threshold value for the measure. If a triangle fails to meet the threshold value, the triangulation is refined by placing a new site at the circumcenter of the failed triangle via Watson's algorithm. Some care must be taken to insure that measures are chosen which are guaranteed to be reduced when the refinement takes place. Using thresholding in this way does not give the user direct control over the actual number of triangle generated in the process of Steiner refinement. Wiltberger takes a different approach by maintaining a *dynamic heap* data structure of the quality measure. (Heap structures are a very efficient way of keeping a sorted list of entries with insertion and query time $O(\log N)$ for N entries.) The triangle with the largest value of the specified measure will be located at the top of the heap at all times during the triangulation. This makes implementation of a Steiner triangulation which *minimizes the maximum value* of the measure very efficient (and unique). In this implementation, the user can either specify the number of triangles to be generated or a threshold value of the measure. Note that multiple measures can be refined lexicographically. Figure 3.26 shows a Steiner triangulation using the Wiltberger algorithm with MaxMin insertion and refinement based on maximum aspect ratio.

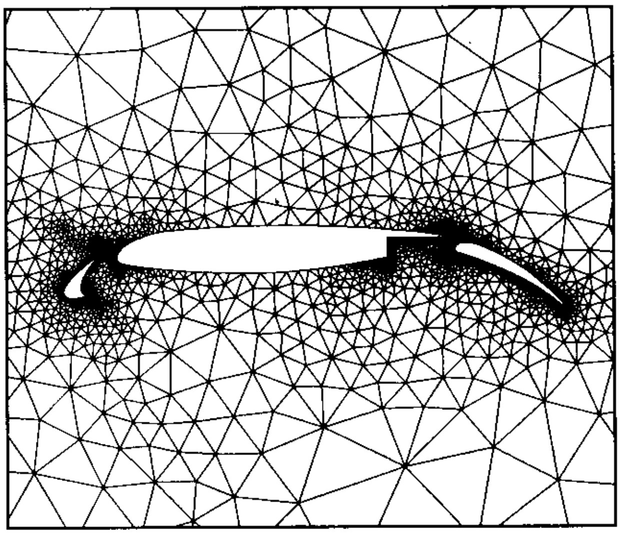


Figure 3.26 Steiner triangulation with sites inserted at circumcenters to reduce maximum cell aspect ratio.

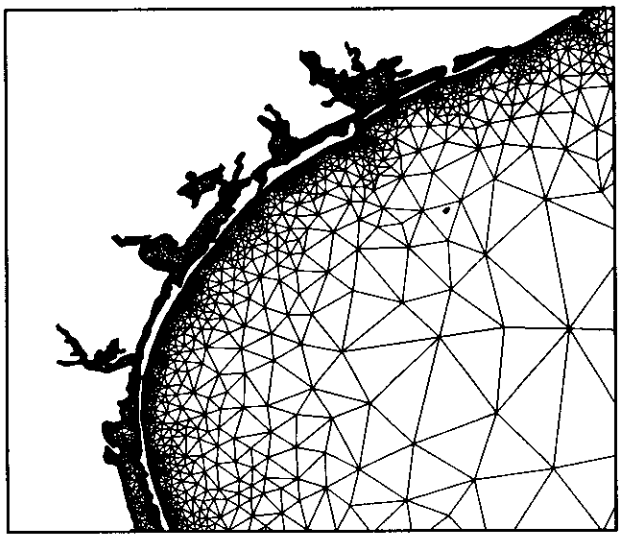


Figure 3.27 Steiner triangulation of Texas coast and the Gulf of Mexico.

This triangulation has proven to be very flexible. For instance, figure 3.27 shows a Steiner triangulation of the Texas coast and Gulf of Mexico.

3.6 Three-Dimensional Triangulations

The Delaunay triangulation extends naturally into three dimensions as the geometric dual of the 3-D Voronoi diagram. The Delaunay triangulation in 3-D can be characterized as the unique triangulation such that the circumsphere passing through the four vertices of any tetrahedron must not contain any other point in the triangulation. As in the 2-D case, the 3-D Delaunay triangulation has the property that it minimizes the maximum containment sphere (globally but not locally). In two dimensions, it can be shown that a mesh entirely comprised of acute triangles is automatically Delaunay. To prove this, consider an adjacent triangle pair forming a quadrilateral. By swapping the position of the diagonal it is easily shown that the minimum angle always increases. Rajan [12] shows the natural extension of this idea to three or more space dimensions. He defines a "self-centered" simplex in $\mathbf{R}^d$ to be a simplex which has the circumcenter of its circumsphere interior to the simplex. In two dimensions, acute triangles are self-centered and obtuse triangles are not. Rajan shows that a triangulation entirely composed of self-centered simplices in $\mathbf{R}^d$ is automatically Delaunay.

3.6a 3-D Bowyer and Watson Algorithms

The algorithms of Bowyer [14] and Watson [15] extend naturally to three dimensions with estimated complexities of $O(N^{5/3})$ and $O(N^{4/3})$ for N randomly distributed vertices. They do not give worst case estimates. It should be noted that in three dimensions, Klee [20] shows that the maximum number of tetrahedra which can be generated from N vertices is $O(N^2)$. Thus an optimal worst case complexity would be a least $O(N^2)$. Under normal conditions this worst case scenario is rarely encountered. Baker [16] reports more realistic actual run times for Watson's algorithm.

3.6b 3-D Edge Swapping Algorithms

Until most recently, the algorithm of Green and Sibson based on edge swapping was thought not to be extendable to three dimensions because it was unclear how to generalize the concept of edge swapping to three or more dimensions. In 1986, Lawson published a paper [21] in which he proved the fundamental combinatorial result:

Theorem: (Lawson, 1986) The convex hull of $d+2$ points in $\mathbf{R}^d$ can be triangulated in at most 2 ways.

Joe [31,32], and Rajan [12] have constructed algorithms based on this theorem. In joint work with A. Gandhi [33], we independently constructed an incremental Delaunay triangulation algorithm based on Lawson's theorem. The remainder of this section will review our algorithm and the basic ideas behind 3-D edge swapping algorithms.

It is useful to develop a taxonomy of possible configurations addressed by Lawson's theorem. Figure 3.28 shows configurations in 3-D of five points which can be triangulated in only one way and hence no change is possible. We call these arrangements "unswappable". Figure 3.29 shows configurations which allow two ways of triangulation. It is possible to flip between the two possible triangulations and we call these arrangements "swappable".

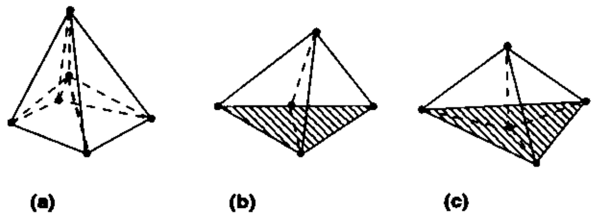


Figure 3.28 Generic nonswappable configurations of 5 points. Shaded region denotes planar surface.

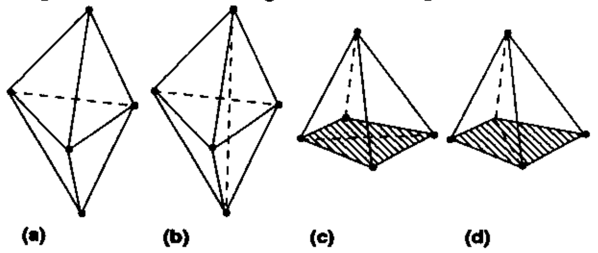


Figure 3.29 Generic swappable configurations of 5 points. Shaded region denotes planar surface.

There are two arrangements that allow two triangulations. Figures 3.29(a) and 3.29(b) show the subclass of companion triangulations that can be transformed from one type to another thereby changing the number of tetrahedra from 2 to 3 or vice-versa. Figures 3.29(c) and 3.29(d) show the other subclass of configurations that can be transformed from one type to another while keeping constant the number of tetrahedra (2). These figures reveal an important difference between the two and three-dimensional algorithms. *The number of tetrahedrons involved in the swapping operation need not be constant.*

The 3-D edge swapping algorithm is based on flipping between the two ways of triangulating the configurations in figure 3.29. One good way of finding all sets of five points in the mesh is to loop through all the faces in the mesh and considering the five points that make up the two adjoining tetrahedra for that face. Below we present the facewise edge swapping primitive.

Primitive: EDGE_SWAP(face)

Let $C = \{$ Set of tetrahedra made from the 5 nodes of the two adjoining tetrahedra $\}$

```
If(shape(C) = convex)then
  Let $T$ = current triangulation
  Let $T^*$ = alternate triangulation (if it exists)
  If( Quality($T^*$) > Quality($T$))then
    [Edge Swap $T$ into $T^*$]
  Endif
Endif
```

The first step is to find *all* the tetrahedra that are described by the five nodes of the two tetrahedra adjacent to the face in question. There is a maximum of four tetrahedra that can be built from five nodes. Since any two tetrahedra made from the same five points will have to share three points (i.e., a face) it is sufficient to only look at the four neighboring tetrahedra of any of the two tetrahedra already known. This constitutes a linear time algorithm for finding all the non-overlapping tetrahedra made from the five points.

If these tetrahedra form a convex shape, then the configuration is described by one of the configurations in figures 3.28 and 3.29 (configurations of the convex hull) and edge swapping is permitted. If, for example, only two of the three tetrahedra in figure 3.29(b) were present, the two tetrahedra will form a concave shape. Obviously, edge swapping concave shapes is not possible without possibly creating overlapping tetrahedra in the mesh. For swappable configurations, a check is performed to see if the local mesh quality measure (discussed further below) will improve by edge swapping into the alternate triangulation. If it does, the swap is performed; otherwise the triangulation is unchanged. This technique offers a distinct advantage over others (Bowyer's or Watson's algorithm) in that it allows the use of any arbitrary mesh quality measure.

Computational Aspects of 3-D Edge Swapping

Before proceding further, it is useful to discuss the computational aspects of some of the operations needed for the edge swapping algorithm.

• Determining Convexity: This operation tests whether the shape formed by the tetrahedron $\mathcal{T}_1$ and $\mathcal{T}_2$ is convex. Let the vertices of the tetrahedra be numbered $\mathcal{T}_1 = (1,2,3,4)$ and $\mathcal{T}_2 = (1,2,3,5)$ i.e., $(1,2,3)$ is the face shared by $\mathcal{T}_1$ and $\mathcal{T}_2$ and nodes 4 and 5 are at the two ends of $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. We make use of the notion of barycentric coordinates to perform the convexity test. The $b_{1,2,3,4}$ satisfying

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 1 \\ x_5 \\ y_5 \\ z_5 \end{bmatrix}$$

are called the barycentric coordinates of node 5. They indicate the position of 5 in relation to the nodes of tetrahedron $\mathcal{T}_1$. For each s, the sign of b_s indicates the position of 5 relative to the plane H_s passing through the triangular face opposite node s. Thus $b_s = 0$ when 5 is in H_s, $b_s > 0$ when 5 is on the same side of H_s as node s, and $b_s < 0$ when 5 is on the opposite side of H_s from node s. Clearly, $b_{1,2,3,4} > 0$ if 5 lies inside tetrahedron $(1,2,3,4)$. If we imagine a cone formed by planes $H_{1,2,3}$ of $\mathcal{T}_1$, then $\mathcal{T}_1$ and $\mathcal{T}_2$ would form a convex shape if and only if node 5 lies in the cone on the side opposite from node 4 (figure 3.30).

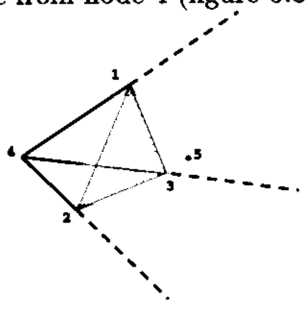


Figure 3.30 Convexity cone for node 4.

The conditions to satisfy this requirement are $b_4 < 0$ and $b_{1,2,3} > 0$. In order to test if three cells form a convex shape, the nodes are renumbered as if the three cell configuration were edge swapped into the corresponding two cell configuration for purposes of the barycentric test. For example, consider a three cell configuration with numbering $(1,2,3,4)$, $(2,3,4,5)$, and $(1,2,4,5)$, then the corresponding two cell configuration would have $(1,3,5)$ as the common face and nodes 2, and 4 at the ends of the two tetrahedra. Notice that if the three tetrahedra formed a convex shape, then it would be possible to edge swap it to a convex two tetrahedra configuration. If the three cells formed a concave shape, however, the transformed two cell triangulation would contain overlapping tetrahedra which the test above would label as concave.

Using Cramer's rule to solve the $Ax = b$ problem posed above requires computing determinants of the form

$$\begin{vmatrix} 1 & 1 & 1 & 1 \\ a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{vmatrix}$$

five times. An optimization is possible by exploiting the property of determinants that subtracting one row or column from another leaves the determinant unchanged. If we subtract the first column from the rest, we simplify the above 4x4 determinant into a 3x3 one.

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ a_{11} & a_{12}-a_{11} & a_{13}-a_{11} & a_{14}-a_{11} \\ a_{21} & a_{22}-a_{21} & a_{23}-a_{21} & a_{24}-a_{21} \\ a_{31} & a_{32}-a_{31} & a_{33}-a_{31} & a_{34}-a_{31} \end{vmatrix} = \begin{vmatrix} a_{12}-a_{11} & a_{13}-a_{11} & a_{14}-a_{11} \\ a_{22}-a_{21} & a_{23}-a_{21} & a_{24}-a_{21} \\ a_{32}-a_{31} & a_{33}-a_{31} & a_{34}-a_{31} \end{vmatrix}$$

• Delaunay Circumsphere test: The 3-D Delaunay triangulation is defined as the unique triangulation such that the circumsphere of any tetrahedron contains no other point in the mesh. To determine where point E lies in relation to the circumsphere of tetrahedron (A, B, C, D), denoted by($\bigcirc$ $ABCD$), we use the *InSphere* primitive :

$$InSphere(E) \begin{cases} < 0 & \text{if } E \text{ is inside } \bigcirc\ ABCD \\ = 0 & \text{if } E \text{ is on } \bigcirc\ ABCD \\ > 0 & \text{if } E \text{ is outside } \bigcirc\ ABCD \end{cases}$$

where *InSphere* is computed from the following determinant:

InSphere(E)

$$= \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ x_A & x_B & x_C & x_D & x_E \\ y_A & y_B & y_C & y_D & y_E \\ z_A & z_B & z_C & z_D & z_E \\ w_A^2 & w_B^2 & w_C^2 & w_D^2 & w_E^2 \end{vmatrix} \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_A & x_B & x_C & x_D \\ y_A & y_B & y_C & y_D \\ z_A & z_B & z_C & z_D \end{vmatrix}$$

$$= \begin{vmatrix} x_B-x_A & x_C-x_A & x_D-x_A & x_E-x_A \\ y_B-y_A & y_C-y_A & y_D-y_A & y_E-y_A \\ z_B-z_A & z_C-z_A & z_D-z_A & z_E-z_A \\ w_B^2-w_A^2 & w_C^2-w_A^2 & w_D^2-w_A^2 & w_E^2-w_A^2 \end{vmatrix} \begin{vmatrix} x_B-x_A & x_C-x_A & x_D-x_A \\ y_B-y_A & y_C-y_A & y_D-y_A \\ z_B-z_A & z_C-z_A & z_D-z_A \end{vmatrix}$$

and $w_P^2 = x_P^2 + y_P^2 + z_P^2$

The first determinant is the 3-D extension of Guibas' *InCircle* primitive [1]. It represents the volume of a pentatope whose vertices are the points A, B, C, D, E projected onto the 4-D paraboloid $(x^2 + y^2 + z^2)$. (A pentatope is the simplest polytope in 4-D just as a tetrahedron is the simplest polytope in 3-D and a triangle in 2-D. A pentatope can be constructed by joining the tetrahedron to a fifth point outside its 3-space.) The coordinates in 3-space of these five points remain unchanged; they simply acquire a value in their fourth coordinate equal to the square of their distances from the origin. The volume of this polytope is positive if point E lies outside $\bigcirc$ $ABCD$ and negative if point E lies inside $\bigcirc$ $ABCD$, provided that tetrahedron (A, B, C, D) has a positive volume (as given by the second determinant). The determinant is degenerate if point E lies exactly on $\bigcirc$ $ABCD$.

This test is motivated by the observation in 3-D that the intersection of a cylinder and a unit paraboloid is an ellipse lying in a plane (figure 3.31). So, any four co-circular points in 2-D will project to four co-planar points and the volume of the tetrahedra made from these four co-planar points will be zero. The paraboloid is a surface that is convex upward and the points interior to a circle get projected to the paraboloid below the intersection plane and the points exterior to it get projected above the the intersection plane. If a point lies outside the circumcircle of three other points, the tetrahedron made from these four points will have positive volume provided the three points were ordered in a counter-clockwise fashion. The volume will be negative if the point lies inside the circumcircle.

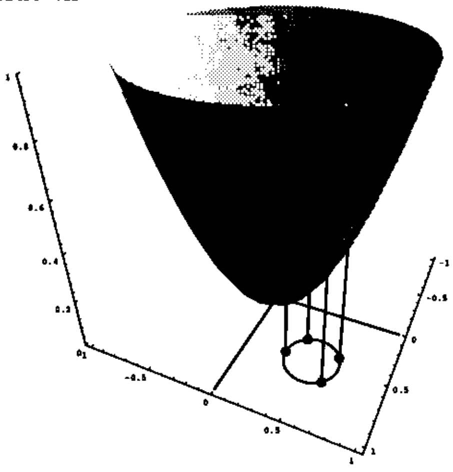


Figure 3.31 Projection of cocircular points onto unit paraboloid.

The second determinant is the 3-D extension of Guibas' *CCW* (counter clock-wise) primitive [1] which computes the volume of tetrahedron (A, B, C, D). Thus the *InSphere* primitive works irrespective of how the points A, B, C, and D are ordered. If it can be guaranteed that all the tetrahedra in a mesh have their vertices ordered to have positive volumes then the need to compute the second determinant is eliminated. The *InSphere* primitive becomes ill-behaved when the points A, B, C, and D all lie nearly on a plane because the position of the circumsphere with respect to the points (i.e., whether the circumsphere is above or below the plane) becomes very sensitive to small perturbations in the coordinates of the five points.

3-D Mesh Optimization

The 3-D edge swapping algorithm can be used to optimize existing triangulations. In fact, there is no way to triangulate a given set of points based on the minmax or maxmin of the face angles directly. An alternative is to start with an existing triangulation and optimize it. This requires that we cycle through all the faces in a mesh and apply the edge swapping procedure at each step. This process is continued until no more swaps are possible.

Algorithm: Three-dimensional mesh optimization.

```
while (swaps occurred in the last cycle over faces)
  for all faces
    EDGE_SWAP (face)
  endfor
endwhile
```

In the following paragraphs, we discuss a few swap-criteria and examine the meshes they produce.

Global Edge Swapping

The *InSphere* criteria is binary in the sense that either the triangulation of a set of five points satisfies the criteria or it does not. It can also be shown that of the two ways to triangulate a set of five points, if one way fails the *InSphere* criteria, then the other one will pass and vice-versa. Cases 1 and 3 in figure 3.15 and cases 1, 3, and 5 in figure 3.16 will always pass the *InSphere* criteria.

The Delaunay triangulation is unique for a given set of points. Lawson also noticed the relation between local and global properties of the Delaunay *InSphere* criteria: a triangulation is Delaunay if and only if the triangulations of sets of five points corresponding to all the interior faces in the mesh satisfy the *InSphere* criteria. This means that if every face satisfies the Delaunay criteria, then the whole mesh must be a Delaunay triangulation.

Joe [32] has proven, however, that processing faces in an arbitrary way may result in getting stuck in local optima. This is an important difference between two and three dimensional combinatorial edge swapping.

3-D MinMax and MaxMin Triangulations

The edge swapping algorithm can be applied locally to produce a triangulation that minimizes the maximum face angle. In 2-D, the edge swapping algorithm (working with edge angles) gets stuck in local minima and depending on the order in which the edges were traversed, different local minima are reached. In practice, the local minima all seem very close to the global minimum which makes edge swapping a practical way to get a nearly optimal MinMax triangulation. We observe that in 3-D as well, there are many local minima and the order of face traversal determines which one is found. It is hard to determine how far these local minima are from the global minimum but we believe that edge swapping is a practical way to get nearly optimal MinMax meshes.

Lawson has shown that in 2-D, Delaunay triangulations have the property that the minimum edge angle is maximized (i.e., MaxMin triangulation). So in 2-D, the MaxMin triangulation is unique and the edge swapping algorithm will converge to it. In 3-D, however, the Delaunay triangulation is not the same as MaxMin triangulation and the edge swapping algorithm working with the MaxMin criteria has the same property of getting stuck in local minima as the MinMax. Again, it is hard to judge how close the local minima are from the global minimum but we still conclude that edge swapping is a fairly efficient technique for the construction of MaxMin triangulations.

3-D Minimum Edge Triangulation

Another mesh of interest is the minimum edge triangulation. Since finite-volume flow solvers work edge-wise, it is beneficial to reduce the number of edges in a mesh. This is easily accomplished by edge swapping such that we always swap from case 3.29a to case 3.29b. Each time this operation is performed, one edge and one tetrahedron are removed from the mesh. Again, different meshes will be produced depending upon how faces are traversed and the final mesh may only be at a local minimum.

Incremental Delaunay Triangulation

The edge swapping algorithm provides an effective way for inserting a point into an existing triangulation. Simply find the tetrahedra into which the point is to be inserted and test its faces according to the circumsphere criteria to determine if edge swapping should take place. If a set of 5 points corresponding to a face is retriangulated, we proceed to test all the outer faces of the new triangulation for swappability and so on. This propagates a front that retriangulates the mesh. It is known that any new face created during the retriangulation is indeed a part of the final mesh as well, and so back-propagation is not required. This may not be true for other mesh quality measures and back propagation then becomes necessary.

Rajan proves that it is possible to find a certain sequence of edge swaps which will guarantee that Delaunay triangulation is recovered when a site is added to an existing Delaunay triangulation. In practice, however, we find that this ordering of edge swaps does not seem to be necessary in order to recover the Delaunay triangulation. In fact, *it is our conjecture that this is always the case.*

This insertion algorithm can be used to adaptively refine meshes. To do this, sites are inserted at the centers of the circumspheres of tetrahedra with large aspect ratios (or other suitable measures). This insertion site does not always lie within the cell T_i marked for refinement. To find the cell in which the new site lies, a walking algorithm is employed. Starting at T_i, barycentrics are computed to determine which face of T_i the new site lies behind. The next step is to traverse to the cell behind that face. This procedure is applied recursively until the cell in which the new site falls within is found. The idea of introducing new sites at the centers of the circumspheres of tetrahedra works well because each new site introduced is equidistant to the 4 points of the large aspect ratio tetrahedra. This produces high quality meshes in 2-D and seems to work well in 3-D.

3.6c 3-D Surface Triangulation

The Wiltberger algorithm has been extended to include the triangulation of surface patches. Although the concept of Dirichlet tessellation is well defined on a smooth manifolds using the concept of geodesic distance, in practice this is too expensive. Finding geodesic distance is a variational problem that is not easily solved. We have implemented a simpler procedure in which surface grids in 3-D are constructed from rectangular surface patches (assumed at least C^0 smooth) using a generalization of the 2-D Steiner triangulation scheme.

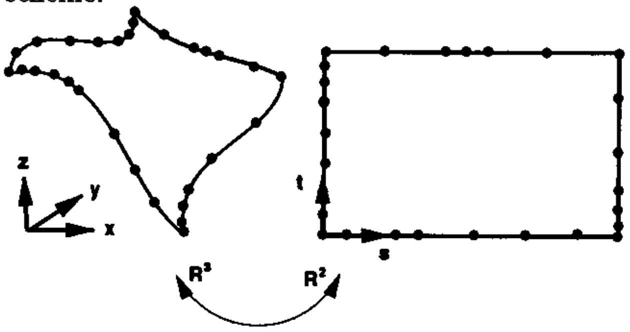


Figure 3.32 Mapping of rectangular patches on (s,t) plane.

Points are first placed on the perimeter of each patch using an adaptive refinement strategy based on absolute error and curvature measures. The surface patches are projected onto the plane, see figure 3.32. Simple stretching of the rectangular patches permits the user to produce preferentially stretched meshes. (This is useful near the leading edge of a wing for example.)

The triangulation takes place in the two dimensional (s,t) plane. The triangulation is adaptively refined using Steiner point insertion to minimize the maximum user specified absolute error and curvature tolerance on each patch. The absolute error is approximated by the perpendicular distance from the triangle centroid (projected back to 3-space) to the true surface as depicted in figure 3.33. The user can further refine based on triangle aspect ratio in the (s,t) plane if desired.

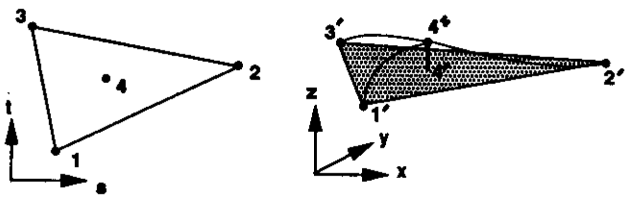


Figure 3.33 Calculation of triangulation absolute error by measurement of distance from face centroid to true surface.

Figure 3.34 shows a typical adaptive surface grid generated using the Steiner triangulation method.

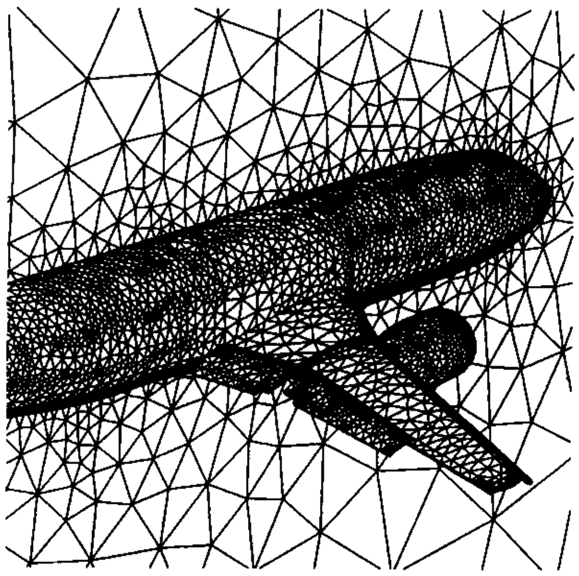


Figure 3.34 Adaptive Steiner triangulation of surface mesh about Boeing 737 with flaps deployed.

4.0 Some Theory Related to Finite-Volume Solvers

4.1 Scalar Conservation Law Equations

For purposes of these notes, we consider numerical methods for solving *conservation law* equations.

Definition: A conservation law asserts that the rate of change of the total amount of a substance with density z in a fixed region Ω is equal to the flux $\mathbf{F}$ of the substance through the boundary $\partial\Omega$.

$$\frac{\partial}{\partial t}\int_{\Omega} z\, da + \int_{\partial\Omega} \mathbf{F}(z)\cdot\mathbf{n}\, dl = 0 \quad \text{(integral form)}$$

The choice of a numerical algorithm used to solve a conservation law equation is often influenced by the form in which the conservation law is presented. A finite-difference practitioner would apply the divergence theorem to the integral form and let the area of Ω shrink to zero thus obtaining the divergence form of the equation.

$$\frac{\partial}{\partial t} z + \nabla\cdot\mathbf{F}(z) = 0 \qquad \text{(divergence form)}$$

The finite-element practitioner constructs the divergence form then multiplies by an arbitrary test function ϕ and integrates by parts.

$$\frac{\partial}{\partial t}\int_{\Omega} \phi z\, da - \int_{\Omega} \nabla\phi\cdot\mathbf{F}(z)\, da + \int_{\partial\Omega} \phi\mathbf{F}(z)\cdot\mathbf{n}\, dl = 0 \quad \text{(weak form)}$$

Algorithm developers starting from these three forms can produce seemingly different numerical schemes. In reality, the final discretizations are usually very similar. Some differences do appear in the handling of boundary conditions, solution discontinuities, and nonlinearities. When considering flows with discontinuities, the integral form appears advantageous since conservation of fluxes comes for free and the proper jump conditions are assured. At discontinuities, the divergence form of the equations implies satisfaction in the sense of distribution theory. Consequently, at discontinuities special care is needed to construct finite difference schemes which produce physically relevant solutions. Because the test functions have compact support, the weak form of the equations also guarantees satisfaction of the jump conditions over the extent of the support. The divergence form of the equations is rarely used in the discretization of conservation law equations on unstructured meshes because of the difficulty in ensuring conservation. On the other hand, the integral and weak forms are both used extensively in numerical modeling of conservation laws on unstructured meshes. In the next section, the simplest of numerical schemes based on integral and weak forms of the conservation law are compared to illustrate their similarities. These schemes can be viewed as the "central-difference" counterparts on unstructured grids. For advection dominated flows, these algorithms are inadequate and additional terms must be added. This topic is undertaken in detail in future sections.

4.2 Comparison of Finite-Volume and Galerkin Finite-Element Methods

Although the integral and weak forms of the equations appear to be quite different, numerical schemes based on these forms often produce identical discretizations. To demonstrate this point, consider the Galerkin discretization (with linear elements) of a general model advection-diffusion equation ($\mu > 0$):

$$\frac{\partial}{\partial t} z + \nabla\cdot\mathbf{F}(z) = \nabla\cdot\mu\nabla z$$

Multiplying by a test function ϕ and integrating by parts over the region Ω produces the weak form of the equation.

$$\begin{aligned}\frac{\partial}{\partial t}\int_{\Omega} \phi z\, da - \int_{\Omega} \nabla\phi\cdot\mathbf{F}(z)\, da + \int_{\partial\Omega} \phi\, \mathbf{F}(z)\cdot\mathbf{n}\, dl \\ = -\int_{\Omega} \mu\nabla\phi\cdot\nabla z\, da + \int_{\partial\Omega} \mu\phi\nabla z\cdot\mathbf{n}\, dl\end{aligned} \tag{4.0}$$

In the finite-element method, the entire domain is first divided into smaller elements. In this case, the elements are triangles T_j, such that $\Omega = \cup T_j$, $T_j \cap T_l = \emptyset$, $l \neq j$. In Fig. 4.1a we show a representative vertex with adjacent neighbors. (To simplify the discussion in the remainder of these notes, we adopt the convention that the index "j" refers to a global index of a mesh whereas the index "i" always refers to a local index.) The linear variation of the solution in each triangle T_j can be expressed in terms of the three local nodal values of the solution, $z^h_{T_j,i}$, $i = 1,2,3$, and three element shape functions n_i, $i = 1,2,3$.

$$z^h(x,y)_{T_j} = \sum_{i=1}^{3} n_i(x,y)\; z^h_{T_j,i} \qquad \text{(local representation)}$$

Each element shape function n_i can be interpreted as a piecewise linear surface which takes on a unit value at v_i and vanishes at the other two vertices of the triangle as well as everywhere outside the triangle. The solution can also be expressed globally in terms of nodal values of the solution and global shape functions.

$$z^h(x,y) = \sum_{nodes} N_j(x,y)\; z_j^h$$

(global representation)

In this form, the global shape functions are piecewise linear pyramids which are formed from the union of all local shape functions with have unit value at v_j. These global shape functions also enjoy compact support, i.e. they vanish outside the region Ω_j formed from the union of all triangles incident to v_j. A global shape function for vertex v_j is shown in Fig. 4.1b.

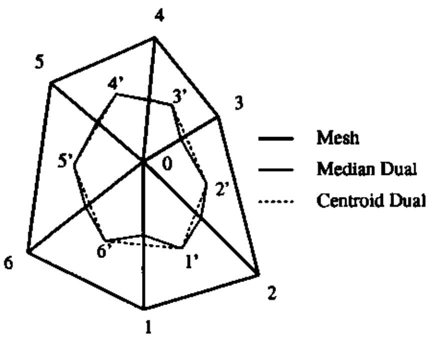


Figure 4.1a Local mesh with centroid and median duals.

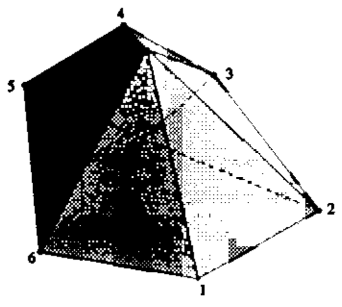


Figure 4.1b Global shape function for vertex v_0 (not labeled).

The Galerkin finite-element method assumes that the class of test functions is identical to the class of functions approximating the solution. The simplest test functions of this sort are the individual shape functions. To obtain a Galerkin discretization for a typical vertex v_j, simply set $\phi^h = N_j$ and evaluate (4.0) in Ω_j. Since ϕ vanishes on $\partial\Omega_j$ equation (4.0) simplifies to the following form:

$$\begin{aligned}\frac{\partial}{\partial t}\int_{\Omega_j} \phi^h z^h\, da - \int_{\Omega_j} \nabla\phi^h \cdot \mathbf{F}(z^h)\, da \\ = -\int_{\Omega_j} \mu\nabla\phi^h \cdot \nabla z^h\, da\end{aligned} \quad (4.1)$$

Before evaluating equation (4.1), it is useful to introduce more notation concerning the geometry of figure 4.1a. Figure 4.2 depicts the index and normal convention which will be used throughout these notes. The triangle with vertices 0, i, and $i+1$ is denoted as $T_{i+1/2}$. This index convention will be used for other quantities such as areas and gradients which are computed in $T_{i+1/2}$.

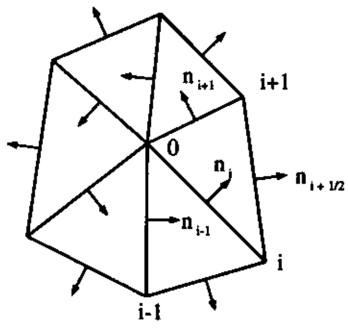


Figure 4.2 Vertex v_0 and adjacent neighbors.

It is convenient to define normals, $\vec{\mathbf{n}}$, for straight edges which are scaled by the length of the edge. Using this notation, a simple formula exists for the gradient of the numerical solution in a triangle $T_{i+1/2}$.

$$\nabla z^h_{i+1/2} = \frac{-1}{2A_{i+1/2}}\left(z_0^h \vec{\mathbf{n}}_{i+1/2} + z_i^h \vec{\mathbf{n}}_{i+1} - z_{i+1}^h \vec{\mathbf{n}}_i\right) \quad (4.2)$$

The gradient of the test function in each triangle takes a similar form (replace z by ϕ in the previous formula with $\phi_0 = 1, \phi_i = 0, \phi_{i+1} = 0$).

$$\nabla\phi_{i+1/2} = \frac{-1}{2A_{i+1/2}}\vec{\mathbf{n}}_{i+1/2} \quad (4.3)$$

The discrete form of eqn. (4.0) is now written as

$$\begin{aligned}\frac{\partial}{\partial t}\int_{\Omega_j} \phi z^h\, da + \sum_{i=1}^{d(v_0)} \frac{\vec{\mathbf{n}}_{i+1/2}}{2A_{i+1/2}} \cdot \int_{T_{i+1/2}} \mathbf{F}(z^h)\, da \\ = \sum_{i=1}^{d(v_0)} \frac{\vec{\mathbf{n}}_{i+1/2}}{2A_{i+1/2}} \cdot \int_{T_{i+1/2}} \mu\nabla z^h\, da\end{aligned} \quad (4.4)$$

The flux integral can be evaluated by exact integration (when possible) or numerical quadrature. In this case, the latter is assumed.

$$\int_{T_{i+1/2}} \mathbf{F}(z^h)\, da = \frac{A_{i+1/2}}{3}\left(\mathbf{F}(z_0^h) + \mathbf{F}(z_i^h) + \mathbf{F}(z_{i+1}^h)\right) \quad (4.5)$$

The diffusion term is also evaluated with ∇z^h constant in $T_{i+1/2}$ and $\overline{\mu}_{i+1/2}$ the area weighted average μ.

$$\int_{T_{i+1/2}} \mu\nabla z^h\, da = A_{i+1/2}\, \overline{\mu}_{i+1/2} \nabla z^h_{i+1/2} \quad (4.6)$$

This simplifies (4.0) considerably.

$$\begin{aligned}\frac{\partial}{\partial t}\int_{\Omega_j}\phi z^h\,da \\ +\sum_{i=1}^{d(v_0)}\frac{1}{6}\vec{\mathbf{n}}_{i+1/2}\cdot\left(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h)+\mathbf{F}(z_{i+1}^h)\right) \\ =\sum_{i=1}^{d(v_0)}\frac{1}{2}\overline{\mu}_{i+1/2}\ \vec{\mathbf{n}}_{i+1/2}\cdot\nabla_{i+1/2}z^h\end{aligned} \tag{4.7}$$

Equation (4.7) represents a Galerkin discretization of the model equation assuming piecewise linear functions. Note that as far as the geometry is concerned, only the exterior normals of Ω_j appear. Conspicuously absent are the normal vectors for interior edges. This strengthens our confidence that we can show an equivalence with a finite-volume discretization on *nonoverlapping* control volumes. To show this equivalence, note that the flux term can be manipulated using the identity $\sum_{i=1}^{d(v_0)}\vec{\mathbf{n}}_{i+1/2}=\mathbf{0}$ into a form in which the relevant geometry is any path connecting adjacent triangle centroids (**R** denotes the spatial position vector):

$$\begin{aligned}&\sum_{i=1}^{d(v_0)}\frac{1}{6}\vec{\mathbf{n}}_{i+1/2}\cdot\left(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h)+\mathbf{F}(z_{i+1}^h)\right)\\ &=\sum_{i=1}^{d(v_0)}\frac{1}{6}\left(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h)\right)\cdot\left(\vec{\mathbf{n}}_{i+1/2}+\vec{\mathbf{n}}_{i-1/2}\right)\\ &=\sum_{i=1}^{d(v_0)}\frac{1}{6}\left(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h)\right)\cdot\int_{\mathbf{R}_{i-1}}^{\mathbf{R}_{i+1}}\mathbf{n}\,dl\\ &=\sum_{i=1}^{d(v_0)}\frac{1}{2}\left(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h)\right)\cdot\int_{\frac{1}{3}(\mathbf{R}_0+\mathbf{R}_i+\mathbf{R}_{i-1})}^{\frac{1}{3}(\mathbf{R}_0+\mathbf{R}_i+\mathbf{R}_{i+1})}\mathbf{n}\,dl\end{aligned} \tag{4.8}$$

The diffusion term also simplifies using this identity.

$$\begin{aligned}&\sum_{i=1}^{d(v_0)}\frac{1}{2}\vec{\mathbf{n}}_{i+1/2}\cdot\overline{\mu}_{i+1/2}\nabla_{i+1/2}z^h\\ &=\sum_{i=1}^{d(v_0)}\overline{\mu}_{i+1/2}\nabla_{i+1/2}z^h\cdot\int_{\frac{1}{2}(\mathbf{R}_0+\mathbf{R}_i)}^{\frac{1}{3}(\mathbf{R}_0+\mathbf{R}_{i+1})}\mathbf{n}\,dl\end{aligned} \tag{4.9}$$

To obtain a single consistent path for the integrations appearing in equations(4.8) and (4.9) requires that the path pass through the centroid of each triangle and the mid-side of each interior edge. The path formed by connecting these points by line segments is precisely the median dual of the mesh. This dual completely covers the domain (no holes) and represents a consistent and conservative finite-volume discretization of the domain which is spatially equivalent to the Galerkin approximation. The scheme can now be written in a finite-volume form

$$\frac{\partial}{\partial t}\phi^h z^h\,da+\sum_{i=1}^{d(v_0)}(\mathbf{H}\cdot\vec{\mathbf{n}})_i=0 \tag{4.10}$$

where **H** is the numerical flux of the finite-volume discretization

$$\begin{aligned}(\mathbf{H}\cdot\vec{\mathbf{n}})_i=&\frac{1}{2}(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h))\cdot\int_{\mathbf{R}'_{i-1/2}}^{\mathbf{R}'_{i+1/2}}\mathbf{n}\,dl\\ &-\overline{\mu}_{i-1/2}\nabla_{i-1/2}z^h\cdot\int_{\mathbf{R}'_{i-1/2}}^{\mathbf{R}_i^m}\mathbf{n}\,dl\\ &-\overline{\mu}_{i+1/2}\nabla_{i+1/2}z^h\cdot\int_{\mathbf{R}_i^m}^{\mathbf{R}'_{i+1/2}}\mathbf{n}\,dl\end{aligned} \tag{4.11}$$

and $\mathbf{R}'_{i+1/2}$ is the centroid of $T_{i+1/2}$, $\mathbf{R}'_{i+1/2}=\frac{1}{3}(\mathbf{R}_0+\mathbf{R}_i+\mathbf{R}_{i+1})$ and $\mathbf{R}_i^m$ is the midpoint of the edge $e(v_0,v_i)$, $\mathbf{R}_i^m=\frac{1}{2}(\mathbf{R}_0+\mathbf{R}_i)$.

Conclusion: *The spatial discretization produced by the Galerkin finite-element scheme with linear elements has an equivalent finite-volume discretization on nonoverlapping control volumes with bounding curves which pass through the centroid of triangles and midside of edges. One such set of control volumes satisfying these constraints is the median dual.*

We now need to ask if the time integrals produce identical "mass" matrices for the Galerkin finite-element and finite-volume schemes. The answer to this question is no. In fact, these matrices are not the same in one space dimension. The Galerkin mass matrix for a simple 1-D mesh with uniform spacing produces a row of the mass matrix with the following weights:

$$\frac{\partial}{\partial t}\int_{\Omega_j}\phi^h z^h dx=\frac{\partial}{\partial t}\Delta x\frac{1}{6}(z_{j-1}+4z_j+z_{j+1})$$

(Finite – Element)

The finite volume scheme on "median" dual produces the following weights:

$$\frac{\partial}{\partial t}\int_{\Omega_j}z^h dx=\frac{\partial}{\partial t}\Delta x\frac{1}{8}(z_{j-1}+6z_j+z_{j+1})$$

(Finite – Volume)

Although the finite-volume matrix gives better temporal stability, the finite-element mass matrix is more accurate.

4.3 Edge Formulas

The first term appearing on the right-hand-side of equation (4.10)

$$\frac{1}{2}(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h))\cdot\int_{\mathbf{R}'_{i-1/2}}^{\mathbf{R}'_{i+1/2}}\mathbf{n}\,dl$$

suggests a computer implementation using an *edge* data structure. The fluxes in this term are evaluated at the two endpoints of an edge. The geometrical terms could be evaluated edgewise if the midpoint of the edge and the centroids of the two adjacent cells are known. Recall that the edge data structure (described in section 1.3) for a 2-D mesh supplies this information for each interior edge of the mesh, i.e. the structure provides for each edge

(1) The two vertices which form the edge.

(2) The two adjacent cell centroids (or a pointer to centroid values) which share the edge.

More generally, if the solution is assumed to vary linearly within each triangle then edge formulas can be derived for discretized forms of the gradient, divergence, Hessian, and Laplacian operators. As we will see, the formulas can be derived from either a finite-volume or finite-element point-of-view with essentially identical results.

4.3a Gradient and Divergence Edge Formulas

As a first example, we will derive an edge formula for the integral averaged gradient of a function u, $\int\nabla u\,da$, for the the region Ω_0 described by the union of all triangles which share the vertex v_0, see figure 4.1a. If the discrete solution u^h varies linearly in each triangle T then the gradient is constant and the integration exact.

$$\int_{\Omega_0}\nabla u^h\,da=\sum_{T\in\Omega_0}(\nabla u^h)_T A_T \tag{4.12}$$

Equation (4.12) would suggest computing the gradient in each triangle sharing v_0 and accumulating the area weighted sum. If integral averaged gradients are required at all vertices then the gradient in each triangle could be computed and the area weighted result scattered to the three vertices of the triangle for accumulation. We refer to this as the element-by-element approach. A Green's formula would suggest a different approach for the same task.

$$\int_{\Omega_0}\nabla u\,da=\oint_{\partial\Omega_0}u\,\mathbf{n}\,dl \tag{4.13}$$

Identical results are obtained by approximating the right-hand-side of (4.13) by trapezoidal quadrature (exact for piecewise linear u^h)

$$\oint_{\partial\Omega_0}u^h\,\mathbf{n}\,dl=\sum_{i\in\mathcal{I}_0}\frac{1}{2}(u_i^h+u_{i+1}^h)\vec{\mathbf{n}}_{i+1/2} \tag{4.14}$$

where $\mathcal{I}_0=\{1,2,...,6\}$ and $\vec{\mathbf{n}}_{i+1/2}$ is the vector perpendicular to the edge $e(v_i,v_{i+1})$ with magnitude equal to the length of the edge. The summation can be rearranged to yield

$$\oint_{\partial\Omega_0}u^h\,\mathbf{n}\,dl=\sum_{i\in\mathcal{I}_0}\frac{1}{2}u_i^h(\vec{\mathbf{n}}_{i+1/2}+\vec{\mathbf{n}}_{i-1/2}). \tag{4.15}$$

A constant solution can be added to (4.15) since the gradient of a constant function is exactly zero in this discretization. In particular, we add the value of u^h at vertex v_0.

$$\oint_{\partial\Omega_0}u^h\,\mathbf{n}\,dl=\sum_{i\in\mathcal{I}_0}\frac{1}{2}(u_0^h+u_i^h)(\vec{\mathbf{n}}_{i+1/2}+\vec{\mathbf{n}}_{i-1/2}) \tag{4.16}$$

Once again using the fact that for any closed curve $\oint\mathbf{n}\,dl=\oint d\vec{\mathbf{n}}=0$ which implies that

$$\vec{\mathbf{n}}_{i+1/2}+\vec{\mathbf{n}}_{i-1/2}=\int_{v_{i-1}}^{v_{i+1}}d\vec{\mathbf{n}}$$

for *any* path connecting v_{i-1} and v_{i+1}. This path integral represents a vector which is parallel in direction and three times the magnitude of the vector $\vec{\mathbf{n}}$ obtained by computing the integral for any simple path connecting the centroids of the two triangles which share the edge $e(v_0,v_i)$. $\int_{v_{i-1}}^{v_{i+1}}d\vec{\mathbf{n}}=3\int_{v_{i'-1}}^{v_{i'}}d\vec{\mathbf{n}}=3\vec{\mathbf{n}}_{0i}$. This reduces the gradient formula to the following form:

$$\oint_{\partial\Omega_0}u^h\,\mathbf{n}\,dl=\sum_{i\in\mathcal{I}_0}\frac{3}{2}(u_0^h+u_i^h)\vec{\mathbf{n}}_{0i} \tag{4.17}$$

The vertex lumped average gradient at vertex is then given by

$$(\nabla u^h)_{v_0}=\frac{3}{A_{\Omega_0}}\sum_{i\in\mathcal{I}_0}\frac{1}{2}(u_0^h+u_i^h)\vec{\mathbf{n}}_{0i}. \tag{4.18}$$

It is well known that the region bounded by the "median" dual at vertex v_0, (shown in figure 4.1a) has an area A_0 which is exactly $\frac{1}{3}A_{\Omega_0}$. Therefore, using the median dual we obtain a formula which appears to represent some approximate quadrature of the right-hand-side of (4.13) on nonoverlapping regions.

$$(\nabla u^h)_{v_0}=\frac{1}{A_0}\sum_{i\in\mathcal{I}_0}\frac{1}{2}(u_0^h+u_i^h)\vec{\mathbf{n}}_{0i} \tag{4.19}$$

A naive interpretation of equation (4.19) would probably conclude that this equation is a rather poor approximation to (4.13). It is not obvious from (4.19) that the gradient of a linear function u is computed exactly. From the origin of this formula, we now know that this formula can be obtained from a trapezoidal quadrature on a slightly larger region and is exact within the class of linear polynomials.

Keep in mind that a constant solution could have been subtracted instead of added from equation (4.15) which would have given a different but equivalent form of (4.19).

$$(\nabla u^h)_{v_0} = \frac{1}{A_0} \sum_{i \in \mathcal{I}_0} \frac{1}{2}(u_i^h - u_0^h)\vec{\mathbf{n}}_{0i} \qquad (4.20)$$

This formula does not appear to resemble any approximate quadrature of (4.13).

Equation (4.19) suggests an algorithm using an edge data structure which is quite different from the element-by-element method (4.12). The edge-based calculation consists of the following steps:

Sample Gradient Computation

(1) (Precomputation) For each edge $e(v_i, v_j)$ *gather* the centroid coordinates of the two adjacent cells, v_i' and v_j'.

(2) (Precomputation) For each edge *compute* the dual edge normal $\vec{\mathbf{n}}_{ij}$ from the centroid coordinates $\vec{\mathbf{n}}_{ij} = \int_{v_i'}^{v_j'} d\,\vec{\mathbf{n}}$. (Orient from v_i to v_j if $i < j$).

(3) For each edge $e(v_i, v_j)$ *gather* the values of the function at the two vertices, u_i^h and u_j^h.

(4) For each edge *compute* the arithmetic average and multiply by the dual edge normal, $\frac{1}{2}(u_i + u_j)\mathbf{n}_{ij}$.

(5) For each edge *scatter and accumulate* the result at vertex v_i.

(6) For each edge *negate, scatter and accumulate* the same result at vertex v_j.

(7) For each vertex *compute* the final gradient by dividing the accumulated result by area of the median dual, A_0.

This algorithm conforms perfectly within the edge data structure. In practice all the geometrical factors could be precomputed and stored in memory by edge thereby eliminating a gather. The sample algorithm described above serves as a template for all the remaining algorithms described in the rest of this section.

The gradient and divergence operators are related so that it is not surprising that the discretization of the divergence operator produces a similar formula:

$$\int_{\Omega_0} div(\mathbf{F}^h)da = \sum_{i \in \mathcal{I}_0} \frac{3}{2}(\mathbf{F}_0^h + \mathbf{F}_i^h) \cdot \vec{\mathbf{n}}_{0i} \quad (4.21)$$

The Galerkin weighted finite element integrals with linear elements produce essentially identical results. In this case a piecewise linear weighting function ϕ^h is introduced (see figure 4.1b). The gradient and divergence formulas (introduced earlier)

$$\int_{\Omega_0} \phi^h \, \nabla u^h da = \sum_{i \in \mathcal{I}_0} \frac{1}{2}(u_0^h + u_i^h)\vec{\mathbf{n}}_{0i} \qquad (4.22)$$

$$\int_{\Omega_0} \phi^h \, div(\mathbf{F}^h)da = \sum_{i \in \mathcal{I}_0} \frac{1}{2}(\mathbf{F}_0^h + \mathbf{F}_i^h) \cdot \vec{\mathbf{n}}_{0i} \quad (4.23)$$

differ from the previous formulas by a constant factor of 1/3. For example, if a lumped approximation to the left-hand-side of (4.22) is assumed, then (4.19) is recovered since

$$\int_{\Omega_0} \phi^h \nabla u^h da \approx (\nabla u^h)_{v_0} A_0. \qquad (4.24)$$

4.3b 2-D Hessian and Laplacian Edge Formulas

We begin by approximating the following matrix of second derivatives

$$\nabla \mu (\nabla u)^T = \begin{bmatrix} (\mu u_x)_x & (\mu u_y)_x \\ (\mu u_x)_y & (\mu u_y)_y \end{bmatrix} \qquad (4.25)$$

using a standard Galerkin approximation for the region Ω_0 formed from the union of all triangles that share the vertex v_0. To do so, multiply (4.25) by the weight function ϕ and perform integration by parts over Ω_0 assuming $\phi = 0$ on $\partial\Omega_0$.

$$\begin{aligned} \int_{\Omega_0} \phi \nabla \, \mu (\nabla u)^T \, da &= - \int_{\Omega_0} \mu (\nabla \phi) \; (\nabla u)^T \, da \\ &= - \sum_{i \in \mathcal{I}_0} \int_{T_{i+1/2}} \mu (\nabla \phi) \; (\nabla u)^T \, da \end{aligned} \qquad (4.26)$$

where $T_{i+1/2} \equiv simplex(v_0, v_i, v_{i+1})$. Using the notation of figure 4.2, gradients of the piecewise linear functions ϕ^h (figure 4.1b) and u^h are

$$(\nabla \phi^h)_{T_{i+1/2}} = -\frac{1}{2A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} \qquad (4.27)$$

and

$$(\nabla u^h)_{T_{i+1/2}} = \frac{-1}{2A_{i+1/2}}(u_0^h \vec{\mathbf{n}}_{i+1/2} + u_i^h \vec{\mathbf{n}}_{i+1} - u_{i+1}^h \vec{\mathbf{n}}_i) \tag{4.28}$$

where $A_{i+1/2}$ is the area of $T_{i+1/2}$ and $\vec{\mathbf{n}}_{i+1/2}$ is the vector normal to the edge $e(v_i, v_{i+1})$ with magnitude equal to the length of the edge.

For piecewise linear u^h, the gradient is constant in each triangle. The integral average matrix of second derivatives simplifies to the following form:

$$\begin{aligned}\int_{\Omega_0} \phi^h \nabla\, \mu(\nabla u^h)^T\, da &= \sum_{i\in\mathcal{I}_0} \frac{1}{2A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} \int_{T_{i+1/2}} \mu(\nabla u^h)^T da \\ &= \sum_{i\in\mathcal{I}_0} \frac{1}{2A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} (\nabla u^h)^T_{T_{i+1/2}} \int_{T_{i+1/2}} \mu da \\ &= \sum_{i\in\mathcal{I}_0} \frac{\overline{\mu}_{i+1/2}}{2} \vec{\mathbf{n}}_{i+1/2} (\nabla u^h)^T_{T_{i+1/2}}\end{aligned} \tag{4.29}$$

where $\overline{\mu}_{i+1/2}$ is the integral average of μ in $T_{i+1/2}$. Inserting the triangle gradient formula, we obtain a discretized formula for the Galerkin integral.

$$\begin{aligned}\int_{\Omega_0} \phi^h \nabla\, \mu(\nabla u^h)^T\, da &= -\frac{1}{4} \sum_{i\in\mathcal{I}_0} \frac{\overline{\mu}_{i+1/2}}{A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} (u_0^h \vec{\mathbf{n}}^T_{i+1/2} \\ &\quad + u_i^h \vec{\mathbf{n}}^T_{i+1} - u_{i+1}^h \vec{\mathbf{n}}_i^T)\end{aligned} \tag{4.30}$$

Regrouping terms and removal of a constant solution yields the following simplified form

$$\begin{aligned}\int_{\Omega_0} \phi^h \nabla\, \mu(\nabla u^h)^T\, da &= \int_{\Omega_0} \nabla \mu(\nabla(u^h - u_0^h))^T\, da \\ &= \sum_{i\in\mathcal{I}_0} M_i (u_i^h - u_0^h)\end{aligned} \tag{4.31}$$

with

$$M_i = -\frac{1}{4}\left[\frac{\overline{\mu}_{i+1/2}}{A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} (\vec{\mathbf{n}}_{i+1})^T - \frac{\overline{\mu}_{i-1/2}}{A_{i-1/2}} \vec{\mathbf{n}}_{i-1/2} (\vec{\mathbf{n}}_{i-1})^T\right]. \tag{4.32}$$

Even though this formula is very simple, it is not compatible with the edge data structure mentioned earlier. Using some simple identities, we will now rewrite the weight formula in a form which is compatible with the edge data structure.

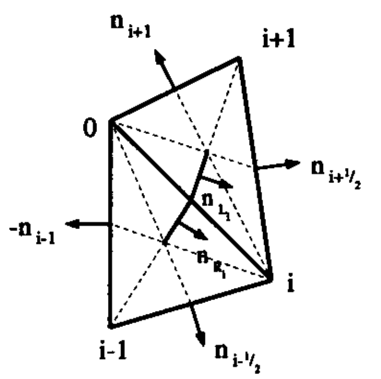


Figure 4.3 Local geometry configuration.

Referring to figure 4.3, we have the following vector identities:

$$\vec{\mathbf{n}}_{i-1/2} = 3\vec{\mathbf{n}}_{R_i} - \frac{1}{2}\vec{\mathbf{n}}_i, \quad \vec{\mathbf{n}}_{i-1} = 3\vec{\mathbf{n}}_{R_i} + \frac{1}{2}\vec{\mathbf{n}}_i \tag{4.33}$$

and similarly

$$\vec{\mathbf{n}}_{i+1/2} = 3\vec{\mathbf{n}}_{L_i} + \frac{1}{2}\vec{\mathbf{n}}_i, \quad \vec{\mathbf{n}}_{i+1} = -3\vec{\mathbf{n}}_{L_i} + \frac{1}{2}\vec{\mathbf{n}}_i. \tag{4.34}$$

It is useful to decompose the tensor product terms into symmetric and skew-symmetric parts, for example:

$$-\vec{\mathbf{n}}_{i-1/2}(\vec{\mathbf{n}}_{i-1})^T = \underbrace{(\frac{1}{4}\vec{\mathbf{n}}_i\vec{\mathbf{n}}_i^T - 9\vec{\mathbf{n}}_R\vec{\mathbf{n}}_R^T)}_{Symmetric} + \underbrace{\frac{3}{2}(\vec{\mathbf{n}}_i\vec{\mathbf{n}}_R^T - \vec{\mathbf{n}}_R\vec{\mathbf{n}}_i^T)}_{Skew-symmetric}$$

and

$$\vec{\mathbf{n}}_{i+1/2}(\vec{\mathbf{n}}_{i+1})^T = \underbrace{(\frac{1}{4}\vec{\mathbf{n}}_i\vec{\mathbf{n}}_i^T - 9\vec{\mathbf{n}}_L\vec{\mathbf{n}}_L^T)}_{Symmetric} + \underbrace{\frac{3}{2}(\vec{\mathbf{n}}_i\vec{\mathbf{n}}_L^T - \vec{\mathbf{n}}_L\vec{\mathbf{n}}_i^T)}_{Skew-symmetric}$$

Upon dividing by the area terms, some simple algebra reveals that

$$\frac{\vec{\mathbf{n}}_{i-1/2}(\vec{\mathbf{n}}_{i-1})^T}{A_{i-1/2}} = \frac{(9\vec{\mathbf{n}}_{R_i}\vec{\mathbf{n}}_{R_i}^T - \frac{1}{4}\vec{\mathbf{n}}_i\vec{\mathbf{n}}_i^T)}{A_{i-1/2}} + \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{4.35}$$

and similarly

$$\frac{\vec{\mathbf{n}}_{i+1/2}(\vec{\mathbf{n}}_{i+1})^T}{A_{i+1/2}} = \frac{(\frac{1}{4}\vec{\mathbf{n}}_i\vec{\mathbf{n}}_i^T - 9\vec{\mathbf{n}}_{L_i}\vec{\mathbf{n}}_{L_i}^T)}{A_{i+1/2}} + \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \tag{4.36}$$

So in summary we have that

$$\begin{aligned} M_i = & -\frac{1}{4}\left[\frac{\overline{\mu}_{i+1/2}}{A_{i+1/2}}\vec{\mathbf{n}}_{i+1/2}(\vec{\mathbf{n}}_{i+1})^T \right. \\ & \left. - \frac{\overline{\mu}_{i-1/2}}{A_{i-1/2}}\vec{\mathbf{n}}_{i-1/2}(\vec{\mathbf{n}}_{i-1})^T\right] \\ = & -\frac{1}{4}\left[\overline{\mu}_{i+1/2}\frac{\frac{1}{4}\vec{\mathbf{n}}_i\vec{\mathbf{n}}_i^T - 9\vec{\mathbf{n}}_{L_i}\vec{\mathbf{n}}_{L_i}^T}{A_{i+1/2}} \right. \\ & + \overline{\mu}_{i-1/2}\frac{\frac{1}{4}\vec{\mathbf{n}}_i\vec{\mathbf{n}}_i^T - 9\vec{\mathbf{n}}_{R_i}\vec{\mathbf{n}}_{R_i}^T}{A_{i-1/2}} \\ & \left. + (\overline{\mu}_{i+1/2} - \overline{\mu}_{i-1/2})\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right] \end{aligned} \quad . \quad (4.37)$$

The second form is compatible with an edge data structure where edge vertices and adjacent cell centroids are known.

4.3c 2-D Edge Discretization of $\nabla \cdot \mu \nabla u$

Calculation of this term amounts to summing diagonal entries of the previous result. Cancellation of terms leaves a reduced form.

$$\begin{aligned} \int_{\Omega_0} \phi^h \nabla \cdot \mu \nabla u^h \, da = & \text{Trace} \int_{\Omega_0} \phi^h \nabla \, \mu (\nabla u^h)^T \, da \\ = & \sum_{i \in \mathcal{I}_0} W_i (u_i^h - u_0^h) \end{aligned} \quad (4.38)$$

$$W_i = -\frac{1}{4}\left[\overline{\mu}_{i+1/2}\frac{\vec{\mathbf{n}}_{i+1/2}\cdot\vec{\mathbf{n}}_{i+1}}{A_{i+1/2}} + \overline{\mu}_{i-1/2}\frac{\vec{\mathbf{n}}_{i-1/2}\cdot\vec{\mathbf{n}}_{i-1}}{A_{i-1/2}}\right] \quad (4.39)$$

The area of a triangle can be expressed in terms of the magnitude of the cross product of the scaled edge normals.

$$A_{i+1/2} = \frac{1}{2}|\vec{\mathbf{n}}_{i+1/2} \times \vec{\mathbf{n}}_{i+1}|$$

$$A_{i-1/2} = \frac{1}{2}|\vec{\mathbf{n}}_{i-1/2} \times \vec{\mathbf{n}}_{i-1}|$$

$$W_i = -\frac{1}{2}\left[\overline{\mu}_{i+1/2}\frac{(\vec{\mathbf{n}}_{i+1/2}\cdot\vec{\mathbf{n}}_{i+1})}{|\vec{\mathbf{n}}_{i+1/2}\times\vec{\mathbf{n}}_{i+1}|} - \overline{\mu}_{i-1/2}\frac{(\vec{\mathbf{n}}_{i-1/2}\cdot\vec{\mathbf{n}}_{i-1})}{|\vec{\mathbf{n}}_{i-1/2}\times\vec{\mathbf{n}}_{i-1}|}\right] \quad (4.40)$$

Finally we can express the dot and cross products in terms of the local angles as sketched in figure 4.4.

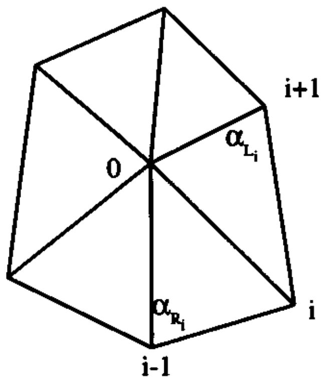


Figure 4.4 Local angles for triangles sharing edge $e(v_0, v_i)$.

$$\frac{\vec{\mathbf{n}}_{i+1/2}\cdot\vec{\mathbf{n}}_{i+1}}{|\vec{\mathbf{n}}_{i+1/2}\times\vec{\mathbf{n}}_{i+1}|} = -\frac{\cos(\alpha_{L_i})}{\sin(\alpha_{L_i})} = -\text{cotan}(\alpha_{L_i}) \quad (4.41)$$

and

$$-\frac{\vec{\mathbf{n}}_{i-1/2}\cdot\vec{\mathbf{n}}_{i-1}}{|\vec{\mathbf{n}}_{i-1/2}\times\vec{\mathbf{n}}_{i-1}|} = -\frac{\cos(\alpha_{R_i})}{\sin(\alpha_{R_i})} = -\text{cotan}(\alpha_{R_i}) \quad (4.42)$$

Inserting these formulas yields a particularly simple form of the weight factors W_i:

$$W_i = \frac{1}{2}\left[\overline{\mu}_{i+1/2}\text{cotan}(\alpha_{L_i}) + \overline{\mu}_{i-1/2}\text{cotan}(\alpha_{R_i})\right] \quad (4.43)$$

Equation (4.43) is particularly useful in theoretical studies.

4.3d 3-D Hessian and Laplacian Edge Formulas

As in the 2-D case, we begin with the Galerkin integral equation for the Hessian-like matrix of derivatives.

$$\int_\Gamma \phi^h \nabla \, \mu (\nabla u^h)^T \, dv = -\int_\Gamma \mu (\nabla \phi^h) \, (\nabla u^h)^T \, dv$$

In this formula V_Γ is the volume formed by the union of all tetrahedra that share vertex v_0. Following a procedure identical to the 2-D case, we can derive the analogous 3-D edge formula for the matrix of second derivatives

$$\int_{V_{\Gamma_0}} \phi^h \nabla \, \mu (\nabla u^h)^T \, dv = \sum_{i \in \mathcal{I}_0} M_i (u_i - u_0) \quad (4.44)$$

where

$$M_i = -\frac{1}{9}\sum_{k=1}^{d(v_0,v_i)} \frac{\overline{\mu}_{k+1/2}}{V_{k+1/2}}\vec{\mathbf{s}}_{k+1/2}(\vec{\mathbf{s}}'_{k+1/2})^T \quad (4.45)$$

$\mathcal{I}_0$ is the set of indices of all adjacent neighbors of v_0 connected by incident edges, k a local cyclic index describing the associated vertices which form a polygon of degree $d(v_0, v_i)$ surrounding the edge $e(v_0, v_i)$. The subscript $k + 1/2$ indicates quantities associated with the tetrahedron with vertices v_0, v_i, v_k and v_{k+1} as shown in figure 4.5.

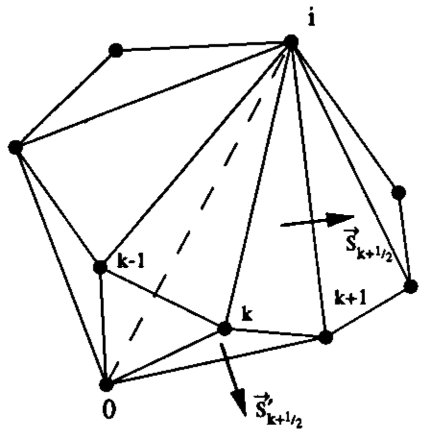


Figure 4.5 Set of tetrahedra sharing edge $e(v_0, v_i)$ with local cyclic index k.

4.3e 3-D Edge Discretization of $\nabla \cdot \mu\nabla u$

Following the same procedure as in 2-D, we obtain:

$$\int_{\Gamma_0} \phi^h \nabla \cdot \mu \nabla u^h \, dv = \text{Trace} \int_{\Gamma_0} \phi^h \nabla\, \mu (\nabla u^h)^T dv$$
$$= \sum_{i \in \mathcal{I}_0} W_i (u_i - u_0) \tag{4.46}$$

where

$$W_i = -\frac{1}{9} \sum_{k=1}^{d(v_0,v_i)} \frac{\overline{\mu}_{k+1/2}}{V_{k+1/2}} \vec{s}_{k+1/2} \cdot \vec{s}'_{k+1/2} \tag{4.47}$$

It can be shown that the volume of a tetrahedron is given by

$$V_{k+1/2} = \frac{2}{3} \frac{|\vec{s}_{k+1/2} \times \vec{s}'_{k+1/2}|}{|\Delta \mathbf{R}_{k+1/2}|} \tag{4.48}$$

where $|\Delta \mathbf{R}_{k+1/2}|$ is the length of the edge shared by the faces associated with $\vec{s}_{k+1/2}$ and $\vec{s}'_{k+1/2}$.

$$W_i = -\frac{1}{6} \sum_{k=1}^{d(v_0,v_i)} \overline{\mu}_{k+1/2} |\Delta R_{k+1/2}| \frac{\vec{s}_{k+1/2} \cdot \vec{s}'_{k+1/2}}{|\vec{s}_{k+1/2} \times \vec{s}'_{k+1/2}|} \tag{4.49}$$

Finally we can rewrite the dot and cross product in terms of the cotangent of the face angle.

$$\frac{\vec{s}_{k+1/2} \cdot \vec{s}'_{k+1/2}}{|\vec{s}_{k+1/2} \times \vec{s}'_{k+1/2}|} = -\frac{\cos(\alpha_{k+1/2})}{\sin(\alpha_{k+1/2})}$$
$$= -\operatorname{cotan}(\alpha_{k+1/2})$$

As in the 2-D case, the weights W_i now have a particularly simple form:

$$W_i = \frac{1}{6} \sum_{k=1}^{d(v_0,v_i)} \overline{\mu}_{k+1/2} |\Delta R_{k+1/2}| \operatorname{cotan}(\alpha_{k+1/2}) \tag{4.50}$$

4.4 Godunov Finite-Volume Schemes

In this section, we consider upwind algorithms for scalar hyperbolic equations. In particular, we concentrate on upwind schemes based on Godunov's method [34] and defer the discussion of "upwind" schemes based on the fluctuation decomposition method or the Petrov-Galerkin formulation (SUPG, GLS) to the lectures of Profs. Deconinck, Hughes, and Johnson.

The development presented here follows many of the ideas developed previously for structured meshes. For example, in the extension of Godunov's scheme to second order accuracy in one space dimension, van Leer [35] developed an advection scheme based on the *reconstruction* of discontinuous piecewise linear distributions together with Lagrangian hydrodynamics. Soon thereafter, Colella and Woodward [36] and Woodward and Colella [37] further extended these ideas to include discontinuous piecewise parabolic approximations with Eulerian or Lagrangian hydrodynamics. Harten et. al. [38,39] later extended related schemes to arbitrary order and clarified the entire process. These techniques have been applied to structured meshes in multiple space dimensions by applying one-dimensional-like schemes along individual coordinate lines. This has proven to be a highly successful approximation but does not directly extend to unstructured meshes. In reference [40], we proposed a scheme for multidimensional reconstruction on unstructured meshes using discontinuous piecewise linear distributions of the solution in each control volume. Monotonicity of the reconstruction was enforced using a limiting procedure similar to that proposed by van Leer [35] for structured grids. In a later paper (Barth and Frederickson [41]), we developed numerical schemes for unstructured meshes utilizing a reconstruction algorithm of arbitrary order. Portions of the discussion presented here is taken from these papers.

4.4a Generalized Godunov Scheme

We begin by considering the integral conservation law for some domain, Ω and its tessellation $\mathcal{T}(\Omega)$ comprised of cells, c_j, $\Omega = \cup c_j$, $c_k \cap c_j = \emptyset$, $k \neq j$. The integral equation is valid for the entire domain Ω as well as in each cell (or possibly dual cell):

$$\frac{\partial}{\partial t}\int_{c_j} u\, da + \int_{\partial c_j} \mathbf{F}(u)\cdot \mathbf{n}\, dl = 0 \qquad (4.51a)$$

Fundamental to Godunov's method is the cell average of the solution, $\overline{u}$, in each cell.

$$\int_{c_j} u\ da = \overline{u}_j A_j \qquad (4.52)$$

In Godunov's method and the higher order accurate extension considered here, these cell averages are treated as the fundamental unknowns (degrees of freedom).

$$\frac{\partial}{\partial t}(\overline{u}_j A_j) + \int_{\partial c_j} \mathbf{F}(u)\cdot \mathbf{n}\, dl = 0 \qquad (4.51b)$$

The solution algorithm for (4.51b) is a relatively standard procedure for extensions of Godunov's scheme in Eulerian coordinates [34-39]. The basic idea is to start with piecewise constant data in each cell with value equal to the integral cell average. Using information from cell averages, $k-th$ order piecewise polynomials are *reconstructed*:

$$u^k(x,y) = \sum_{m+n\le k} \alpha_{(m,n)} P_{(m,n)}(x-x_c, y-y_c) \qquad (4.53)$$

where $P_{(m,n)}(x-x_c, y-y_c) = (x-x_c)^m (y-y_c)^n$ and (x_c, y_c) is the cell centroid. The process of reconstruction amounts to finding the polynomial coefficients, $\alpha_{(m,n)}$. Near steep gradients and discontinuities, these polynomial coefficients maybe altered based on monotonicity arguments. Because the reconstructed polynomials vary discontinuously from cell to cell, a unique value of the solution does not exist at cell interfaces. This nonuniqueness is resolved via exact or approximate solutions of the Riemann problem. In practice, this is accomplished by supplanting the true flux function in (4.51) with a numerical flux function (described below) which produces a single unique flux given two solution states. Once the flux integral in (4.51) is carried out (either exactly or by numerical quadrature), the cell average of the solution can be evolved in time. In most cases, standard techniques for integrating ODE equations are used for the time evolution, i.e. Euler implicit, Euler explicit, Runge-Kutta. The result of the evolution process is a new collection of cell averages. The process can then be repeated. The process can be summarized in the following steps:

(1) **Reconstruction in Each Cell**: Given integral cell averages in all cells, reconstruct piecewise polynomial coefficients $\alpha_{(m,n)}$ for use in equation (4.51). For solutions containing discontinuities and/or steep gradients, monotonicity enforcement may be required.

(2) **Flux Evaluation on Each Edge**: Consider each cell boundary, ∂c_j, to be a collection of edges (or dual edges) from the mesh. Along each edge (or dual edge), perform a high order accurate flux quadrature.

(3) **Evolution in Each Cell**: Collect flux contributions in each cell and evolve in time using any time stepping scheme, i.e., Euler explicit, Euler implicit, Runge-Kutta, etc. The result of this process is once again cell averages.

By far, the most difficult of these steps is the polynomial reconstruction given cell averages. In the following paragraphs, we describe design criteria for a general reconstruction operator.

Reconstruction

The reconstruction operator serves as a finite-dimensional (possibly pseudo) inverse of the cell-averaging operator $\mathbf{A}$ whose j-th component $\mathbf{A}_j$ computes the cell average of the solution in c_j.

$$\overline{u}_j = \mathbf{A}_j u = \frac{1}{a_j}\int_{c_j} u(x,y)\, da \qquad (4.54)$$

In addition, we place the following additional requirements:

(1) **Conservation of the mean**: Simply stated, given cell averages $\overline{u}$, we require that all polynomial reconstructions u^k have the correct cell average.

$$\text{if } u^k = \mathbf{R}^k \overline{u} \text{ then } \overline{u} = \mathbf{A} u^k$$

This means that $\mathbf{R}^k$ is a right inverse of the averaging operator $\mathbf{A}$.

$$\mathbf{A}\mathbf{R}^k = I \qquad (4.55)$$

Conservation of the mean has an important implication. Unlike finite-element schemes, *Godunov schemes have a diagonal mass matrix.*

(2) **k-exactness**: We say that a reconstruction operator $\mathbf{R}^k$ is *k-exact* if $\mathbf{R}^k\mathbf{A}$ reconstructs polynomials of degree k or less exactly.

$$\text{if} \quad u \in \mathcal{P}_k \quad \text{and} \quad \overline{u} = \mathbf{A}u, \quad \text{then} \quad u^k = \mathbf{R}^k\overline{u} = u$$

In other words, $\mathbf{R}^k$ is a left-inverse of $\mathbf{A}$ restricted to the space of polynomials of degree at most k.

$$\mathbf{R}^k\mathbf{A}\Big|_{\mathcal{P}_k} = I \tag{4.56}$$

This insures that exact solutions contained in $\mathcal{P}_k$ are in fact solutions of the discrete equations. For sufficiently smooth solutions, the property of k-exactness also issures that when piecewise polynomials are evaluated at cell boundaries, the difference between solution states diminishes with increasing k at a rate proportional to h^{k+1} were h is a maximum diameter of the two cells. Figure 4.6a shows a global quartic polynomial $u \in \mathcal{P}_4$ which has been averaged in each interval.

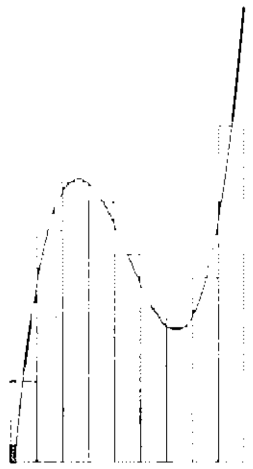


Figure 4.6a Cell averaging of quartic polynomial.

Figure 4.6b shows a quadratic reconstruction $u^k \in \mathcal{P}_2$ given the cell averages. Close inspection of figure 4.6b reveals small jumps in the piecewise polynomials at interval boundaries. These jumps would decrease even more for cubics and vanish altogether for quartic reconstruction. Property (1) requires that the area under each piecewise polynomial is exactly equal to the cell average.

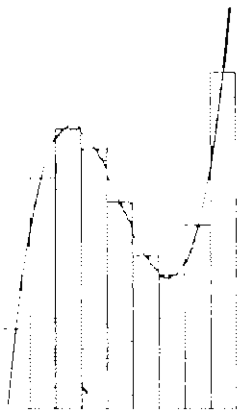


Figure 4.6b Piecewise quadratic reconstruction.

Flux Evaluation

The task here is to evaluate the flux integral appearing in (4.51).

$$\int_{\partial c_j} \mathbf{F}(u) \cdot \mathbf{n}\, dl \tag{4.57}$$

At cell interfaces, two distinct values of the solution can be obtained anywhere on the boundary of the control volume by direct evaluation of the piecewise polynomials in the two cells sharing the interface. For brevity, the states will be denoted by u^+ and u^- which should be interpreted as $(u^k)^+$ and $(u^k)^-$ where $\pm$ refers to which piecewise polynomial was used in the evaluation. Rather than use a numerical flux function derived from the exact solution of the Riemann problem, we prefer numerical flux functions based on mean value linearizations. As we will see, this actually makes certain stability proofs much clearer. Define $f(u, \mathbf{n}) = \mathbf{F}(u) \cdot \mathbf{n}$ and $a(u, \mathbf{n}) = f(u, \mathbf{n})'$, the mean value flux function is given by

$$\begin{aligned} \mathrm{h}(u^+, u^-, \mathbf{n}) =& \frac{1}{2}\left(f(u^+, \mathbf{n}) + f(u^-, \mathbf{n})\right) \\ &- \frac{1}{2}|a(\tilde{u}, \mathbf{n})|\left(u^+ - u^-\right) \end{aligned} \tag{4.58}$$

where $f(u^+) - f(u^-) = a(\tilde{u}, \mathbf{n})(u^+ - u^-)$ and $\tilde{u} = \theta u^- + (1-\theta)u^+$ for some $\theta \in [0, 1]$. Using the numerical flux function, we approximate (4.57) by

$$\int_{\partial c_j} \mathrm{h}(u^+, u^-, \mathbf{n})\, dl$$

In practice, this flux integral is never evaluated exactly, except when the data is piecewise constant. When piecewise linear functions are used, a midpoint quadrature formula is usually employed. This is used rather than the slightly more accurate trapezoidal quadrature because it requires only one flux evaluation per edge segment while the trapezoidal quadrature requires two. When considering schemes with reconstruction order k greater than one, we suggest in [41] that Gauss quadrature formulas be used. Recall that N point Gauss quadrature formulas integrate $2N-1$ polynomials exactly. These quadrature formulas give the highest accuracy for the lowest number of function evaluations. For the k-exact reconstruction discussed below, $N \geq (k+1)/2$ point Gauss quadrature formulas are used.

4.5 k-exact Reconstruction

In this section, a brief account is given of the reconstruction scheme presented in Barth and Frederickson [41] for arbitrary order reconstruction. Upon first inspection, the use of high order reconstruction appears to be an expensive proposition. The present reconstruction strategy optimizes the efficiency of the reconstruction by precomputing as a *one time* preprocessing step the set of weights $\mathbf{W}_j$ in each cell $\mathbf{c}_j$ with neighbor set $\mathcal{N}_{c_j}$ such that

$$\alpha_{(m,n)} = \sum_{i \in \mathcal{N}_{c_j}} W_{(m,n)i} \overline{u}_i \tag{4.59}$$

where $\alpha_{(m,n)}$ are the polynomial coefficients. This effectively reduces the problem of reconstruction to multiplication of predetermined weights and cell averages to obtain polynomial coefficients.

During the preprocessing to obtain the reconstruction weights $\mathbf{W}_j$ a coordinate system with origin at the centroid of $\mathbf{c}_j$ is assumed to minimize roundoff errors. To insure that the reconstruction is invariant to affine transformations, we then temporarily transform (rotate and scale) to another coordinate system $(\overline{x}, \overline{y})$ which is normalized to the cell $\mathbf{c}_j$

$$\begin{bmatrix} \overline{x} \\ \overline{y} \end{bmatrix} = \begin{bmatrix} \mathbf{D}_{1,1} & \mathbf{D}_{1,2} \\ \mathbf{D}_{2,1} & \mathbf{D}_{2,2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

with the matrix $\mathbf{D}$ is chosen so that

$$\mathbf{A}_j(\overline{x}^2) = \mathbf{A}_j(\overline{y}^2) = 1$$

$$\mathbf{A}_j(\overline{x}\,\overline{y}) = \mathbf{A}_j(\overline{y}\,\overline{x}) = 0$$

Polynomials on $\mathbf{c}_j$ are temporarily represented using the polynomial basis functions

$$\overline{P} = [1, \overline{x}, \overline{y}, \overline{x}^2, \overline{xy}, \overline{y}^2, \overline{x}^3, ...].$$

Note that polynomials in this system are easily transformed to the standard cell-centroid basis

$$\overline{x}^m \overline{y}^n = \sum_{s+t \leq k} \binom{m}{s} \binom{n}{t} \mathbf{D}_{1,1}^s \mathbf{D}_{1,2}^{m-s} \mathbf{D}_{2,1}^t \mathbf{D}_{2,2}^{n-t} x^{s+t} y^{m+n-s-t}$$

Since $0 \leq s+t \leq k$ and $0 \leq m+n-s-t \leq k$, we can reorder and rewrite in terms of the standard and transformed basis polynomials

$$\overline{P}_{(m,n)} = \sum_{s+t \leq k} G_{m,n}^{s,t} P_{(s,t)} \tag{4.60}$$

Satisfaction of conservation of the mean is guaranteed by introducing into the transformed coordinate system *zero mean* basis polynomials $\overline{P}^0$ in which all but the first have zero cell average, i.e $\overline{P}^0 = [1, \overline{x}, \overline{y}, \overline{x}^2 - 1, \overline{x}\,\overline{y}, \overline{y}^2 - 1, \overline{x}^3 - A_j(\overline{x}^3), ...]$. Note that using these polynomials requires a minor modification of (4.60) but retains the same form:

$$\overline{P}^0_{(m,n)} = \sum_{s+t \leq k} \overline{G}_{m,n}^{s,t} P_{(s,t)} \tag{4.61}$$

Given this preparatory work, we are now ready to describe the formulation of the reconstruction algorithm.

Minimum Energy (Least-Squares) Reconstruction

We note that the set of cell neighbors $\mathcal{N}_j$ must contain at least $(k+1)(k+2)/2$ cells $\mathbf{c}_j$ if the reconstruction operator $\mathbf{R}_j^k$ is to be k-exact. That $(k+1)(k+2)/2$ cells is not sufficient in all situations is easily observed. If, for example, the cell-centers all lie on a single straight line one can find a linear function u such that $\mathbf{A}_j(u) = 0$ for every cell $\mathbf{c}_j$, which means that reconstruction of u is impossible. In other cases a k-exact reconstruction operator $\mathbf{R}_j^k$ may exist, but due to the geometry may be poorly conditioned.

Our approach is to work with a slightly larger support containing more than the minimum number of cells. In this case the operator $\mathbf{R}_j^k$ is likely to be nonunique, because various subsets would be able to support reconstruction operators of degree k. Although all would reproduce a polynomial of degree k exactly, if we disregard roundoff, they would differ in their treatment of non-polynomials, or of polynomials of degree higher than k. Any k-exact reconstruction operator $\mathbf{R}_j^k$ is a weighted average of these basic ones. Our approach is to choose the one of minimum Frobenius norm. This operator is optimal, in a certain sense, when the function we are reconstructing is not exactly a polynomial of degree k, but one that has been perturbed by the addition of Gaussian noise, for it minimizes the expected deviation from the unperturbed polynomial in a certain rather natural norm.

As we begin the formulation of the reconstruction preprocessing algorithm, the reader is reminded that the task at hand is to calculate the weights $\mathbf{W}_j$ for each cell $\mathbf{c}_j$ which when applied via (4.59) produce piecewise polynomial approximations. We begin by first rewriting the piecewise polynomial (4.53) for cell $\mathbf{c}_j$ in terms of the reconstruction weights (4.59)

$$u^k(x,y) = \sum_{m+n\le k} P_{(m,n)} \sum_{i\in\mathcal{N}_{c_j}} W_{(m,n)i}\overline{u}_i$$

or equivalently

$$u^k(x,y) = \sum_{i\in\mathcal{N}_{c_j}} \overline{u}_i \sum_{m+n\le k} W_{(m,n)i}P_{(m,n)}$$

Polynomials of degree k or less are equivalently represented in the transformed coordinate system using zero mean polynomials

$$u^k(x,y) = \sum_{i\in\mathcal{N}_{c_j}} \overline{u}_i \sum_{m+n\le k} W'_{(m,n)i}\overline{P}^0_{(m,n)} \quad (4.62)$$

Using (4.61), we can relate weights in the transformed system to weights in the original system

$$W_{(s,t)i} = \sum_{m+n\le k} \overline{G}^{s,t}_{m,n} W'_{(m,n),i} \quad (4.63)$$

We satisfy k-exactness by requiring that (4.62) is satisfied for all linear combinations of $\overline{P}^0_{(s,t)}(x,y)$ such that $s+t\le k$. In particular, if $u^k(x,y) = \overline{P}^0_{(s,t)}(x,y)$ for some $s+t\le k$ then

$$\overline{P}^0_{(s,t)}(x,y) = \sum_{m+n\le k} \overline{P}^0_{(m,n)} \sum_{i\in\mathcal{N}_{c_j}} W'_{(m,n)i}A_i(\overline{P}^0_{(s,t)})$$

This is satisfied if for all $s+t, m+n \le k$

$$\sum_{i\in\mathcal{N}_{c_j}} W'_{(m,n)i}A_i(\overline{P}^0_{(s,t)}) = \delta^{st}_{mn}$$

Transforming basis polynomials back to the original coordinate system we have

$$\sum_{i\in\mathcal{N}_{c_j}} W'_{(m,n)i} \sum_{u+v\le k} \overline{G}^{u,v}_{s,t} A_i(P_{(u,v)}) = \delta^{st}_{mn} \quad (4.64)$$

This can be locally rewritten in matrix form as

$$\mathbf{W}'_j\mathbf{A}'_j = \mathbf{I} \quad (4.65a)$$

and transformed in terms of the standard basis weights via

$$\mathbf{W}_j = \mathbf{G}\mathbf{W}'_j \quad (4.65b)$$

Note that $\mathbf{W}'_j$ is a $(k+1)(k+2)/2$ by $\mathcal{N}_j$ matrix and $\mathbf{A}'_j$ has dimensions $\mathcal{N}_j$ by $(k+1)(k+2)/2$. To solve (4.65a) in the optimum sense described above, an $\mathbf{L}_j\mathbf{Q}_j$ decomposition of $\mathbf{A}'_j$ is performed where the orthogonal matrix $\mathbf{Q}_j$ and the lower triangular matrix $\mathbf{L}_j$ have been constructed using a modified Gram-Schmidt algorithm (or a sequence of Householder reflections). The weights $\mathbf{W}'_j$ are then given by

$$\mathbf{W}'_j = \mathbf{Q}^*_j\mathbf{L}^{-1}_j$$

Applying (4.63) these weights are transformed to the standard centroid basis and the preprocessing step is complete.

We now show a few results presented earlier in reference [41]. The first calculation involves the reconstruction of a sixth order polynomial with random normalized coefficients which has been cell averaged onto a random mesh. Figures 4.7a-b show a sample mesh and the absolute L_2 error of the reconstruction for various meshes and reconstruction degree.

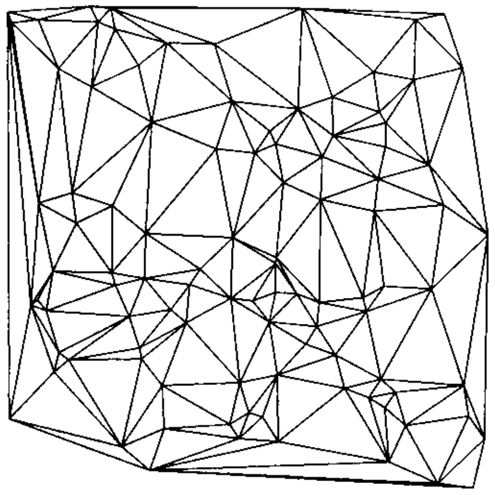


Figure 4.7a Random mesh.

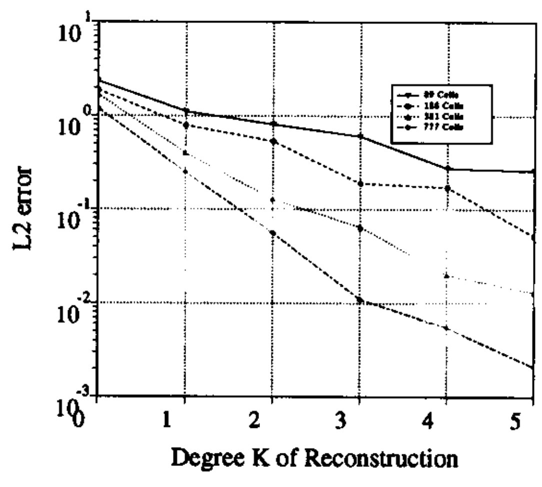


Figure 4.7b L_2 error of reconstruction.

The reconstruction algorithm has also been tested on more realistic problems. Figures 4.8a-c show a mesh and reconstructions (linear and quadratic) of a cell averaged density field corresponding to a Ringleb flow, an exact hodograph solution of the gasdynamic equations, see [42].

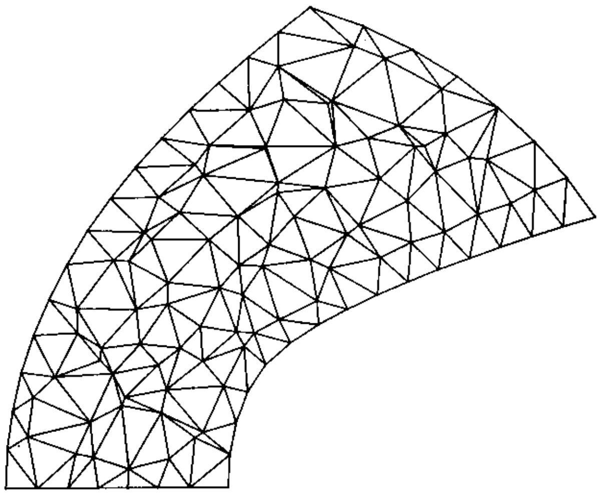


Figure 4.8a Randomized mesh for Ringleb flow.

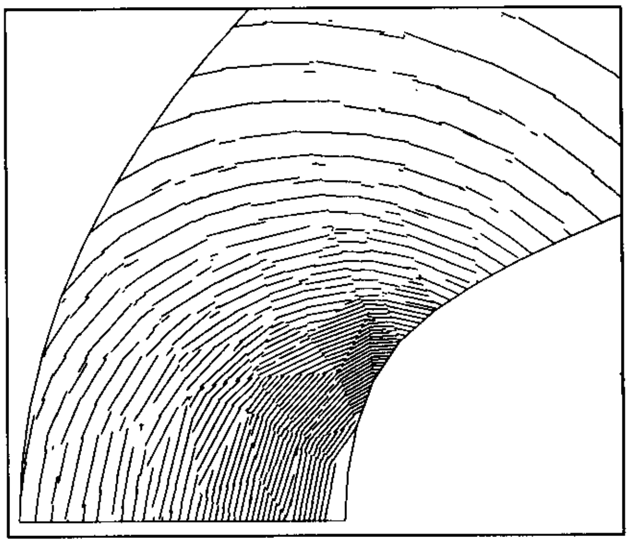


Figure 4.8b Piecewise linear reconstruction of Ringleb flow.

The reader should note that the use of piecewise contours gives a crude visual critique as to how well the solution is represented by the piecewise polynomials. The improvement from linear to quadratic is dramatic in the case of Ringleb flow. A later section will show actual numerical solutions computed using this reconstruction operator.

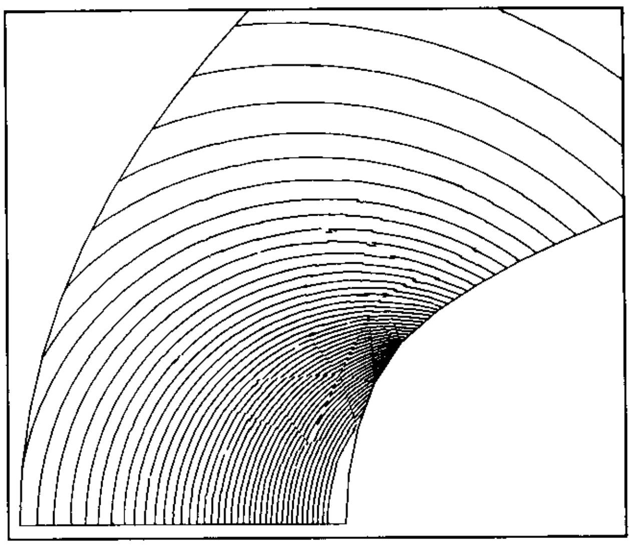


Figure 4.8c Piecewise quadratic reconstruction of Ringleb flow.

4.6 Upwind Advection Scheme with $k = 0$ Reconstruction

This is the simplest (first order) approximation in which the polynomial behavior in each cell, $\mathbf{c}_j$, is a constant value equal to the cell average.

$$u^{k=0}(x,y) = \overline{u}_j \quad \text{for } u^k \in c_j \tag{4.66}$$

The flux formula then simplifies to the following form (for clarity $\overline{u}_j$ is locally numbered $\overline{u}_0$ as shown in figure 4.1a)

$$\begin{aligned} \mathrm{h}(u^+, u^-, \vec{\mathbf{n}})_i =& \mathrm{h}(\overline{u}_i, \overline{u}_0, \vec{\mathbf{n}}_i) \\ =& \frac{1}{2}(f(\overline{u}_0, \vec{\mathbf{n}}_i) + f(\overline{u}_i, \vec{\mathbf{n}}_i)) \\ & -\frac{1}{2}|a(\tilde{u}_i, \vec{\mathbf{n}}_i)|(\overline{u}_i - \overline{u}_0) \end{aligned} \tag{4.67}$$

In this formula, $\vec{\mathbf{n}}_i = \int_{\mathbf{R}'_{i-1/2}}^{\mathbf{R}'_{i+1/2}}$ for any simple path. By summing over all edges of the control volume, the entire scheme for c_j is written

$$\begin{aligned} \frac{\partial}{\partial t} \int_{c_j} \overline{u}_0 \, da + & \sum_{i=1}^{d(c_j)} \frac{1}{2} (f(\overline{u}_0, \vec{\mathbf{n}}_i) + f(\overline{u}_i, \vec{\mathbf{n}}_i)) \\ - & \sum_{i=1}^{d(c_j)} \frac{1}{2} |a(\tilde{u}_i, \vec{\mathbf{n}}_i)| \, (\overline{u}_i - \overline{u}_0) = 0 \end{aligned} \tag{4.68}$$

It is not difficult to prove stability and monotonicity of this scheme.

Monotonicity and Stability

Recall that the flux function was constructed from a mean value linearization such that

$$f(u_i,\vec{n}_i) - f(u_0,\vec{n}_i) = a(\tilde{u}_i,\mathbf{n}_i)\,(u_i-u_0) \quad (4.69)$$

with $\tilde{u}_i = \theta u_0 + (1-\theta)u_i$, $\theta \in [0,1]$. This permits regrouping terms into the following form:

$$\frac{\partial}{\partial t}\int_{c_j} \overline{u}_0\,da + \sum_{i=1}^{d(c_j)} f(\overline{u}_0,\vec{n}_i) + \sum_{i=1}^{d(c_j)} \frac{1}{2}\left(a(\tilde{u}_i,\vec{n}_i) - |a(\tilde{u}_i,\vec{n}_i)|\right)(\overline{u}_i - \overline{u}_0) = 0 \quad (4.70)$$

For any closed control volume, we have that

$$\sum_{i=1}^{d(c_j)} f(\overline{u}_0,\vec{n}_i) = 0$$

Combining the remaining terms yields a final form for analysis ($a = a^+ + a^-$, $|a| = a^+ - a^-$):

$$\frac{\partial}{\partial t}\int_{c_j} \overline{u}_0\,da + \sum_{i=1}^{d(c_j)} a(\tilde{u}_i,\vec{n}_i)^-(\overline{u}_i - \overline{u}_0) = 0 \quad (4.71)$$

To verify the monotonicity of the scheme at steady state, set the time term to zero and solve for $\overline{u}_0$.

$$\overline{u}_0 = \frac{\sum_{i=1}^{d(c_j)} a(\tilde{u}_i,\vec{n}_i)^-\ \overline{u}_i}{\sum_{i=1}^{d(c_j)} a(\tilde{u}_i,\vec{n}_i)^-} = \sum_{i=1}^{d(c_j)} \alpha_i \overline{u}_i \quad (4.72)$$

All weights α_i are positive and sum to unity. The scheme is monotone since $\overline{u}_0$ is a positive weighted average of all neighbors. This implies a *maximum principle* since $\overline{u}_0$ is bounded from above and below by the maximum and minimum of neighboring values (and itself), $\overline{u}_{max}$ and $\overline{u}_{min}$.

$$\overline{u}_{min} \le \overline{u}_0 \le \overline{u}_{max} \quad (4.73)$$

For explicit time stepping, a CFL-like condition is obtained for monotonicity. For Euler explicit time stepping, we have the time approximation,

$$\frac{\partial}{\partial t}\frac{1}{A_c}\int_c \overline{u}_0\,da \approx \frac{\overline{u}_0^{n+1} - \overline{u}_0^n}{\Delta t}$$

which results in the following scheme:

$$\begin{aligned}\overline{u}_0^{n+1} &= \overline{u}_0^n - \frac{\Delta t}{A_c}\sum_{i=1}^{d(c_j)} a(\tilde{u}_i,\vec{n}_i)^-(\overline{u}_i^n - \overline{u}_0^n)\\ &= \sum_{i=0}^{d(c_j)} \alpha_i \overline{u}_i^n\end{aligned} \quad (4.74)$$

It should be clear that coefficients in (4.74) sum to unity. To prove monotonicity in time and space, it is sufficient to show positivity of coefficients. By inspection we have that $\alpha_i \ge 0 \ \ \forall\, i > 0$. To guarantee monotonicity requires that $\alpha_0 \ge 0$.

$$\alpha_0 = 1 + \frac{\Delta t}{A_c}\sum_{i=1}^{d(c_j)} a(\tilde{u}_i,\vec{n}_i)^- \ \ge 0 \quad (4.75)$$

Thus, a CFL-like condition is obtained which insures monotonicity and stability.

$$\Delta t \le -\frac{A_c}{\sum_{i=1}^{d(c_j)} a(\tilde{u}_i,\vec{n}_i)^-} \quad (4.76)$$

Note that in one dimension, this number corresponds to the conventional CFL number. In multiple space dimensions, this inequality is sufficient but not necessary for stability. In practice somewhat larger timestep values may be used.

Conclusion: *The upwind algorithm (4.68) using piecewise constant data satisfies a discrete maximum principle for general unstructured meshes.*

4.7 Upwind Advection Schemes with Linear ($k = 1$) Reconstruction

In this section, we consider advection schemes based on linear reconstruction. The process of linear reconstruction in one dimension is depicted in figure 4.9.

Figure 4.9 Linear Reconstruction of cell-averaged data.

One of the most important observations concerning linear reconstruction is that we can dispense with the notion of cell averages as unknowns by reinterpreting the unknowns as pointwise values of the solution sampled at the centroid (midpoint in 1-D) of the control volume. This well known result greatly simplifies schemes based on linear reconstruction. The linear reconstruction in each interval shown in figure 4.9 was obtained by a simple central-difference formula given point values of the solution at the midpoint of each interval.

In section 4.3, results for the Ringleb flow with linear reconstruction were presented. The reconstruction strategy presented there satisfies all the design requirements of the reconstruction operator. For linear reconstruction, simpler formulations are possible which exploit the edge data structure. Several of these reconstruction schemes are given below. Note that for steady-state computations, conservation of the mean in the data reconstruction is not necessary. The implication of violating this conservation is that a *nondiagonal* mass matrix appears in the time integral. Since time derivatives vanish at steady-state, the effect of this mass matrix vanishes at steady-state. The reconstruction schemes presented below assume that solution variables are placed at the vertices of the mesh, which may not be at the precise centroid of the control volume, thus violating conservation of the mean. The schemes can all be implemented using an edge data structure and satisfy k-exactness for linear functions.

4.7a Green-Gauss Reconstruction

This reconstruction exploits the gradient calculation (4.19) studied earlier in section 4.3:

$$(\nabla u)_{v_0} = \frac{1}{A_0} \sum_{i \in \mathcal{I}_0} \frac{1}{2}(u_i + u_0)\vec{\mathbf{n}}_{0i} \qquad (4.77)$$

where $\vec{\mathbf{n}}_{0i}$ is the vector normal associated with the edge $e(v_0, v_i)$. This approximation extends naturally to three dimensions, see Barth [43].

4.7b Linear Least-Squares (L_2) Reconstruction

To derive this reconstruction technique, consider a vertex v_0 and suppose that the solution varies linearly over the support of adjacent neighbors of the mesh. In this case, the change in vertex values of the solution along an edge $e(v_i, v_0)$ can be calculated by

$$(\nabla u^h)_0 \cdot (\mathbf{R}_i - \mathbf{R}_0) = u_i - u_0 \qquad (4.78)$$

This equation represents the scaled projection of the gradient along the edge $e(v_i, v_0)$. A similar equation could be written for all incident edges subject to an arbitrary weighting factor. The result is the following matrix equation, shown here in three dimensions:

$$\begin{bmatrix} w_1\Delta x_1 & w_1\Delta y_1 & w_1\Delta z_1 \\ \vdots & \vdots & \vdots \\ w_n\Delta x_n & w_n\Delta y_n & w_n\Delta z_n \end{bmatrix} \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \begin{pmatrix} w_1(u_1-u_0) \\ \vdots \\ w_n(u_n-u_0) \end{pmatrix} \qquad (4.79)$$

or in symbolic form $\mathcal{L}\ \nabla u = \mathbf{f}$ where

$$\mathcal{L} = \left[\vec{\mathbf{L}}_1 \quad \vec{\mathbf{L}}_2 \quad \vec{\mathbf{L}}_3 \right] \qquad (4.80)$$

in three dimensions. Exact calculation of gradients for linear u is guaranteed if any three row vectors $w_i(\mathbf{R}_i - \mathbf{R}_0)$ span all of 3 space. This implies linear independence of $\vec{\mathbf{L}}_1$, $\vec{\mathbf{L}}_2$, and $\vec{\mathbf{L}}_3$. The system can then be solved via a Gram-Schmidt process, i.e.,

$$\begin{bmatrix} \vec{\mathbf{V}}_1 \\ \vec{\mathbf{V}}_2 \\ \vec{\mathbf{V}}_3 \end{bmatrix} \left[\vec{\mathbf{L}}_1 \quad \vec{\mathbf{L}}_2 \quad \vec{\mathbf{L}}_3 \right] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (4.81)$$

The row vectors $\vec{\mathbf{V}}_i$ are given by $\vec{\mathbf{V}}_i = \frac{\vec{\mathbf{U}}_i}{(\vec{\mathbf{L}}_i \cdot \vec{\mathbf{U}}_i)}$ where

$$\begin{aligned} \vec{\mathbf{U}}_1 =& (l_{33}l_{22} - l_{23}l_{23})\vec{\mathbf{L}}_1 - (l_{33}l_{12} - l_{23}l_{13})\vec{\mathbf{L}}_2 \\ & -(l_{22}l_{13} - l_{23}l_{12})\vec{\mathbf{L}}_3 \\ \vec{\mathbf{U}}_2 =& (l_{33}l_{11} - l_{13}l_{13})\vec{\mathbf{L}}_2 - (l_{33}l_{12} - l_{13}l_{23})\vec{\mathbf{L}}_1 \\ & -(l_{11}l_{23} - l_{13}l_{12})\vec{\mathbf{L}}_3 \\ \vec{\mathbf{U}}_3 =& (l_{11}l_{22} - l_{12}l_{12})\vec{\mathbf{L}}_3 - (l_{22}l_{13} - l_{12}l_{23})\vec{\mathbf{L}}_1 \\ & -(l_{11}l_{23} - l_{12}l_{13})\vec{\mathbf{L}}_2 \end{aligned}$$

and $l_{ij} = (\vec{\mathbf{L}}_i \cdot \vec{\mathbf{L}}_j)$.

Note that reconstruction of N independent variables in $\mathbf{R}^d$ implies $\binom{d+1}{2} + d\,N$ inner product sums. Since only $d\,N$ of these sums involves the solution variables themselves, the remaining sums could be precalculated and stored in computer memory. This makes the present scheme competitive with the Green-Gauss reconstruction. Using the edge data structure, the calculation of inner product sums can be calculated for *arbitrary* combinations of polyhedral cells. In all cases linear functions are reconstructed exactly. We demonstrate this idea by example:

```
For $k = 1, n(e)$                                  ! Loop through edges of mesh
    $j_1 = e^{-1}(k,1)$                            ! Pointer to edge origin
    $j_2 = e^{-1}(k,2)$                            ! Pointer to edge destination
    $dx = w(k) \cdot (x(j_2) - x(j_1))$            ! Weighted $\Delta x$
    $dy = w(k) \cdot (y(j_2) - y(j_1))$            ! Weighted $\Delta y$
    $l_{11}(j_1) = l_{11}(j_1) + dx \cdot dx$      ! $l_{11}$ orig sum
    $l_{11}(j_2) = l_{11}(j_2) + dx \cdot dx$      ! $l_{11}$ dest sum
    $l_{12}(j_1) = l_{12}(j_1) + dx \cdot dy$      ! $l_{12}$ orig sum
    $l_{12}(j_2) = l_{12}(j_2) + dx \cdot dy$      ! $l_{12}$ dest sum
    .
    .
    .
    $du = w(k) \cdot (u(j_2) - u(j_1))$            ! Weighted $\Delta u$
    $lu_3(j_1) = lu_3(j_1) + dz \cdot du$          ! $lu_3$ orig sum
    $lu_3(j_2) = lu_3(j_2) + dz \cdot du$      ! $lu_3$ dest sum
Endfor
```

This formulation provides freedom in the choice of weighting coefficients, w_i. These weighting coefficients can be a function of the geometry and/or solution. Classical approximations in one dimension can be recovered by choosing geometrical weights of the form $w_i = 1./|\mathbf{R}_i - \mathbf{R}_0|^t$ for values of $t = 0, 1, 2$. The L_2 gradient calculation technique is optimal in a weighted least squares sense and determines gradient coefficients with least sensitivity to Gaussian noise. This is an important property when dealing with highly distorted (stretched) meshes.

4.7c Data Dependent Reconstruction

Both the Green-Gauss and L_2 gradient calculation techniques can be generalized to include data dependent (i.e. solution dependent) weights. In the case of Green-Gauss formulation, the sum

$$\sum_{i\in\mathcal{I}_0} \frac{1}{2}(u_0 + u_i)\vec{\mathbf{n}}_{0i}$$

is replaced by

$$\sum_{i\in\mathcal{I}_0} p^-_{0i}\frac{1}{2}(u_0+u_i)\vec{\mathbf{n}}_{0i}+p^+_{0i}\frac{1}{2}\left((\nabla u)_0\cdot(\mathbf{R}_i-\mathbf{R}_0)\right)\vec{\mathbf{n}}_{0i} \tag{4.82}$$

If the $p^\pm_{0i}$ are chosen such that $p^-_{0i} + p^+_{0i} = 1$ then the gradient calculation is exact whenever the solution varies linearly over the support. In two space dimensions, equation (4.82) implies the solution of a linear 2×2 system of the form

$$\begin{bmatrix} A_0 - m_{xx} & -m_{xy} \\ -m_{yx} & A_0 - m_{yy} \end{bmatrix}\begin{pmatrix} u_x \\ u_y \end{pmatrix} = \sum_{i\in\mathcal{I}_0} p^-_{0i}\frac{1}{2}(u_0+u_i)\vec{\mathbf{n}}_{0i}$$

where

$$m_{xx} = \sum_{i\in\mathcal{I}_0} p^+_{0i}\Delta x_i n_{x_i}, \quad m_{yy} = \sum_{i\in\mathcal{I}_0} p^+_{0i}\Delta y_i n_{y_i}$$

$$m_{xy} = \sum_{i\in\mathcal{I}_0} p^+_{0i}\Delta x_i n_{y_i}, \quad m_{yx} = \sum_{i\in\mathcal{I}_0} p^+_{0i}\Delta y_i n_{x_i}$$

Care must be exercised in the selection of $p^\pm$ in order that the system be invertible. This is similar to the spanning space requirement of the L_2 gradient calculation technique.

4.7d Monotonicity Enforcement

When solution discontinuites and steep gradients as present, additional steps must be taken to prevent oscillations from developing in the numerical solution. One way to do this was pioneered by van Leer [35] in the late 1970's. The basic idea is to take the reconstructed piecewise polynomials and enforce strict monotonicity in the reconstruction. Monotonicity in this context should be interpreted to mean that the value of the reconstructed polynomial does not exceed the minimum and maximum of neighboring cell averages. In other words, the final reconstruction must guarantee that no new extrema have been created. This will be referred to as 'monotonicity property 1.' When a new extremum is produced, the slope of the reconstruction in that interval is reduced until monotonicity is restored. This implies that at a local minimum or maximum in the cell averaged data the slope in 1-D is *always* reduced to zero, see for example figure 4.10.

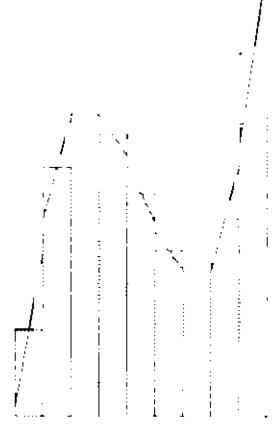


Figure 4.10 Linear Reconstruction with monotone limiting.

Another property (referred to hereafter as 'property 2') of the monotonicity enforcement is motivated by the stability proof associated with the higher order accurate schemes (presented in section 4.7e). In one dimension, property 2 can be characterized as the requirement that the new reconstruction not produce a reconstructed solution variation, $\int |du|$, which is larger than the piecewise constant value. If property 2 is violated then the slopes must be reduced until the solution variation is satisfied. This situation is depicted in figure 4.11. For arbitrary unstructured grids, a *sufficient* condition is that the differences in the extrapolated states at a cell interface quadrature point be of the same sign as the difference in the piecewise constant values, i.e.

$$\frac{u^+ - u^-}{\overline{u}^+ - \overline{u}^-} \geq 0, \quad \text{(property 2)}$$

when combined with property 1, the following inequality exists:

$$1 \geq \frac{u^+ - u^-}{\overline{u}^+ - \overline{u}^-} \geq 0 \tag{4.83}$$

This inequality is crucial in the stability proof given below.

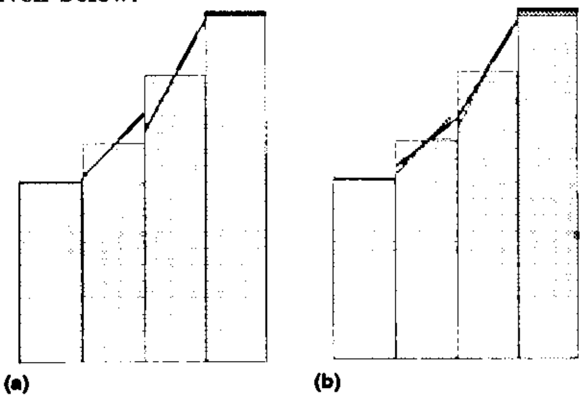


Figure 4.11 (a) Reconstruction profile with increased variation violating monotonicity property 2. (b) Profile after modification to satisfy monotonicity property 2.

In Barth and Jespersen [40], we gave a simple recipe for invoking property 1. Consider writing the linearly reconstructed data in the following form:

$$u^k(x,y)_j = \overline{u}_j + \nabla u^k_{c_j} \cdot (\mathbf{R} - \mathbf{R}_j) \qquad (4.84a)$$

Now consider a "limited" form of this piecewise linear distribution.

$$u^k(x,y)_j = \overline{u}_j + \Phi_j \nabla u^k_{c_j} \cdot (\mathbf{R} - \mathbf{R}_j) \qquad (4.84b)$$

The idea is to find the largest admissible Φ_j while invoking a monotonicity principle that values of the linearly reconstructed function must not exceed the maximum and minimum of neighboring centroid values (including the centroid value in c_j). To do this, first compute

$$u_j^{min} = \min(\overline{u}_j, \overline{u}_{neighbors})$$

and

$$u_j^{max} = \max(\overline{u}_j, \overline{u}_{neighbors})$$

then require that

$$u_j^{min} \le u(x,y)_j^k \le u_j^{max} \qquad (4.85)$$

For linear reconstructions, extrema in $u(x,y)_j^k$ occur at the vertices of the control volume and sufficient conditions for (4.85) can be easily obtained. For each vertex of the cell compute $u_{i*} = u^k(x_i,y_i)_j$, $i = 1, N_{c_j}$ to determine the limited value, ϕ_i, which satisfies (4.84):

$$\phi_i = \begin{cases} \min(1, \frac{u_j^{max} - \overline{u}_j}{u_i - \overline{u}_j}), & \text{if } u_i - \overline{u}_j > 0 \\ \min(1, \frac{u_j^{min} - \overline{u}_j}{u_i - \overline{u}_j}), & \text{if } u_i - \overline{u}_j < 0 \\ 1 & \text{if } u_i - \overline{u}_j = 0 \end{cases}$$

with $\Phi_j = \min(\phi_1, \phi_2, \phi_3, ..., \phi_{N_{c_j}})$. In practice, the reconstructed polynomial may be calculated at the flux quadrature points instead of the vertices of the control volume with a negligible degradation in monotonicity. In the implementation of property 2, we prefer a "symmetric" reduction of slopes. In other words, at interfaces violating property 2, both of the two cells sharing that interface reduce their slope until (4.83) is satisfied.

When the above procedures are combined with the flux function given earlier (4.58),

$$\begin{aligned} \mathrm{h}(u^+, u^-, \mathbf{n}) =& \frac{1}{2}\left(f(u^+,\mathbf{n}) + f(u^-,\mathbf{n})\right) \\ & -\frac{1}{2}|a(\tilde{u},\mathbf{n})|\left(u^+ - u^-\right) \end{aligned} \qquad (4.58)$$

the resulting scheme has very good shock resolving characteristics. To demonstrate this fact, we consider the scalar nonlinear hyperbolic problem suggested by Struijs, Deconinck, *et al* [44]. The equation is a multidimensional form of Burger's equation.

$$u_t + (u^2/2)_x + u_y = 0$$

We solve the equation in a square region $[0, 1.5] \times [0, 1.5]$ with boundary conditions: $u(x,0) = 1.5 - 2x$, $x \le 1$, $u(x,0) = -.5$, $x > 1$, $u(0,y) = 1.5$, and $u(1.5,y) = -.5$. Figures 4.12 and 4.13 show carpet plots and contours of the solution on regular and irregular meshes.

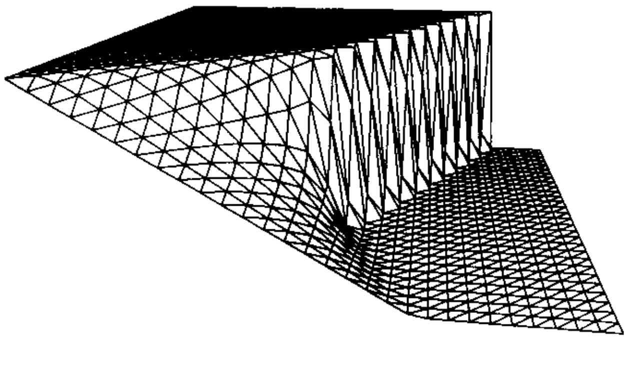


Figure 4.12a Carpet plot of Burger's equation solution on regular mesh.

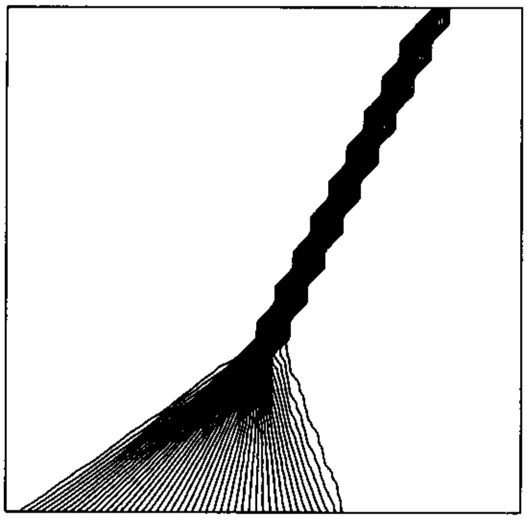


Figure 4.12b Solution Contours on regular mesh.

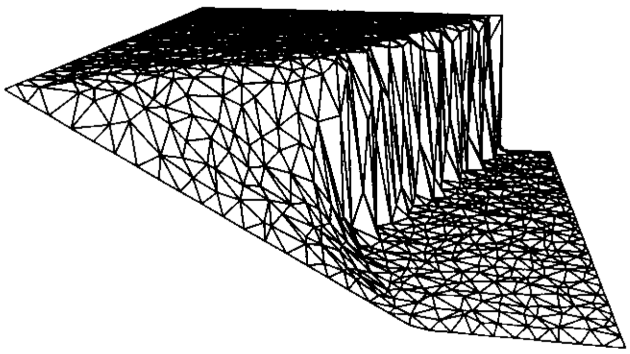


Figure 4.13a Carpet plot of Burger's equation solution on irregular mesh.

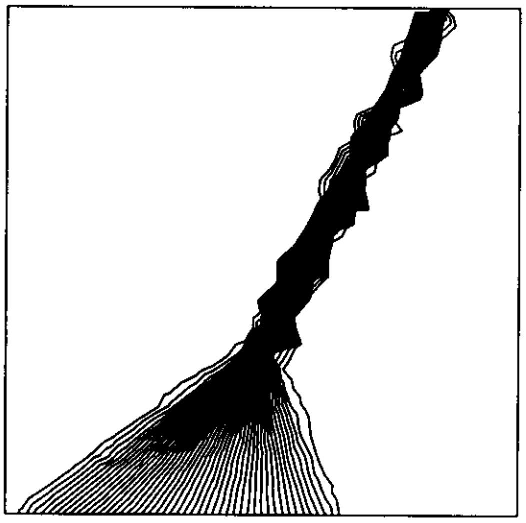


Figure 4.13b Solution contours.

Note that the carpet plots indicate that the numerical solution on both meshes is monotone. Even so, most people would prefer the solution on the regular mesh. This is an unavoidable consequence of irregular meshes. The only remedy appears to be mesh adaptation. Similar results for the Euler equations will be shown on irregular meshes in a future section.

4.7e Stability Analysis via Energy Methods

Consider once again the local mesh shown in figure 4.1a with local index about a vertex v_0. In the analysis performed below, we consider energy stability of schemes of the following form

$$\frac{\partial}{\partial t}\overline{u}_0 A_0 = -\sum_{i=1}^{d(c_j)} \mathbf{h}(u^+, u^-, \vec{\mathbf{n}})_{0i} \qquad (4.86)$$

using linear reconstruction with limiting. Note that in this analysis all boundary effects will be ignored. In section 4.5, stability of the first order upwind scheme was proven using monotonicity analysis. Before considering the higher order schemes, we briefly digress to show stability of the first order upwind scheme using energy methods. Using the same techniques, energy stability of the high order schemes with reconstruction and limiting will be shown.

Energy Analysis for the $k = 0$ scheme

In this case, the flux takes the simple form and the scheme for a single vertex v_0 can be written as indicated below

$$\underbrace{(A_0\overline{u}_0)}_{\mathcal{D}\mathbf{u}}{}_t + \underbrace{\sum_{i=1}^{d(c_j)} \frac{1}{2}\left(f(\overline{u}_0, \vec{\mathbf{n}}_i) + f(\overline{u}_i, \vec{\mathbf{n}}_i)\right)}_{\mathcal{L}_a\mathbf{u}}$$
$$- \underbrace{\sum_{i=1}^{d(c_j)} \frac{1}{2}|a(\tilde{u}_i, \vec{\mathbf{n}}_i)|\,(\overline{u}_i - \overline{u}_0)}_{\mathcal{L}_d\mathbf{u}} = 0 \qquad (4.87)$$

or in symbolic operator form, where $\mathbf{u}$ denotes the solution vector, i.e. $\mathbf{u} = [\overline{u}_1, \overline{u}_2, \overline{u}_3, ..., \overline{u}_N]^T$. In this symbolic form, the scheme is written as

$$(D\mathbf{u})_t + \mathcal{L}_a\mathbf{u} - \mathcal{L}_d\mathbf{u} = 0 \qquad (4.88)$$

where D is a positive diagonal matrix containing the area of each control volume. $\mathcal{L}_a$ and $\mathcal{L}_s$ represent the advective and diffusive operators in this linear scheme. The energy of the system (4.88) is given by the following equation:

$$\left(\mathbf{u}^T D_0 \mathbf{u}\right)_t + \mathbf{u}^T(\mathcal{L}_a + \mathcal{L}_a^T)\,\mathbf{u} - \mathbf{u}^T(\mathcal{L}_d + \mathcal{L}_d^T)\,\mathbf{u} = 0 \qquad (4.89)$$

It is a straightforward exercise to show that in the linear case, $\mathcal{L}_a$ and $\mathcal{L}_a^T$ are skew-symmetric (isoenergetic) operators, hence

$$\mathbf{u}^T\left(\mathcal{L}_a + \mathcal{L}_a^T\right)\mathbf{u} = 0.$$

The diffusive operator $\mathcal{L}_d$ is symmetric which reduces the energy equation to the following form:

$$\frac{1}{2}\left(\mathbf{u}^T D_0 \mathbf{u}\right)_t - \mathbf{u}^T \mathcal{L}_d \mathbf{u} = 0 \tag{4.90}$$

From symmetry and application of the eigenvalue circle theorem, it is easily shown that $\mathcal{L}_d$ is a symmetric, negative semi-definite matrix operator which implies that

$$\mathbf{u}^T \mathcal{L}_d \mathbf{u} \le 0$$

for all $\mathbf{u}$. This establishes that the scheme is energy stable since

$$\left(\mathbf{u}^T D_0 \mathbf{u}\right)_t \le 0$$

Energy Analysis for the $k = 1$ scheme

We now consider the advection scheme with linear reconstruction. The interface states for the edge of the control volume separating cells c_0 and c_i are denoted by u_0^+ and u_i^-, respectively. The scheme is written in the familiar form:

$$\underbrace{(A_0\overline{u}_0)_t}_{\mathcal{D}\mathbf{u}} + \underbrace{\sum_{i=1}^{d(c_j)} \frac{1}{2}\left(f(u_0^+, \vec{\mathbf{n}}_i) + f(u_i^-, \vec{\mathbf{n}}_i)\right)}_{\mathcal{L}_a\mathbf{u}}$$
$$\underbrace{- \sum_{i=1}^{d(c_j)} \frac{1}{2}|a(\tilde{u}_i, \vec{\mathbf{n}}_i)|\left(u_i^- - u_0^+\right)}_{\mathcal{L}_d\mathbf{u}} = 0 \tag{4.91}$$

Consider rewriting equation (4.91) using the identity

$$u_i^- - u_0^+ = \left(\frac{u_i^- - u_0^+}{\overline{u}_i - \overline{u}_0}\right)(\overline{u}_i - \overline{u}_0) = \psi(\overline{u}_i - \overline{u}_0)$$

which tacitly assumes that the ratio exists. Monotonicity properties 1 and 2 guarantee that $\psi \in [0,1]$. Thus, equation (4.91) is rewritten in the nonlinear form:

$$(A_0\overline{u}_0)_t + \sum_{i=1}^{d(c_j)} \frac{1}{2}\left(f(u_0^+, \vec{\mathbf{n}}_i) + f(u_i^-, \vec{\mathbf{n}}_i)\right)$$
$$- \sum_{i=1}^{d(c_j)} \frac{1}{2}\psi_i|a(\tilde{u}_i, \vec{\mathbf{n}}_i)|(\overline{u}_i - \overline{u}_0) = 0$$

From symmetry and the eigenvalue circle theorem we have that

$$\mathbf{u}^T \mathcal{L}_d \mathbf{u} \le 0 \tag{4.92}$$

It remains to be shown that the advection operator $\mathcal{L}_a$ is either isoenergetic (in the linear case) or decays energy in the system. Not all extrapolation formulas guarantee that this is true. A full discussion of this topic is beyond the scope of these notes and is a subject of current research. Note that in reference [45], we indicated a preference for a standard Galerkin discretization of $\mathcal{L}_a$. Since this operator is isoenergetic, when combined with the diffusion operator described above, the entire scheme is provably stable in an energy norm.

4.8 Maximum Principles and the Delaunay Triangulation

The edge formulas presented earlier not only provide an efficient procedure for calculating quantities such as the gradient and divergence, but also provide certain theoretical results which are difficult to ascertain otherwise. For example, Ciarlet and Raviart [46] consider Galerkin schemes for solving elliptic equations using linear finite-elements. They derive sufficient conditions for the existence of a discrete maximum principle for Laplace's equation if all angles in the triangulation are less than $\pi/2 - \epsilon$ for some positive ϵ. Using the edge formulas derived in section 4.3, sufficient and necessary conditions can be derived for a discrete maximum principle which are quite different from the Ciarlet result. A brief outline of the proof is given below.

Example: Derive conditions for a discrete maximum principle using a Galerkin approximation with linear elements.

Using a reduced form of (4.43), the canonical edge formula for the discrete Laplacian operator is given by

$$\int_{\Omega_0} \phi \Delta u^h da = \mathcal{L}(u^h)_{v_0} =$$
$$\sum_{i \in \mathcal{I}_0} \frac{1}{2}[\cot\text{an}(\alpha_{L_i}) + \cot\text{an}(\alpha_{R_i})]\,(u_i - u_0) \tag{4.93}$$

where the angles α_{L_i} and α_{R_i} are depicted below.

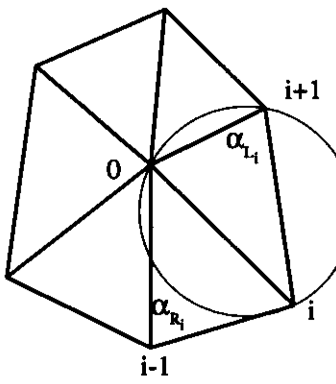


Figure 4.14 Circumcircle test for adjacent triangles.

It is well known that a discrete maximum principle exists for arbitrary point distributions and boundary data if and only if the discrete operator is a nonnegative operator, i.e., if

$$\mathcal{L}(u^h)_{v_0} = \sum_{i\in\mathcal{I}_0} w_i u_i^h \tag{4.94}$$

and

$$w_0 < 0,\ w_i \geq 0, i > 0,\ w_0 + \sum_{i\in\mathcal{I}_0} w_i = 0 \tag{4.95}$$

for any interior vertex v_0. For schemes of the form

$$\mathcal{L}(u^h)_{v_0} = \sum_{i\in\mathcal{I}_0} W_i(u_i^h - u_0^h) \tag{4.96}$$

nonnegativity requires that $W_i \geq 0$ for all $i \in \mathcal{I}_0$. This guarantees a maximum principle. Equating equation (4.96) to zero, we obtain

$$u_0^h = \frac{\sum_{i\in\mathcal{I}_0} W_i u_i^h}{\sum_{i\in\mathcal{I}_0} W_i} \tag{4.97}$$

and therefore

$$\min(u_1^h, u_2^h, ..., u_{d_0}^h) \leq u_0^h \leq \max(u_1^h, u_2^h, ..., u_{d_0}^h)$$

A natural question to be addressed concerns the existence and uniqueness of triangulations of an arbitrary point set such that (4.93) guarantees a discrete maximum principle. In two dimensions a unique triangulation always exists. The main result is summarized in the following theorem:

The discrete Laplacian operator (4.93) exhibits a discrete maximum principle for arbitrary point sets in two space dimensions iff the triangulation of these points is a Delaunay triangulation.

The key elements of the proof are given below: Rearrangement of the weights appearing in (4.93) yields

$$\begin{aligned} W_i =& \frac{1}{2}\left[\operatorname{cotan}(\alpha_{L_i}) + \operatorname{cotan}(\alpha_{R_i})\right] \\ =& \frac{1}{2}\left[\frac{\cos(\alpha_{L_i})}{\sin(\alpha_{L_i})} + \frac{\cos(\alpha_{R_i})}{\sin(\alpha_{R_i})}\right] \\ =& \frac{1}{2}\left[\frac{\sin(\alpha_{L_i}+\alpha_{R_i})}{\sin(\alpha_{L_i})\sin(\alpha_{R_i})}\right] \end{aligned} \tag{4.98}$$

Since $\alpha_{L_i} < \pi$, $\alpha_{R_i} < \pi$, the denominator is always positive and nonnegativity requires that $\alpha_{L_i} + \alpha_{R_i} \leq \pi$. Some trigonometry reveals that for the configuration of figure 4.14 with circumcircle passing through $\{v_0, v_i, v_{i+1}\}$ the sum $\alpha_{R_i} + \alpha_{L_i}$ depends on the location of v_{i-1} with respect to the circumcircle in the following way:

$$\begin{aligned} &\alpha_{R_i} + \alpha_{L_i} < \pi, && v_{i-1} \text{ exterior} \\ &\alpha_{R_i} + \alpha_{L_i} > \pi, && v_{i-1} \text{ interior} \\ &\alpha_{R_i} + \alpha_{L_i} = \pi, && v_{i-1} \text{ cocircular} \end{aligned} \tag{4.99}$$

Also note that we could have considered the circumcircle passing through $\{v_0, v_i, v_{i-1}\}$ with similar results for v_{i+1}. The condition of nonnegativity implies a circumcircle condition for all pairs of adjacent triangles whereby the circumcircle passing through either triangle cannot contain the fourth point. This is precisely the unique characterization of the Delaunay triangulation which would complete the proof.

Keep in mind that from equation (4.98) we have that $\operatorname{cotan}(\alpha) \geq 0$ if $\alpha \leq \pi/2$. Therefore a sufficient but not necessary condition for nonnegativity (and a Delaunay triangulation) is that all angles of the mesh be less than or equal to $\pi/2$. This is a standard result in finite element theory and applies in two or more space dimensions. The construction of nonnegative operators has important implications with respect to the diagonal dominance of implicit schemes, the eigenvalue spectrum of the discrete operator, stability of relaxation schemes, etc.

We can ask if the result concerning Delaunay triangulation and the maximum principle extends to three space dimensions. As we will show, the answer is no. Section 4.3d gives the corresponding edge formulas for Hessian and Laplacian discretizations in 3-D. The resulting formula for the three dimensional Laplacian is

$$\int_{V_{\Gamma_0}} \phi^h \nabla^2 u^h \, dv = \sum_{i\in\mathcal{I}_0} W_i(u_i - u_0) \tag{4.100}$$

where

$$W_i = \frac{1}{6}\sum_{k=1}^{d(v_0,v_i)} |\Delta \mathbf{R}_{k+1/2}| \cot(\alpha_{k+1/2}). \tag{4.101}$$

In this formula V_{Γ_0} is the volume formed by the union of all tetrahedra that share vertex v_0. $\mathcal{I}_0$ is the set of indices of all adjacent neighbors of v_0 connected by incident edges, k a local cyclic index describing the associated vertices which form a polygon of degree $d(v_0, v_i)$ surrounding the edge $e(v_0, v_i)$, $\alpha_{k+1/2}$ is the face angle between the two faces associated with $\vec{s}_{k+1/2}$ and $\vec{s}'_{k+1/2}$ which share the edge $e(v_k, v_{k+1})$ and $|\Delta \mathbf{R}_{k+1/2}|$ is the magnitude of the edge (see figure below).

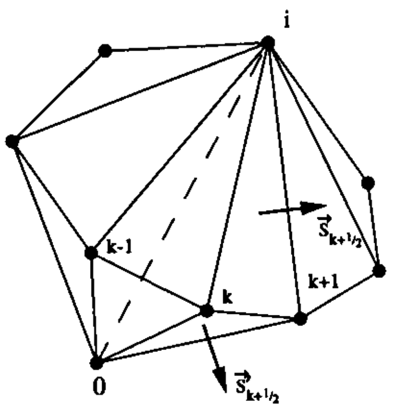


Figure 4.15 Set of tetrahedra sharing interior edge $e(v_0, v_i)$ with local cyclic index k.

A maximum principle is guaranteed if all $W_i \geq 0$. We now will procede to describe a valid Delaunay triangulation with one or more $W_i < 0$. It will suffice to consider the Delaunay triangulation of N points in which a single point v_0 lies interior to the triangulation and the remaining $N-1$ points describe vertices of boundary faces which completely cover the convex hull of the point set.

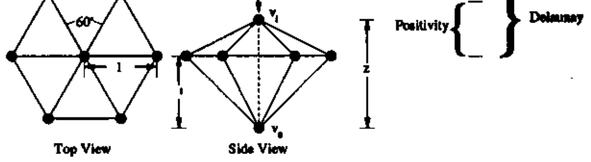


Figure 4.16 Subset of 3-D Delaunay Triangulation which does not maintain nonnegativity.

Consider a subset of the N vertices, in particular consider an interior edge incident to v_0 connecting to v_i as shown in figure 4.16 by the dashed line segment and all neighbors adjacent to v_i on the hull of the point set. In this experiment we consider the height of the interior edge, z, as a free parameter. Although it will not be proven here, the remaining $N-8$ points can be placed without conflicting with any of the conclusions obtained for looking at the subset.

It is known that a necessary and sufficient condition for the 3-D Delaunay triangulation is that the circumsphere passing through the vertices of any tetrahedron must be point free [21]; that is to say that no other point of the triangulation can lie interior to this sphere. Furthermore a property of locality exists [21] so that we need only inspect adjacent tetrahedra for the satisfaction of the circumsphere test. For the configuration of points shown in figure 4.16, convexity of the point cloud constrains $z \geq 1$ and the satisfaction of the circumsphere test requires that $z \leq 2$.

$$1 \leq z \leq 2 \quad \text{(Delaunay Triangulation)}$$

From (2.13) we find that $W_i \geq 0$ if and only if $z < 7/4$.

$$1 \leq z \leq \frac{7}{4}, \quad \text{(Nonnegativity)}$$

This indicates that for $7/4 < z \leq 2$ we have a valid Delaunay triangulation which does not satisfy a discrete maximum principle. In fact, the Delaunay triangulation of 400 points randomly distributed in the unit cube revealed that approximately 25% of the interior edge weights were of the wrong sign (negative).

Keep in mind that from (4.101) we can obtain a sufficient but not necessary condition for nonnegativity that all face angles be less than or equal to $\pi/2$.

The formulas for the prototypical viscous term $\nabla \cdot \mu \nabla u$ are only slightly more complicated than the Laplacian formulas. In 2-D we have the following weights

$$W_i = \frac{1}{2}\left[\overline{\mu}_{L_i}\cot(\alpha_{L_i}) + \overline{\mu}_{R_i}\cot(\alpha_{R_i})\right] \tag{4.102}$$

or in 3-D

$$W_i = \frac{1}{6}\sum_{k=1}^{d(v_0,v_i)} \overline{\mu}_{k+1/2}|\Delta \mathbf{R}_{k+1/2}| \cot(\alpha_{k+1/2}) \tag{4.103}$$

where $\overline{\mu}$ is the average value of μ in the specified simplex. Since μ and $\overline{\mu}$ are always assumed positive quantities, we have the following theorem:

A discrete maximum principle associated with the discretization of $\nabla \cdot \mu \nabla u$ with weights given by (4.102) and (4.103) is guaranteed iff $W_i \geq 0$ for all interior edges of the mesh. A sufficient but not necessary condition is that all angles (2-D) or faces angles (3-D) be less than or equal to $\pi/2$.

The proof follows immediately from nonnegativity of (4.102) and (4.103). The sufficient but not necessary condition is a minor extension of the result by Ciarlet [46].

5.0 Finite-Volume Solvers for the Euler Equations

In this section, we consider the extension of upwind scalar advection schemes to the Euler equations of gasdynamics. As we will see, the changes are relatively minor since most of the difficult work has already been done in designing the scalar scheme.

5.1 Euler Equations in Integral Form

The physical laws concerning the conservation of mass, momentum, and energy for an arbitrary region Ω can be written in the following integral form:

Conservation of Mass

$$\frac{\partial}{\partial t}\int_{\Omega} \rho \, da + \int_{\partial\Omega} \rho(\mathbf{V}\cdot\mathbf{n}) \, dl = 0 \tag{5.1}$$

Conservation of Momentum

$$\frac{\partial}{\partial t}\int_{\Omega} \rho\mathbf{V} \, da + \int_{\partial\Omega} \rho\mathbf{V}(\mathbf{V}\cdot\mathbf{n}) \, dl + \int_{\partial\Omega} p\mathbf{n} \, dl = 0 \tag{5.2}$$

Conservation of Energy

$$\frac{\partial}{\partial t}\int_{\Omega} E \, da + \int_{\partial\Omega} (E+p)(\mathbf{V}\cdot\mathbf{n}) \, dl = 0 \tag{5.3}$$

In these equations $\rho, \mathbf{V}, p$, and E are the density, velocity, pressure, and total energy of the fluid. The system is closed by introducing a thermodynamical equation of state for a perfect gas:

$$p = (\gamma - 1)(E - \frac{1}{2}\rho(\mathbf{V}\cdot\mathbf{V})) \tag{5.4}$$

These equations can be written in a more compact vector equation:

$$\frac{\partial}{\partial t}\int_{\Omega} \mathbf{u} \, da + \int_{\partial\Omega} \mathbf{F}(\mathbf{u})\cdot\mathbf{n} \, dl = 0 \tag{5.5}$$

with

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho\mathbf{V} \\ E \end{pmatrix}, \quad \mathbf{F}(\mathbf{u})\cdot\mathbf{n} = \begin{pmatrix} \rho(\mathbf{V}\cdot\mathbf{n}) \\ \rho\mathbf{V}(\mathbf{V}\cdot\mathbf{n}) + p\mathbf{n} \\ (E+p)(\mathbf{V}\cdot\mathbf{n}) \end{pmatrix}$$

In the next section, we show the natural extension of the scalar advection scheme to include (5.5).

5.2 Extension of Scalar Advection Schemes to Systems of Equations

The extension of the scalar advection schemes to the Euler equations requires two rather minor modifications:

(1) *Vector Flux Function.* The scalar flux function is replaced by a vector flux function. In the present work, the mean value linearization due to Roe [47] is used. The form of this vector flux function is identical to the scalar flux function (4.58), i.e.

$$\begin{aligned} \mathbf{h}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n}) =& \frac{1}{2}\left(\mathbf{f}(\mathbf{u}^+, \mathbf{n}) + \mathbf{f}(\mathbf{u}^-, \mathbf{n})\right) \\ & - \frac{1}{2}|A(\tilde{\mathbf{u}})|\left(\mathbf{u}^+ - \mathbf{u}^-\right) \end{aligned} \tag{5.6}$$

where $\mathbf{f}(\mathbf{u}, \mathbf{n}) \equiv \mathbf{F}(\mathbf{u})\cdot\mathbf{n}$, and $A \equiv d\mathbf{f}/d\mathbf{u}$ is the flux Jacobian.

(2) *Componentwise limiting.* The solution variables are reconstructed componentwise. In principle, any set of variables can be used in the reconstruction (primitive variables, entropy variables, etc.). Note that conservation of the mean can make certain variable combinations more difficult to implement than others because of the nonlinearities that may be introduced. The simplest choice is obviously the conserved variables themselves. When conservation of the mean is not important (steady-state calculations), we prefer the use of primitive variables in the reconstruction step.

The resulting scheme for the Euler equations has the same shock resolving characteristics as the scalar scheme. Figures 5.1a-b show a simple Steiner triangulation and the resulting solution obtained with a linear reconstruction scheme for transonic Euler flow ($M_\infty = .80, \alpha = 1.25°$) over a NACA 0012 airfoil section.

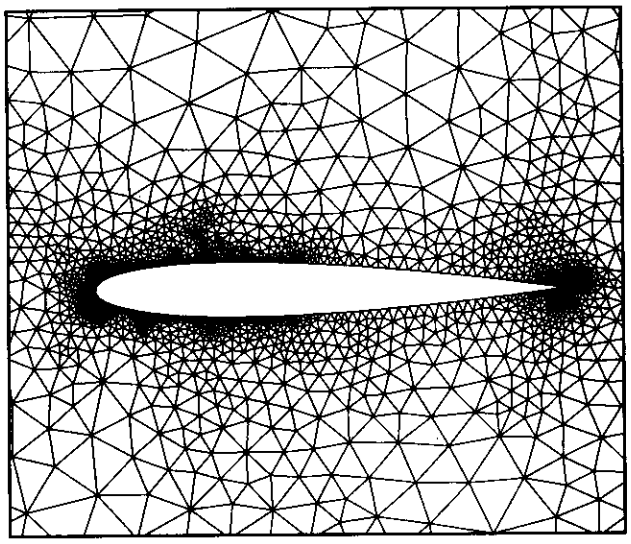


Figure 5.1a Initial triangulation of airfoil, 3155 vertices.

Even though the grid is very coarse with only 3155 vertices, the upper surface shock is captured cleanly with a profile that extends over two cells of the mesh.

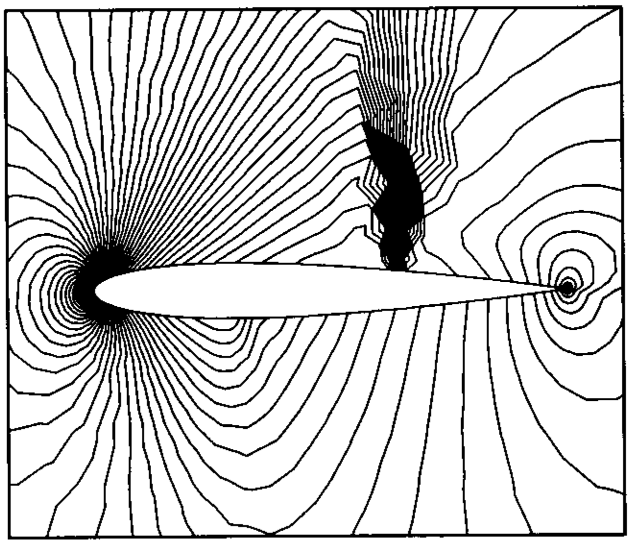


Figure 5.1b Mach number contours on initial triangulation, $M_\infty = .80, \alpha = 1.25°$.

Clearly, the power of the unstructured grid method is the ability to locally adapt the mesh to resolve flow features. Figures 5.2a-b show an adaptively refined mesh and solution for the same flow. The mesh has been locally refined based on *a posteriori* error estimates. These estimates were obtained by performing k-exact reconstruction in each control volume using linear and quadratic functions. A complete discussion of error estimation and solution adaption will be given by Professor Johnson in these notes. The paper by Warren *et al* [29] also provides some interesting insights into the area of mesh adaptation for flows containing discontinuities.

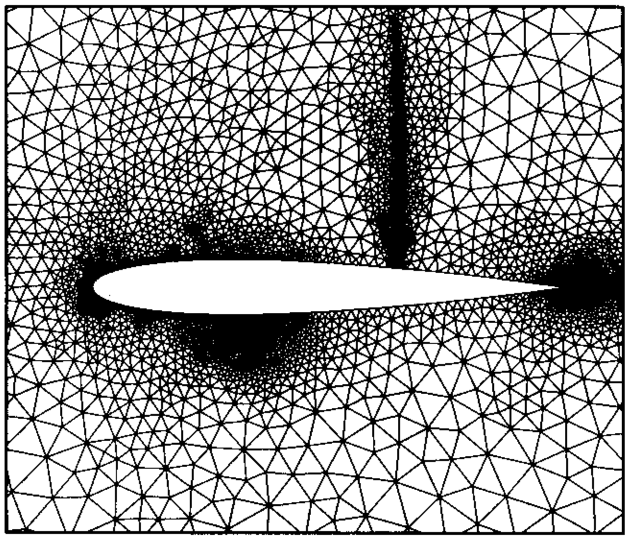


Figure 5.2a Solution adaptive triangulation of airfoil, 6917 vertices.

The flow features in figure 5.2b are clearly defined with a weak lower surface shock now visible. Figure 5.3 shows the surface pressure coefficient distribution on the airfoil. The discontinuities are crisply captured by the scheme.

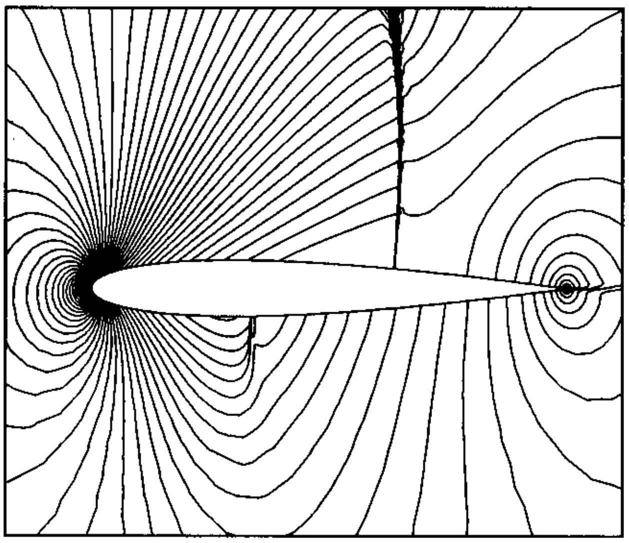


Figure 5.2b Mach number contours on adapted airfoil.

The other major advantage of unstructured grids is the ability to automatically mesh complex geometries. The next example shown in figure 5.4a-b is a Steiner triangulation and solution about a multi-component airfoil.

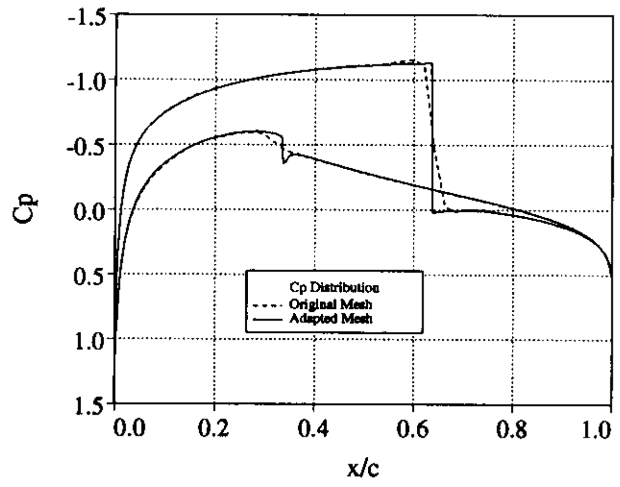


Figure 5.3 Comparison of C_p distributions on initial and adapted meshes.

Using the incremental Steiner algorithm discussed previously, the grid can constructed from curve data in about ten minutes time on a standard engineering workstation using less than a minute of actual CPU time.

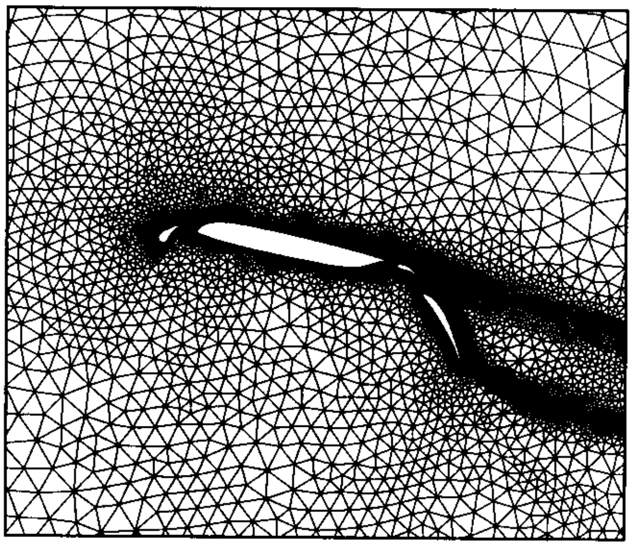


Figure 5.4a Steiner Grid about multi-component airfoil.

The flow calculation shown in figure 5.4b was performed on a CRAY supercomputer taking just a few minutes of CPU time using a linear reconstruction scheme with implicit time advancement. Details of the implicit scheme are given in the next section.

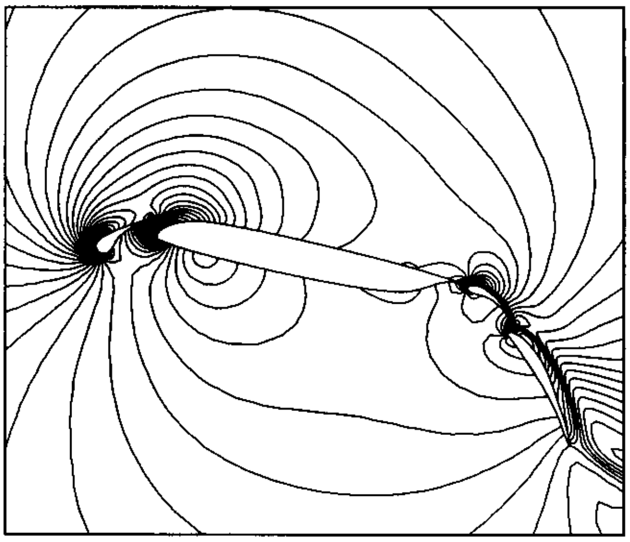


Figure 5.4b Mach number contours about multi-component airfoil, $M_\infty = .2, \alpha = 0°$.

We previously mentioned the importance of using accurate flux quadrature formulas. In fact, for k-exact reconstruction, we suggest N point Gauss quadratures with $N \geq (k+1)/2$. In Figs. 5.5a-b this importance is illustrated by plotting density contours for a numerical calculation of the Ringleb flow (previously described) using quadratic reconstruction $k = 2$. Our formula suggests that two point quadratures should be used in this case.

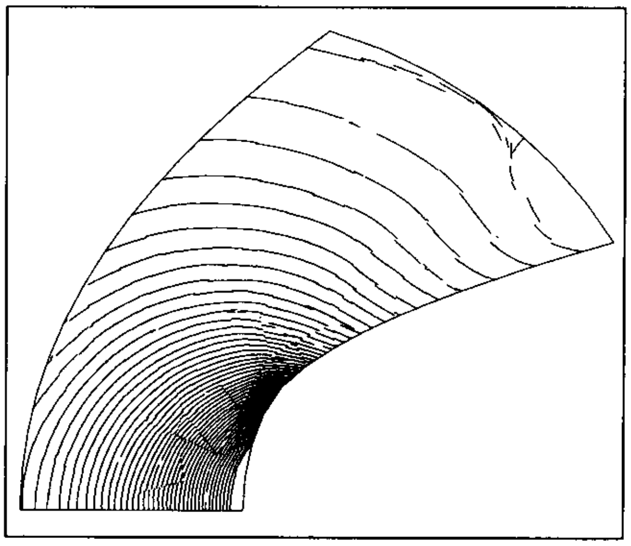


Figure 5.5a Ringleb flow density contours using quadratic reconstruction and one-point Gauss quadrature ($k = 2, N = 1$).

Figure 5.5a shows contours for a calculation using one-point Gauss quadrature and Fig. 5.5b shows contours for a calculation using two-point quadratures. The improvement in Fig. 5.5b is dramatic. Increasing the number of quadrature points to three leaves the solution unchanged.

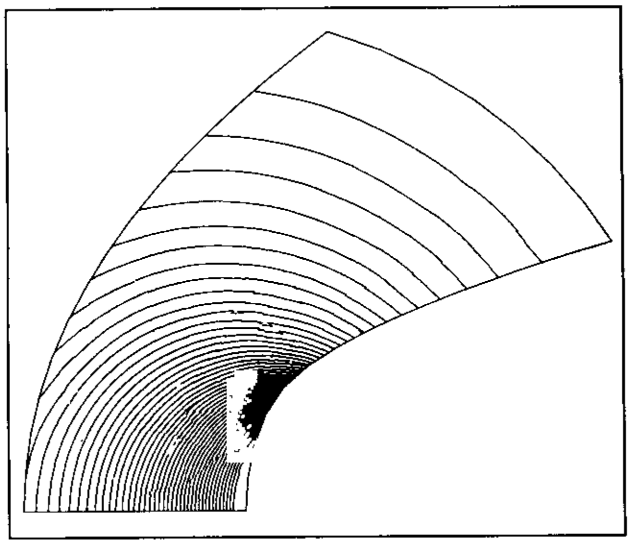


Figure 5.5b Ringleb flow density contours using quadratic reconstruction and two-point Gauss quadrature ($k = 2, N = 2$).

The algorithms outlined in section 4 have been extended to include the Euler equations in three dimensions. In reference [43], we showed the natural extension of the edge data structure in the development of an Euler equation solver on tetrahedral meshes. One of the calculations presented in this paper simulated Euler flow about the ONERA M6 wing. The tetrahedral mesh used for the calculations was a subdivided 151x17x33 hexahedral C-type mesh with spherical wing tip cap. The resulting tetrahedral mesh contained 496,350 tetrahedra, 105,177 vertices, 11,690 boundary vertices, and 23,376 boundary faces. Figure 5.6 shows a closeup of the surface mesh near the outboard tip.

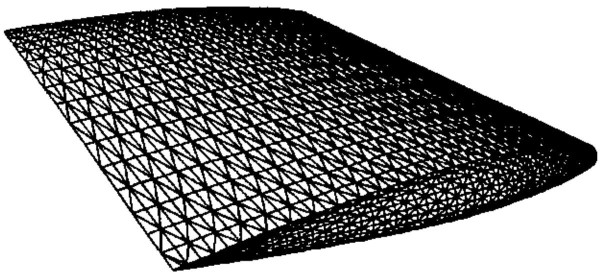


Figure 5.6 Closeup of M6 Wing Surface Mesh Near Tip.

Transonic calculations, $M_\infty = .84, \alpha = 3.06°$, were performed on the CRAY Y-MP computer using the upwind code with both the Green-Gauss and L_2 gradient reconstruction. Figure 5.7 shows surface pressure contours on the wing surface and C_p profiles at several span stations.

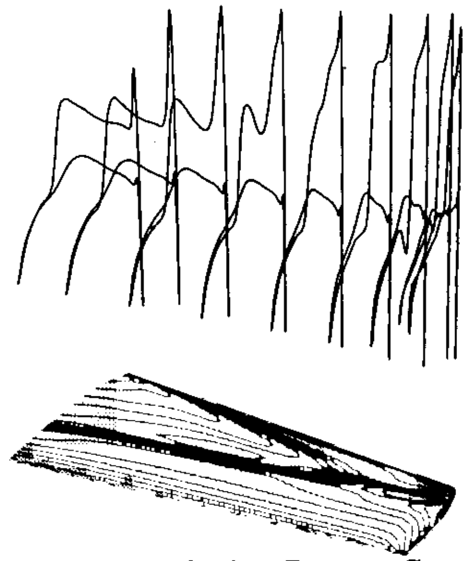


Figure 5.7 M6 Wing Surface Pressure Contours and Spanwise C_p Profiles (M_∞= .84, α=3.06°).

Pressure contours clearly show the lambda type shock pattern on the wing surface. Figures 5.8a-c compare pressure coefficient distributions at three span stations on the wing measured in the experiment, y/b=.44,.65.,.95.

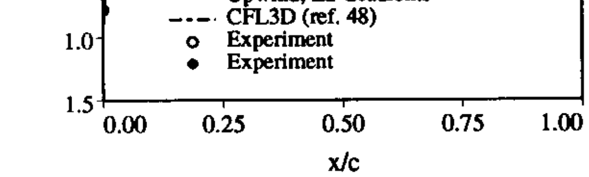


Figure 5.8a M6 Wing Spanwise Pressure Distribution, $y/b = .44$.

Each graph compares the upwind code with Green-Gauss and L_2 gradient calculation with the CFL3D results appearing in [48] and the experimental data [49]. Numerical results on the tetrahedral mesh compare very favorably with the CFL3D structured mesh code. The results for the outboard station appear better for the present code than the CFL3D results. This is largely due to the difference in grid topology and subsequent improved resolution in that area.

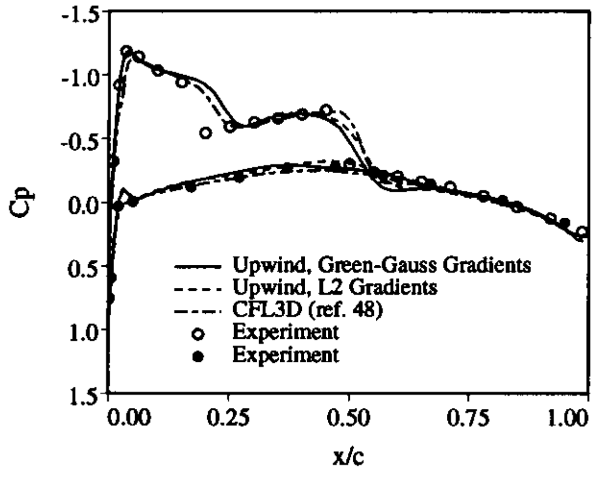


Figure 5.8b M6 Wing Spanwise Pressure Distribution $y/b = .65$.

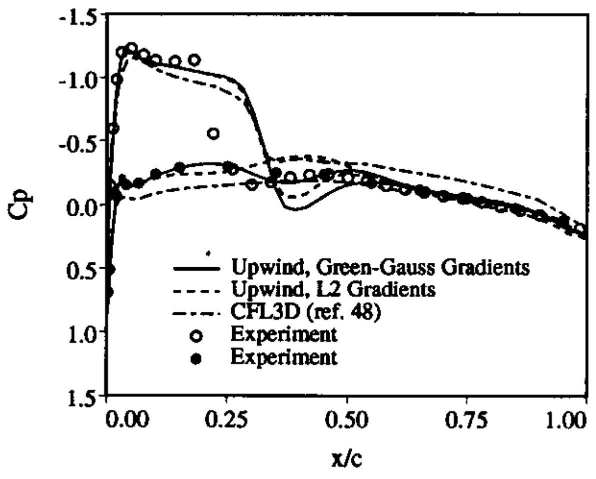


Figure 5.8c M6 Wing Spanwise Pressure Distribution $y/b = .95$.

5.3 Implicit Linearizations

In this section, we consider the task of linearizing the discrete spatial operator for purposes of backward Euler time integration. Defining the solution vector $\mathbf{u} = [\overline{u}_1, \overline{u}_2, \overline{u}_3, ..., \overline{u}_N]^T$, the basic scheme is written as

$$D\mathbf{u}_t = \mathbf{R}(\mathbf{u}) \tag{5.7}$$

where D is a positive diagonal matrix. Performing a backward Euler time integration, equation (5.7) is rewritten as

$$D(\mathbf{u}^{n+1} - \mathbf{u}^n) = \Delta t\ \mathbf{R}(\mathbf{u}^{n+1}). \tag{5.8}$$

where n denotes the iteration (time step) level. Linearizing the right-hand-side (RHS) of (5.8) in time produces the following form:

$$D(\mathbf{u}^{n+1} - \mathbf{u}^n) = \Delta t \left(\mathbf{R}(\mathbf{u}^n) + \frac{d\mathbf{R}^n}{d\mathbf{u}}(\mathbf{u}^{n+1} - \mathbf{u}^n) \right)$$

By rearranging terms, we can arrive at the so called "delta" form of the backward Euler scheme

$$\left[D - \Delta t\ \frac{d\mathbf{R}^n}{du} \right] (\mathbf{u}^{n+1} - \mathbf{u}^n) = +\Delta t\ \mathbf{R}(\mathbf{u}^n) \tag{5.9}$$

Note that for large time steps, the scheme becomes equivalent to Newton's method for finding roots of a nonlinear system of equations. Newton's method is known to be quadratically convergent for isolated roots. Each iteration of the scheme requires the solution of an algebraic system of linear equation. In practice, we use either sparse direct methods as discussed in ref. [45] or preconditioned minimum residual methods. Both of these topics will be addressed by Professor Hughes and other lecturers. The success or failure of these methods hinges heavily on the accuracy of the time linearization. For the schemes discussed in sections 4 and 5, the most difficult task is the linearization of the numerical flux vector with respect to the two solution states. For example, given the flux vector

$$\begin{aligned} \mathbf{h}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n}) =& \frac{1}{2} \left(\mathbf{f}(\mathbf{u}^+, \mathbf{n}) + \mathbf{f}(\mathbf{u}^-, \mathbf{n}) \right) \\ & - \frac{1}{2} |A(\tilde{\mathbf{u}})| \left(\mathbf{u}^+ - \mathbf{u}^- \right) \end{aligned} \tag{5.10}$$

we require the Jacobian terms $\frac{d\mathbf{h}}{d\mathbf{u}^+}$ and $\frac{d\mathbf{h}}{d\mathbf{u}^-}$. In reference [50], we derived the exact Jacobian linearization of Roe's flux function. In this same paper, approximate linearizations of (5.10) were investigated. The linearization of (5.10) is straightforward, except for terms which arise from differentiation of $|A(\tilde{\mathbf{u}})|$. A simple approximation is to neglect these terms in the linearization process. This produces the following approximate linearizations:

$$\frac{d\mathbf{h}}{d\mathbf{u}^+} = \frac{1}{2}(A(\mathbf{u}^+) - |A(\tilde{\mathbf{u}})|) \quad \text{(Approx 1)}$$

$$\frac{d\mathbf{h}}{d\mathbf{u}^-} = \frac{1}{2}(A(\mathbf{u}^-) + |A(\tilde{\mathbf{u}})|) \quad \text{(Approx 1)}$$

It is not difficult that to prove that the error associated with this approximation is $O(\|\mathbf{u}^+ - \mathbf{u}^-\|)$ which makes the linearization attractive for the implicit calculation of smooth flows. Near discontinuities, this term becomes $O(1)$ which can slow the convergence of the scheme considerably. One important attribute of this approximation is that it retains *time conservation* of the scheme. This amounts to a telescoping property of fluxes in time. For time accurate problems involving moving discontinuities, this property is essential to obtain correct shock speeds. Another approximate form considered in [50] uses the following simple approximation

$$\frac{d\mathbf{h}}{d\mathbf{u}^+} = A(\tilde{\mathbf{u}})^- \quad \text{(Approx 2)}$$

$$\frac{d\mathbf{h}}{d\mathbf{u}^-} = A(\tilde{\mathbf{u}})^+ \quad \text{(Approx 2)}$$

This linearization also differs from the exact linearization by terms $O(\|\mathbf{u}^+ - \mathbf{u}^-\|)$. One important feature of this linearization is that it produces a LHS operator for the first order upwind scheme which is (block) diagonally dominant. For those solution methods or preconditioning methods based on classical relaxation schemes, this property establishes convergence of the relaxation method. Scalar equation analysis also indicates that when this linearization is used with backward Euler time integration and first order upwind space discretization, the resulting scheme is monotone for all time step size. Unfortunately, this linearization violates time conservation and should not be used for time accurate calculations.

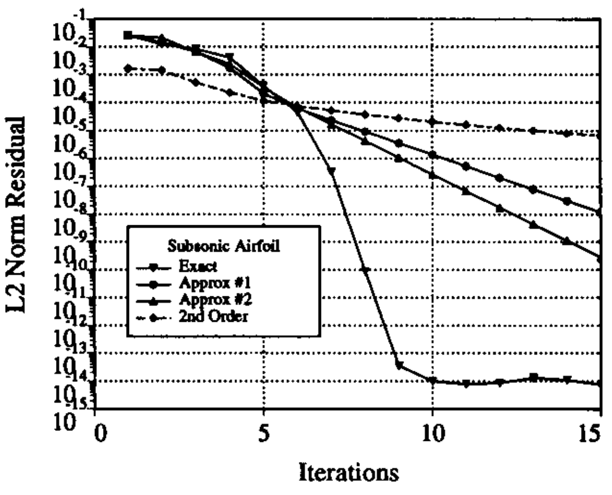


Figure 5.9 Convergence histories for exact and approximate linearizations. Solid lines show convergence histories for calculations carried out using first order upwind RHS in(5.9). Dashed line depicts scheme run with exact linearization of first order scheme on the LHS and second order RHS discretization.

Using the edge data structure, the assembly of the LHS matrix in (5.9) for the first order scheme is very straightforward. The flux associated with each edge $e(v_i, v_j)$ of the mesh is linearized with respect its two arguments, $\mathbf{u}_i$ and $\mathbf{u}_j$. This means that the linearization contributes to the formation of the block matrix elements in the i-th row j-th column, i-th row i-th column, j-th row i-th column, and j-th row j-th column positions. This leads to a highly vectorizable (using gather-scatter hardware) algorithm for matrix assembly (and matrix multiplies). For the higher order reconstruction schemes, the usual strategy is to only construct LHS matrix terms associated with the first order upwind scheme while using a higher order RHS operator. The mismatch of operators destroys any hope of quadratic convergence for large time steps. Figure 5.9 graphs the convergence history for a typical calculation using the linearizations discussed above. The flow problem being solved is subsonic flow over a single airfoil. In this case, the flow is smooth and all linearizations should be applicable. In this figure, we see that when the RHS and LHS operators both correspond to the first order upwind scheme and the exact Jacobian linearization is used, quadratic convergence is achieved. The schemes using approximate linearizations do not approach quadratic convergence but are very effective in reducing the initial residual. In reality, most computations are terminated after reducing the residual about four orders of magnitude. For the present example, this would amount to 7 steps using the exact linearization or 8-9 steps using the approximate forms. Using a higher order accurate RHS slows the convergence even further. Nevertheless, a four order reduction in residual is achieved after 30-40 steps.

6.0 Numerical Solution of the Navier-Stokes Equations with Turbulence

6.1 Turbulence Modeling for Unstructured Grids

Simulating the effects of turbulence on unstructured meshes via the compressible Reynolds-averaged Navier-Stokes equations and turbulence modeling is a relatively unexplored topic. In early work by Rostand [51], an algebraic turbulence model was incorporated into an unstructured mesh flow solver. This basic methodology was later refined by Mavriplis [52] for the flow about multi-element airfoil configurations. Both of these implementations utilize locally structured meshes to produce one-dimensional-like boundary-layer profiles from which algebraic models can determine an eddy viscosity coefficient for use in the Reynolds-averaged Navier-Stokes equations. The use of local structured meshes limits the general applicability of the method.

The next level of turbulence modeling involves the solution of one or more auxiliary differential equations. Ideally these differential equations would only require initial data and boundary conditions in the same fashion as the Reynolds-averaged mean equations. The use of turbulence models based on differential equations greatly increases the class of geometries that can be treated "automatically." Unfortunately this does not make the issue of turbulence modeling a "solved" problem since most turbulence models do not perform well across the broad range of flow regimes usually generated by complex geometries. Also keep in mind that most turbulence models for wall-bounded flow require knowledge of distance to the wall for use in "damping functions" which simulate the damping effect of solid walls on turbulence. The distance required in these models is measured in "wall units" which means that physical distance from the wall y is scaled by the local wall shear, density, and viscosity.

$$y^+ = \sqrt{\frac{\tau_{wall}}{\rho_{wall}}}\frac{y}{\nu} \tag{6.1}$$

Scaling by wall quantities is yet another complication but does not create serious implementation difficulties for unstructured meshes as we will demonstrate shortly.

6.2 A One-Equation Turbulence Transport Model

In a recent report with Baldwin [53], we proposed and tested (on structured meshes) a single equation turbulence transport model. In this report, the model was tested on various subsonic and transonic flow problems: flat plates, airfoils, wakes, etc. The model consists of a single scalar advection-diffusion equation with source term for a field variable which is the product of turbulence Reynolds number and kinematic viscosity, $\nu\widetilde{R}_T$. This variable is proportional to the eddy viscosity except very near a solid wall.

$$\begin{aligned}\frac{D(\nu\widetilde{R}_T)}{Dt} &= (c_{\epsilon_2}f_2 - c_{\epsilon_1})\sqrt{\nu\widetilde{R}_T P}\\ &+(\nu + \frac{\nu_t}{\sigma_R})\nabla^2(\nu\widetilde{R}_T) - \frac{1}{\sigma_\epsilon}(\nabla\nu_t)\cdot\nabla(\nu\widetilde{R}_T)\end{aligned} \tag{6.2}$$

In this equation, P is the production of turbulent kinetic energy and is related to the mean flow velocity rate-of-strain and the kinematic eddy viscosity ν_t. In equation (6.2), the following functions are required:

$$\begin{aligned}
\frac{1}{\sigma_\epsilon} &= (c_{\epsilon_2} - c_{\epsilon_1})\sqrt{c_\mu}/\kappa^2\\
\sigma_R &= \sigma_\epsilon\\
\nu_t &= c_\mu(\nu\widetilde{R}_T)D_1 D_2\\
\mu_t &= \rho\nu_t\\
D_1 &= 1-\exp(-y^+/A^+)\\
D_2 &= 1-\exp(-y^+/A_2^+)\\
P &= \nu_t\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)\frac{\partial U_i}{\partial x_j} - \frac{2}{3}\nu_t\left(\frac{\partial U_k}{\partial x_k}\right)^2\\
f_2(y^+) &= \frac{c_{\epsilon_1}}{c_{\epsilon_2}} + (1-\frac{c_{\epsilon_1}}{c_{\epsilon_2}})(\frac{1}{\kappa y^+} + D_1 D_2)\Bigg(\sqrt{D_1 D_2}\\
&+\frac{y^+}{\sqrt{D_1 D_2}}\Big(\frac{1}{A^+}\exp(-y^+/A^+)\ D_2\\
&+\frac{1}{A_2^+}\exp(-y^+/A_2^+)\ D_1\Big)\Bigg)
\end{aligned}$$

The following constants have been recommended in [53]:

$$\begin{aligned}\kappa &= 0.41, \quad c_{\epsilon_1} = 1.2, \quad c_{\epsilon_2} = 2.0\\ c_\mu &= 0.09, \quad A^+ = 26, \quad A_2^+ = 10\end{aligned}$$

We also recommend the following boundary conditions for (6.2):

1. **Solid Walls**: Specify $\widetilde{R}_T = 0$.
2. **Inflow** ($\mathbf{V}\cdot\mathbf{n} < 0$): Specify $\widetilde{R}_T = (\widetilde{R}_T)_\infty < 1$.
3. **Outflow** ($\mathbf{V}\cdot\mathbf{n} > 0$): Extrapolate $\widetilde{R}_T$ from interior values.

Equation (6.2) depends on distance to solid walls in two ways. First, the damping function f_2 appearing in equation (6.2) depends directly on distance to the wall (in wall units). Secondly, ν_t depends on $\nu\widetilde{R}_t$ and damping functions which require distance to the wall.

$$\nu_t = c_\mu D_1(y^+)D_2(y^+)\nu\widetilde{R}_t$$

It is important to realize that the damping functions f_2, D_1, and D_2 deviate from unity only when very near a solid wall. For a typical turbulent boundary-layer (see figure 6.1) accurate distance to the wall is only required for mesh points which fall below the logarithmic region of the boundary-layer.

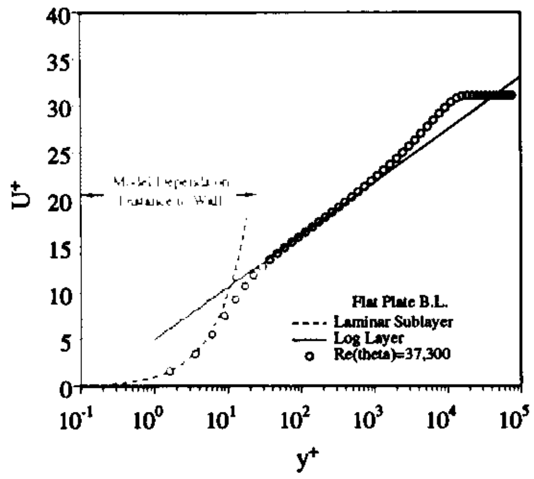


Figure 6.1 Typical flat plate boundary-layer from ref. [53] showing dependence of turbulence model on distance to wall.

The relative insensitivity of distance to the wall means that accurate estimation of distance to the wall is only required for a small number of points that are extremely close to a boundary surface. The remaining points can be assigned any approximate value of physical distance since the damping functions are essentially at their asymptotic values. A general procedure for calculation of distance to the wall in wall units is to precompute and store, for each vertex of the mesh, the minimum distance from the vertex to the closest solid wall (examples are shown later in figures 6.2b and 6.3b). This strategy can only fail if two bodies are in such close proximity that the near wall damping functions never reach their asymptotic values. Realistically speaking, the validity of most turbulence models would be in serious question in this situation anyway. In general, the minimum distance from vertex to boundary edge does not occur at the end points of a boundary edge but rather interior to a boundary edge. For each vertex, information concerning which boundary edge achieves the minimum distance and the weight factor for linear interpolation along the boundary edge must be stored. Data can then be interpolated to the point of minimum distance whenever needed. In the course of solving (6.2), distance to a solid wall in wall units is calculated by retrieving physical distance to the wall and the local wall quantities needed for (6.1) as interpolated along the closest boundary edge.

The numerical calculations presented in this section represent a successful application of the ideas discussed in previous sections. Figures 6.2a and 6.3a show examples of Min-Max triangulations about single and multi-element airfoils. The first geometry consists of a single RAE 2822 airfoil. Navier-Stokes flow is computed about this geometry assuming turbulent flow with the following free-stream conditions: $M_\infty = .725, \alpha = 2.31°, Re = 6.5$ million. Wind tunnel experiments for the RAE 2822 geometry at these test conditions have been performed by Cook, McDonald, and Firmin [54]. The RAE 2822 airfoil mesh shown in figure 6.2a contains approximately 14000 vertices and 41000 edges. The second geometry consists of a two element airfoil configuration with wind tunnel walls. The inflow conditions assume turbulent flow with $M_\infty = .09$ and $Re = 1.8$ million. Details of the geometry and wind tunnel test results can be found in the report by Adair and Horne [55]. The two element mesh shown in figure 6.3a contains approximately 18000 vertices and 55000 edges.

Both meshes were constructed in two steps. The first step was to generate a Delaunay triangulation of the point cloud. As mentioned earlier, the method of Delaunay triangulation can generate poor triangulations for highly stretched point distributions. Both meshes suffered from nearly collapsed triangles with two small interior angles. As a second step, a Min-Max triangulation was constructed by edge swapping the Delaunay triangulation. Edge swapping repaired both triangulations. Both airfoil geometries were calculated assuming turbulent flow using the one-equation turbulence model (6.2). Level sets of the generalized distance function used in the turbulence model are shown in figures 6.2b and 6.3b.

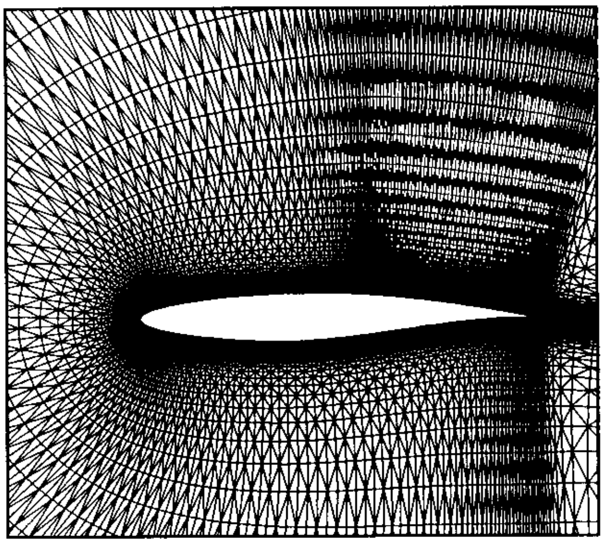


Figure 6.2a Mesh near RAE 2822 airfoil.

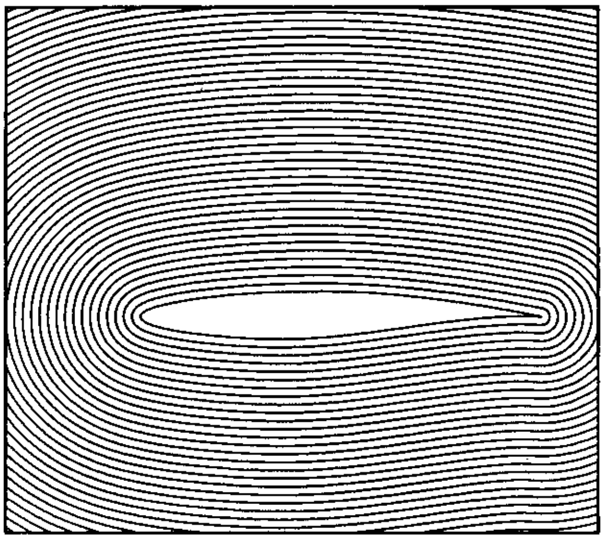


Figure 6.2b Contours of distance function for turbulence model.

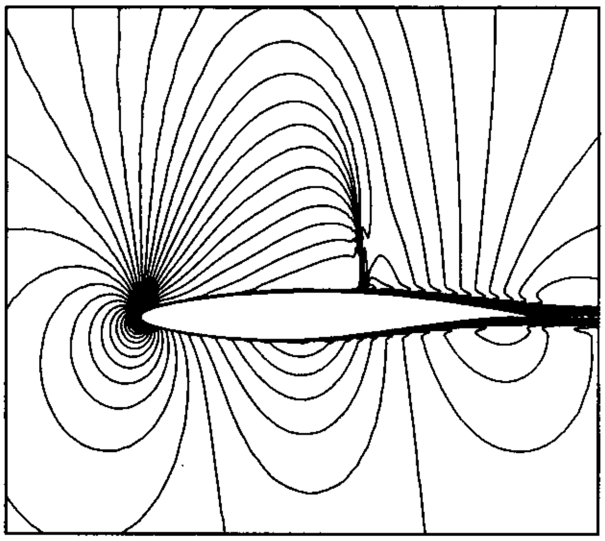


Figure 6.2c Closeup of Mach number contours near airfoil.

Figures 6.2c-d plot Mach number contours and pressure coefficient distributions for the RAE 2822 airfoil. The pressure coefficient distribution compares favorably with the experiment of Cook, McDonald, and Firmin [29]. Leading edge trip strips were used on the experimental model but not simulated in the computations. This may explain the minor differences in the leading edge pressure distribution.

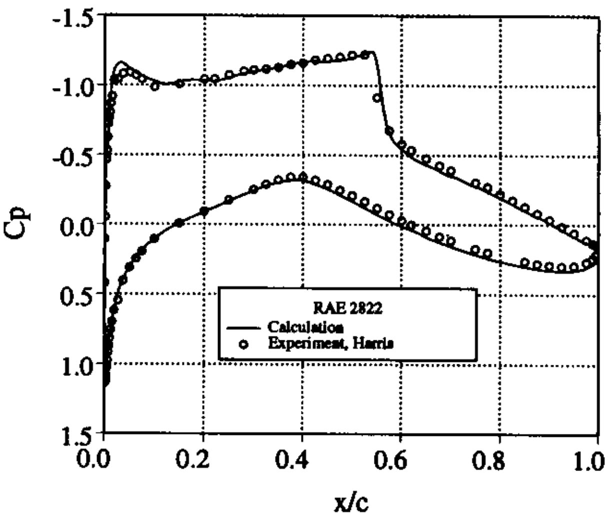


Figure 6.2d Pressure coefficient distribution on airfoil.

Navier-Stokes computations for the two element airfoil configuration as shown in figures 6.3c-d. The effects of wind tunnel walls have been modeled in the computation by assuming an inviscid wall boundary condition. Mach number contours are shown in figure 6.3c. Observe that the contours appear very smooth, even in regions where the mesh becomes very irregular. This is due to the insistance that linear functions be accurately treated in the flow solver reguardless of mesh irregularities.

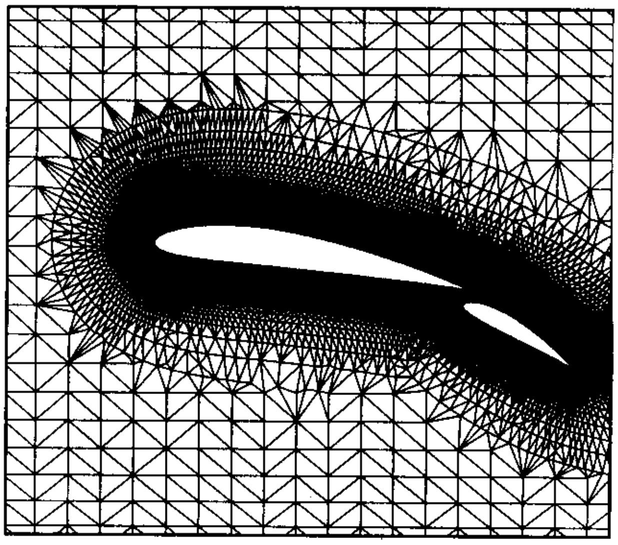


Figure 6.3a Mesh near Multi-element airfoil.

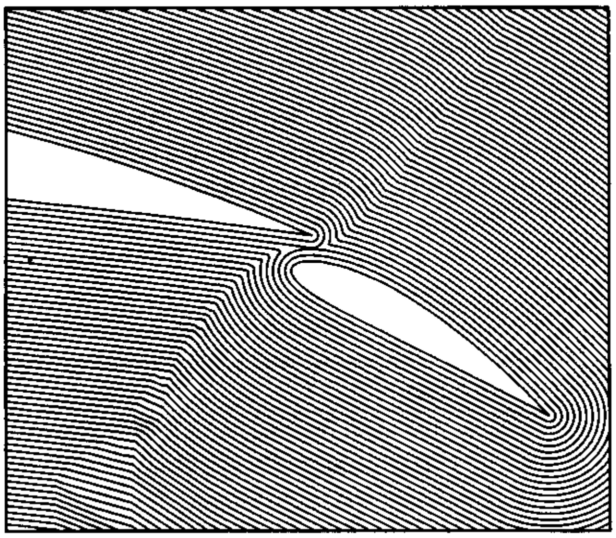


Figure 6.3b Contours of distance function for turbulence model.

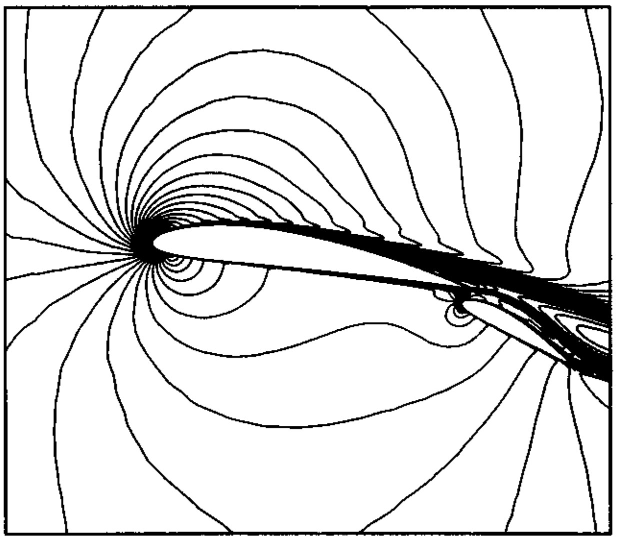


Figure 6.3c Closeup of Mach number contours near airfoil.

Pressure coefficient distribution on the main airfoil and flap are graphed in figure 6.3d. Comparison of calculation and experiment on the main element is very good. The suction peak values of pressure coefficient on the flap element are slightly below the experiment. The experimentors also note a small separation bubble at the trailing edge of the flap which was not found in the computations.

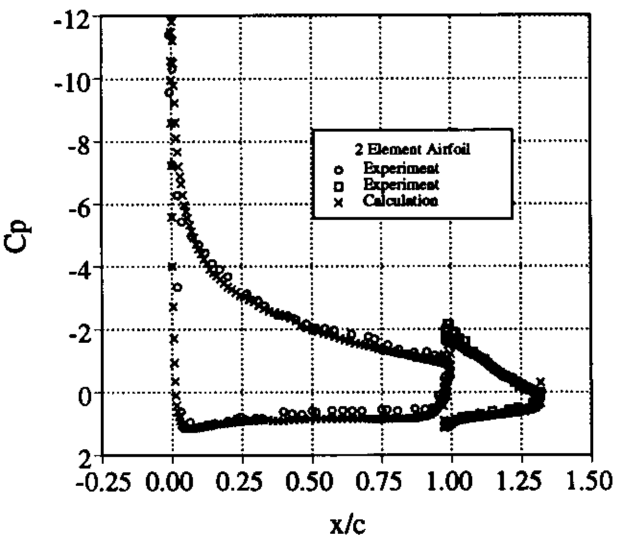


Figure 6.3d Pressure coefficient distribution on airfoil.

References

1. Guibas, L.J., and Stolfi, J.,"Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams",ACM Trans. Graph., Vol. 4, 1985, pp. 74-123.
2. Dobkin, D.P., and Laszlo, M.J.,"Primitives for the Manipulation of Three-dimensional Subdivisions", Algorithmica, Vol. 4, 1989, pp. 3-32.
3. Brisson, E.,"Representing Geometric Structures in *d* Dimensions: Topology and Order", In Proceedings of the 5th ACM Symposium on Computational Geometry., 1989, pp. 218-227.
4. Hammond, S., and Barth, T.J., "An Efficient Massively Parallel Euler Solver for Unstructured Grids", AIAA paper 91-0441, Reno, 1991.
5. Chrobak, M. and Eppstein, D.,"Planar Orientations with Low Out-Degree and Compaction of Adjacency Matrices", Theo. Comp. Sci., Vol. 86, No. 2, 1991, pp.243-266.
6. Rosen, R., "Matrix Band Width Minimization", Proc. ACM Nat. Conf., 1968, pp. 585-595.
7. Cuthill, E., and McKee, J.,"Reducing the Band Width of Sparse Symmetric Matrices", Proc. ACM Nat. Conf., 1969, pp. 157-172.
8. George, J.A,"Computer Implementation of the Finite Element Method", Techical Report No. STAN-CS-71-208, Computer Science Dept., Stanford University, 1971.
9. Venktakrishnan, V., Simon, H.D., Barth, T.J., "A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids", NASA

Ames R.C., Tech. Report RNR-91-024, 1991.

10. Simon, H.D., "Partitioning of Unstructured Problems for Parallel Processing," NASA Ames R.C., Tech. Report RNR-91-008, 1991.
11. Lawson, C. L., "Software for C^1 Surface Interpolation", Mathematical Software III, (Ed., John R. Rice), Academic Press, New York, 1977.
12. Rajan, V. T., "Optimality of the Delaunay Triangulation in $\mathbf{R}^d$", Proceedings of the 7th ACM Symposium on Computational Geometry, 1991, pp. 357-372.
13. Rippa, S., "Minimal Roughness Property of the Delaunay Triangulation", CAGD, Vol. 7, No. 6., 1990, pp-489-497.
14. Bowyer, A., "Computing Dirichlet Tessellations", The Computer Journal, Vol. 24, No. 2, 1981, pp. 162—166.
15. Watson, D. F, "Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes," The Computer Journal, Vol. 24, No. 2, 1981, pp. 167—171.
16. Baker, T. J. ,"Automatic Mesh Generation for Complex Three-Dimensional Regions Using a Constrained Delaunay Triangulation", Engineering with Computers, Vol. 5, 1989, pp. 161-175.
17. Green, P. J. and Sibson, R., "Computing the Dirichlet Tesselation in the Plane", The Computer Journal, Vol. 21, No. 2, 1977, pp. 168—173.
18. Tanemura, M., Ogawa, T., and Ogita, N.,"A New Algorithm for Three-Dimensional Voronoi Tesselation", J. Comput. Phys., Vol. 51, 1983, pp. 191-207.
19. Merriam, M. L.,"An Efficient Advancing Front Algorithm for Delaunay Triangulation", AIAA paper 91-0792, Reno, NV, 1991.
20. Klee, V., "On the Complexity of d-dimensional Voronoi diagrams", Archiv der Mathematik, Vol. 34, 1980.
21. Lawson, C. L., "Properties of n-dimensional Triangulations" CAGD, Vol. 3, April 1986, pp. 231–246.
22. Babuška, I., and Aziz, A. K., "On the Angle Condition in the Finite Element Method", SIAM J. Numer. Anal., Vol. 13, No. 2, 1976.
23. Edelsbrunner, H., Tan, T.S, and Waupotitsch, R.,"An $O(n^2 \log n)$ Time Algorithm for the MinMax Angle Triangulation," Proceedings of the 6th ACM Symposium on Computational Geometry, 1990, pp. 44-52.
24. Wiltberger, N. L., Personal Communication, NASA Ames Research Center, M.S. 258-1, Moffett Field, CA, 1991.
25. Gilbert, P.N., "New Results on Planar Triangulations", Tech. Rep. ACT-15, Coord. Sci. Lab., University of Illinois at Urbana, July 1979.
26. Nira, D., Levin, D., Rippa, S.,"Data Dependent Triangulations for Piecewise Linear Interpolation", J. Numer. Anal., Vol. 10, No. 1, 1990, pp. 137-154.
27. Nira, D., Levin, D., Rippa, S.,"Algorithms for the Construction of Data Dependent Triangulations", **Algorithms for Approximation, II**, Chapman, and Hall, London, 1990, pp. 192-198.
28. Holmes, G. and Snyder, D., "The Generation of Unstructured Triangular Meshes using Delaunay Triangulation," in *Numerical Grid Generation in CFD*, pp. 643-652, Pineridge Press, 1988.
29. Warren, G., Anderson, W.K., Thomas, J.L., and Krist, S.L.,"Grid Convergence for Adaptive Methods,", AIAA paper 91-1592-CP, Honolulu, Hawaii, June 24-27,1991.
30. Anderson, W.K.,"A Grid Generation and Flow Solution Method for the Euler Equations on Unstructured grids," NASA Langley Research Center, USA, unpublished manuscript, 1992.
31. Joe, B.,"Three-Dimensional Delaunay Triangulations From Local Transformations", SIAM J. Sci. Stat. Comput., Vol. 10, 1989, pp. 718-741.
32. Joe, B.,"Construction of Three-Dimensional Delaunay Triangulations From Local Transformations", CAGD, Vol. 8, 1991, pp. 123-142.
33. Gandhi, A. S., and Barth T. J.,"3-D Unstructured Grid Generation and Refinement Usinge 'Edge-Swapping' ", NASA TM in preparation, 1992.
34. Godunov, S. K., "A Finite Difference Method for the Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics", Mat. Sb., Vol. 47, 1959.
35. Van Leer, B., "Towards the Ultimate Conservative Difference Schemes V. A Second Order Sequel to Godunov's Method", J. Comp. Phys., Vol. 32, 1979.
36. Colella, P., Woodward, P., "The Piecewise Parabolic Method for Gas-Dynamical Simulations", J. Comp. Phys., Vol. 54, 1984.
37. Woodward, P., Colella, P., "The Numerical Simulation of Two-Dimensioal Fluid Flow with Strong Shocks" J. Comp. Phys., Vol. 54, 1984.
38. Harten, A. , Osher, S., "Uniformly High-

Order Accurate Non-oscillatory Schemes, I.," MRC Technical Summary Report 2823, 1985.

39. Harten, A., Engquist, B., Osher, S., Chakravarthy, "Uniformly High Order Accurate Essentially Non - Oscillatory Schemes III, ICASE report 86-22, 1986.
40. Barth, T. J., and Jespersen, D. C., "The Design and Application of Upwind Scehemes on Unstructured Meshes", AIAA-89-0366, Jan. 9-12, 1989.
41. Barth, T. J., and Frederickson, P. O., "Higher Order Solution of the Euler Equations on Unstructured Grids Using Quadratic Reconstruction", AIAA-90-0013, Jan. 8-11, 1990.
42. Chiocchia, G., " Exact Solutions to Transonic and Supersonic Flows", AGARD Advisory Report AR-211,1985.
43. Barth, T. J.,"A Three-Dimensional Upwind Euler Solver of Unstructured Meshes," AIAA Paper 91-1548, Honolulu, Hawaii, 1991.
44. Struijs, R., Vankeirsblick, and Deconinck, H., "An Adaptive Grid Polygonal Finite Volume Method for the Compressible Flow Equations," AIAA-89-1959-CP, 1989.
45. Barth, T.J.,"Numerical Aspects of Computing Viscous High Reynolds Number Flows on Unstructured Meshes", AIAA paper 91-0721, January, 1991.
46. Ciarlet, P.G., Raviart, P.-A., "Maximum Principle and Uniform Convergence for the Finite Element Method", Comp. Meth. in Appl. Mech. and Eng., Vol. 2., 1973, pp. 17-31.
47. Roe, P.L.,"Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes", J. Comput. Phys., Vol 43, 1981.
48. Thomas, J.L., van Leer, B., and Walters, R.W., "Implicit Flux-Split Schemes for the Euler Equations," AIAA paper 85-1680, 1985.
49. Schmitt, V., and Charpin, F., "Pressure Distributions on the ONERA M6-Wing at Transonic Mach Numbers," in "Experimental Data Base for Computer Program Assessment," AGARD AR-138, 1979.
50. Barth, T.J.,"Analysis of Implicit Local Linearization Techniques for Upwind and TVD Schemes," AIAA Paper 87-0595, 1987.
51. Rostand, P., "Algebraic Turbulence Models for the Computation of Two-dimensional High Speed Flows Using Unstructured Grids," ICASE Report 88-63, 1988.
52. Mavriplis, D., "Adaptive Mesh Generation for Viscous Flows Using Delaunay Triangulation," ICASE Report No. 88-47,1988.
53. Baldwin, B.S., and Barth, T.J.,"A One-Eqn Turbulence Transport Model for High Reynolds Number Wall-Bounded Flows," NASA TM-102847, August 1990.
54. Cook, P.H., McDonald M.A., Firmin, M.C.P., "AEROFOIL RAE 2822 Pressure Distributions, and Boundary Layer and Wake Measurements," AGARD Advisory Report No. 139, 1979.
55. Adair, D., and Horne, W.C., "Characteristics of Merging Shear Layers and Turbulent Wakes of a Multi-element Airfoil," NASA TM 100053, Feb. 1988.