Preface: An Elementary Introduction

Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill

This first section is an elementary introduction provided for those with little familiarity with grid (mesh) generation in order to establish a base from which the technical development of the chapters in this handbook can proceed. (The terms *grid* and *mesh* are used interchangeably throughout with identical meaning.) The intent is not to introduce numerical solution procedures, but rather to introduce the idea of using numerically generated grid (mesh) systems as a foundation of such solutions.

P-1 Discretizations

The numerical solution of partial differential equations (PDEs) requires first the discretization of the equations to replace the continuous differential equations with a system of simultaneous algebraic difference equations. There are several bifurcations on the way to the construction of the solution process, the first of which concerns whether to represent the differential equations at discrete points or over discrete cells. The discretization is accomplished by covering the solution field with discrete points that can, of course, be connected in various manners to form a network of discrete cells. The choice lies in whether to represent the differential equations at the discrete points or over the discrete cells.

P-1.1 Point Discretization

In the former case (called *finite difference*), the derivatives in the PDEs are represented at the points by algebraic difference expressions obtained by performing Taylor series expansions of the solution variables at several neighbors of the point of evaluation. This amounts to taking the solution to be represented by polynomials between the points. This can be unrealistic if the solution varies too strongly between the points. One remedy is, of course, to use more points so that the spacing between points is reduced. This, however, can be expensive, since there will then be more points at which the equations must be evaluated.

This is exacerbated if the points are equally spaced and strong variations in the solution occur over scattered regions of the field, since numerous points will be wasted in regions of small variation. An alternative, of course, is to make the points unequally spaced.

P-1.2 Cell Discretization

The other possibility of this first bifurcation is to return the PDEs to their more fundamental integral form and then to represent the integrals over discrete cells. Here there is yet another bifurcation — whether to represent the solution variables over the cell in terms of selected functions and then to integrate

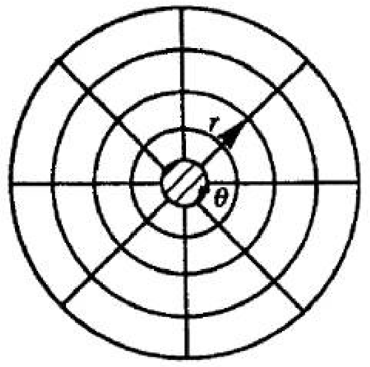


FIGURE 1

these functions analytically over the volume (*finite element*), or to balance the fluxes through the cell sides (*finite volume*).

The finite element approach itself comes in two basic forms: the *variational,* where the PDEs are replaced by a more fundamental integral variational principle (from which they arise through the calculus of variations), or the *weighted residual* (Galerkin) approach, in which the PDEs are multiplied by certain functions and then integrated over the cell.

In the finite volume approach, the solution variables are considered to be constant within a cell, and the fluxes through the cell sides (which separate discontinuous solution values) are best calculated with a procedure that represents the dissolution of such a discontinuity during the time step (Riemann solver).

P-2 Curvilinear (Structured) Grids

The finite difference approach, using the discrete points, is associated historically with rectangular Cartesian grids, since such a regular lattice structure provides easy identification of neighboring points to be used in the representation of derivatives, while the finite element approach has always been, by the nature of its construction on discrete cells of general shape, considered well suited for irregular regions, since a network of such cells can be made to fill any arbitrarily shaped region and each cell is an entity unto itself, the representation being on a cell, not across cells.

P-2.1 Boundary-Fitted Grids

The finite difference method is not, however, limited to rectangular grids and has long been applied on other readily available analytical coordinate systems (cylindrical, spherical, elliptical, etc.) that still form a regular lattice. albeit curvilinear, that allows easy identification of neighboring points. These special curvilinear coordinate systems are all orthogonal, as are the rectangular Cartesian systems, and they also can exactly cover special regions (e.g., cylindrical coordinates covering the annular region between two concentric circles) in the same way that a Cartesian grid fills a rectangular region. The *cardinal feature* in each case is that some coordinate line is coincident with each portion of the boundary.

In fact, these curvilinear systems can be considered to be *logically rectangular,* and from a programming standpoint are no different, conceptually, from the Cartesian system. Thus, for example, the cylindrical grid in Figure 1, where the radial coordinate r varies from r_1 on the inner boundary to r_2 on the outer and the azimuthal coordinate θ varies from 0 to 2π, can be diagrammed logically as shown in Figure 2.

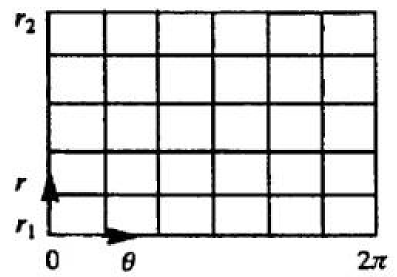


FIGURE 2

The continuity of the azimuthal coordinate can be represented by defining extra "phantom" columns to the left of 0 and to the right of 2π and setting values on each phantom column equal to those on the corresponding "real" columns inside of 2π and 0, respectively. This latter, logically rectangular, view of the cylindrical grid is the one used in programming anyway, and without being told of the cylindrical configuration, a programmer would not realize any difference here from programming in Cartesian coordinates — there would simply be a different set of equations to be programmed on what is still a logically rectangular grid, e.g., the Laplacian on a Cartesian grid (with $\xi = x$ and $\eta = y$),

$$\nabla^2 f = f_{\xi\xi} + f_{\eta\eta}$$

becomes (with $\xi = \theta$ and $\eta = r$)

$$\nabla^2 f = \frac{f_{\xi\xi}}{\eta^2} + f_{\eta\eta} + \frac{f_\eta}{\eta}$$

on a cylindrical grid. The key point here is that in the logical (i.e., programming) sense there is really no essential difference between Cartesian grids and the cylindrical systems: both can be programmed as nested loops; the equations simply are different.

Another key point is that the cylindrical grid *fits the boundary* of a cylindrical region just as the Cartesian grid fits the boundary of a rectangular region. This allows boundary conditions to be represented in the same manner in each case also (see Figure 3). By contrast, the use of a Cartesian grid on a cylindrical region requires a stair-stepped boundary and necessitates interpolation in the application of boundary conditions (Figure 4) — the proverbial square peg in a round hole.

P-2.2 Block Structure (The Sponge Analogy)

The best way to visualize the correspondence of a curvilinear grid in the physical field with a logically rectangular grid in the computational field is through the *sponge analogy.* Consider a rectangular sponge within which an equally spaced Cartesian grid has been drawn. Now wrap the sponge around a circular cylinder and connect the two ends of the sponge together. Clearly the original Cartesian grid in the sponge now has become a curvilinear grid fitted to the cylinder. But the rectangular logical form of the grid lattice is still preserved, and a programmer could still operate in the logically underformed sponge in constructing the loop and the difference expressions, simply having been given different equations to program. The correspondence of "phantom" points just outside one of the connected faces of the sponge

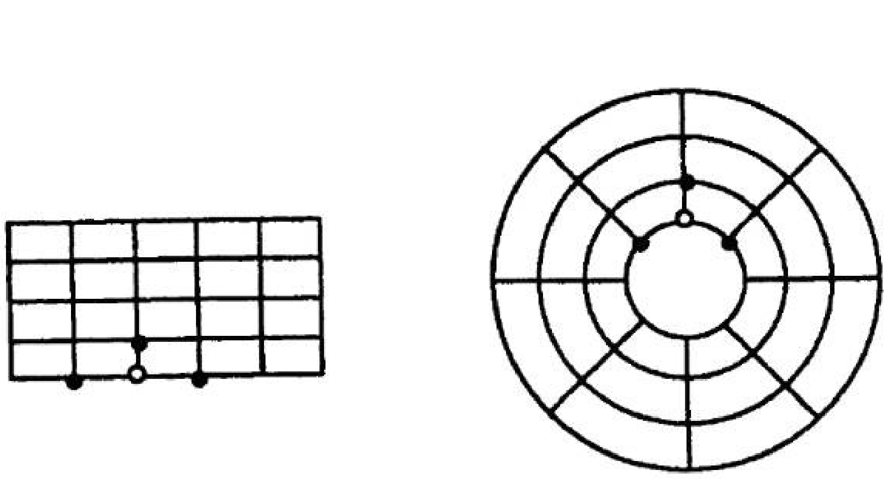


FIGURE 3

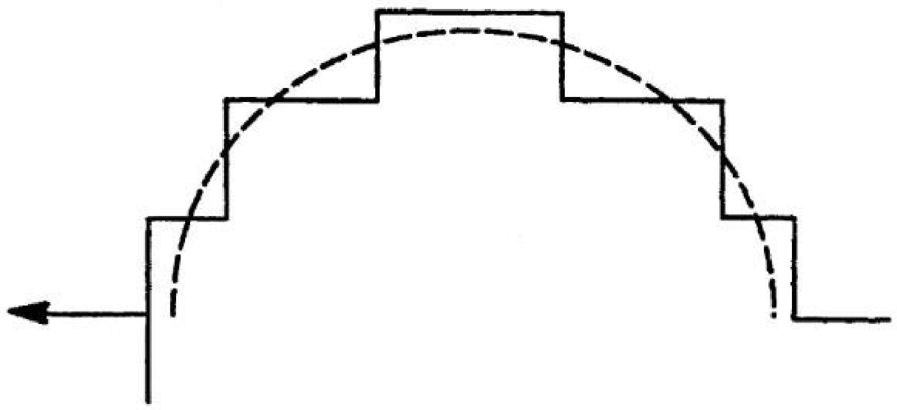


FIGURE 4

with "real" points just inside the face to which it is connected is clear — this is simply the correspondence of 370° with 10° in a cylindrical system.

Such a sponge could just as well be around a cylinder of noncircular cross section, regardless of the cross-sectional shape. To carry the analogy further, the sponge could, in principle, be wrapped around a body of any shape, or could be expanded and compressed to fill any region (e.g., expanding to fill a sphere), again producing a curvilinear grid filling the region and having the same correspondence to a logically rectangular grid (Figure 5). The programmer need not know, in fact, what has been done to the sponge. It is also clear from this analogy that the sponge could deform in time; i.e., the curvilinear grid could move in physical space, while the logically rectangular grid could still be considered fixed in computational space (image the sponge filling a beating heart). Again, the programmer need not be told that the boundaries are moving, but simply again be given a different set of equations that will include a transformation of the time derivatives as well.

It is not hard to see, however, that for some boundary shapes the sponge may have to be so greatly deformed that the curvilinear grid will be so highly skewed and twisted that it is not usable in a numerical solution. The solution to this problem is to use not one, but rather a group of sponges to fill the physical field. Each sponge has its own logically rectangular grid that deforms to a curvilinear grid when the sponge is put in place in the field. Each sponge now abuts with adjacent sponges, and the correspondence

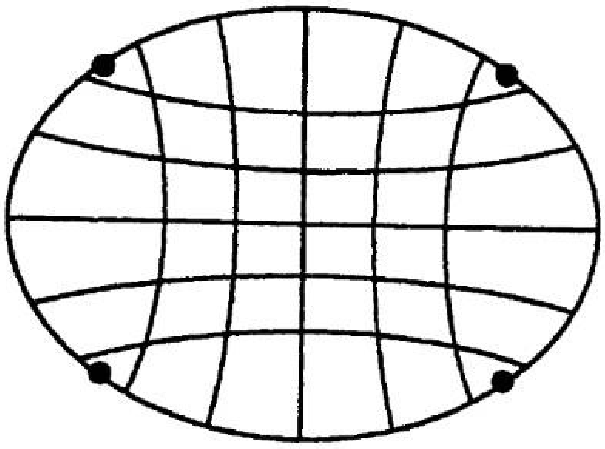


FIGURE 5

across an interface is analogous to that across the two connected faces of the single sponge in the cylindrical case above — here it is simply that the "phantom" points just outside one sponge correspond to "real" points just inside a different sponge.

Block-structured grid codes are based on this multiple-sponge analogy, with the physical field being filled with a group of grid blocks with correspondence of grid lines, and in fact complete continuity, across the interfaces between blocks. This approach has been carried to a high degree of application in the aerospace industry (cf. Chapter 13), with complete aircraft configurations being treated with a hundred or so blocks. Current grid generation systems seek to make the setup of this block structure both graphical and easy for the user. The ultimate goal is to automate the process (cf. Chapter 10).

2.3 Grid Generation Approaches

With these obvious advantages of specialized curvilinear coordinate systems fitted to the boundaries of cylindrical, spherical, elliptical, and other analytic regions, it has been natural to use grids based on these systems for finite difference solutions on such regions. In the late 1960s the visual analogy between potential solutions (electrostatic potential, potential flow, etc.) that are solutions of Laplace's equation, $\nabla^2\phi = 0$, and curvilinear grids led to the idea of generating grid lines in arbitrary regions as the solution of Laplace's equation. Thus, whereas potential flow is described in terms of a stream function ψ and a velocity potential ϕ that are orthogonal and satisfy $\nabla^2\psi = 0$, $\nabla^2\phi = 0$ (Figure 6), a curvilinear grid could be generated by solving the system $\nabla^2\xi = 0$, $\nabla^2\eta = 0$ with η a constant on the upper and lower boundaries in the above region, while ξ is constant on the left and right boundaries (Figure 7).

Here again, for purposes of programming, the grid forms a logically rectangular lattice (Figure 8). The problem of generating a curvilinear grid to fit any arbitrary region thus becomes a *boundary value problem* — the generation of interior values for the curvilinear coordinates from specified values on the boundary of the region (cf. Chapter 4). In order to set this up, note that we have for the boundary value problem the generation of interior values of the curvilinear coordinates ξ and η from specified constant values on opposing boundaries (Figure 9).

Clearly ξ and η must vary monotonically and over the same range over the boundaries on which they are not specified, else the grid would overlap on itself. Thus, on the lower and upper boundaries, ξ here must vary monotonically from ξ_1 on the left to ξ_2 on the right. Similarly, on the left and right boundaries,

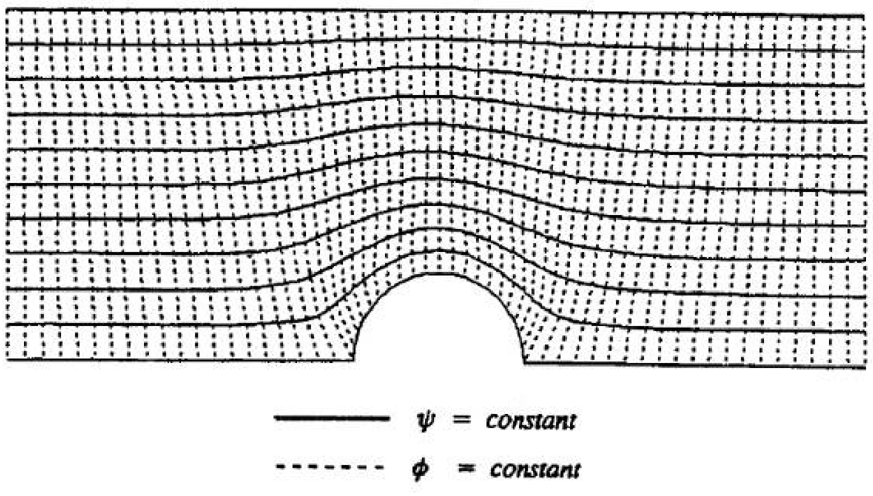


FIGURE 6

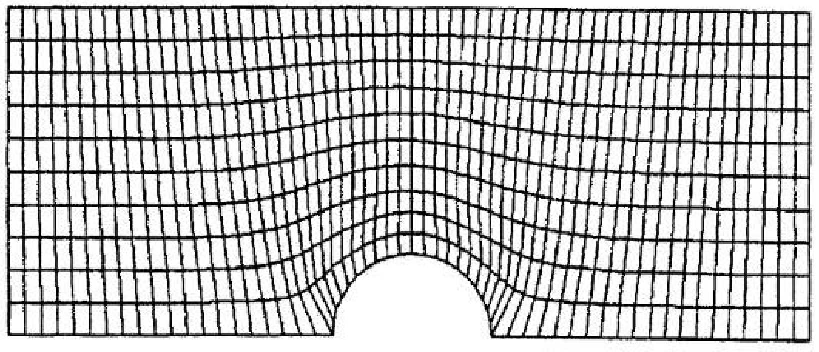


FIGURE 7

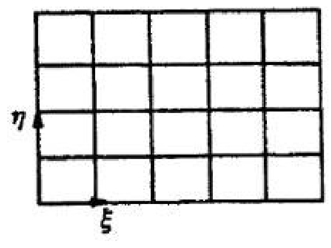


FIGURE 8

η must vary monotonically from η_1 at the bottom to η_2 at the top. The next question is what this variation should be. This is, in fact, up to the user. Ultimately, the discrete grid will be constructed by plotting lines of constant ξ and lines of constant η at equal intervals of each, with the size of the interval determined by the number of grid lines desired. Thus, if there are to be 10 grid lines running from side to side between the top and bottom of the region, 10 points would be selected on the left and right sides — with their locations being up to the user. Once these points are located, η can be said to assume, at the 10 points on each side, 10 values at equal intervals between its top and bottom values, η_1 and η_2. With this specification on the sides, the curvilinear coordinate η is thus specified on the entire boundary of

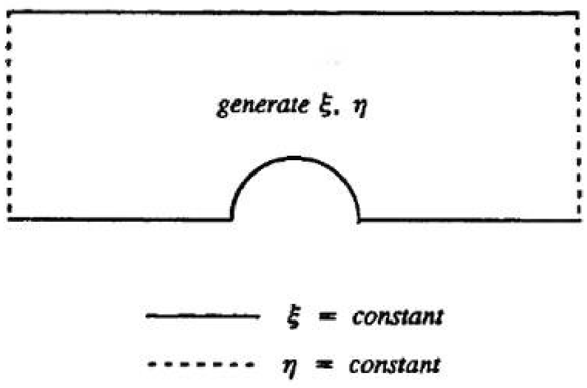


FIGURE 9

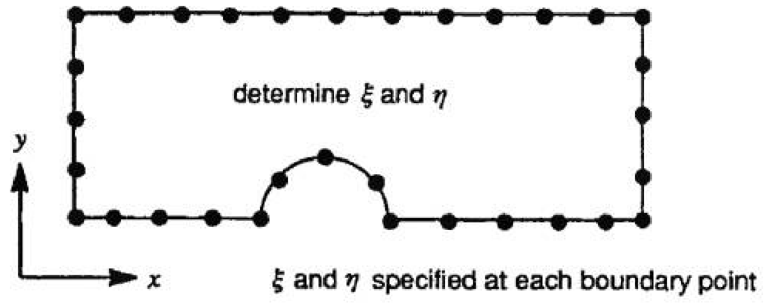

determine ξ and η
y
x
ξ and η specified at each boundary point

FIGURE 10

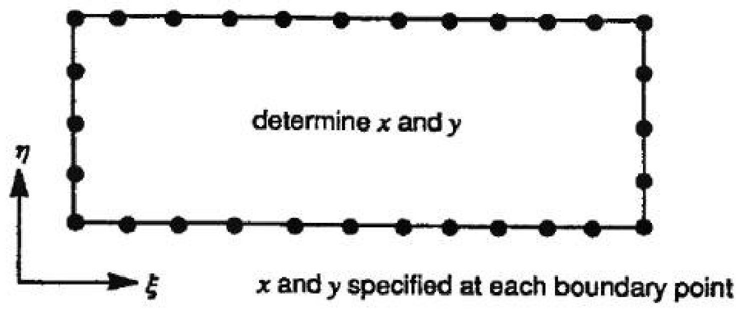


FIGURE 11

the region, and its interior values can be determined as a boundary value problem. A similar specification of ξ on the bottom and top boundaries by placing points on these boundaries sets up the determination of ξ in the interior from its boundary values. Now the problem can be considered a boundary value problem in the physical field for the curvilinear coordinates ξ and η (Figure 10) or can be considered a boundary value problem in the logical field for the Cartesian coordinates, x and y (Figure 11).

Note that the boundary points are by nature equally spaced on the boundary of the logical field regardless of the distribution on the boundaries of the physical field. Continuing the potential analogy, the curvilinear grid can be generated by solving the system $\nabla^2\xi = 0$, $\nabla^2\eta = 0$, in the first case, or by solving the transformation of these equations (transformation relations are covered in Chapter 2), in the

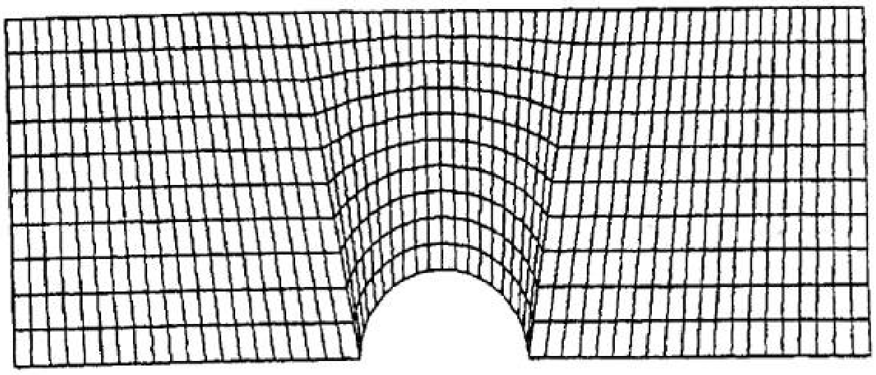


FIGURE 12

$$\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} = 0$$
$$\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} = 0$$
$$\alpha = x_\eta^2 + y_\eta^2$$
$$\gamma = x_\xi^2 + y_\xi^2$$
$$\beta = x_\xi x_\eta + y_\xi y_\eta$$

second case. Although the equation set is longer in the second case, the solution region is rectangular, and the differencing can be done on a uniformly spaced rectangular grid. This is, therefore, the preferred approach. Note that the placing of points in any desired distribution on the boundary of the physical region, where x and y are the independent variables, amounts to setting (x,y) values at equally spaced points on the rectangular boundary of the logical field, where ξ and η are the independent variables. This is the case regardless of the shape of the physical boundary.

This boundary value problem for the curvilinear grid can be generalized beyond the analogy with potential solutions, and in fact is in no way tied to the Laplace equation. The simplest approach is to generate the interior values by interpolation from the boundary values — a process called *algebraic* grid generation (cf. Chapter 3). There are several variants of this process. Thus for the region considered above, a grid could be generated by interpolating linearly between corresponding points on the top and bottom boundaries (Figure 12). Note that the point distributions on the side boundaries have no effect here. Alternatively, the interpolation could be between pairs of points on the side boundaries (Figure 13). The second case is, however, obviously unusable since the grid overlaps the boundary. Here the lack of influence from the points on the bottom boundary is disastrous.

Another alternative is *transfinite interpolation* in which the interpolation is done in one (either) direction as above, but then the resulting error on the two sides not involved is interpolated in the other direction and subtracted from the first result. This procedure includes effects from all of the boundary and consequently matches the point distribution that is set on the entire boundary. This is the preferred approach, and it provides a framework for placing any one-dimensional interpolation into a multiple-dimensional form. It is possible to include any type of interpolation, such as cubic, which gives orthogonality at the boundaries, in the transfinite interpolation format.

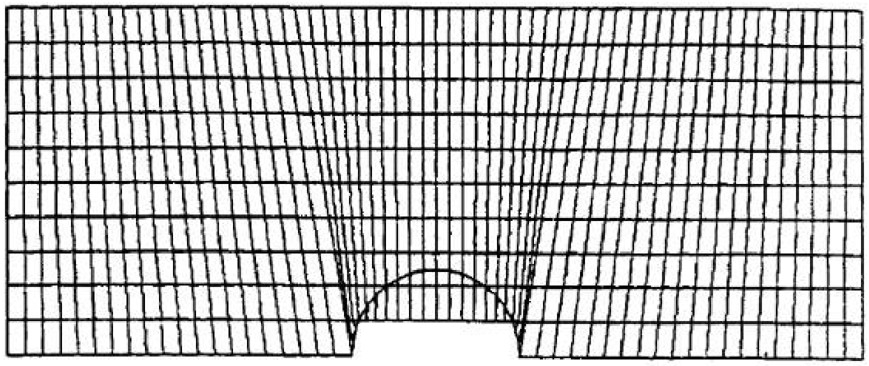


FIGURE 13

It is still possible in some cases for the grid to overlap the boundaries with transfinite interpolation, and there is no control over the skewness of the grid. This gives incentive to now return to the grids generated from solving the Laplace equation.

The Laplace equation is, by its very nature, a smoother, tending to average values at points with those at neighboring points. It can be shown from the calculus of variations, in fact, that grids generated from the Laplace equation are the smoothest possible. There arises, however, the need to concentrate coordinate lines in certain areas of anticipated strong solution variation, such as near solid walls in viscous flow. This can be accomplished by departing from the Laplace equation and *designing* a partial differential equation system for grid generation: designing because, unlike physics, there are no laws governing grid generation waiting to be discovered.

The first approach to this, historically, was the obvious: simply replace the Laplace equation with Poisson equations $\nabla^2\xi = P$, $\nabla^2\eta = Q$ and leave the *control functions* on the right-hand sides to be specified by the user (with appeal to Urania, the muse of science, for guidance). This does in fact work but the approach has evolved over the years, guided both by logical intuition and the calculus of variations, to use a similar set of equations but with a somewhat different right-hand side. Also, the user has been relieved of the responsibility for specifying the control functions, which are now generally evaluated automatically by the code from the boundary point distributions that are set by the user (cf. Chapter 4). These functions may also be adjusted by the code to achieve orthogonality at the boundary and/or to reduce the grid skewness or otherwise improve the grid quality (cf. Chapter 6).

Algebraic grid generation, based on transfinite interpolation, is typically used to provide an initial solution to start an iterative solution of the partial differential equation for this *elliptic* grid generation system that provides a smoother grid, but with selective concentration of lines, and is less likely to result in overlapping of the boundary.

This elliptic grid generation has an analogy to stretching a membrane attached to the boundaries (cf. Chapter 33) Grid lines inscribed on the underformed membrane move in space as the membrane is selectively stretched, but the region between the boundaries is always covered by the grid. Another form of grid generation from partial differential equations has an analogy with the waves emanating from a stone tossed into a pool This *hyperbolic* grid generation uses a set of hyperbolic equations, rather than the Poisson equation, to grow an orthogonal grid outward from a boundary (cf. Chapter 5). This approach is, in fact, faster than the elliptic grid generation, since no iterative solution is involved, but it is not possible to fit a specified outer boundary. Hyperbolic grid generation is thus limited in its use to open regions. As with the elliptic system, it is possible to control the spacing of the grid lines, and the orthogonality helps prevent skewness.

The control of grid line spacing can be extended to dynamically couple the grid generation system with the physical solution to be performed on the grid in order to resolve developing gradients in the solution wherever such variations appear in the field (cf. Chapter 34 and 35). With such *adaptive* grids, certain solution variables, such as pressure or temperature, are made to feed back to the control functions in the grid generations system to adjust the grid before the next cycle of the physical solution algorithm on the grid.

P-2.4 Variations

Structured grids today are typically generated and applied in the block-structured form described above with the multiple-sponge analogy. A variation is the *chimera* (from the monster of Greek mythology, composed of disparate parts) approach in which separate grids are generated about various boundary components, e.g., bodies in the field, and these separate grids are simply overlaid on a background grid and perhaps on each other in a hierarchy (cf. Chapter 11). The physical solution process on this composite grid proceeds with values being transferred between grids by interpolation. This approach has a number of advantages: (1) simplicity in grid generation since the various grids are generated separately, (2) bodies can be added to, or taken out of, the field easily, (3) bodies can carry their grids when moving relative to the background (think of simulating the kicking of a field goal with the ball and its grid tumbling end over end), (4) the separate grids can be used selectively to concentrate points in regions of developing gradients that may be in motion. The disadvantages are the complexity of setup (but this is being attacked in new code development) and the necessity for the interpolation between grids.

Another approach of interest is the *hybrid* combination with separate structured grids over the various boundaries, connected by unstructured grids (cf. Chapter 23). There is great incentive to use structured grids over boundaries in viscous flow simulation because the boundary layer requires very small spacing out from the wall, resulting either in very long skewed triangular cells or a prohibitively and unnecessarily large number of small cells when unstructured grids are used. This hybrid approach is less well developed but can be expected to receive more attention.

P-2.5 Transformation

The use of numerically generated nonorthogonal curvilinear grids in the numerical solution of PDEs is not, in principle, any more difficult than using Cartesian grids: the differencing and solution techniques can be the same; there are simply more terms in the equations. For instance, the first derivative f_x could be represented in difference form on a Cartesian grid as

$$\left(f_x\right)_{ij} = \frac{f_{i+1,j} - f_{i-1,j}}{2\nabla x}$$

or if the spacing is not uniform, though the grid is still rectangular, by

$$\left(f_x\right)_{ij} = \frac{f_{i+1,j} - f_{i-1,j}}{x_{i+1,j} - x_{i-1,j}}$$

To use a curvilinear grid, this derivative is transformed so that the curvilinear coordinate (ξ,η) rather than the Cartesian coordinate x,y, are the independent coordinates. Thus

$$f_x = \frac{(x_\xi f_\eta - x_\eta f_\xi)}{J}$$

where $J = x_\xi\ y_\eta - x_\eta\ y_\xi$ is the Jacobian of the transformation and represents the cell volume. This then could be written in a difference form, taking $\Delta\xi$ and $\Delta\eta$ to be unity without loss of generality, using

$$\left(f_\xi\right)_{ij} = \frac{1}{2}\left({}_{i+1,j} - {}_{i-1,j}\right)$$
$$\left(f_\eta\right)_{ij} = \frac{1}{2}\left(f_{i+1,j} - f_{i,j-1}\right)$$

with analogous expressions for x_ξ, x_η, y_ξ, y_η.

Movement of the grid, either to follow moving boundaries or to dynamically adapt to developing solution gradients, is not really a complication, since the time derivative can also be transformed as

$$\left(f_t\right)_r = \left(f_t\right)_\xi - \left(f_x\dot{x} + f_y\dot{y}\right)$$

where the time derivative on the left is taken at a fixed position in space, i.e., is the time derivative appearing in the PDEs while the one on the right is that seen by a particular grid point moving with a speed $(\dot{\mathbf{x}},\dot{\mathbf{y}})$. The spatial derivatives (f_x, f_y) are transformed as was discussed above. There is no need to interpolate solution values from the grid at one time step to the displaced grid at the next time step, since that transfer is accomplished by the grid speed terms $(\dot{\mathbf{x}},\dot{\mathbf{y}})$ in the above transformation relation.

The straightforwardness of the use of curvilinear grids is further illustrated by the appearance of the generic convection–diffusion equations;

$$f_t + \nabla\cdot(\mathbf{u}f) + \nabla\cdot(\nu\nabla f) + S = O$$

where $\mathbf{u}$ is the velocity, ν is a diffusion coefficient, and S is a source term, after transformation:

$$A_t + \sum_{i=1}^{3}\left(U^i + \nu\nabla^2\xi^i\right)A_{\xi^i} + \sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij}\left(\nu A_{\xi^j}\right)_{\xi^i} + A\sum_{i=1}^{3} a^i \cdot u_{\xi^i} + S = 0$$

where now the time derivative is understood to be that seen by a certain (moving) grid point. Here the elements of the contravariant metric tensor g^{ij} are given by

$$g^{ij} = a^i \cdot a^j$$

where the $\mathbf{a}^i$ are the contravariant base vectors (which are simply normals to the cell sides):

$$a^i = \left(a_j \times a_k\right) / \sqrt{g} \ (i, j, k \ cyclic)$$

with the $\mathbf{a}_i$ the covariant base vectors (tangents to the coordinate lines):

$$a_i = r_{\xi^i}$$

where **r** is the Cartesian coordinate of a grid point, and $\sqrt{g}$ is the Jacobian of the transformation (the cell volume):

$$\sqrt{g} = a_1 \cdot \left(a_2 \times a_3\right)$$

Also, the contravariant velocity (normal to the cell sides) is

$$U^i = a \cdot (u - r)$$

where **u** is the fluid velocity and **r** is the velocity of the moving grid. For comparison, the Cartesian grid formulation is

$$A_t + \sum_{i=1}^{3} u_i A_{x_i} + \sum_{i=1}^{3} \sum_{j=1}^{3} \delta_{ij} \left(v A_{x_j}\right)_{x_i} + A \sum_{i=1}^{3} \left(u_i\right)_{x_i} + S = 0$$

The formulation has thus been complicated by the curvilinear grid only in the sense that the coefficient u_i has been replaced by the coefficient $U^i + v(\nabla^2 \xi^i)$, and the Kronecker delta in the double summation has been replaced by g^{ij} (thus expanding that summation from three terms to nine terms), and through the insertion of variable coefficients in the last summation. When it is considered that the transformed equation is to be solved on a fixed rectangular field with a uniform square grid, while the original equation would have to be solved on a field with moving curved boundaries, the advantages of using the curvilinear systems are clear.

These advantages are further evidenced by consideration of boundary conditions. In general, boundary conditions for the example being treated would be of the form

$$\alpha A + \beta \mathbf{n} \cdot \left(u \nabla A\right) = \gamma$$

where **n** is the unit normal to the boundary and α, β, and γ are specified. These conditions transform to

$$\alpha A + \beta \frac{v}{\sqrt{g^{ii}}} \sum_{j=1}^{3} g^{ij} A_{\xi^j} = \gamma$$

for a boundary on which ξ^i is constant. For comparison, the original boundary conditions can be written in the form

$$\alpha A + \beta v \sum_{i=1}^{3} n_j A_{x_j} = \gamma$$

The transformed boundary conditions thus have the same form as the original conditions, but with the coefficient n_j replaced by $g^{ij}/\sqrt{g^{ii}}$. The important simplification is the fact that the boundary to which the transformed conditions are applied is fixed and flat (coincident with a curvilinear coordinate surface). This permits a discrete representation of the derivative $A_{\xi j}$ along the transformed boundary without the need for interpolation. By contrast, the derivative A_{xj} in the original conditions cannot be discretized along the physical boundary without interpolation since the boundary is curved and may be in motion.

Although the transformed equation clearly contains more terms, the differencing is the same as on a rectangular grid, i.e., it is done on the logically rectangular computational lattice, and the solution field is logically rectangular. Note that it is not necessary to discover and implement a transformation for each new boundary shape — rather the above formulation applies for all, simply with different values of (x, y, z) at the grid points.

The transformed PDE can also be expressed in conservative form as

$$\left(\sqrt{g}A\right)_t + \sum_{i=1}^{3}\left[\sqrt{g}\left(U^i A + v\sum_{i=1}^{3} g^{ij} A_{\xi^j}\right)\right]_{\xi^i} + \sqrt{g}S = 0$$

for use in the finite volume approach. For more information on transformations, see Chapter 2.

P-3 Unstructured Grids

P-3.1 Connectivities and Data Structures

The basic difference between structured and unstructured grids lies in the form of the data structure which most appropriately describes the grid. A structured grid of quadrilaterals consists of a set of coordinates and connectivities that naturally map into elements of a matrix. Neighboring points in a mesh in the physical space are the neighboring elements in the mesh matrix (Figure 14).

Thus, for example, a two-dimensional array $x(i,j)$ can be used to store the x-coordinates of points in a 2D grid. The index i can be chosen to describe the position of points in one direction, while j describes the position of points in the other direction. Hence, in this way, the indices **i** and **j** represent the two families of curvilinear lines. These ideas naturally extend to three dimensions.

For an unstructured mesh the points cannot be represented in such a manner and additional information has to be provided. For any particular point, the connection with other points must be defined explicitly in the connectivity matrix (Figure 15).

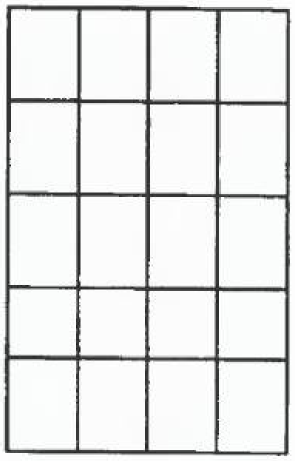
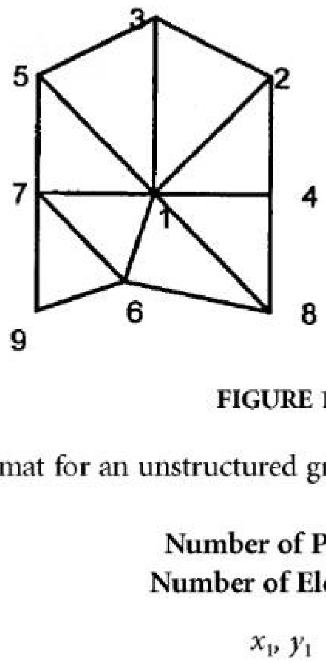


FIGURE 14



Element,	Nodes
1	1, 5, 7
2	1, 4, 2
3	1, 2, 3
4	1, 3, 5
5	1, 7, 6
6	1, 6, 8
7	1, 8, 4
8	7, 9, 6

FIGURE 15

A typical form of data format for an unstructured grid in two dimensions is

Number of Points,
Number of Elements

$$
\begin{gathered}
x_1, y_1 \\
x_2, y_2 \\
x_3, y_3 \\
\ldots \\
n_1, n_2, n_3 \\
n_4, n_5, n_6 \\
n_7, n_8, n_9 \\
\ldots
\end{gathered}
$$

where (x_1, y_1) are the coordinates of point i, and n_i, 1=1,N are the point numbers with, for example, the triad (n_1, n_2, n_3) forming a triangle.

Other forms of connectivity matrices are equally valid, for example, connections can be based upon edges. The real advantage of the unstructured mesh is, however, because the points and connectivities do not possess any global structure. It is possible, therefore, to add and delete nodes and elements as the geometry requires or, in a flow adaptivity scheme, as flow gradients or errors evolve. Hence the unstructured approach is ideally suited for the discretization of complicated geometrical domains and complex flowfield features. However, the lack of any global directional features in an unstructured grid makes the application of line sweep solution algorithms more difficult to apply than on structured grids.

P-3.2 Grid Generation Approaches

In contrast to the generation of structured grids, algorithms to construct unstructured grids are frequently based upon geometrical ideas. There are now many techniques available, many of which are described within this Handbook. For this elementary overview it is not appropriate to discuss details but to comment on general procedures.

P-3.2.1 Triangle and Tetrahedra Creation by Delaunay Triangulation

The Delaunay approach to unstructured grid generation is now popular. The basic concepts go back as far as Dirichlet, who in a paper in 1850 discussed the basic geometrical concept. Dirichlet proposed a method whereby a given domain could be systematically decomposed into a set of packed convex polygons. Given two points in the plane, P and Q, the perpendicular bisector of the line joining the two points subdivides the plane into two regions, V and W. The region V is closer to P than it is to Q. Extending these ideas, it is clear that for a given set of points in the plane, the regions V_i are territories that can be assigned to each point so that V_i represents the space closer to P_i than to any other point in the set. 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 polygons, called Voronoï regions, covering the entire domain.

From this description, it is apparent that in two dimensions, the territorial boundary that forms a side of a Voronoï polygon must be midway between the two points 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 a boundary in common are joined by straight lines, the result is a triangulation of the convex hull of the set of points P_i. This triangulation is known as the Delaunay triangulation.

Equivalent constructions can be defined in higher dimensions. In three dimensions, the territorial boundary that forms a face of a Voronoï polyhedron is equidistant between the two points it separates. If all point pairs that have a common face in the Voronoï construction are connected, then a set of tetrahedra is formed that covers the convex hull of the data points.

For the number of points which may be required in grid for computational analysis, it might appear that the above procedure would be difficult and computationally expensive to construct. However, there are several algorithms that can form the construction in a very efficient manner. These are discussed at length in Chapters 1, 16 and 20. The approach is very flexible in that it can automatically create grids with the minimum of user interaction for arbitrary geometries.

P-3.2.2 Triangle and Tetrahedra Creation by the Advancing Front Method

A grid generation technique based on the simultaneous point generation and connection is the *advancing front method.* Unlike the Delaunay approach, advancing front methods are not based on any geometrical criteria. They encompass the logical procedure of starting with a boundary grid of edges, in two dimensions, triangular faces, in three dimensions, and creating a point and constructing an element. Slowing the initial boundary *advances* into the domain until the domain is filled with elements. The placing of points within the domain is, like the Delaunay approach, controlled by a combination of a background mesh and sources that provides the required data to ensure adequate resolution of the domain. The algorithms that generate grids in this way are based on fast geometrical search routines. Details are to be found in Chapter 17.

It is possible to combine techniques from both the Delaunay and the Advancing Front methods to produce effective grid generation procedures – a sort of combination that tries to utilize the advantages of both approaches. Chapter 18 discusses one such approach.

The Delaunay triangulation produces elements that are isotropic in nature. Although the Advancing Front method can produce elements with stretching, it cannot produce high quality meshes with stretching factors applicable to some problems, such as high Reynolds number viscous flows. Hence, it is necessary to augment the standard procedures outlined above. In general, this is done by introducing a mapping that ensures that regular isotropic grids can be generated but once mapped back to the physical space are distorted in a well defined manner to give appropriate element stretching. Such a method is described in detail in Chapter 20.

P-3.2.3 Unstructured Grids of Quadrilaterals and Hexahedra

The preference of some developers for quadrilateral or hexahedral element based unstructured meshes has resulted in effort devoted to the generation of such meshes. In two dimensions, it is possible to modify the Advancing Front algorithm to construct quadrilaterals, although the additional complexity in extending this approach to three dimensions has not yet been overcome for practical geometries. An alternative approach that has seen some success is that of "paving." This approach relies upon iteratively layering or paving rows of elements in the interior of a region. As rows overlap or coincide they are carefully connected together. It is fair to conclude that almost without exception the methods for the construction of unstructured hexahedral based grids are heuristic in nature, requiring considerable effort to include the many possible geometrical occurrences. Chapter 21 discusses in detail aspects of this kind of grid generation.

P-3.2.4 Surface Mesh Generation

The generation of unstructured grids on surfaces is, in itself, one of the most difficult and yet important aspects of mesh generation in three dimensions. The surface mesh influences the field mesh close to the boundary. Surface meshes have the same requirement for smoothness and continuity as the field meshes for which they act as boundary conditions, but in addition, they are required to conform to the geometry surfaces, including lines of intersection and must accurately resolve regions of high curvature.

The approach usually taken to generate grids on surfaces is to represent the geometry in parametric coordinates. A parametric representation of a surface is straightforward to construct and provides a description of a surface in terms of two parametric coordinates. This is of particular importance, since the generation of a mesh on a surface then involves using grid generation techniques developed for two space dimensions. A full description of these procedures is given in Chapter 19.

P-3.3 Grid Adaptation Techniques

To resolve features of a solution field accurately it is, in general, necessary to introduce grid adaptivity techniques. Adaptivity is based on the equidistribution of errors principle, namely,

$$w_i ds_i = \text{constant}$$

where w_i is the error or activity indicator at node i and ds_i is the local grid point spacing at node i.

Central to adaptivity techniques and the satisfaction of this equidistribution principle is to define an appropriate indicator w_i. Adaptivity criteria are based on an assessment of the error in the solution of the governing equations or are constructed to detect features of the field. These estimators are intimately connected to the analysis equations to be solved. For example, some of the main features of a solution of the Euler equations can be shock waves, stagnation points and vortices, and any indicator should accurately identify these flow characteristics. However, for the Navier-Stokes equations, it is important not only to refine the mesh in order to capture these features but, in addition, to adequately resolve viscous dominated phenomena such as the boundary layers. Hence, it seems likely that, certainly in the near future, adaptivity criteria will be a combination of measures, each dependent on some aspects of the flow and, in turn, on the flow equations.

There is also an extensive choice of criteria based on error analysis. Such measures include, a comparison of computational stencils of different orders of magnitude, comparison of the same derivatives on different meshes, e.g., Richardson extrapolation, and resort to classical error estimation theory. No generally applicable theory exists for errors associated with hyperbolic equations, hence, to date combinations of rather *ad hoc* methods have been used.

Once an adaptivity criterion has been established, the equidistribution principle is achieved through a variety of methods, including point enrichment, point derefinement, node movement and remeshing, or combinations of these. For more information on grid adaption techniques, see Chapter 35.

P-3.3.1 Grid Refinement

Grid refinement, or *h*-refinement, involves the addition of points into regions where adaptation is required. Such a procedure clearly provides additional resolution at the expense of increasing the number of points in the computation.

Grid refinement on unstructured grids is readily implemented. The addition of a point or points involves a local reconnection of the elements, and the resulting grid has the same form as the initial grid. Hence, the same solver can be used on the enriched grid as was used on the initial grid.

It is important that the adaptivity criteria resolve both the discontinuous features of the solution (i.e. shock waves, contacts) and the smooth features as the number of grid points are increased. A desirable feature of any adaptive method to ensure convergence is that the local cell size goes to zero in the limit of an infinite number of mesh points.

Grid refinement on a structured or multiblock grids is not so straightforward. The addition of points will, in general, break the regular array of points. The resulting distributed grid points no longer naturally fit into the elements of an array. Furthermore, some points will not "conform" to the grid in that they have a different number of connections to other points. Hence grid refinement on structured grids requires a modification to the basic data structure and also the existence of so-called non-conforming nodes requires modifications to the solver. Clearly, point enrichment on structured grids is not as natural a process as the method applied on unstructured grids and hence is not so widely employed. Work has been undertaken to implement point enrichment on structured grids and the results demonstrate the benefits to be gained from the additional effort in modifications to the data structure and the solve.

P-3.3.2 Grid Movement

Grid movement satisfies the equidistribution principle through the migration of points from regions of low activity into regions of high activity. The number of nodes in this case remains fixed. Traditionally, algorithms to move points involve some optimization principle. Typically, expressions for smoothness, orthogonality and weighting according to the analysis field or errors are constructed and then an optimization is performed such that movement can be driven by a weight function, but not at the expense of loss of smoothness and orthogonality. Such methods are in general, applicable to both structured and unstructured grids.

An alternative approach is to use a weighted Laplacian function. Such a formulation is often used to smooth grids, and of course the formal version of the formulation is used as the elliptic grid generator presented earlier.

P-3.3.3 Combinations of Node Movement, Point Enrichment and Derefinement

An optimum approach to adaptation is to combine node movement and point enrichment with derefinement. These procedures should be implemented in a dynamic way, i.e., applied at regular intervals within the simulation. Such an approach also provides the possibility of using movement and enrichment to independently capture different features of the analysis.

P-3.3.4 Grid Remeshing

One method of adaptation which, to date, has been primarily used on unstructured grids, is adaptive remeshing. As already indicated, unstructured meshes can be generated using the concept of a background mesh. For an initial mesh, this is usually some very coarse triangulation that covers the domain and on which the spatial distribution is consistent with the given geometry. For adaptive remeshing, the solution achieved on an initial mesh is used to define the local point spacing on the background mesh which was itself the initial mesh used for the simulation. The mesh is regenerated using the new point spacing on the background mesh. Such an approach can result in a second adapted mesh that contains fewer points than that contained in the initial mesh. However, there is the overhead of regeneration of the mesh which in three dimensions can be considerable. Nevertheless, impressive demonstrations of its use have been published.