1 Fundamental Concepts and Approaches

Joe F. Thompson
Nigel P. Weatherill

1.1 Introduction

This introductory chapter uses fluid mechanics as an example field problem for reference; the applicability of the concepts discussed is, however, not in any way limited to this area.

Fluid mechanics is described by nonlinear equations, which cannot generally be solved analytically, but which have been solved using various approximate methods including expansion and perturbation methods, sundry particle and vortex tracing methods, collocation and integral methods, and finite difference, finite volume, and finite element methods. Generally the finite difference, finite volume, and finite element discretization methods have been the most successful, but to use them it is necessary to discretize the field using a grid (mesh). (The terms *grid* and *mesh* are used interchangeably throughout with identical meaning.) The mesh can be structured or unstructured, but it must be generated under some of the various constraints described below, which can often be difficult to satisfy completely. In fact, at present it can take orders of magnitude more person-hours to construct the grid than it does to construct and analyze the physical solution on the grid. This is especially true now that solution codes of wide applicability are becoming available. Computational fluid dynamics (CFD) is a prime example, and grid generation has been cited repeatedly as a major pacing item (cf. Thompson [1996]). The same is true for other areas of computational field simulation.

The proceedings of the several international conferences on grid generation (Thompson [1982], Hauser and Taylor [1986], Sengupta, et al. [1988], Arcilla, et al. [1991], Eiseman, et al. [1994], Soni et al. [1996]) as well as those of the NASA conferences (Smith [1980], Smith [1992], Choo [1995]) provide numerous illustrations of application to CFD and some other fields.

A recent comprehensive text is Carey [1997].

1.2 Mesh Generation Considerations

The generated mesh must be sufficiently dense that the numerical approximation is an accurate one, but it cannot be so dense that the solution is impractical to obtain. Generally, the grid spacing should be smoothly and sufficiently refined to resolve changes in the gradients of the solution. If the grid is also boundary-conforming and curvilinear, the application of boundary conditions is simplified. Boundary-conforming curvilinear grids may also allow the use of various approximate equations such as boundary-layer equations. The grid should also be constructed with computational efficiency in mind. The accuracy of a numerical approximation can also be impaired, if a grid changes discontinuously or is too skewed. Various computers often require well-organized data, and memory requirements can grow to impractical limits unless the data is organized well. Finally, the choice of a grid should not lead to overly complex computer codes.

A mesh is a set of points distributed over a calculation field for a numerical solution of a set of partial differential equations (PDEs). This set may be structured, e.g., formed by the intersections of curvilinear coordinate surfaces, or unstructured, i.e., with no relation to coordinate directions. In the first case the points form quadrilateral cells in 2D, or hexahedral cells in 3D (with nonplanar sides). The unstructured mesh generally consists of triangles and tetrahedra in 2D and 3D, respectively, in its most basic form, but may be made of hexahedra or elements of any shape in general.

The structured grid can be generated algebraically by interpolation from boundaries, e.g., transfinite interpolation, or by solving a set of partial differential equations in the region. An entire subject, complete with textbook (Thompson, Warsi, and Mastin [1985], now on the Web at www.erc.msstate.edu), has developed around the generation of structured grids having the fundamental characteristic that some curvilinear coordinate surface is coincident with each boundary segment, i.e., boundary conforming. A later text is Knupp and Steinberg [1993]. Castillo [1991] provides a compilation of mathematical aspects as well. Structured grid generation is also covered in the recent text of Carey [1997]. Several earlier surveys of the field are still useful for basic understanding (Thompson, Warsi, and Mastin [1982], Thompson [1984], Thompson [1985], Eiseman [1985]).

Structured boundary-conforming meshes have been widely applied in computational fluid dynamics. Basically, the algebraic generation systems (Chapter 3) are faster, but the grids generated from partial differential equations are generally smoother. The hyperbolic generation systems (Chapter 5) are faster than the elliptic systems, but are more limited in the geometries that can be treated. The elliptic systems (Chapter 4) are the most generally applicable with complicated boundary configurations, but transfinite interpolation is also effective in a composite grid framework.

The generation of unstructured meshes can be done by tessellation of a point distribution that could be random but is more likely to have been produced by some ordered procedure. This tessellation is not unique, and involves some type of nearest-neighbor search, such as the Delaunay triangulation (Chapter 16). Other approaches are the advancing front procedure (Chapter 17) and the finite octree method (Chapter 15). The recent text of Carey [1997] covers unstructured grid generation as well as structured grid generation. An earlier text on unstructured grids is George [1991].

General configurations can conceivably be treated with either type of mesh, and hybrid combinations (Chapter 23) are also possible, using individual structured meshes near boundaries, with these subregions being connected by an unstructured mesh. Still another approach is overlaid grids (chimera) (Chapter 11), in which separate boundary-conforming structured grids are generated for each component of a complex configuration, and data is communicated between the various component grids by interpolation.

Of particular importance is the development of dynamically adaptive meshes (Chapters 33–36) coupled with the physical solution. In this mode the mesh is locally refined by the selective addition of points, and/or is moved to concentrate points, in order to resolve developing gradients in the physical solution on the mesh. Both of these approaches have seen considerable development and show much promise in particular areas.

Implementation of solution algorithms on structured, unstructured, and overlaid grids places differing requirements on the algorithms. Various conflicts arise between the grid and solution procedures in regard to requirements and ease of operation. In particular, unstructured grids require a much more complex solution data structure, but are more easily generated and adapted. Structured grids provide a more natural representation of normal derivative boundary conditions and allow more straightforward approximations based on prevailing directions, e.g., parallel or normal to a boundary or flow direction. The structure also leads to a much simpler data set construction, and allows the use of directional time splitting and flux representations. On the other hand, unstructured grids can be much more readily imagined for complicated boundary configurations.

1.3 Structured Grids

1.3.1 Composite Grids

Structured grid generation had its roots in the U.S. in the work of Winslow and Crowley at Lawrence Livermore National Lab in the late 1960s (Winslow [1967]), and in Russia from Godunov and Prokopov [1967] at about the same time. (There is also that enigmatic Biblical reference to the "four corners of the earth," once thought to proclaim a flat earth but now seen to be a prophesy of structured grid generation.) Another very fundamental component was the work of Bill Gordon at Drexel on transfinite interpolation for the automotive industry, introduced to the emerging grid generation community at the grid conference in Nashville in 1982 (Gordon and Thiel [1982]).

1.3.1.1 Terminology

The use of composite grids has been the key to the treatment of general 3D configurations with structured grids. Here in general, *composite* refers to the fact that the physical region is divided into subregions, within each of which a structured grid is generated. These subgrids may be patched together at common interfaces, may be overlaid, or may be connected by an unstructured grid. Considerable confusion has arisen in regard to terminology for composite grids, making it difficult to immediately classify papers on the subject.

Composite grids in which the subgrids share common interfaces are referred to as *block*, *patched*, *embedded*, or *zonal* grids in the literature. The use of the first two of these terms is fairly consistent with this type of grid (*patched* comes from the common interfaces, *block* from the logically rectangular structure), but the last two are sometimes also applied to overlaid grids. *Overlaid* (overset) grids are often called *chimera* grids after the composite monster of Greek mythology. Unfortunately, the common interface grids can also be said to overlap, since they typically use surrounding layers of points to achieve continuity. Embedded grids can be almost anything, and the term is probably best avoided. The use of *zonal* comes mostly from CFD applications where the suggestion of applying different solution equations sets in different flow regions is made. Perhaps *block* or *patched* would be best for the common interface grids, *chimera* for the overlaid (avoiding *overlaid*) grids, and *hybrid* for the structured–unstructured combinations.

1.3.1.2 Forms

With this terminology adopted, the block (or patched) grids may be completely continuous at the interfaces, have slope or line continuity, or be discontinuous (sharing a common interface but not common points thereon). (*Block* seems to cover all of these possibilities, but *patched* is being stretched a bit in the latter case.) Complete continuity is achieved through a surrounding layer of *(image, phantom)* points at which values are kept equal to those at corresponding *object* points inside an adjacent block. This requires a data indexing procedure to link the blocks across the interfaces. With complete continuity, the interface is not fixed (not even in shape), but is determined in the course of the solution. This type of interface necessitates an elliptic generation system. Slope continuity requires that the grid generation procedure incorporate some control over the intersection angle at boundaries (usually, but not necessarily, orthogonality), as can be done through Hermite interpolation in algebraic generation systems or through iterative adjustment of the control functions in elliptic systems. In this case the points on the interface are fixed, and the subgrids are generated independently, except for the use of the common interface points and a common (presumably orthogonal) angle of intersection with the interface. The PDE coding construction is greatly simplified with either complete or slope continuity, since then no algorithm modifications are necessary at the interfaces.

The *chimera* (overlaid) grids are composed of completely independent component grids that overlap a background grid, other component grids and/or other component boundary elements, creating *holes* in the component grids. This requires flagging procedures to locate grid points that lie out of the field of computation, but such holes can be handled even in tridiagonal solvers by placing ones at the corresponding positions on the matrix diagonal and all zeros off the diagonal. These overlaid grids also require interpolation to transfer data between grids, and that subject is the principal focus of effort in regard to the use of this type of composite grid.

The *hybrid* structured–unstructured grids avoid this interpolation by replacing the overlaid region with an unstructured grid connecting logically rectangular structured component grids. This can require modification of solution codes, however.

1.3.2 Block-Structured Grids

Block-structured grids opened the door to real-world CFD in the late 1980s, and many real applications are still based on these grids (see Chapters 12 and 13). The idea appears in the proceedings of the grid conference in Nashville in 1982 (Rubbert and Lee [1982]), but it was Weatherill and Forsey [1984], and Miki and Takagi [1984] that attracted attention to the block-structured approach. Today's structured grid codes are based on this approach. Although the grid is logically rectangular within each block, the blocks fit together in an unstructured manner. Block-structured generation systems that maintain complete continuity across block interfaces allow difference representations to be applied on the block interfaces as in the rest of the field. Complete continuity across block interfaces in the field is accomplished by treating the interface in the manner of a branch cut, with correspondence between "phantom" points outside one block with "real" points inside the adjacent block.

The curvilinear grid system can be constructed simply by setting values in a rectangular array of position vectors,

$$\mathbf{r}_{ijk}\left(i = 1,2,\ldots,I \quad j = 1,2,\ldots,J \quad k = 1,2,\ldots,K\right)$$

and identifying the indices *i, j, k* with the three curvilinear coordinates. The position vector **r** is a three-vector giving the values of the *x, y,* and *z* Cartesian coordinates of a grid point. Since all increments in the curvilinear coordinates cancel out of the transformation relations for derivative operators, there is no loss of generality in defining the discretization to be on integer values of these coordinates.

Fundamental to this curvilinear coordinate system is the coincidence of some coordinate surface with each segment of the boundary of the physical region, in the same manner that surfaces of constant radius coincide with the inner and outer boundary segments of the region between two concentric spheres filled with a polar coordinate system. This is accomplished by placing a two-dimensional array of points on a physical boundary segment and entering these values into the array $\mathbf{r}_{ijk}$ of position vectors, with one index constant, e.g., in , $\mathbf{r}_{ijk}$ with *i* from 1 to *I* and *j* from 1 to *J*. The curvilinear coordinate *k* is thus constant on this physical boundary segment. With values set on the sides of the rectangular array of position vectors in this manner, the generation of the grid is accomplished by determining the values in its interior, e.g., by interpolation or a PDE solution. The set of values $\mathbf{r}_{ijk}$ then forms the nodes of a curvilinear coordinate system filling the physical region. A physical region bounded by six generally curved sides can thus be considered to have been transformed to a logically rectangular computational region on which the curvilinear coordinates are the independent variables.

Although in principle it is possible to establish a correspondence between any physical region and a single empty logically rectangular block for general three-dimensional configurations, the resulting grid is likely to be much too skewed and irregular to be usable when the boundary geometry is complicated. A better approach with complicated physical boundaries is to segment the physical region into contiguous subregions, each bounded by six curved sides (four in 2D) — each of which transforms to a logically rectangular block in the computational region. Each subregion has its own curvilinear coordinate system irrespective of that in the adjacent subregions (see Figure 13.5).

This then allows both the grid generation and numerical solutions on the grid to be constructed to operate in a logically rectangular computational region, regardless of the shape or complexity of the full physical region. The full region is treated by performing the solution operations in all of the logically rectangular computational blocks. With the block-structured framework, partial differential equation solution procedures written to operate on logically rectangular regions can be incorporated into a code for general configurations in a straightforward manner, since the code only needs to treat a rectangular block. The entire physical field then can be treated in a loop over all the blocks. Transformation relations for partial differential equations are covered in Chapter 2 and in Thompson, Warsi, and Mastin [1985], on the Web. Discretization error related to the grid is covered in Chapter 32. The evaluation and control of grid quality (Chapter 33) is an ongoing area of active research.

The generally curved surfaces bounding the subregions in the physical region form internal interfaces across which information must be transferred, i.e., from the sides of one logically rectangular computational block being paired with another on the same, or different, block, since both correspond to the same physical surface. Grid lines at the interfaces may meet with complete continuity, with or without slope continuity, or may not meet at all.

Complete continuity of grid lines across the interface requires that the interface be treated as a branch cut on which the generation system is solved just as it is in the interior of blocks. The interface locations are then not fixed, but are determined by the grid generation system. This is most easily handled in coding by providing an extra layer of points surrounding each block. Here the grid points on an interface of one block are coincident in physical space with those on another interface of the same or another block, and also the grid points on the surrounding layer outside the first interface are coincident with those just inside the other interface, and vice versa. This coincidence can be maintained during the course of an iterative solution of an elliptic generation system by setting the values on the surrounding layers equal to those at the corresponding interior points after each iteration. All the blocks are thus iterated to convergence together, so that the entire composite grid is generated at once. The same procedure is followed by PDE solution codes on the block-structured grid.

The construction of codes for complicated regions is greatly simplified by the block structure since, with the use of the surrounding layer of points on each block, a PDE code is only required basically to operate on a logically rectangular computational region. The necessary correspondence of points on the surrounding layers (image points) with interior points (object points) is set up by the grid code and made available to the PDE code.

1.3.3 Elliptic Systems

Elliptic grid generation is treated in detail in Chapter 4. This section provides an overview of the technology as applied in the EAGLE system (Thompson [1987]), as an example of the technology applied in several current grid generation codes.

1.3.3.1 Generation System

An elliptic grid generation system used in many codes is

$$\sum_{m=1}^{3}\sum_{n=1}^{3} g^{mn}\mathbf{r}_{\xi^m\xi^n} + \sum_{n=1}^{3} g^{nn}P_n\mathbf{r}_{\xi^n} = 0 \tag{1.1}$$

where the g^{mn} are the elements of the contravariant metric tensor,

$$g^{mn} = \nabla\xi^m \cdot \nabla\xi^n$$

These elements are more conveniently expressed for computation in terms of the elements of the covariant metric tensor, g_{mn},

$$g_{mn} = \mathbf{r}_{\xi^m} \cdot \mathbf{r}_{\xi^n}$$

which can be calculated directly. Thus

$$g^{mn} = \left(g_{ik}g_{j\ell} - g_{i\ell}g_{jk}\right) / g$$
$$(m,i,j) \text{ cyclic, } (n,k,\ell) \text{ cyclic}$$

where g, the square of the Jacobian, is given by

$$g = \det\left|g_{mn}\right| = \mathbf{r}_{\xi^1} \cdot \left(\mathbf{r}_{\xi^2} \times \mathbf{r}_{\xi^2}\right)$$

In these relations, $\mathbf{r}$ is the Cartesian position vector of a grid point ($\mathbf{r} = \mathbf{i}x + \mathbf{j}y + \mathbf{k}z$) and the ξ^i ($i = 1,2,3$) are the three curvilinear coordinates. The P_n ($n = 1,2,3$) are the control functions that serve to control the spacing and orientation of the grid lines in the field.

The first and second coordinate derivatives are normally calculated using second-order central differences. One-sided differences dependent on the sign of the control function P_n (backward for $P_n < 0$ and forward for $P_n > 0$) are useful to enhance convergence with very strong control functions. The control functions are evaluated either directly from the initial algebraic grid and then smoothed, or by interpolation from the boundary-point distributions.

1.3.3.2 Control Functions

The now-standard procedure in block-structured systems is to first generate surface grids on block faces — both boundary and in-field block interfaces — from point distributions placed on the face edges by distribution functions. Then volume grids are generated within the blocks. In both this surface and volume grid generation, the first step is normally TFI, to be followed by elliptic generation with control functions interpolated into the field in accordance with boundary point distribution and surface curvature.

The three components of the elliptic grid generation system, Eq. 1.1, provide a set of three equations that can be solved simultaneously at each point for the three control functions, P_n ($n = 1,2,3$), with the derivatives here represented by central differences. This produces control functions that will reproduce the algebraic grid from the elliptic system in a single iteration, of course. Thus, evaluation of the control functions in this manner would be of trivial interest except that these control functions can be smoothed before being used in the elliptic generation system. This smoothing is done by replacing the control function at each point with the average of the four neighbors in the two curvilinear directions (one in 2D) other than that of the function. Thus P_i is smoothed in the ξ^j and ξ^k directions, where i, j, k are cyclic. No smoothing is done in the direction of the function because to do so would smooth the spacing distribution.

An algebraic grid is generated by transfinite interpolation (Chapter 3) from the boundary point distribution, to serve as the starting solution for the iterative solution of the elliptic system. With the boundary point distribution set from the hyperbolic sine or tangent functions, which have been shown to give reduced truncation error (Chapters 3 and 32), this algebraic grid has a good spacing distribution but may have slope breaks propagated from corners into the field. The use of smoothed control functions evaluated from the algebraic grid produces a smooth grid that retains essentially the spacing of the algebraic grid.

The elliptic generation system can be solved by point SOR iteration using a field of locally optimum acceleration parameters. These optimum parameters make the solution robust and capable of convergence with strong control functions.

Control functions can also be evaluated on the boundaries using the specified boundary point distribution in the generation system, with certain necessary assumptions (orthogonality at the boundary) to eliminate some terms, and then can be interpolated from the boundaries in this manner. More general regions can, however, be treated by interpolating elements of the control functions separately. Thus control functions on a line on which ξ^n varies can be expressed as

$$P_n = A_n = +\frac{S_n}{\rho_n} \qquad (1.2)$$

where A_n is the logarithmic derivative of the arc length, S_n is the arc length spacing, and P_n the radius of curvature of the surface on which ξ^n is constant.

The arc length spacing, S_n, and the arc length contribution, A_n, to the control function can be interpolated into the interior of the block from the four sides on which they are known by two-dimensional interpolation. The radius of curvature, ρ_n, however is interpolated one-dimensionally between the two surfaces on which it is evaluated. The control function is finally formed by adding the arc length spacing divided by the radius of curvature to the arc length contribution according to Eq. 1.2. This procedure allows very general regions with widely varying boundary curvature to be treated. A more general construction of the control functions is given in Chapter 4.

1.3.3.3 Boundary Orthogonality

The standard approach used to achieve orthogonality and specified off-boundary spacing on boundaries has been the iterative adjustment of control functions in elliptic generation systems, first introduced by Sorenson in the GRAPE code in the 1980s (Sorenson [1989]). Various modifications of this basic concept have been introduced in several codes, and the general approach is now common (see Chapter 6).

A second-order elliptic generation system allows either the point locations on the boundary or the coordinate line slope at the boundary to be specified, but not both. It is possible, however, to iteratively adjust the control functions in the generation system until not only a specified line slope but also the spacing of the first coordinate surface off the boundary is achieved, with the point locations on the boundary specified. These relations can be applied on each successive coordinate surface off the boundary, with the off-surface spacing determined by a hyperbolic sine distribution from the spacing specified at the boundary. The control function increments are attenuated away from the boundary, and contributions are accumulated from all orthogonal boundary sections. Since the iterative adjustment of the control functions is a feedback loop, it is necessary to limit the acceleration parameters for stability. This allows the basic control function structure evaluated from the algebraic grid, or from the boundary-point distributions, to be retained, and thus relieves the iterative process from the need to establish this basic form of the control functions. The extent of the orthogonality into the field can also be controlled. This orthogonality feature is also applicable on specified grid surfaces within the field, allowing grid surfaces in the field to be kept fixed while retaining continuity of slope of the grid lines crossing the surface. This is quite useful in controlling the skewness of grid lines in some troublesome areas.

Alternatively, boundary orthogonality can be achieved through Neumann boundary conditions, which allow the boundary points to move over a surface spline. The boundary point locations by Newton iteration on the spline to be at the foot of normals to the adjacent field points. This is done as follows: The Neumann point on the section currently closest to the field point **R** is first located. This is done by sweeping the section in ever expanding squares centered on the Neumann point. (These squares are actually limited by the section edges, of course, and hence, may become rectangles.) Next the quadrant about this closest point above which the field point lies is determined by comparing the dot products of the distance vector (from the closest point to the field point) with the tangent vectors ($\mathbf{r}_\xi$, $\mathbf{r}_\eta$) to the two grid lines on the section. The quadrant is identified by the signs of these two dot products. The Neumann boundary point in question, **r**, is then moved to the foot of the normal from the adjacent field point to the surface. This position is found as the solution of the nonlinear system

$$\mathbf{r}_\xi \cdot (\mathbf{R} - \mathbf{r}) = 0,\ \mathbf{r}_\eta \cdot (\mathbf{R} - \mathbf{r}) = 0 \tag{1.3}$$

by Newton iteration. The location of the closest current boundary point and the examination of dot products described above has determined the surface cell, i.e., the quadrant, on which this solution lies so that the iteration can be confined to a single cell.

Provision can also be made for extrapolated zero-curvature boundary conditions and for mirror-image reflective boundary conditions on symmetry planes.

1.3.3.4 Surface Grids

In the case of a curved surface, the surface is splined and the surface grid is generated in terms of surface parametric coordinates. The generation of a grid on a general surface (Chapter 9) is a two-dimensional grid problem in its own right, which can also be done either by interpolation or a PDE solution. In general, this is a 2D boundary value problem on a curved surface, i.e., the determination of the locations of points on the surface from specified distributions of points on the four edges of the surface. This is best approached through the use of surface parametric coordinates, whereby the surface is first defined by a 2D array of points, $\mathbf{r}_{mn}$, e.g., a set of cross sections. The surface is then splined, and the spline coordinates (u,v; surface parametric coordinates) are then made the dependent variables for the interpolation or PDE generation system. The generation of the surface grid can then be accomplished by first specifying the boundary points in the array $\mathbf{r}_{ij}$ on the four edges of the surface grid; converting these Cartesian coordinate values to spline coordinate values (u_{ij}, v_{ij}) on the edges; then determining the interior values in the arrays u_{ij} and v_{ij} from the edge values by interpolation or PDE solution; and finally converting these spline values to Cartesian coordinates $\mathbf{r}_{ij}$ (see Figure 9.1).

1.3.4 Hyperbolic System

Elliptic generation systems operate throughout the entirety of a region, while hyperbolic systems move outward from boundaries. An alternate approach to boundary orthogonality and spacing is to incorporate a hyperbolic generation system near the boundary, transitioning to an elliptic system in the far field.

It is also possible to base a grid generation system on hyperbolic partial differential equations, rather than elliptic equations (Chapter 5). In this case the grid is generated by numerically solving a hyperbolic system, marching in the direction of one curvilinear coordinate between two boundary curves in two dimensions, or between two boundary surfaces in three dimensions. The hyperbolic system, however, allows only one boundary to be specified, and is therefore of interest only for use in calculation on physically unbounded regions where the precise location of a computational outer boundary is not important. The hyperbolic grid generation system has the advantage of being generally faster than elliptic generation systems but, as just noted, is applicable only to certain configurations. Hyperbolic generation systems can be used to generate orthogonal grids.

In two dimensions the condition of orthogonality is simply $g_{12} = 0$. If either the cell area $\sqrt{g}$ or the cell diagonal length (squared), $g_{11} + g_{22}$, is a specified function of the curvilinear coordinates, i.e.,

$$\sqrt{g} = F(\xi,\eta) \text{ or } g_{11} + g_{22} = F(\xi,\eta)$$

then the system consisting of $g_{12} = 0$ and either of the two equations just above is hyperbolic. Since the system is hyperbolic, a noniterative marching solution can be constructed proceeding in one coordinate direction away from a specified boundary.

The cell volume distribution in the field can be controlled by the specified *control function F*. One form of this specification is as follows: Let points be distributed on a circle having a perimeter equal to that of the specified boundary at the same arc length distribution as on that boundary. Then specify a radial distribution of concentric circles about this circle according to some distribution function, e.g., the hyperbolic tangent. Then use the volume distribution from this unequally spaced cylindrical coordinate system as F. The specification of the cell volume prevents the coordinate system from overlapping even off a concave boundary. In this case the line spacing will expand rapidly away from the boundary in order to keep the cell volume from vanishing. Although this prevents overlap, the rapid expansion that occurs can lead to problems with truncation error in some cases. This approach is extendable to 3D with the coordinate lines emanating from the boundary being orthogonal to the other two coordinates, but the latter two lines not being orthogonal. There apparently is no system, hyperbolic or elliptic, that will give complete orthogonality in 3D in general.

This hyperbolic grid generation system is faster than the elliptic generation systems by one or two orders of magnitude, the computational time required being equivalent to about that for one iteration in a solution of the elliptic system. The specification of the cell volume distribution avoids the grid line overlapping that otherwise can occur with concave boundaries in a method involving projection away from a boundary. The grid may, however, be somewhat distorted when concave boundaries are involved. The cell volume specification also allows control of the grid line spacing, but again concave boundaries may cause the intended spacing to occur in the wrong coordinate direction, since it is only the volume, and not the spacing in the two separate coordinate directions, that is controlled. As has been noted, the grid is constructed to be orthogonal.

The hyperbolic generation system is not as general as the elliptic systems, however, since the entire boundary of the region cannot be specified. Boundary slope discontinuities are propagated into the field, so that the metric elements will be discontinuous along coordinate lines emanating from boundary slope discontinuities. Finally, since hyperbolic partial differential equations can have shock-like solutions in some circumstances, it is possible for very unsuitable grids to result with some specifications of boundary point and cell volume distributions. This is in contrast with the elliptic generation system, which tends to emphasize smoothness because of the nature of elliptic partial differential equations.

1.3.5 Algebraic System

Transfinite interpolation (TFI) has become the standard for algebraic grid generation systems, and is now incorporated in most large codes. TFI can accomplish interpolation from any combination of faces, edges, and corners — with boundary orthogonality and with blending functions interpolated from boundary point distributions.

Algebraic grid generation is treated in detail in Chapter 3. This section provides an overview of the technology as applied in the EAGLE system, for example (Thompson [1988]).

1.3.5.1 Transfinite Interpolation

An algebraic three-dimensional generation system based on transfinite interpolation (using either Lagrange or Hermite interpolation) generates an initial solution to start the iterative solution of the elliptic generation system. The interpolation, in general complete transfinite interpolation from all boundaries, can be restricted to any combination of directions or lesser degrees of interpolation, and the form (Lagrange, Hermite, or incomplete Hermite) can be different in different directions or in different blocks. The blending functions can be linear or, more appropriately, based on interpolated arc length from the boundary point distributions. (This arc length is interpolated by 2D transfinite interpolation from four sides of the block.)

Hermite interpolation, based on cubic blending functions, allows orthogonality at the boundary. Incomplete Hermite uses quadratic functions and hence can give orthogonality atone of two opposing boundaries, while Lagrange, with its linear functions, does not give orthogonality.

The transfinite interpolation is done by the appropriate combination of 1D projectors, F_i, for the type of interpolation specified. (Each projector is simply the 1D interpolation in the direction indicated.) For interpolation from all sides of the section, if all three directions are indicated and the section is a volume, this interpolation is from all six sides, and the combination of projectors is the Boolean sum of the three projectors:

$$F_1 + F_2 + F_3 - F_1F_2 - F_2F_3 - F_3F_1 + F_1F_2F_3$$

With interpolation in only the two directions j and k, or if the section is a surface on which ξ^i is constant, the combination is the Boolean sum of F_j and F_k:

$$F_j + F_k - F_jF_k \quad (i, j, k) \text{ cyclic}$$

With interpolation in only a single direction i, or if the section is a line on which ξ^i varies, the interpolation is between the two sides on which ξ^i is constant using only the single projector F_i.

With interpolation from the edges of the section, with all three directions indicated and the section a volume, the interpolation is from all 12 edges using the Boolean combination

$$F_1F_2 + F_2F_3 + F_3F_1 - 2F_1F_2F_3$$

Interpolation from the eight corners of the section is done using $F_1\,F_2\,F_3$. There are also other possible combinations.

Blocks can be divided into subblocks for the purpose of generation of the algebraic grid and the control functions. Point distributions on the sides of the subblocks can either be specified or automatically generated by transfinite interpolation from the edge of the side. This allows additional control over the grid in general configurations, and is particularly useful in cases where point distributions need to be specified in the interior of a block or to prevent grid overlap in highly curved regions. This also allows points in the interior of the field to be excluded if desired, e.g., to represent holes in the field.

1.3.6 Adaptive Grid Schemes

Adaptive grid systems are treated in detail in Chapters 33 and 34. This section provides an overview of the technology as applied in the EAGLE system as an example, (Tu and Thompson [1991], Kim and Thompson [1990]).

Dynamically adaptive grids continually adapt to follow developing gradients in the physical solution. This adaptation can reduce the oscillations associated with inadequate resolution of large gradients, allowing sharper shocks and better representation of boundary layers. Another advantageous feature is the fact that in the viscous regions where real diffusion effects must not be swamped, the numerical dissipation from upwind biasing is reduced by the adaptation. Dynamic adaptation is at the frontier of numerical grid generation and may well prove to be one of its most important aspects, along with the treatment of real three-dimensional configurations through the composite grid structure.

There are three basic strategies that may be employed in dynamically adaptive grids coupled with the partial differential equations of the physical problems. (Combinations are also possible, of course.)

1.3.6.1 Redistribution of a Fixed Number of Points

In this approach, points are moved from regions of relatively small error or solution gradient to regions of large error or gradient. As long as the redistribution of points does not seriously deplete the number of points in other regions of possible significant gradients, this is a viable approach. The increase in spacing that must occur somewhere is not of practical consequence if it occurs in regions of small error or gradient, even though in a formal mathematical sense the global approximation is not improved. The redistribution approach has the advantage of not increasing the computer time and storage during the solution, and of being straightforward in coding and data structure. The disadvantages are the possible deleterious depletion of points in certain regions and the possibility of the grid's becoming too skewed.

1.3.6.2 Local Refinement of a Fixed Set of Points

In this approach, points are added (or removed) locally in a fixed point structure in regions of relatively large error or solution gradient. Here there is, of course, no depletion of points in other regions, and therefore no formal increase of error occurs. Since the error is locally reduced in the area of refinement, the global error does formally decrease. The practical advantage of this approach is that the original point structure is preserved. The disadvantages are that the computer time and storage increase with the refinement, and that the coding and data structure are difficult, especially for implicit flow solvers.

1.3.6.3 Local Increases in Algorithm Order

In this approach, the solution method is changed locally to a higher-order approximation in regions of relatively large error or solution gradient without changing the point distribution. This again increases the formal global accuracy elsewhere. The advantage is that the point distribution is not changed at all. The disadvantage is the great complexity of implementation in implicit flow solvers.

1.3.6.4 Formulations

Adaptive redistribution of points traces its roots to the principle of equidistribution of error, by which a point distribution is set so as to make the product of the spacing and a weight function constant over all the points:

$$w\nabla x = \text{constant}$$

With the point distribution defined by a function $x(\xi)$, where ξ varies by a unit increment between points, the equidistribution principle can also be expressed as

$$wx_{\xi} = \text{constant} \tag{1.4}$$

This one-dimensional equation can be applied in each direction in an alternating fashion, but a direct extension to multiple dimensions can also be made in either of two ways as follows.

From the calculus of variations, the above equation can be shown to be the Euler variational equation for the function $x(\xi)$, which minimizes the integral

$$I = \int w(x) x_{\xi}^{2} dx$$

Generalizing this, a competitive enhancement of grid smoothness, orthogonality, and concentration can be accomplished by representing each of these features by integral measures over the grid and minimizing a weighted average of the three.

The second approach is to note the correspondence between Eq. 1.4 and the one-dimensional form of the following commonly used elliptic grid generation system, Eq. 1.1. Here the *control functions, P_n*, serve to control the grid line spacing and orientation. The 1D form of Eq. 1.1 is

$$x_{\xi\xi} + Px_{\xi} = 0 \tag{1.5}$$

Differentiation of Eq. 1.4 yields

$$wx_{\xi\xi} + w_{\xi}x_{\xi} = 0 \tag{1.6}$$

Then, from Eq. 1.5 and 1.6,

$$\frac{x_{\xi\xi}}{x_{\xi}} = -P = -\frac{w_{\xi}}{w}$$

from which the control function can be taken as

$$P = \frac{w_\xi}{w} \tag{1.7}$$

It is logical then to represent the control functions in 3D as

$$P_n = \frac{w_{\xi^n}}{w}, n = 1,2,3 \tag{1.8}$$

This can be generalized to 3D as

$$P_i = \sum_j \frac{g_{ij}}{g^{ii}} \frac{(w_i)_{\xi^i}}{w_i} \tag{1.9}$$

which, in fact, does arise from a variational form (Warsi and Thompson [1990]). An example of the use of adaptive grids is shown in Figure 34.9.

1.4 Unstructured Grids

Unstructured grid generation has its roots in the finite element world of structures modeling. The real introduction to the CFD community came in the 1980s primarily from Baker, Weatherill, and Lohner. Unstructured grids have inherent simplicity of construction in that, by definition, no structure is required. Also it is not inherently necessary to communicate the actual topology of the configuration to the grid generator. Although largely synonymous with tetrahedral grids, unstructured grids may alternatively be composed of hexahedral cells (without directional structure). The term might strictly encompass any combination of cell shapes, but in the grid generation literature combinations of regions with structure (e.g., structured or prismatic grids near body surfaces) with regions without structure are generally called hybrid grids. For that matter, block-structured grids are unstructured in the large.

Traditionally, unstructured grids have been used with the finite element method. There is, therefore, an extensive literature that covers techniques to generate unstructured grids (cf. Carey [1997], George [1991], Thacker [1980]).

In this introductory chapter, it is not possible to present, in detail, all the different techniques. Instead, emphasis here will be given to one particular approach, based upon the Delaunay triangulation, which provides a powerful unstructured grid generation method. This will be used to illustrate the flexibility and characteristics of unstructured grid methods when applied to complicated geometries in two and three dimensions and in grid adaptation. Brief details of other methods will be given.

1.4.1 The Delaunay Triangulation

Structured grid generation methods place an emphasis on creating the position of points. The subsequent connections between points are defined automatically given the (i, j, k) ordering. Such ordering does not exist in unstructured grids and hence connections between points, in addition to the position of points, have to be defined by an unstructured grid method.

Grid generation based on the Delaunay triangulation (Chapter 16) uses a particularly simple criterion for connecting points to form conforming, nonintersecting elements. This geometrical construction has been known for many years, but only relatively recently has it been used for grid generation for computational

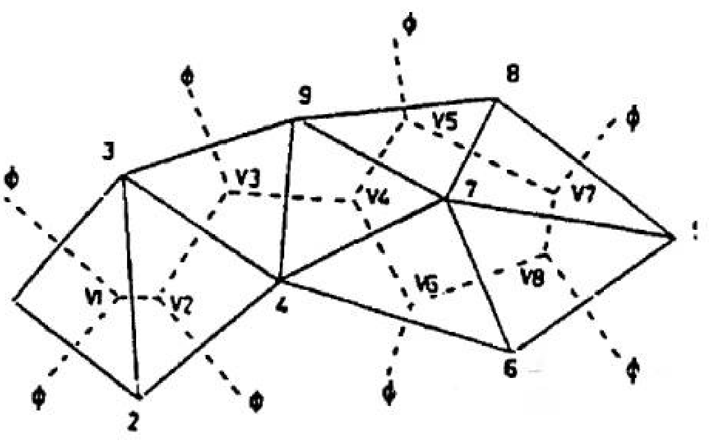

FIGURE 1.1 The Delaunay triangulation (solid line), and Voronoï regions (hashed line).

fluid dynamics. The geometrical criterion provides a mechanism for connecting points. The task of point generation must be considered independently. Hence, grid generation by Delaunay triangulation involves the two distinct problems of point connection and point creation.

1.4.1.1 Delaunay–Voronoï Geometrical Construction

Dirichlet [1850] first proposed a method whereby a given domain, in arbitrary space, could be systematically decomposed into a set of packed convex regions. For a given set of points (P_i), the space is subdivided into regions (V_i), in such a way that the region (V_i) is the space closer to P_i than to any other point. This geometrical construction of tiles is known as the Dirichlet tessellation. This tessellation of a closed domain results in a set of non-overlapping convex regions called Voronoï regions (Voronoï [1908]) that cover the entire domain. More formally, if a set of points is denoted by (P_i), then the Voronoï regions (V_i) can be defined as

$$(V_i) = \{P : \|p - P_i\| < \|p - P_j\|\}, \forall j \neq i \tag{1.10}$$

i.e., the Voronoï regions (V_i) are the set of points P that are closer to P_i than to any other point. The sum of all points p forms a Voronoï region.

From this definition, it is clear that, in two dimensions, the territorial boundary that forms a side of a Voronoï polygon must be midway between the two points that it separates and is thus a segment of the perpendicular bisector of the line joining these two points. If all point pairs that have some segment of boundary in common are joined by straight lines, the result is a triangulation within the convex hull of the set of points (P_i). This triangulation is known as the Delaunay triangulation (Delaunay [1934]). An example of this geometrical construction is given in Figure 1.1.

The construction is also valid in three dimensions. Territorial boundaries are faces that form Voronoï polyhedra and are equidistant between point pairs. If points with a common face are connected, then a set of tetrahedra is formed that covers the convex hull of points.

The Delaunay triangulation possesses some interesting properties. One such property is the in-circle criterion, which states that the circumcircles of the triangles $T(P_i)$ contain no points of the set (P_i). This applies in arbitrary dimensions and is the property used to construct an algorithm for the triangulation. As a consequence of the in-circle criterion, in two dimensions, the triangulation $T(P_i)$ also satisfied the maximum–minimum criterion, which states that if the diagonal of any strictly convex quadrilateral is replaced by the opposite one, the minimum of the six internal angles will not decrease. This is a particularly attractive property, since it ensures that the triangulation maximizes the angle regularity of the triangles, and in this way is analogous to the smoothness property of grids generated by elliptic partial differential equations.

Voronoï Vertex	Delaunay Triangle	Neighbor Voronoï Vertex
1	1 2 3	2 ϕ ϕ
2	2 3 4	1 3 ϕ
3	3 4 9	2 4 ϕ
4	4 7 9	3 5 6
5	7 8 9	4 7 ϕ
6	4 6 7	4 8 ϕ
7	5 8 7	5 8 ϕ
8	5 7 6	6 7 ϕ

FIGURE 1.2 The data structure for the Voronoï diagram and Delaunay triangulation shown in Figure 1.1.

The structure of the Voronoï diagram and Delaunay triangulation can be described by constructing two lists for each Voronoï vertex: a list of the points that define a triangle for a given vertex of the Voronoï construction (so-called forming points), and a free data structure containing the neighboring Voronoï vertices to a given Voronoï vertex. As an example, Figure 1.2 contains the vertex structure for the construction shown in Figure 1.1.

This data structure naturally extends to applications in three dimensions, where each Voronoï vertex has four forming points (tetrahedra of the Delaunay triangulation) and four neighboring Voronoï vertices.

1.4.1.2 Algorithm to Construct the Delaunay Triangulation

There are several algorithms used to construct the Delaunay triangulation. One approach, which is flexible in that it readily applies to two and three dimensions, is due to Bowyer [1981]. Each point is introduced into an existing Delaunay satisfying structure, which is locally broken and then reconstructed to form a new Delaunay-satisfying construction. In the presentation here the terms in italics indicate the interpretation for three dimensions.

Algorithm I

Step 1

Define the convex hull within which all points will lie. It is appropriate to specify four points (*eight points*) together with the associated Voronoï diagram structure.

Step 2

Introduce a new point anywhere within the convex hull.

Step 3

Determine all vertices of the Voronoï diagram to be deleted. A point that lies within a circle (*sphere*) centered at a vertex of the Voronoï diagram and passes through its three (*four*) forming points results in the deletion of that vertex. This follows from the "in-circle" criterion.

Step 4

Find the forming points of all the deleted Voronoï vertices. These are the contiguous points to the new point.

Step 5

Determine the neighboring Voronoï vertices to the deleted vertices that have not themselves been deleted. These data provide the necessary information to enable valid combinations of the contiguous points to be constructed.

Step 6

Determine the forming points of the new Voronoï vertices. The forming points of the new vertices must include the new point together with the two (*three*) points that are contiguous to the new point and form an edge (*face*) of a neighbor triangle (*tetrahedron*). These are the possible combinations obtained from Step 5.

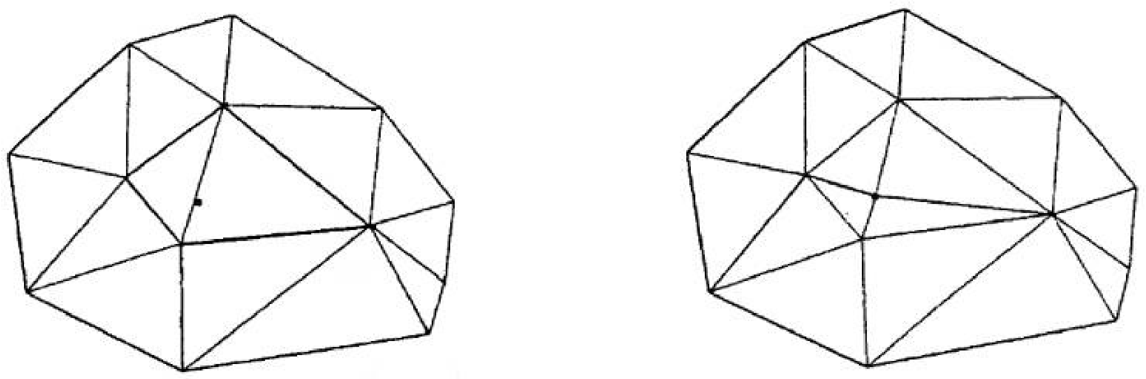
FIGURE 1.3 The addition of a new point results in deletion of some triangles and the construction of new ones.

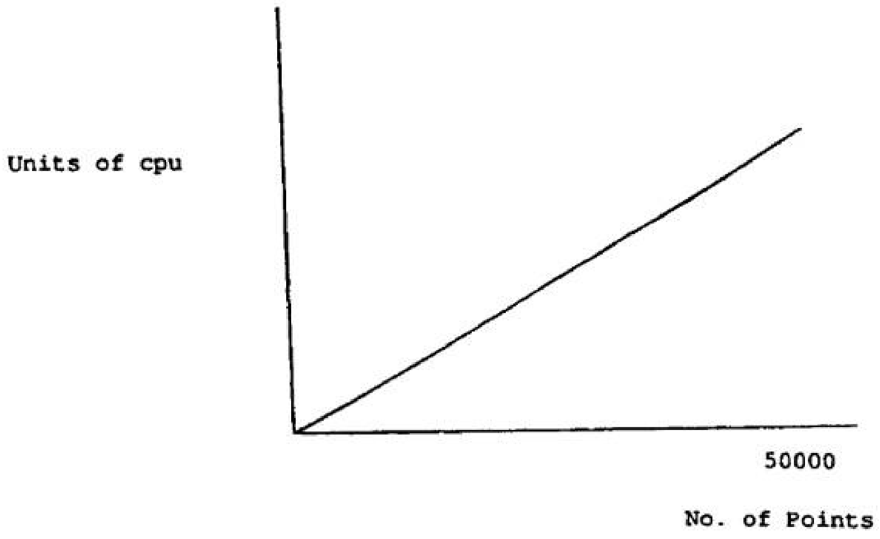

FIGURE 1.4 CPU time v. number of connected points.

Step 7

Determine the neighboring Voronoï vertices to the new Voronoï vertices. Following Step 6, the forming points of all new vertices have been computed. For each new vertex, perform a search through the forming points of the neighboring vertices, as found in Step 5, to identify common pairs (*triads*) of forming points. When a common combination occurs, then the two (*three*) associated vertices are neighbors of the Voronoï diagram.

Step 8

Reorder the Voronoï diagram structure, overwriting the entries of the deleted vertices.

Step 9

Repeat Steps 2–8 for the next point.

Figure 1.3 indicates that for a given point, the local region of influence is detected, i.e., the triangles associated with circles which contain the point. These triangles are deleted, and the new point connected to the nodes which form the enclosing polygon. This new construction is Delaunay satisfying.

The algorithm described here can be used to connect an arbitrary set of points that lie within a convex hull. The efficiency with which this can be achieved depends upon the use of appropriate data structures. The tree structure of neighbor vertices, indicated in Figure 1.2, is central to the implementation. To illustrate performances, Figure 1.4 shows a plot of CPU time against a number of elements generated in 3D on a workstation.

The algorithm described provides an important basis for an unstructured grid method. To illustrate its use and to demonstrate an additional problem, consider the problem of generating a boundary

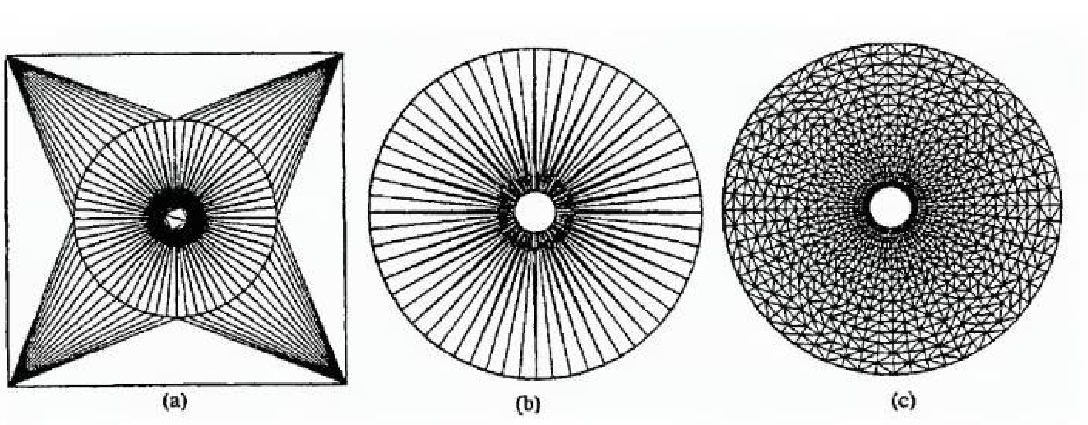

FIGURE 1.5 Delaunay triangulation of points on two circles. (a) Delaunay construction including the convex hull points. (b) Delaunay construction after the removal of the convex hull points. (c) Delaunay construction with points from a polar grid.

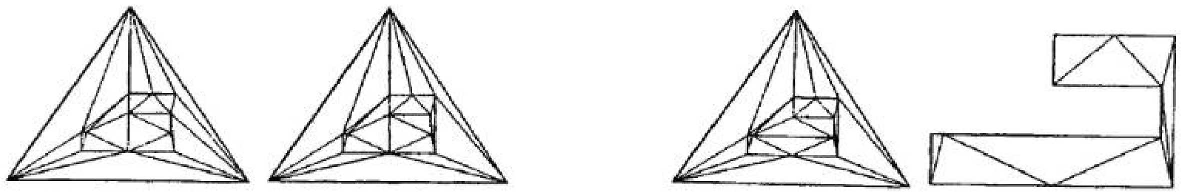

FIGURE 1.6 The boundary is completed by swapping edges in the Delaunay triangulation.

conforming grid within a multiply connected domain defined between two concentric circles. The circles are defined by a set of discrete points. Following the algorithm outlined, these points can be contained within an appropriate hull and then connected together. The result is shown in Figure 1.5a. It is clear that a set of valid triangles has been derived that covers the region of the hull. Two issues are immediately raised. First, to derive a triangulation in the specified region, triangles outside this region must be deleted. Second, if the triangles are to provide a boundary conforming grid it is necessary that edges in the Delaunay triangulation form the given geometrical boundaries of the domain. Unfortunately, given a set of points which define prespecified boundaries there is no guarantee that the resulting Delaunay triangulation will contain the edges which define the domain boundaries. Such a case is also true in three dimensions, where boundary faces must be included in the tetrahedra of the Delaunay triangulation for the resulting grid to be boundary conforming. It is necessary, therefore, to check the integrity of boundaries, and if found not to be complete, appropriate steps must be taken.

Prespecified boundary connectivities can be reconstructed by combinations of edge swapping to recover boundaries in two dimensions is given in Figure 1.6. The given boundary edges are recovered through edge swapping. In 3D, this problem is more severe and requires careful attention.

Once the boundary is complete, it is a simple task to delete triangles exterior to the region of interest. Deletion of unwanted triangles in Figure 1.5a leads to the triangulation shown in Figure 1.5b.

Figure 1.5b represents a valid triangulation of the points that define the two concentric circles. However, the triangles span the entire region and are clearly inappropriate for any form of analysis. Hence, it is necessary to address the problem of point creation.

1.4.2 Point Creation

1.4.2.1 Points Created by an Independent Generation Technique

Points for connection by the Delaunay algorithm could be derived by a method external to the triangulation routine. For example, in the case of the two circles, a polar grid could be generated and the set of points then connected together to form the grid. Such a triangulation with polar grid points is overly

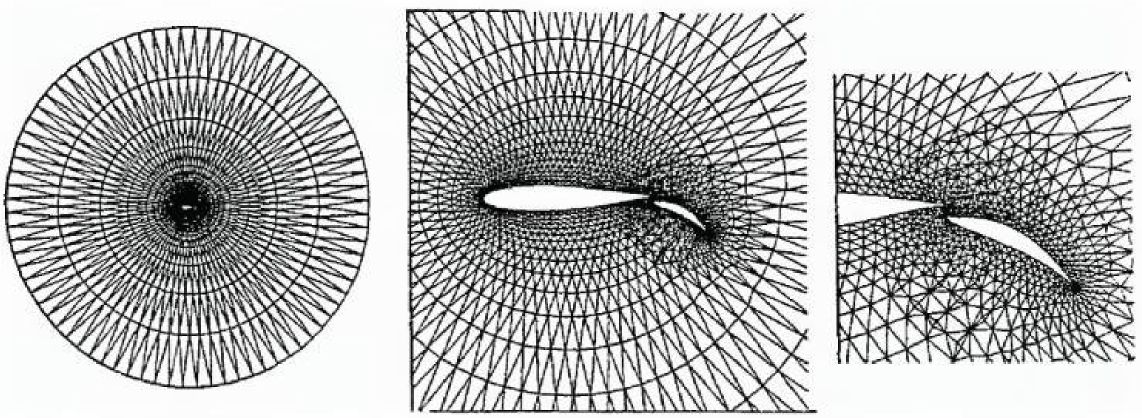
FIGURE 1.7 Unstructured grid with points generated from a structured method.

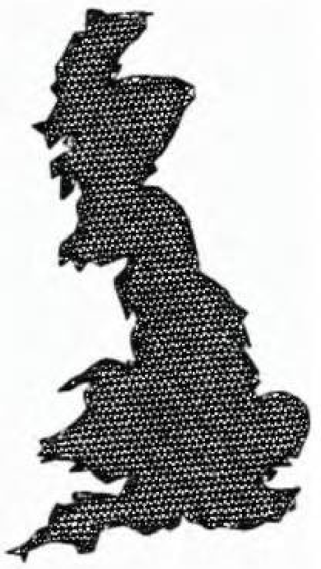
FIGURE 1.8 Delaunay triangulation of a regular set of points superimposed over the domain.

complicated. However, for more realistic domains, which may be more geometrically complex, the approach can prove to be effective. Taking an example from aerospace engineering, Figure 1.7 shows a grid in which two structured grids have been independently generated around the two components and the total set of points connected together to form the unstructured mesh. For more general geometries, alternative, more flexible point creation routines are required.

1.4.2.2 Points Created by Grid Superposition and Successive Subdivision

It is possible to extend the use of an independent grid generation technique to include grid superposition and successive subdivision. The basic idea is to superimpose a regular grid over the domain. The regular grid can be generated using a quadtree or octree data structure that allows point density in the regular grid to be consistent with point spacing at the boundary. An example of this approach is shown in Figure 1.8. In general, this approach results in good spatial discretization in the interior of the domain, although in the vicinity of boundaries the grid quality can be poor.

1.4.2.3 Point Creation Driven by the Boundary Point Distribution

For grid generation purposes, the domain is defined by points on the geometrical boundaries. It will be assumed that this point distribution reflects appropriate geometrical features, such as variation in boundary curvature and gradient. Ideally, any method for automatic point generation should ensure that the boundary point distribution is extended into the domain in a smooth manner. A procedure that has proved successful in creating a smooth point distribution consistent with boundary point spacing and that naturally extends to three dimensions is as follows.

Algorithm II

Step 1

Compute the point distribution function dp_i for each boundary point $\mathbf{r}_i = (x, y)$, where it is assumed that points i+1 and i are contiguous:

$$dp_i = 0.5\left\lfloor \sqrt{(\mathbf{r}_{i+1} - \mathbf{r}_i)^2} + \sqrt{(\mathbf{r}_i - \mathbf{r}_{i-1})^2} \right\rfloor$$

Step 2

Generate the Delaunay triangulation of the boundary points.

Step 3

Initialize $j = 0$.

Step 4

For each triangle T_m within the domain, perform the following:

a. Define a point at the centroid of the triangle T_m, with nodes n_1, n_2, n_3:

$$\mathbf{P}_c = \frac{1}{3}(\mathbf{r}_{n1} + \mathbf{r}_{n2} + \mathbf{r}_{n3})$$

b. Derive the point distribution function dp_c by interpolating the point distribution function from the nodes n_1, n_2, n_3:

$$dp_c = \frac{1}{3}(dp_{n1} + dp_{n2} + dp_{n3})$$

c. If

$$|\mathbf{P}_c - \mathbf{r}_{nk}| < \alpha\, dp_c \quad k = 1,2,3$$

then reject point P_c; next triangle. If

$$|\mathbf{P}_c - \mathbf{r}_{nk}| > \alpha\, dp_c \quad k = 1,2,3$$

then, if

$$|P_j - P_c| > \beta\, dp_c \quad j = 1,\ldots,N$$

accept point $\mathbf{P}_c$ and add to list $\mathbf{P}_j$, $j = 1, N$. If

$$|\mathbf{P}_j - \mathbf{P}_c| < \beta\, dp_c \quad j = 1,\ldots,N$$

then reject point $\mathbf{P}_c$; next triangle

Step 5

If $j = 0$, go to Step 7.

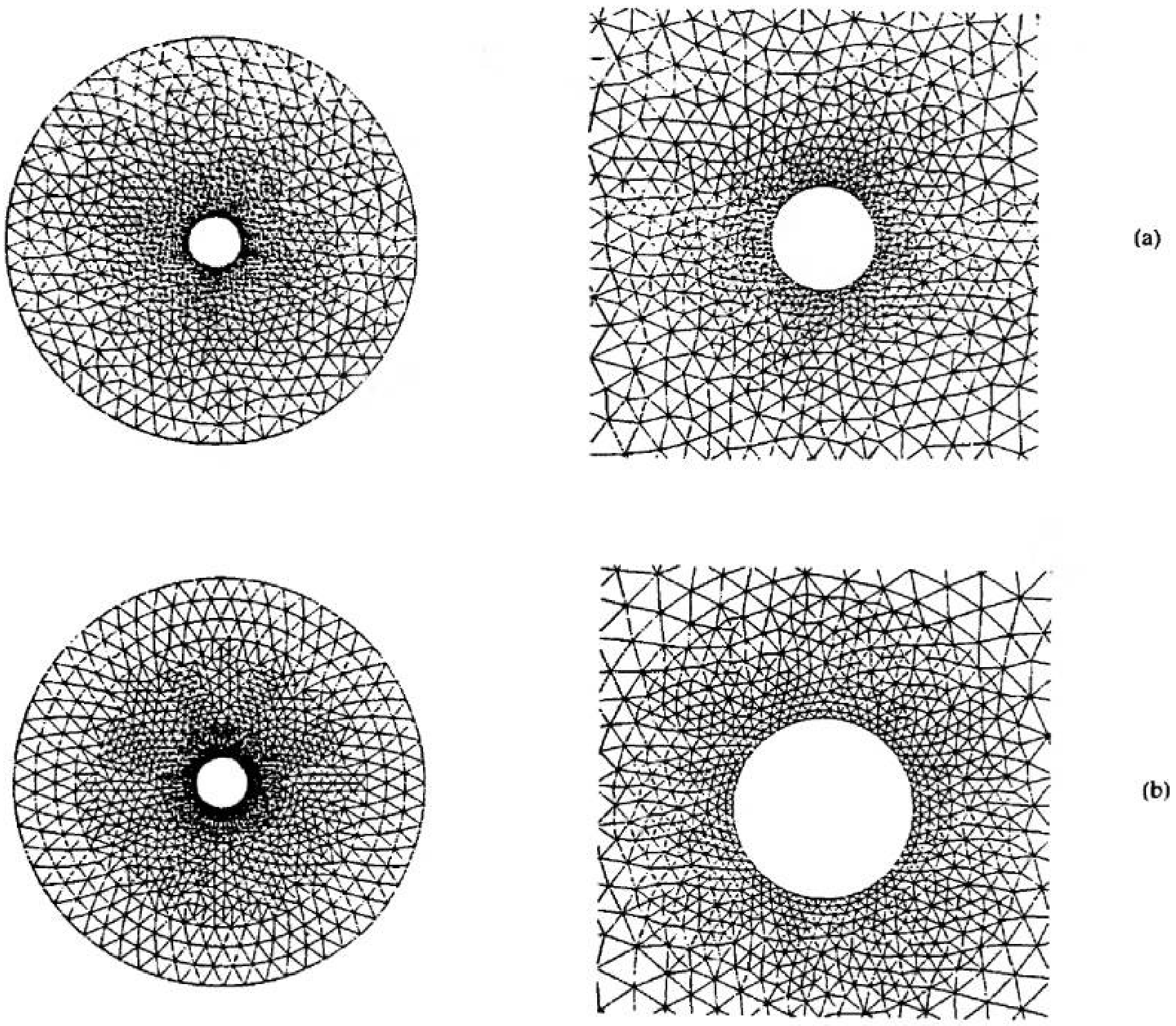

FIGURE 1.9 Grid point creation and distribution controlled by the boundary point distribution.

Step 6

Perform Delaunay triangulation of the derived points $\mathbf{P}_j$, $j = 1, N$. Go to Step 3.

Step 7

Smooth the grid.

It proves beneficial to smooth the position of the grid points using a Laplacian filter. The coefficient α controls the grid point density, while β has an influence on the regularity of the triangulation. Figure 1.9 shows two triangulations produced using this point creation algorithm. In Figure 1.9a, $\alpha = 1$, $\beta = 10$, while in Figure 1.9b $\alpha = 1$, $\beta = 0.02$. The effect of β is clearly evident. A more realistic example is given in Figure 1.10, where a grid is shown for a finite element stress analysis of a tooth.

The algorithm outlined is equally applicable in three dimensions. Figure 1.11 shows an unstructured tetrahedral grid around an airplane. Appropriate point clustering has been achieved close to the plane. A flow solution has been computed on this mesh as a further demonstration of the applicability of the approach.

The procedure outlined creates points consistent with the point distribution on the boundaries. However, in many problems information is known about features within the domain that require a suitable spatial discretization. It proves possible to modify the above algorithm to take such effects into account. Two techniques can be readily implemented. The first utilizes the idea of point and line sources, while the second uses the concept of a background mesh.

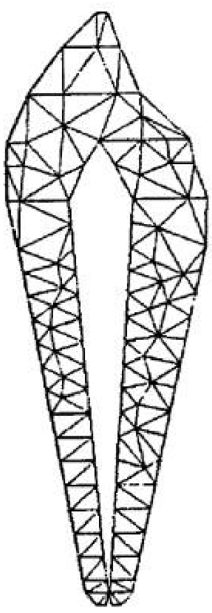
FIGURE 1.10 Unstructured grid used for finite element stress analysis.

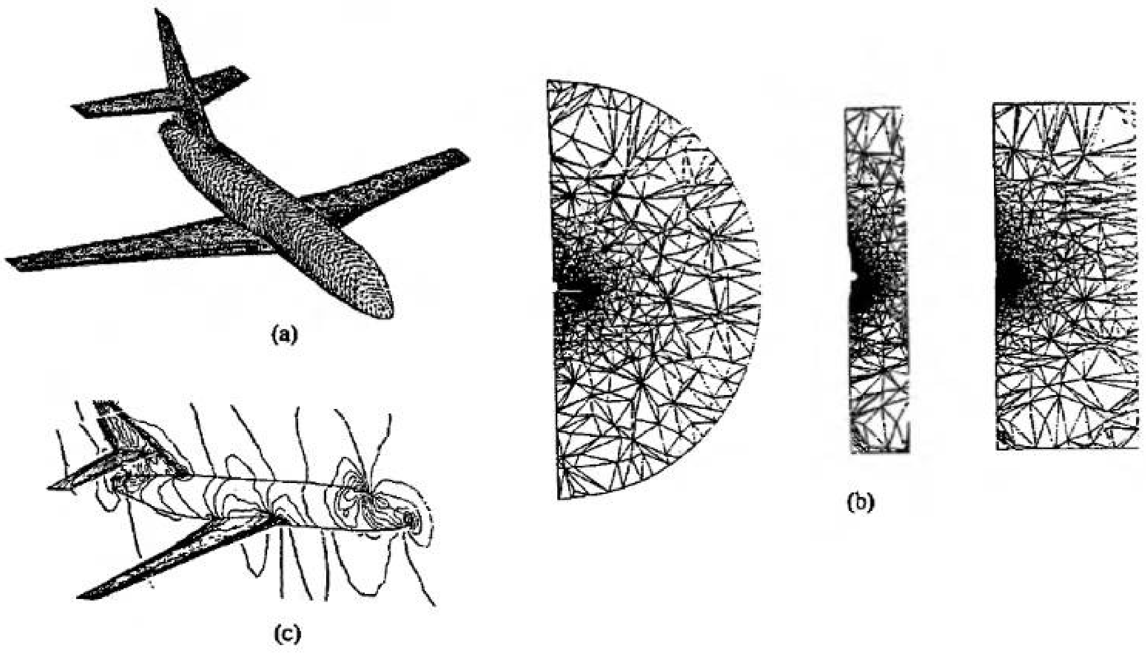

FIGURE 1.11 Automatic point creation in three dimensions driven by boundary point distribution. (a) Surface grid. (b) Cuts through the field. (c) Solution of inviscid flow.

1.4.2.4 Point Creation Controlled by Point and Line Sources

In somewhat of an analogous way to point sources used as control functions with elliptic PDEs, it is possible to define line and point sources to provide grid control for unstructured meshes. Local grid point spacing can be defined as

$$dp_i = \min\left\{A_j e^{B_j\left|\mathbf{R}_j - \mathbf{r}_i\right|}\right\}, j = 1, \ldots \text{number of sources} \tag{1.11}$$

where A_j, B_j, and $\mathbf{R}_j$ (j = 1, number of sources) are user-controlled amplification and decay parameters and the position of sources, respectively. Grid point creation is then performed as outlined, but in Step 4b, the appropriate point distribution function at the centroid is determined by Eq. 1.10. In practice, the substitution of Eq. 1.10 for Step 4b is trivial.

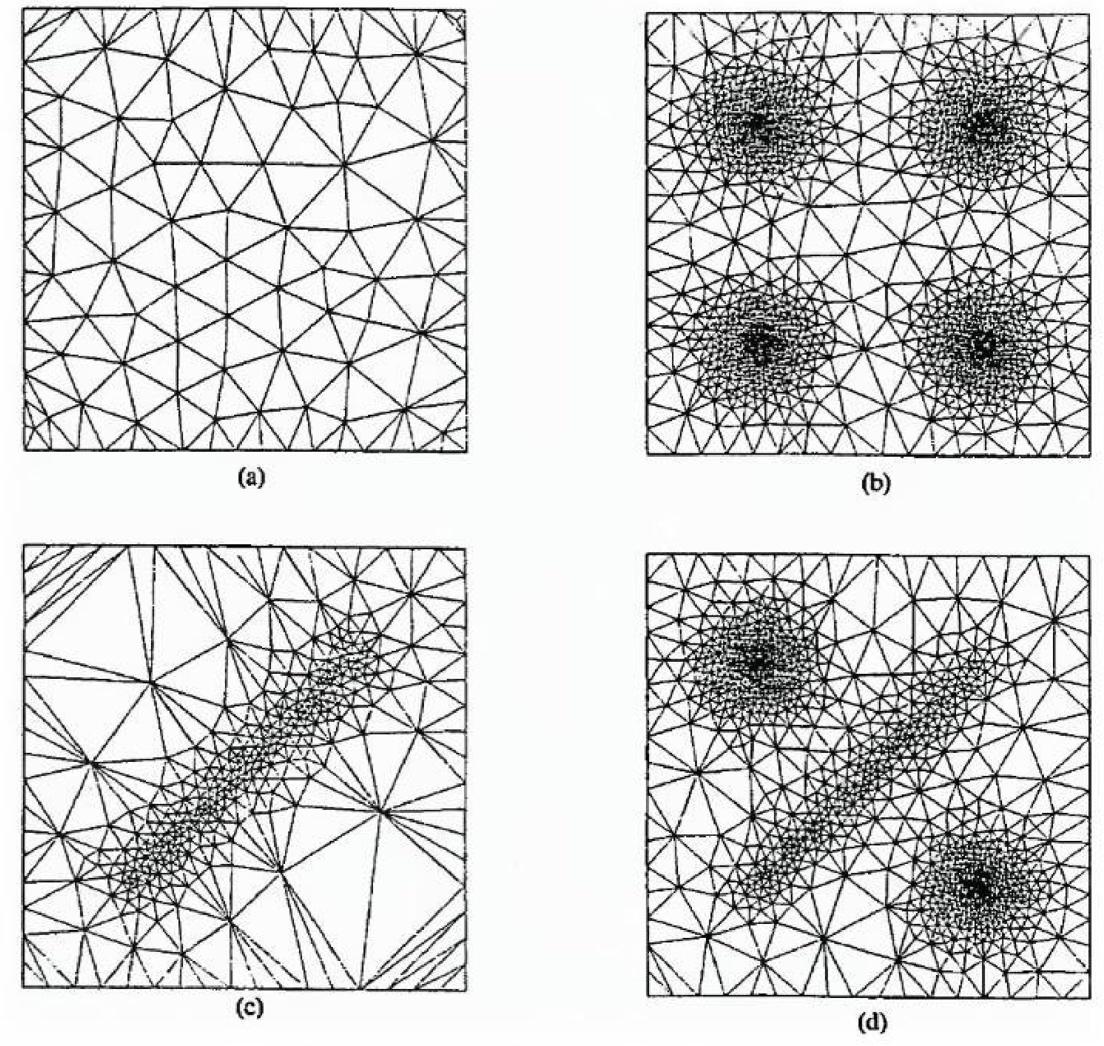

FIGURE 1.12 Point density controlled through point and line sources.

Examples of the use of point sources are given in Figure 1.12a. Figure 1.12a shows the mesh controlled through the boundary point distribution, while in Figure 1.12b a point source has been specified. It is clear that grid spacing is controlled by the source position and associated parameters. Line sources can also be introduced. For simplicity, line sources are treated as a series of point sources. In this way Eq. 1.10 is also applicable. An example of grid control by a line source is given in Figure 1.12c.

Combinations of line and point sources can also be used, such as the example shown in Figure 1.12d. It should be noted that the user-specified information to implement the sources is minimal.

It is clear that the sources provide a mechanism for clustering points. To ensure an adequate point spacing in regions not influenced by the sources, it is appropriate to use a combination of point spacing derived from the sources and the boundary point distribution. In practice, this can be implemented by defining the local length parameter to be

$$dp_i = \min\{A_j e^{B_j|\mathbf{R}_j - \mathbf{r}_i|}, dp_{\text{boundary}}\}, j = 1, \ldots \text{number of sources}$$

where dp_{boundary} is the point spacing parameter defined from the boundary point distribution and derived using Algorithm II. An example of the use of this approach is shown in Figure 1.13, for ocean modeling of the North Atlantic. An unstructured grid generated from the boundary point spacing is shown in Figure 1.13a, while in Figure 1.13b, a single line source has been appropriately positioned to ensure a higher point resolution to capture the Gulf Stream.

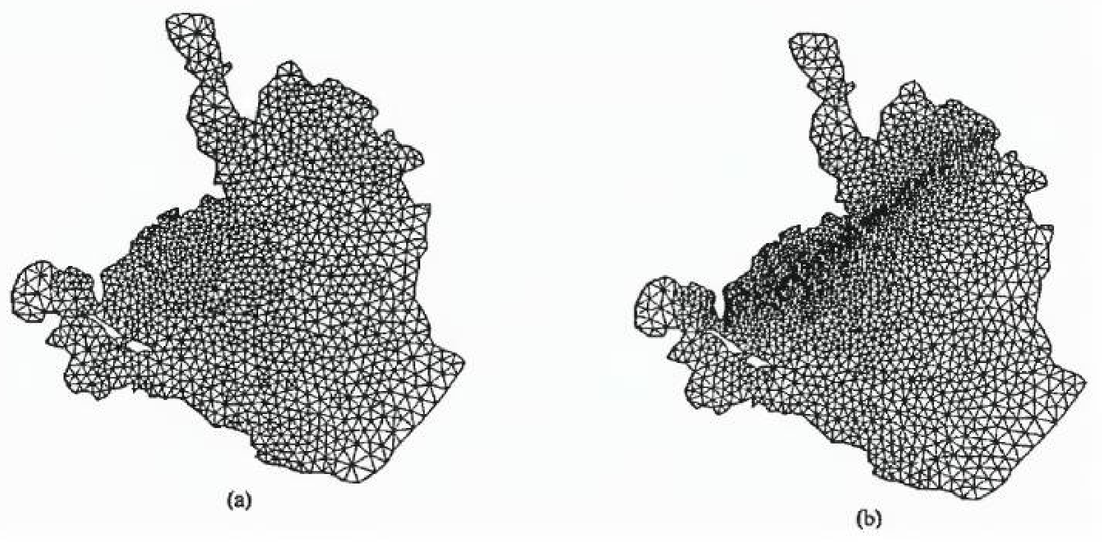

FIGURE 1.13 Unstructured grid for the North Atlantic. (a) Points created from boundary point spacing. (b) Points created from boundary point spacing and line source.

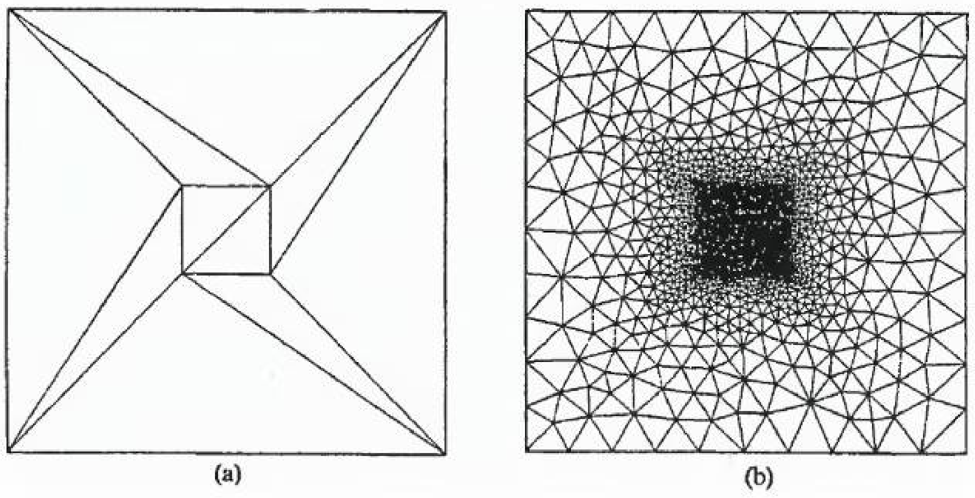

FIGURE 1.14 Grid control using a background mesh.

1.4.2.5 Point Creation Controlled by a Background Mesh

An alternative way to control grid point spacing is by defining a background mesh on which the local point spacing is defined. To implement this approach in the proposed algorithm the point spacing in Step 4b is derived from the background mesh.

As an example, consider the rectangular domain in Figure 1.14a. A background mesh is defined of 10 elements at whose nodes a point spacing is specified. If small spacing is defined at interior nodes and larger spacing on the boundary, then the use of the automatic point creation algorithm results in the mesh shown in Figure 1.14b. Modification to the topology of the background mesh or the point spacing function at nodes allows complete control of the unstructured grid density. In practice, the background mesh is a previous mesh used for analysis in which a measure of the analysis is converted, by an appropriate transformation, to spacings in the physical domain. An example of this is given in the section on adaptation techniques.

The above ideas provide some examples of the way unstructured grids can be generated. There is considerable flexibility to introduce points where required. Algorithms to construct such grids are not over complicated to program and are fast and efficient provided, as already emphasized, appropriate care is taken with respect to data structures.

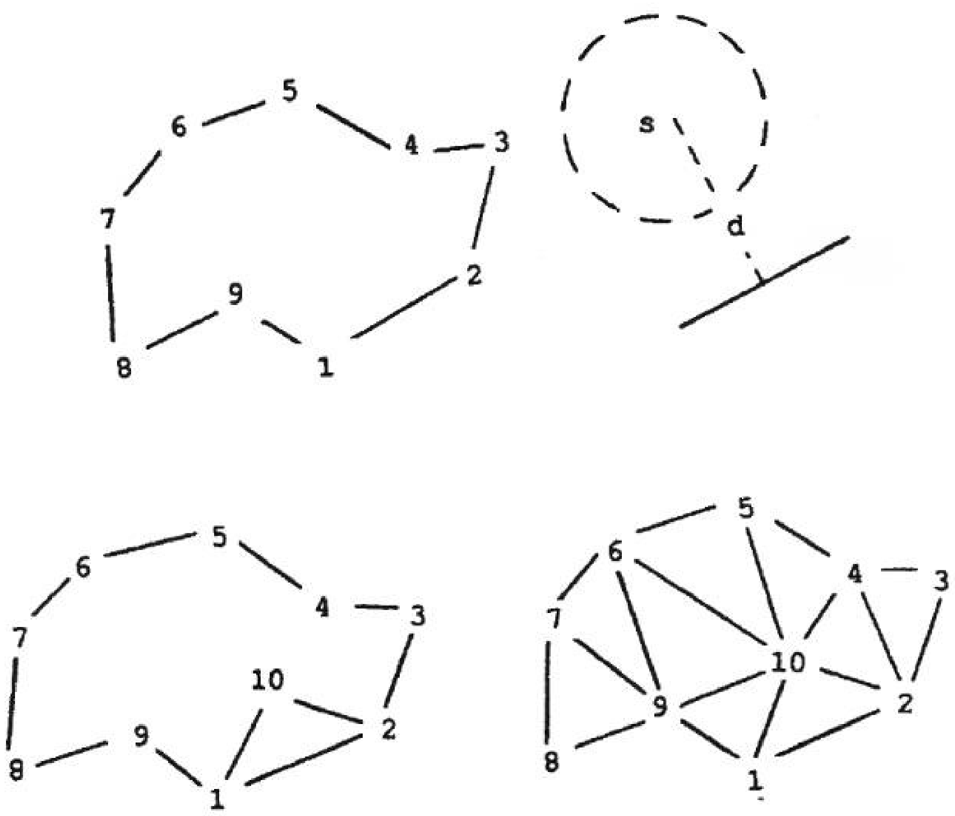

FIGURE 1.15 Test case example indicating the steps in the advancing front technique.

1.4.3 Other Unstructured Grid Techniques

1.4.3.1 Advancing Front Methods

Another class of unstructured grid generators is based upon the idea of an advancing front (Chapter 17). Such methods construct a mesh of a domain from its boundary information. The method is applicable in both two and three dimensions, where triangles and tetrahedra are generated, respectively.

The basic ideas of the method are best illustrated in two dimensions. Consider a region bounded by points (P_i) and edges (P_mP_n). The edges are called the front. A test case example is shown in Figure 1.15. To construct a grid in the domain, perform the following:

1. Choose an edge on the front, say P_1 P_2.
2. Construct the perpendicular bisector of P_1 P_2 and create a point s a distance d into the domain.
3. Create a circle, center s, of radius r.
4. Determine any points which lie within this circle (a_i) and order them in distance from the center s.
5. Form triangles (P_1P_2 a_i) and accept the first triangle that satisfies the following conditions:
   a. Edges (P_1a_i) and (P_2a_i) do not intersect any other edges.
   b. Triangle (P_1P_2 a_i) satisfies an appropriate quality indicator. (Such indicators are based upon regularity of interior angles, etc.)
6. If a_i is a new point, add to the list of points. Add any new edges to the front and delete the old edge (P_1P_2).

This procedure is repeated until the front is empty, i.e., there are not edges left in the front.

In the above algorithm, the grid point density can be controlled by the length d_1, i.e., the distance away from the mid-point of the current edge on the front. This length parameter can be obtained using a background mesh, or a distribution of line or point sources. The effects are the same as those indicated for the Delaunay algorithm.

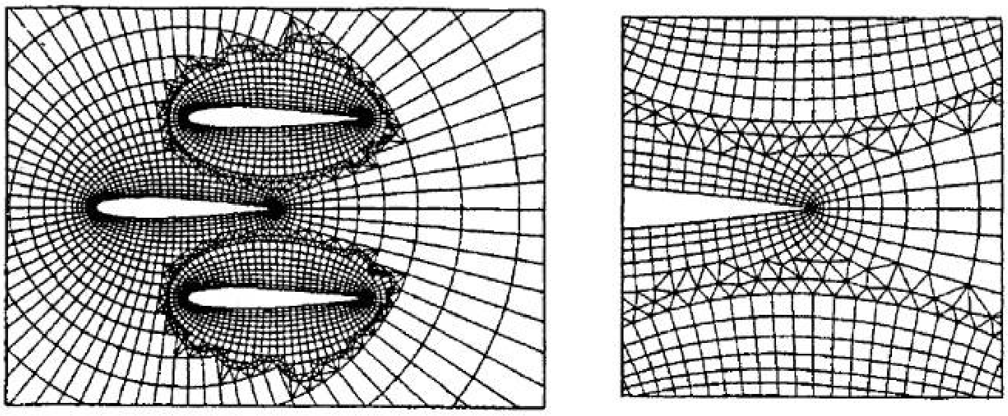

FIGURE 1.16 Hybrid grid for multiple airfoils.

In principle, this basic procedure holds for applications in three dimensions, where the front consists of a set of triangular faces which bound the domain. In practice, the implementation of this basic procedure requires an efficient data structure to ensure realistic computational times.

It is worth commenting that the advancing front technique can be used to also generate grids with elements aligned in given directions. This is achieved by introducing a directional parameter, in addition to a length parameter d. In this way, instead of constructing a line perpendicular to the edge on the front, a line inclined in the specified direction can be generated. Again the directional parameters can be specified in the background mesh.

1.4.3.2 Quadtree and Octree

Grid generation based upon quadtree (2D) and octree (3D) have recently been introduced (Chapter 15). Such methods begin with a point definition of the boundaries. Superimposed over the domain is a sparse regular grid that is subdivided so that at boundaries the cell size is consistent with the boundary point spacing. The data points and cells are contained in the quadtree or octree data structure. The grid is made to be boundary conforming by appropriate cutting and reconnecting to form triangles and tetrahedra.

1.4.3.3 Hybrid Grids

To achieve an optimum compromise between regularity and flexibility, it is possible to combine grid types in the form of hybrid or structured–unstructured grids (Chapter 23). Figure 1.16 shows three airfoils where each is locally discretized using a structured grid that is connected using an unstructured grid. The idea can be also applied in three dimensions. Figure 1.17 shows a surface grid for a fuselage, wing, pylon, and nacelle, where the pylon and nacelle components have been incorporated into a structured grid using locally unstructured grid. Such grid generation techniques require analysis modules that can utilize mixed element types.

1.4.4 Unstructured Grid Generation on Surfaces

In most engineering applications it is not possible to define a surface in a closed form mathematical expression. In general, a given surface is defined as a discrete set of points, that map to a regular array. In such cases, it is possible to define the surface in terms of two parametric coordinates (u, v) where each pair maps to a point on the surface. With this description of a surface it is possible to construct grids on surfaces by utilizing 2D techniques applied in the parametric coordinates (Chapter 19). The final grid on the surface is then obtained by mapping to the physical space. The point connections remain fixed through the transformation.

Complicated surfaces can be defined by using more than one set of rectangular point sets. Figure 1.18 shows a grid in parametric coordinates, which when mapped to physical space becomes an unstructured grid on the surface of a wing. The grid at the tip of the wing is treated separately from the grid on upper and lower surfaces of the wing.

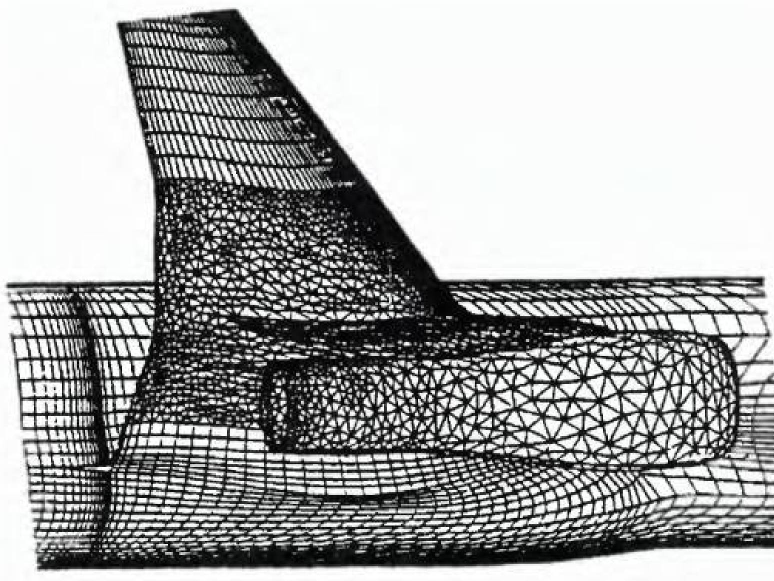

FIGURE 1.17 Hybrid grid for a wing, fuselage, pylon, and nacelle.

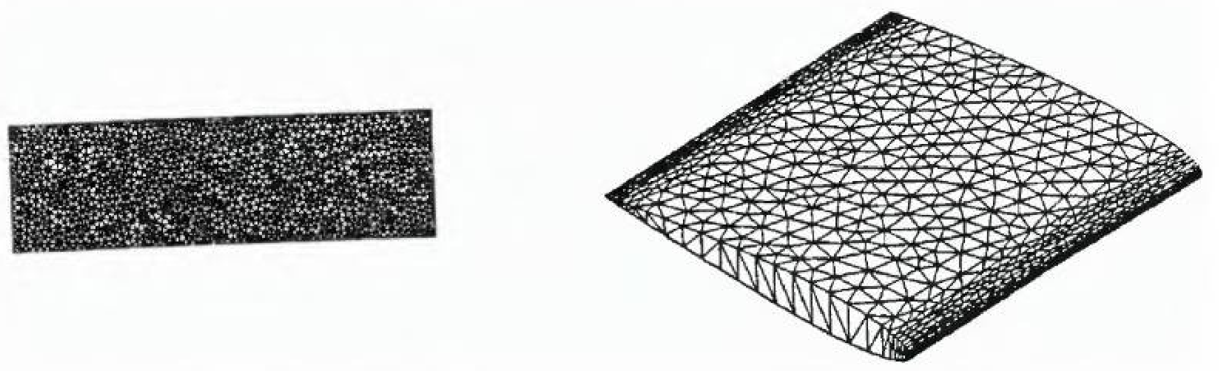

FIGURE 1.18 Grid in parametric space converted to unstructured surface grid in physical space.

1.4.5 Adaptation on Unstructured Grids

The basic principles of adaptation have been given above in Section 1.3.6. Here comments will be restricted to grid adaptation on unstructured grids (Chapter 35).

1.4.5.1 Point Enrichment

Local point enrichment can be achieved on any grid type. It is an effective way to ensure greater resolution of the domain in critical regions. It is most naturally employed on unstructured grids where, on the addition of a point and the subsequent connections, the data format of the grid does not change. Hence, solution modules do not require any modifications. This is not true for structured grids where the addition of a point breaks the (i, j, k) data format and hanging or nonconforming nodes are created.

Points are added to the domain in regions where some measure of error or solution activity is large. Dependent upon the problem, a suitable indicator f is chosen. From a computation on a grid, the indicator f is known at all points. It is then possible to construct operators that indicate where additional points should be added. For example, point enrichment could be driven by detecting changes in f along edges. If

$$\frac{|\phi_a - \phi_b|}{|\phi_i - \phi_j|_{\max}} > \text{tolerance} \tag{1.12}$$

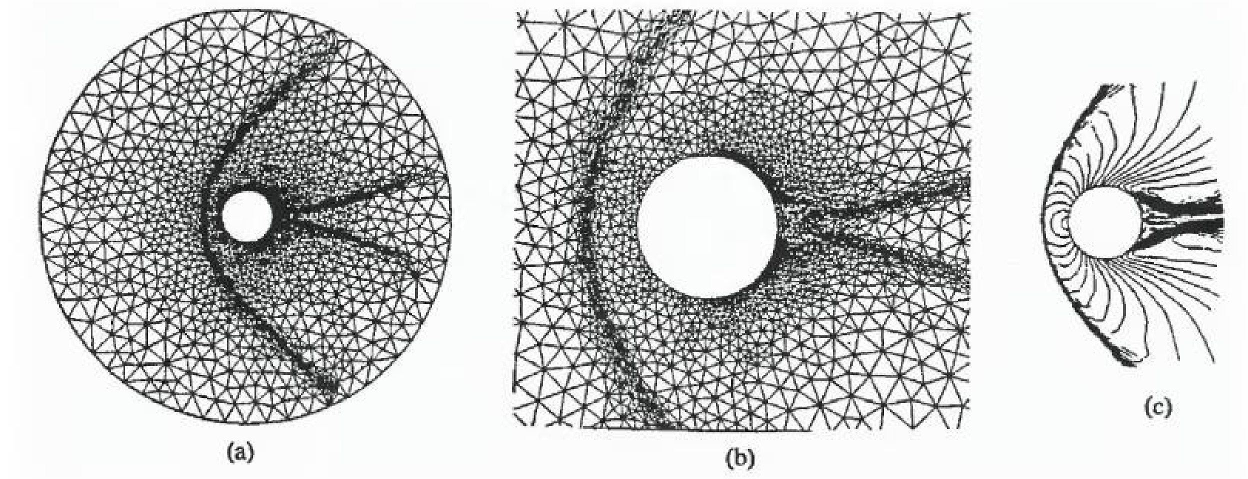

FIGURE 1.19 Mach 3 flow around a cylinder showing point enrichment. Flowfield contours of density also shown.

then add a point along the edge *a* to *b*. Connections to the new node can then be made. Similar expressions can be constructed for triangles, tetrahedra, etc. For some class of problems, more sophisticated error indicators can be used. These can be applied to give a solution which can have a prespecified bound on the errors. In some regions of the domain it may be possible to delete nodes. This process can be driven by criteria similar to the one for enrichment.

Examples of point enrichment and derefinement are given in Figure 1.19 and Figure 1.20a. The first example illustrates the use of point enrichment, driven by gradients of density, on an unstructured grid, for a simulation of Mach 3 flow around a cylinder. Contours of density for the flowfield are shown in Figure 1.19c. It is clear that points have been added where gradients in density are high. Figure 1.20a shows an enriched structured multiblock grid. As indicated earlier, once points are added to such a grid, the data format has to be modified. To provide flexibility for grid point enrichment on such grids, the data format has been converted to quadtree. In the example shown the point addition was driven by gradients in density. Contours of density are shown in Figure 1.20d, again confirming that the point enrichment has occurred in the relevant regions.

1.4.5.2 Node Movement

Node enrichment successfully enhances the resolution of an analysis. However, it can become computationally expensive and provides a diminishing return on successive enrichments. After ensuring that there is sufficient mesh point resolution, node movement can provide the required mechanism to achieve high resolution at a negligible cost. Many techniques have been explored to move points. One that is particularly simple, is applicable to all grid types, and is effective, is based on a weighted Laplacian formulation. A typical form is the following:

$$\mathbf{r}_0^{n+1} = \mathbf{r}_0^n + \omega \frac{\sum_{i=1}^{M} c_{io}\left(\mathbf{r}_i^n - \mathbf{r}_0^n\right)}{\sum_{i=1}^{M} c_{io}} \tag{1.13}$$

where $\mathbf{r} = (x, y)$, $\mathbf{r}_o^{n+1}$ is the position of node o at relaxation level $n + 1$, C_{io} is the adaptive weight function between nodes i and o, and ω is the relaxation parameter. The summation is taken over all edges connecting points o and i, where it is taken that there are M surrounding nodes. The weight C_{io} can be taken as a measure of activity, and a typical form is

$$C_{io} = \alpha_1 = \alpha_2 \left| \frac{\phi_i - \phi_o}{\phi_i + \phi_o} \right| i = 1, \ldots, M \tag{1.14}$$

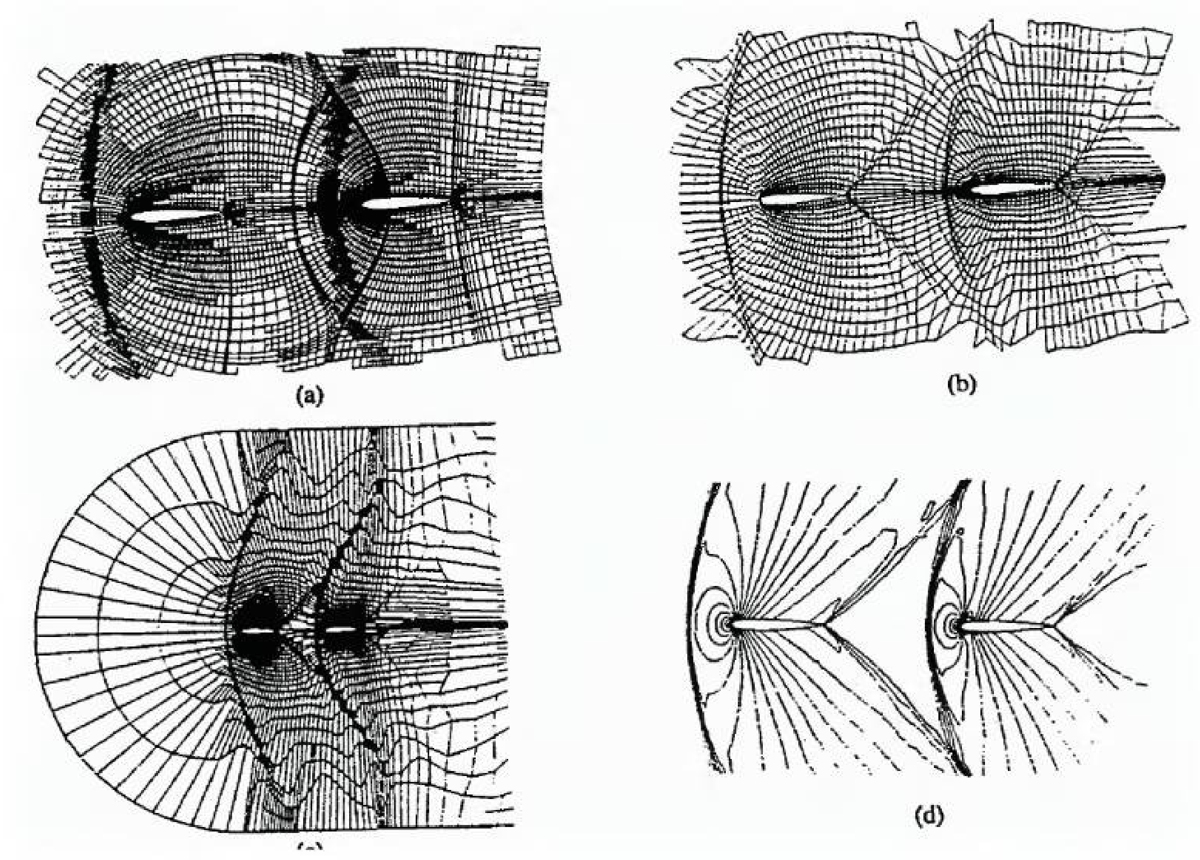

FIGURE 1.20 Adaption on a multiblock grid; (a) point enrichment, (b) node movement, (c) point enrichment, derefinement and node movement, (d) contours of density.

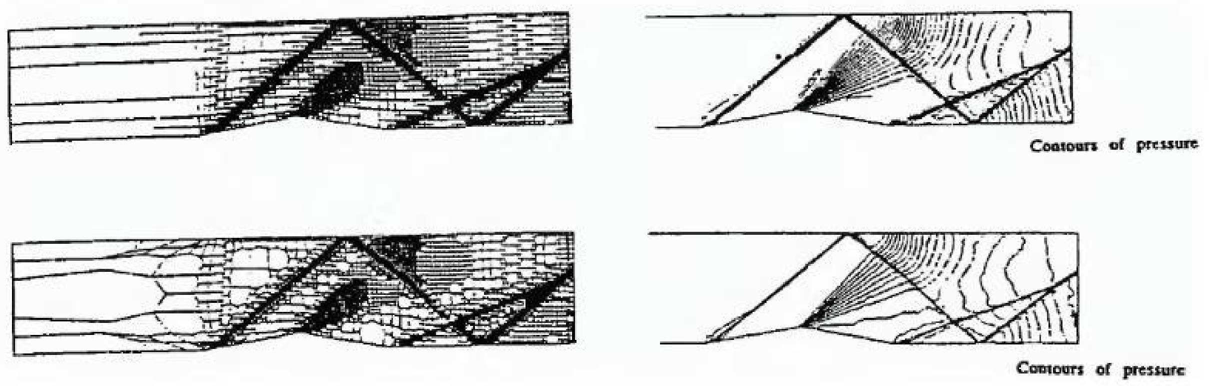

FIGURE 1.21 Mach 2 flow in a channel showing refined/derefined grid with flow contours of density.

where ϕ is the adaptive parameter, α_1 and α_2 are constants.

Figure 1.20b shows a multiblock structured grid that has been adapted using node movement driven by density gradients. As was the case for point enrichment, comparison with the contours of density confirms the point movement has been into regions with high gradients.

For extensive numerical studies it appears that the use of point enrichment, derefinement, and movement should be closely coupled. Sequences of these adaptive procedures give the optimum results, as judged by solution accuracy and computational efficiency. Examples of computations where these adaptive mechanisms have been cycled are given in Figure 1.20c and Figure 1.21. In Figure 1.21, the contours of density on a refined and derefined grid can be compared with those obtained after the grid points have been moved. A clear improvement in shock capturing is evident.

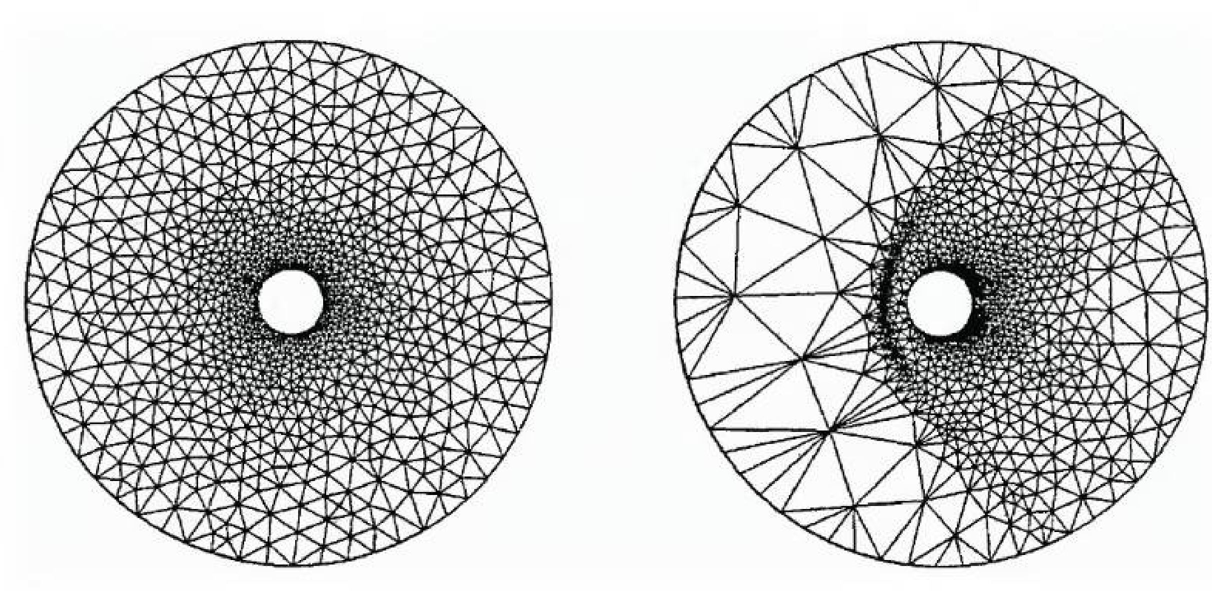
FIGURE 1.22 Mach 3 flow over a cylinder showing remeshing.

1.4.5.3 Remeshing

The concept of grid point generation driven by the spacing specified on a background mesh can be utilized for adaptation. In this case, the result of an analysis can be used to construct spacings, which are then assigned to the mesh, which in turn are used in the background mesh. There are several ways of performing the transformation between results and local length scales, but typically they take the form of

$$dp_i^{\text{new}} = dp_i^{\text{old}} \cdot \frac{\phi_{\text{average}}}{\phi_i} \tag{1.15}$$

where dp_i^{new} and dp_i^{old} are the new and old point distributions, ϕ_i is the adaptive indicator, ϕ_{average} is the indicator averaged throughout the domain. An example of remeshing, where the initial mesh is used as a background mesh and pressure was used as the adaptive indicator, is given in Figure 1.22. It is seen that local point clustering has occurred in the vicinity of the bow shock wave, but not in the region rear of the cylinder, which might be expected from the contours shown in Figure 1.19c. This illustrates a key area in grid adaption, in that, although there are steep gradients in density rear of the cylinder, there are no such gradients in pressure. Hence, the adaption process for remeshing, driven in this example by pressure, does not detect the features in density rear of the cylinder. As yet, for flow problems, there is no universal indicator and hence the selection of the parameter has to be made on a case-by-case basis.

1.4.6 Summary

Unstructured grids provide considerable flexibility for the discretization of complex geometries and grid adaptation. In these sections a brief outline has been given on such techniques. Particular details have been given on the use of the Delaunay triangulation. It should be emphasized that, although the majority of applications have been drawn from aerospace engineering, the ideas and principles discussed are equally applicable to other fields.

References

1. **Arcilla, A. S., Hauser, J., Eiseman, P. R., and Thompson, J. F.,** (Eds.), *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Proceedings of the 3rd International Conference,* North Holland, 1991.
2. **Bowyer, A.,** Computing Dirichlet Tessellations, *The Computer Journal,* Vol. 24, pp. 162–166, 1981.

3. **Carey, G. F.**, *Computational Grids: Generation, Adaptation, and Solution Strategies*, Taylor & Francis, 1997.
4. **Castillo, J. E.**, (Ed.), *Mathematical Aspects of Numerical Grid Generation*, SIAM Press, 1991.
5. **Choo, Y.**, (Ed.), *Proceedings of the Surface Modeling, Grid Generation and Related Issues in Computational Fluid Dynamics Workshop*, NASA Conference Publication 3291, NASA Lewis Research Center, Cleveland, OH, May 1995, p. 359.
6. **Delaunay, B.**, Sur la sphere vide, Bulletin of Academic Science URSS, Class. Science National, 1934, pp. 793–800.
7. **Dirichlet, G. L.**, Uber die Reduction der positiven Quadratischen formen mit drei Underestimmten Ganzen Zahlen, *Z. Reine Angew. Mathematics*, Vol. 40, No. 3, pp. 209–227, 1850.
8. **Eiseman, P. R.**, Grid generation for fluid mechanics computations, *Annual Review of Fluid Mechanics*, Vol. 17, 1985.
9. **Soni, B. K., Thompson, J.F., Eiseman, P.R., and Hauser, J.**, (Eds.), *Numerical Grid Generation in Computational Field Simulation. Proceedings of the 5th International Conference*, MSU Publisher, Mississippi State, MS, U.S., April 1996.
10. **Eiseman, P. R., Hauser, J., Thompson, J. F., and Weatherill, N. P.**, (Eds.), *Numerical Grid Generation in Computational Field Simulation and Related Fields, Proceedings of the 4th International Grid Conference*, Pineridge Press, Swansea Wales, U.K., 1994.
11. **George, P. L.**, Automatic Mesh Generation, Wiley Publications, 1991.
12. **Godunov, S. K. and Propokov, G. P.**, On the computation of conformal transformations and the construction of difference meshes, *Zh. Vychisl. Mat. Mat. Fiz.*, Vol. 7, p. 209, 1967.
13. **Gordon, W. J. and Thiel, L. C.**, Transfinite mappings and their application to grid generation, *Numerical Grid Generation*, Thompson, J. F., (Ed.), North Holland, 1982.
14. **Hauser, J. and Taylor, C.**, (Ed.), *Numerical Grid Generation in Computational Fluid Dynamics, Proceedings of the 1st International Conference*, Pineridge Press, 1986.
15. **Kim, J. K. and Thompson, J. F.**, Three-dimensional solution-adaptive grid generation on a composite block configuration, *AIAA Journal*, Vol. 28, p. 420, 1990.
16. **Knupp, P. and Steinberg, S.**, *Fundamentals of Grid Generation*, CRC Press, Boca Raton, FL, 1993.
17. **Miki, K. and Takagi, T.**, A domain decomposition and overlapping method for the generation of three-dimensional boundary-fitted coordinate systems, *Journal of Computational Physics*, Vol. 53, p. 319, 1984.
18. **Rubbert, P. E. and Lee, K. D.**, Patched coordinate systems, *Numerical Grid Generation*, Thompson, J.F., (Ed.), North-Holland, 1982.
19. **Sengupta, S., Hauser, J., Eiseman, P. R., and Thompson, J. F.**, (Eds.), *Numerical Grid Generation in Computational Fluid Dynamics 1988, Proceedings of the 2nd International Conference,* Pineridge Press, 1988.
20. **Smith, R. E.**, (Ed.), *Numerical Grid Generation Techniques*, NASA Conference Publication 2166, NASA Langley Research Center, 1980.
21. **Smith, R. E.**, (Ed.), *Proceedings of the Software Systems for Surface Modeling and Grid Generation Workshop*, NASA Conference Publication 3143, NASA Langley Research Center, Hampton, VA, 1992, p. 161.
22. **Sorenson, R. L.**, The 3DGRAPE book: theory, users' manual, examples, NASA TM-10224, 1989.
23. **Thacker**, *Int. J. Numer. Meth. Eng.*, 15, p. 1335, 1980.
24. **Thompson, J. F.**, (Ed.), *Numerical Grid Generation*, North Holland, 1982. (Also published as Vol. 10 and 11 of *Applied Mathematics and Computation*, 1982.)
25. **Thompson, J. F.**, Grid generation techniques in computational fluid dynamics, *AIAA Journal*, Vol. 22, p. 1505, 1984.
26. **Thompson, J. F., Warsi, Z. U. A., and Mastin, C. W.**, *Numerical Grid Generation: Foundations and Applications*. North Holland, 1985.
27. **Thompson, J. F.**, A survey of dynamically adaptive grids in the numerical solution of partial differential equations, *Applied Numerical Mathematics*, Vol. 1, p. 3, 1985.

28. **Thompson, J. F.**, A general three-dimensional elliptic grid generation system on a composite block structure, *Computer Methods in Applied Mechanics and Engineering*, Vol. 64, p 377, 1987.
29. **Thompson, J. F.**, A composite grid generation code for 3D regions —the EAGLE code, *AIAA Journal*, Vol. 26, p. 915, 1988,.
30. **Thompson, J. F.**, A reflection on grid generation in the '90s: trends, needs and influences, *Numerical Grid Generation in Computational Field Simulation.* Soni, B. K., Thompson, J. F., Hauser, J., Eiseman, P. R., (Eds.), *Proceedings of the 5th International Conference*, MSU Publisher, Mississippi State, MS, U.S., April 1996, p. 1029.
31. **Thompson, J. F., Warsi, Z. U. A., Mastin, C. W.**, Boundary-fitted coordinate systems for numerical solution of partial differential equations — a review, *J. of Computational Physics*, Vol. 47, p. 1, 1982.
32. **Tu, Y. and Thompson, J. F.**, Three-dimensional solution-adaptive grid generation on composite configurations, *AIAA Journal*, Vol. 29, p. 2025, 1991.
33. **Voronoï, G.**, Nouvelles applications des parametres continus a la theorie des formes quadratiques. recherches sur les parallelloedres primitifs, *Journal Reine Angew. Mathematics*, Vol. 134, 1908.
34. **Warsi, Z. U. A. and Thompson, J. F.**, Application of variational methods in the fixed and adaptive grid generation, *Computers in Mathematics with Applications*, Vol. 19, p. 31–41, 1990.
35. **Weatherill, N. P. and Forsey, C. R.**, Grid generation and flow calculations for complex aircraft geometries using a multi-block scheme, AIAA-84-1665, *AIAA 17th Fluid Dynamics, Plasma Dynamics, and Laser Conference*, Snowmass, CO, 1984.
36. **Winslow, A. M.**, Equipotential zoning of two-dimensional meshes, *J. of Computational Physics*, Vol. 1, p. 149, 1966.
37. **Winslow, A. M.**, Numerical solution of the quasilinear Poisson equation in a nonuniform triangle mesh, *Journal of Computational Physics*, Vol. 135, pp. 128–138, 1997; reprinted from November 1966, Vol. 1, Number 2, pp. 149–172.