von Karman Institute for Fluid Dynamics

Lecture Series 1990–06

# NUMERICAL GRID GENERATION

June 11–15, 1990

## *UNSTRUCTURED MESH METHODS FOR CFD*

J. Peraire, K. Morgan, J. Peiro

Imperial College, London, UK

# CONTENTS

# 1. INTRODUCTION TO FINITE ELEMENT METHODS, ALGORITHMS AND IMPLEMENTATIONS

## *1.1 Structured and unstructured meshes*

The recent rapid development of solution algorithms in the field of computational mechanics means that it it presently possible to attempt the numerical solution of a wide range of practical problems. The essential pre-requisite to a solution process of this type is the construction of an appropriate mesh to represent the computational domain of interest. In this section, we will briefly outline two alternative strategies for accomplishing this task of mesh generation and make some observations about the implications to the analyst arising from the choice of approach which is made. In the discussion, we will assume, for brevity, that the mesh is to be produced for a two dimensional domain.

In the most widely used approach [1,2], the domain is divided into a structured assembly of quadrilateral cells. The structure in the mesh is apparent from the fact that each interior nodal point is surrounded by exactly the same number of mesh cells (or elements), as shown in figure 1.1. Note also that we can immediately identify two directions within the mesh by associating a curvilinear coordinate system $(\xi,\eta)$ with the mesh lines. If we number the nodes consecutively along lines of constant $\eta$, and so that the numbers increase as $\xi$ increases, we can immediately identify the nearest neighbours of any node J on the mesh, as shown in figure 1.2. Generally, such grids are constructed by mapping the domain of interest into a square and then constructing a rectangular mesh over the square. If the equation itself is also mapped, this grid can be used to obtain a solution, otherwise the inverse mapping is applied to obtain the required mesh over the original domain. Various approaches may be regarded as candidates for accomplishing the mapping, such as conformal techniques, the use of differential equations or algebraic methods. All the major discretisation procedures for the equations of fluid flow can normally be implemented on meshes of this type. A major advantage to the computational fluid dynamicist arising from the use of a structured mesh is that he can choose an appropriate solution method from among the large number of algorithms which are available. These algorithms have the advantage that they can normally be implemented in a computationally efficient manner. A disadvantage is the fact that it is not possible to guarantee an acceptable mesh by applying the mapping method, as described above, to regions of general shape. This difficulty can be alleviated by appropriately sub-dividing the computational domain into blocks and then producing a grid by applying the mapping method to each block separately. This results in an extremely powerful method [3], but problems can still be caused by the generation of elements of poor quality and by the elapsed time necessary to produce a grid for domains of extremely complex shape.

The alternative approach is to divide the computational domain into an unstructured assembly of computational cells as illustrated in figure 1.1. The notable feature of an unstructured mesh is that the number of cells surrounding a typical interior node of the mesh is not necessarily constant. It will be apparent that quadrilateral cells could again be used in this context, as shown in figure 1.3, but we will be concentrating our attention upon the use of triangular meshes. The nodes and the elements are now numbered and, to get the necessary information on the neighbours, we store the numbers of the nodes which belong to each element (see figure 1.4). From the detail of a typical unstructured mesh shown in figure 1.1, it is apparent that there is no concept of directionality within a mesh of this type and that,

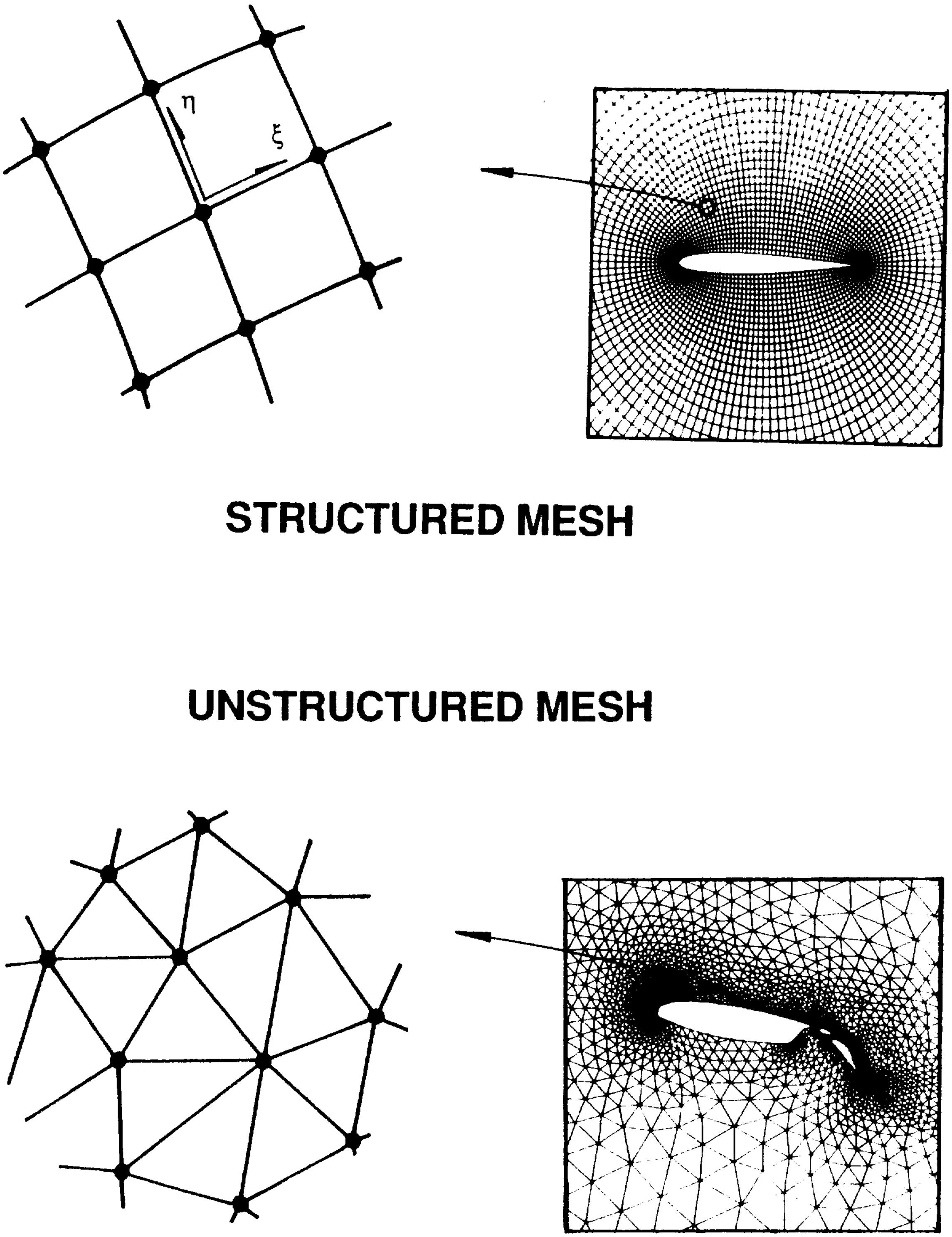


**Figure 1.1** Structured and unstructured mesh discretization of a computational domain.

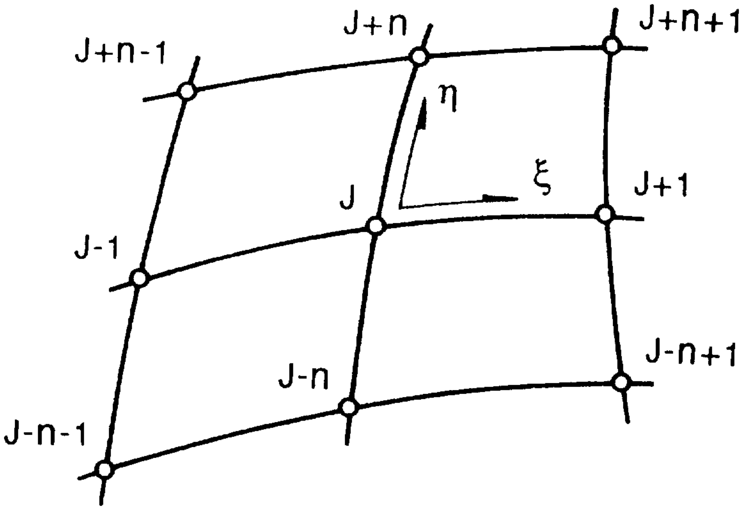


**Figure 1.2** Nearest neighbours of a node J in a structured mesh of n by m points in the $\xi$, $\eta$ directions

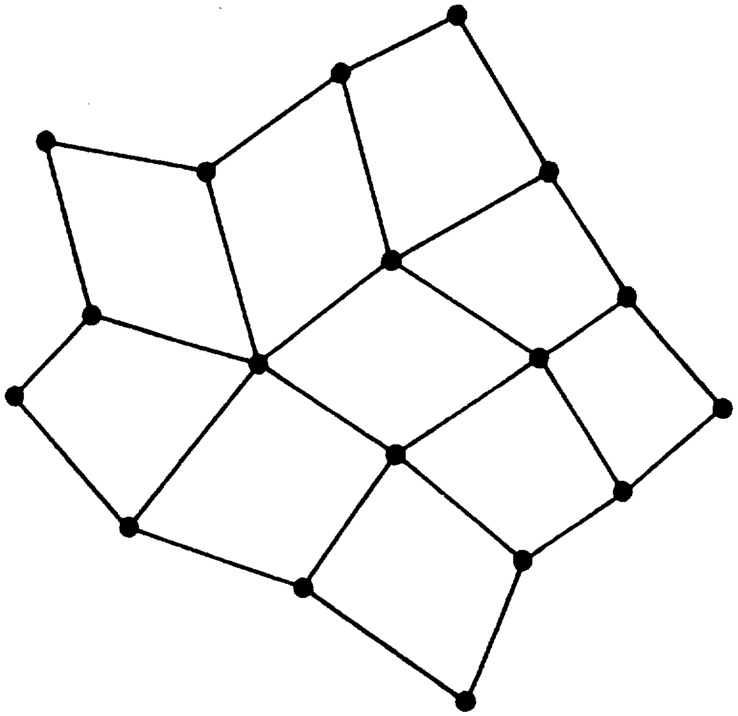


**Figure 1.3** An unstructured quadrilateral mesh.

therefore, solution techniques based upon this concept (e.g. ADI methods) will not be directly applicable. The methods which are normally adopted to generate unstructured triangular meshes are based upon either the Delaunay [4] or the advancing front [5] approaches. Discretisation methods for the equations of fluid flow which are based upon integral procedures, such as the finite volume or the finite element method, are natural candidates for use with unstructured meshes. The principal advantage of the unstructured approach is that it provides a very powerful tool for discretising domains of complex shape [6,7], especially if triangles are used in two dimensions and tetrahedra are used in three dimensions. In addition, unstructured mesh methods naturally offer the possibility of incorporating adaptivity [8]. Disadvantages which follow from adopting the unstructured grid approach are that the number of alternative solution algorithms is currently rather limited and that their computational implementation places large demands on both computer memory and CPU [9]. Further, these algorithms are rather sensitive to the quality of the grid which is being employed and so great care has to be taken in the generation process.

### *1.2 Discretisation techniques*

When an acceptable mesh has been obtained for the computational domain of interest, the analyst is then faced with choosing a discretisation method. This will form the basis of a suitable algorithm for solving the governing differential equation on this mesh. The most widely used discretisation techniques are the finite difference method, the finite volume method and the finite element method [10]. We will illustrate briefly the essentials of these three approaches by considering the application of each method to the solution of a problem of steady linear heat conduction in a two dimensional region, $\Omega$, which is bounded by a closed curve, $\Gamma$.

The temperature distribution $T(x,y)$ will satisfy Laplace's equation

$$\text{div}(\text{grad } T) = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \qquad \text{in } \Omega \qquad (1.1)$$

subject to appropriate boundary conditions. For convenience, we may assume here that the boundary conditions are given in the form

$$T = g(x,y) \qquad \text{on} \quad \Gamma \qquad (1.2)$$

i.e. the value of the temperature is specified at all points of the boundary curve.

The Finite Difference Method

For the purposes of this section, it is sufficient to assume that equation (1.1) is to be solved on a structured square mesh, of the type shown in figure 1.5. We can adopt a convenient coordinate system, such that a typical point on the grid has coordinates ($J\Delta$,$K\Delta$). If we use the subscripts JK to denote an evaluation at this point, equation (1.1) leads to the exact relationship

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{JK} + \left.\frac{\partial^2 T}{\partial y^2}\right|_{JK} = 0 \qquad (1.3)$$

| Element | Nodes | | |
|---|---|---|---|
| 1 | 5 | 8 | 9 |
| 2 | 9 | 8 | 12 |
| 3 | 8 | 7 | 12 |
| 4 | 6 | 7 | 8 |
| 5 | 5 | 6 | 8 |
| 6 | 5 | 9 | 10 |
| 7 | 11 | 5 | 10 |
| 8 | 3 | 5 | 11 |
| 9 | 3 | 4 | 5 |
| 10 | 1 | 4 | 3 |
| 11 | 4 | 6 | 5 |
| 12 | 1 | 6 | 4 |
| 13 | 1 | 2 | 6 |
| 14 | 2 | 7 | 6 |

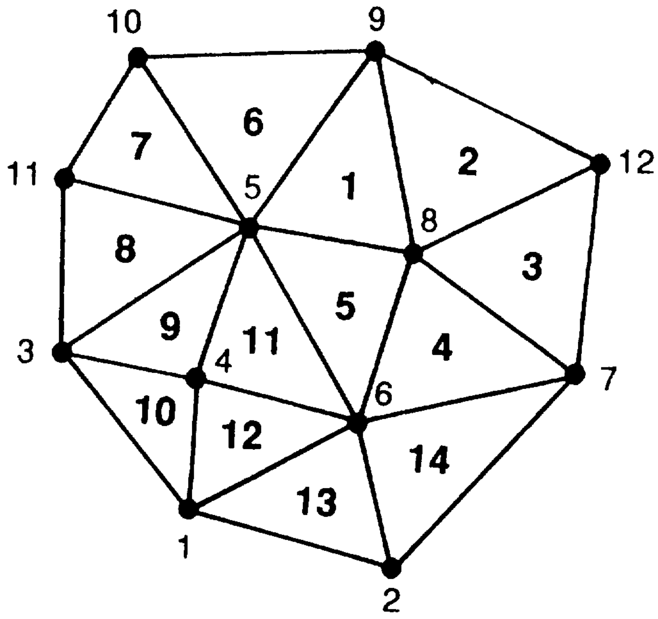


**Figure 1.4** Connectivity array for an unstructured triangular mesh.

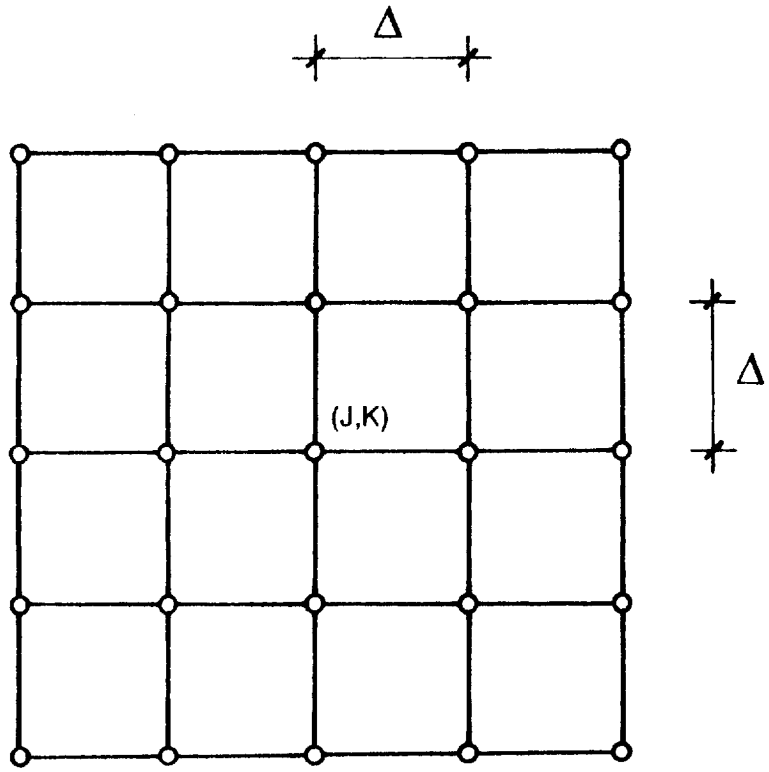


**Figure 1.5** Structured square mesh of constant mesh size Δ.

Using standard central difference representations for the second derivatives, this equation can be approximated for all interior points as

$$\frac{T_{J+1K}-2T_{JK}+T_{J-1K}}{\Delta^2} + \frac{T_{JK+1}-2T_{JK}+T_{JK-1}}{\Delta^2} = 0 \qquad (1.4)$$

which is an equation coupling the values at the five mesh points illustrated in figure 1.6. Writing an equation of this type at each interior point on the grid and incorporating the known values along the boundary, the resulting equation set can be represented in a matrix form

$$\mathbf{K}\,\underline{T} = \underline{f} \qquad (1.5)$$

where **K** is a symmetric constant matrix, $\underline{T}$ is the vector of unknowns and $\underline{f}$ is the force vector which contains information from the boundary conditions. This equation system may be solved by using any suitable procedure. The important point to note here is that the matrix system can be formed directly from the approximation of equation (1.4). This leads to efficient computational implementations, with low storage demands and high vectorisation possibilities.

The Finite Volume Method

The finite volume and finite element methods can also be implemented on structured grids, but we will assume here that we are to use these methods on a given unstructured triangular grid (see figure 1.4). A possible finite volume discretisation for equation (1.1) follows from the requirement that this equation should be satisfied in an integral sense over each cell e i.e.

$$\int_{\Omega_e} \left\{\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right\} d\Omega = 0 \qquad (1.6)$$

where $\Omega_e$ is the area of cell e. Using the divergence theorem, this equation may be written as

$$\int_{\Gamma_e} \frac{\partial T}{\partial n}\, d\Gamma = 0 \qquad (1.7)$$

where $\Gamma_e$ denotes the boundary of cell e and n is the outward normal direction to this boundary. If we associate the unknown $T_e$ with cell e, this equation can be approximated as

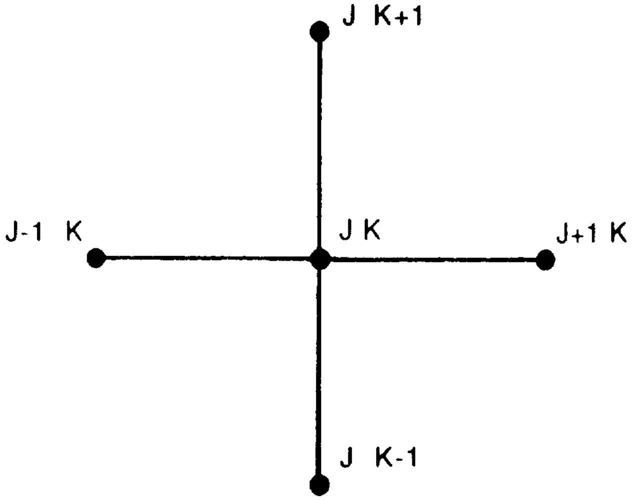


**Figure 1.6** Computational stencil for Laplace's equation.

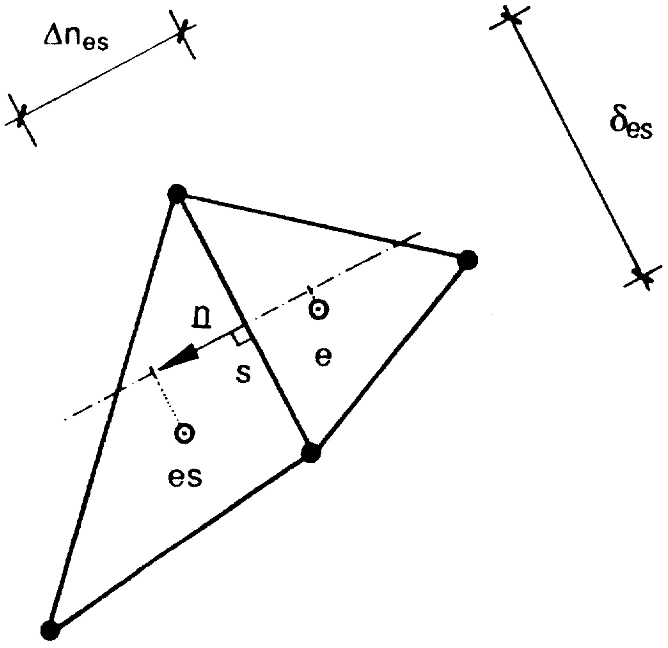


**Figure 1.7** Finite volume discretization

$$\sum_{s=1}^{3} \frac{(T_{es}-T_e)}{\Delta n_{es}} \delta_{es} = 0 \qquad (1.8)$$

where the summation extends over the three sides of the cell e, $T_{es}$ is the unknown in the other cell es which is adjacent to side s, $\delta_{es}$ denotes the length of side s and $\Delta n_{es}$ is the projection onto the normal to side s of the distance between the centroids of the cells e and es (see figure 1.7). When each cell in the mesh is considered in this way, we are led again to a system of equations which can be written in the matrix form of equation (1.5). Note that a convenient data structure in this case would be to number each cell side in the mesh and to store the numbers of the two cells which are adjacent to each side. Equation (1.8) can then be formed by looping over the cell sides and sending the appropriate (equal and opposite) contributions to the adjacent elements. In this form, the conservation properties of the resulting scheme are immediately apparent as the total contribution made by each interior side is zero.

Variational Formulations

Finally, we illustrate how the finite element method can be used to discretise equation (1.1) on an unstructured triangular grid. The starting point is a variational formulation of the problem [11]. Let $\mathcal{T}$ denote the set of all functions T which satisfy the problem boundary condition T = g(x,y) on $\Gamma$ and let $\mathcal{W}$ denote the set of all functions W which satisfy W = 0 on $\Gamma$. In addition, we shall see that the type of variational formulation which is employed will place certain differentiability conditions upon the members of these sets. A possible variational formulation for the above problem could now be : find T in $\mathcal{T}$ such that

$$\int_{\Omega} \left\{\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right\} W \, d\Omega = 0 \qquad (1.9)$$

for every W in $\mathcal{W}$. The sets $\mathcal{T}$ and $\mathcal{W}$ are termed the trial function and the weighting function sets respectively. It will be apparent that the integral appearing in equation (1.9) is valid mathematically provided that the trial functions have continuous first derivatives, while the weighting functions may be discontinuous. It follows immediately that the function T which satisfies this variational formulation will be identically equal to the solution of the problem as posed classically in equations (1.1) and (1.2).

An alternative, so-called weak, formulation can be obtained by using the divergence theorem in equation (1.9) and applying the fact that each function W which is to be considered will vanish on $\Gamma$. The resulting formulation can be stated as : find T in $\mathcal{T}$ such that

$$\int_{\Omega} \left\{\frac{\partial T}{\partial x}\frac{\partial W}{\partial x} + \frac{\partial T}{\partial y}\frac{\partial W}{\partial y}\right\} d\Omega = 0 \qquad (1.10)$$

for each W in $\mathcal{W}$. Note that the trial function set may now be widened, as equation (1.10) requires only that the trial functions be continuous. At the same time, stricter conditions need to be applied to the members of the weighting function set, which must also now be continuous. Assuming that the actual solution is sufficiently smooth so that the steps involved are mathematically valid, it is readily observed that the above steps may be reversed and equation (1.9) regained. It follows that the solution of the weak formulation will also be identical to the solution of the original problem posed in equations (1.1) and (1.2).

### The Galerkin Method

The Galerkin method is a widely used approach for constructing an approximate solution to a problem posed in a variational form [12]. We begin the process by selecting a basis $N_1$, $N_2$, $N_3$,....... for $\mathcal{W}$. This means that each weighting function can be expressed as a linear combination of these basis functions and we indicate this by defining $\mathcal{W}$ by

$$\mathcal{W} = \{ W \mid W = a_1N_1+a_2N_2+a_3N_3+..........;\ W = 0 \text{ on } \Gamma\} \qquad (1.11)$$

where $a_1$, $a_2$, $a_3$, ... denote arbitrary constants. Note that each $N_J$ in the basis must satisfy the condition $N_J = 0$ on $\Gamma$. We can employ a trial function set which is closely linked to $\mathcal{W}$ and is defined by

$$\mathcal{T} = \{ T \mid T = g + b_1N_1+b_2N_2+a_3N_3+..........;\ T = g \text{ on } \Gamma\} \qquad (1.12)$$

where $b_1$, $b_2$, $b_3$, .... denote arbitrary constants. This set has been carefully constructed so that each member of the set satisfies the required conditions on $\Gamma$. If we define $\mathcal{T}_{(p)}$ and $\mathcal{W}_{(p)}$ as the subspaces of $\mathcal{T}$ and $\mathcal{W}$ respectively which are spanned by the first p basis functions i.e.

$$\begin{aligned} \mathcal{T}_{(p)} &= \{ T_{(p)} \mid T_{(p)} = g + b_1N_1+b_2N_2+b_3N_3+.....+b_p N_p;\ T_{(p)} = g \text{ on } \Gamma \} \\ \mathcal{W}_{(p)} &= \{ W_{(p)} \mid W_{(p)} = a_1N_1+a_2N_2+a_3N_3+.......... +a_p N_p.\ W_{(p)} = 0 \text{ on } \Gamma \} \end{aligned} \qquad (1.13)$$

then the Galerkin method is to seek an approximate solution to the variational formulation of equation (1.10) in the form : find $T_{(p)}$ in $\mathcal{T}_{(p)}$ such that

$$\int_\Omega \left\{ \frac{\partial T_{(p)}}{\partial x}\frac{\partial W_{(p)}}{\partial x} + \frac{\partial T_{(p)}}{\partial y}\frac{\partial W_{(p)}}{\partial y} \right\} d\Omega = 0 \qquad (1.14)$$

for each $W_{(p)}$ in $\mathcal{W}_{(p)}$ . However, since $N_1$, $N_2$, $N_3$, ......., $N_p$ forms a basis for $\mathcal{W}_{(p)}$, equation (1.14) can be replaced by the equivalent statement : find $T_{(p)}$ in $\mathcal{T}_{(p)}$ such that

$$\int_\Omega \left\{ \frac{\partial T_{(p)}}{\partial x}\frac{\partial N_J}{\partial x} + \frac{\partial T_{(p)}}{\partial y}\frac{\partial N_J}{\partial y} \right\} d\Omega = 0 \qquad i = 1, 2, 3, ......, p \qquad (1.15)$$

Inserting the assumed form for the function $T_{(p)}$ from equation (1.13) gives

$$\sum_{K=1}^{p}\left[\int_{\Omega}\left\{\frac{\partial N_J}{\partial x}\frac{\partial N_K}{\partial x}+\frac{\partial N_J}{\partial y}\frac{\partial N_K}{\partial y}\right\}d\Omega\right]b_J = \qquad (1.16)$$

$$-\int_{\Omega}\left\{\frac{\partial g}{\partial x}\frac{\partial N_J}{\partial x}+\frac{\partial g}{\partial y}\frac{\partial N_J}{\partial y}\right\}d\Omega \qquad K = 1, 2, 3, \ldots\ldots, p$$

which can be written in the matrix form

$$\mathbf{K}\,\underline{b} = \underline{f} \qquad (1.17)$$

This equation can be solved to determine the unknown coefficients $b_1, b_2, \ldots, b_p$ and so complete the approximation process. These unknowns only give information about the value of the approximation at any point when they are combined as in equation (1.13). Note that K is symmetric and will, in general, be a full matrix.

The Finite Element Method

The Galerkin finite element method results from making a particular choice for the basis functions in equations (1.11) and (1.12). Although more sophisticated representations are possible, we will consider here only the case in which, given a general grid of triangles, we place nodes at the vertices of each triangle and associate an unknown $T_J$ and a piecewise linear shape function $N_J$ with each node J. In this case, the shape functions are constructed (see figure 1.8) such that (a) $N_J$ takes the value unity at node J and the value zero at all other nodes (b) $N_J$ varies linearly (c) $N_J$ is only non-zero on the elements associated with node J. For notational purposes only, it will be convenient to assume that the nodes have been numbered such that nodes 1 to p are interior nodes while nodes p+1 to q lie on the boundary. We define $\Gamma_s$ to be the assembly of the straight sides in the mesh which join the boundary nodes, so that $\Gamma_s$ is an approximation to the exact boundary $\Gamma$. The given function g(x,y) defined on $\Gamma$ in equation (1.2) is approximated on $\Gamma_s$ by the function $g_{(s)}$ constructed as

$$g_{(s)} = \sum_{J=p+1}^{q} T_J\,N_J \qquad T_J = g(x_J, y_J) \qquad (1.18)$$

and this approximation is exact at each of the boundary nodes. We work with spaces, of dimension p, defined by

$$\mathcal{T}_{(p)} = \{\, T_{(p)} \mid T_{(p)} = g_{(s)} + T_1N_1+T_2N_2+T_3N_3+\ldots\ldots.+T_pN_p\ ;\ T_{(p)} = g_{(s)} \text{ on } \Gamma_s\} \qquad (1.19)$$

$$\mathcal{W}_{(p)} = \{\, W_{(p)} \mid W_{(p)} = a_1N_1+a_2N_2+a_3N_3+\ldots\ldots\ldots.+ a_pN_p\ ;\ W_{(p)} = 0 \text{ on } \Gamma_s\}$$

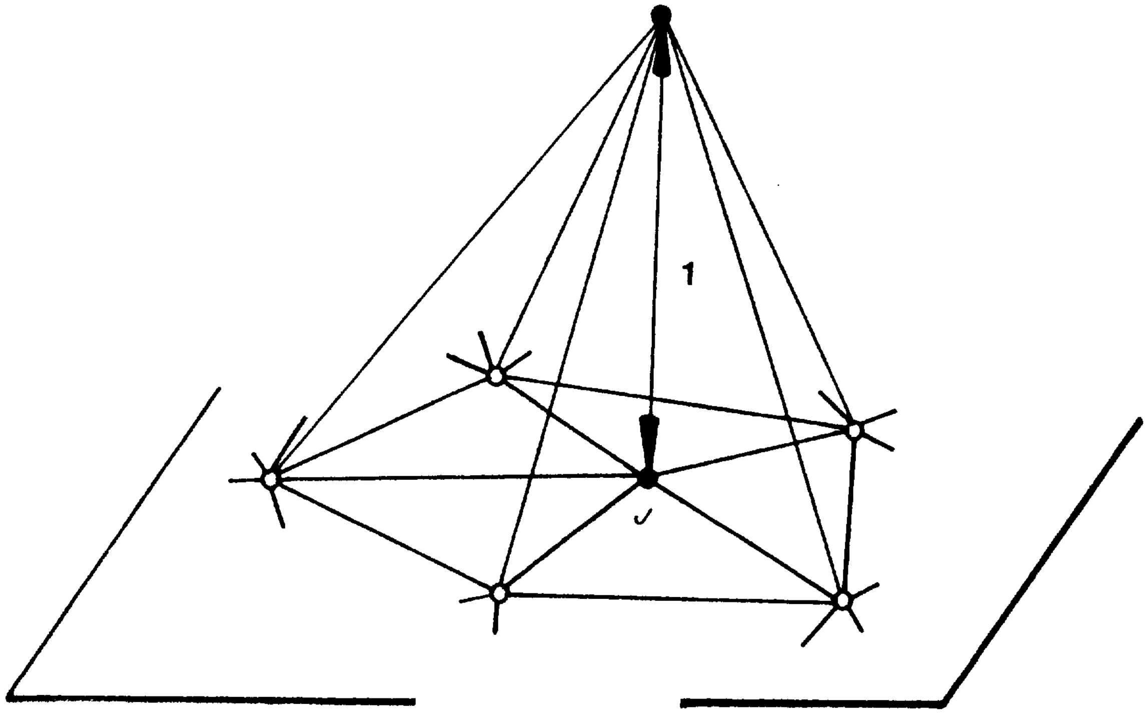


Figure 1.8 Shape function associated to node J

where we have indicated in the definition of $\mathcal{T}_{(p)}$ that the coefficient of $N_J$ is now the value of $T_{(p)}$ at node J. The Galerkin procedure is followed exactly as above, with equation (1.16) being replaced by the requirement that

$$\sum_{K=1}^{p}\left[\int_{\Omega}\left\{\frac{\partial N_J}{\partial x}\frac{\partial N_K}{\partial x}+\frac{\partial N_J}{\partial y}\frac{\partial N_K}{\partial y}\right\}d\Omega\right]T_K = \tag{1.20}$$

$$-\sum_{K=p+1}^{q}\left[\int_{\Omega}\left\{\frac{\partial N_J}{\partial x}\frac{\partial N_K}{\partial x}+\frac{\partial N_J}{\partial y}\frac{\partial N_K}{\partial y}\right\}d\Omega\right]T_K \qquad J = 1,\ 2,\ 3,\ \ldots\ldots,\ p$$

This equation set can again be written in a matrix form

$$\mathbf{K}\,\underline{T} = \underline{f} \tag{1.21}$$

where $\underline{T}$ is now a vector of the unknown approximations to the nodal values of the temperature and a typical entry in the matrix **K** is given by

$$[K]_{JK} = \int_{\Omega}\left\{\frac{\partial N_J}{\partial x}\frac{\partial N_K}{\partial x}+\frac{\partial N_J}{\partial y}\frac{\partial N_K}{\partial y}\right\}d\Omega \tag{1.22}$$

To evaluate these entries, we make use of the local nature of the defined shape functions and the result that the integral over $\Omega$ is equal to the sum of the integrals over the individual triangles $\Omega_e$ [12] i.e.

$$\int_{\Omega}\{\cdot\}\,d\Omega = \sum_{e=1}^{E}\int_{\Omega_e}\{\cdot\}\,d\Omega \tag{1.23}$$

This means that the matrix **K** can be written as

$$\mathbf{K} = \sum_{e=1}^{E}\mathbf{K}^{e} \tag{1.24}$$

where the individual element matrices $\mathbf{K}^e$ have typical entries

$$[K]_{JK}{}^{e} = \int_{\Omega_e}\left\{\frac{\partial N_J{}^{e}}{\partial x}\frac{\partial N_K{}^{e}}{\partial x}+\frac{\partial N_J{}^{e}}{\partial y}\frac{\partial N_K{}^{e}}{\partial y}\right\}d\Omega \tag{1.25}$$

Here $N_J{}^e$ has been used to indicate the value of the shape function $N_J$ on element e. We note that the only shape functions which are non-zero over element e are those associated with the nodes J, K, L of this element. This in turn means that there will be only 9 non-zero entries in $\mathbf{K}^e$ which will occur in those positions which are in both rows J, K,L and in columns J, K, L. Explicit expressions for the shape functions over element e are readily determined from the conditions which were used in their definition. If

$$N_J^e = A_J^e + B_J^e x + C_J^e y \qquad (1.26)$$

where $A_J^e$ , $B_J^e$ , $C_J^e$ are constants, it follows that

$$A_J^e = \frac{(x_K y_L - x_L y_K)}{2\Omega_e} \qquad B_J^e = \frac{(y_K - y_L)}{2\Omega_e} \qquad C_J^e = \frac{(x_L - x_K)}{2\Omega_e} \qquad (1.27)$$

The non-zero entries in the element matrix $\mathbf{K}^e$ can be obtained by direct integration e.g.

$$[\mathbf{K}^e]_{KL} = \frac{1}{4\Omega_e}\{B_K^e B_L^e + C_K^e C_L^e\} \qquad (1.28)$$

With the contributions from a typical element computed in this fashion, it is then possible to assemble the contributions from all the elements in the mesh, according to equation (1.24) and thus to obtain the final form for **K**. The right hand side vector $\underline{f}$ in equation (1.21) can be similarly evaluated. The symmetry of **K** should be apparent from equation (1.28) and it should also be observed that this will be a sparse matrix.

A computer implementation of this method would follow identical lines, with the matrix **K** being determined from a loop over the elements. As each element is considered, the non-zero terms in its element matrix are determined from expressions such as equation (1.28) and these are assembled into the correct locations in **K**, making use of the stored connectivity information, which gives the nodes belonging to each element.

Optimality of the Galerkin Method

For a linear elliptic differential equation, of the type which has been considered above, it is possible to prove that an approximate solution constructed via the Galerkin method possesses a certain optimality property [11]. To demonstrate this, suppose that we have chosen a suitable set of basis functions and a value for the dimension p of the subspace to be considered. We know from equation (1.14) that the Galerkin approximation will satisfy

$$\int_\Omega \left\{ \frac{\partial T_{(p)}}{\partial x}\frac{\partial W_{(p)}}{\partial x} + \frac{\partial T_{(p)}}{\partial y}\frac{\partial W_{(p)}}{\partial y} \right\} d\Omega = 0 \qquad (1.29)$$

for any $W_{(p)}$ in $\mathcal{W}_{(p)}$. As each $W_{(p)}$ is in $\mathcal{W}_{(p)}$, we can deduce that each $W_{(p)}$ is also in $\mathcal{W}$, since $\mathcal{W}_{(p)}$ is a subspace of $\mathcal{W}$. Thus, from equation (1.10) the exact solution T will also satisfy the equation

$$\int_{\Omega} \left\{ \frac{\partial T}{\partial x} \frac{\partial W_{(p)}}{\partial x} + \frac{\partial T}{\partial y} \frac{\partial W_{(p)}}{\partial y} \right\} d\Omega = 0 \tag{1.30}$$

Subtracting equation (1.29) from equation (1.30), we can deduce that the error $\varepsilon = T - T_{(p)}$ in the Galerkin approximation satisfies

$$\int_{\Omega} \left\{ \frac{\partial \varepsilon}{\partial x} \frac{\partial W_{(p)}}{\partial x} + \frac{\partial \varepsilon}{\partial y} \frac{\partial W_{(p)}}{\partial y} \right\} d\Omega = 0 \tag{1.31}$$

for every $W_{(p)}$ in $\mathcal{W}_{(p)}$ Now choose any other function $U_{(p)}$ from the trial function set $\mathcal{T}_{(p)}$. The objective is to show that such a function will always be a worse approximation to the exact solution, according to some measure, than the Galerkin approximation. Define

$$\varepsilon_{\upsilon} = T - U_{(p)} = T - T_{(p)} + T_{(p)} - U_{(p)} = \varepsilon + V_{(p)} \tag{1.32}$$

where

$$V_{(p)} = T_{(p)} - U_{(p)} \tag{1.33}$$

By direct substitution, it can be shown that

$$\begin{aligned} \int_{\Omega} \left\{ \left[\frac{\partial \varepsilon_{\upsilon}}{\partial x}\right]^2 + \left[\frac{\partial \varepsilon_{\upsilon}}{\partial y}\right]^2 \right\} d\Omega &= \int_{\Omega} \left\{ \left[\frac{\partial \varepsilon}{\partial x}\right]^2 + \left[\frac{\partial \varepsilon}{\partial y}\right]^2 \right\} d\Omega \\ &+ 2 \int_{\Omega} \left\{ \frac{\partial \varepsilon}{\partial x} \frac{\partial V_{(p)}}{\partial x} + \frac{\partial \varepsilon}{\partial y} \frac{\partial V_{(p)}}{\partial y} \right\} d\Omega + \int_{\Omega} \left\{ \left[\frac{\partial V_{(p)}}{\partial x}\right]^2 + \left[\frac{\partial V_{(p)}}{\partial y}\right]^2 \right\} d\Omega \end{aligned} \tag{1.34}$$

From the construction of equation (1.33), $V_{(p)}$ must be a member of the set $\mathcal{W}_{(p)}$ and so the second term on the right hand side vanishes, by equation (1.31). The third term on the right hand side is strictly non-negative and so we deduce that

$$\int_{\Omega} \left\{ \left[\frac{\partial \varepsilon_{\upsilon}}{\partial x}\right]^2 + \left[\frac{\partial \varepsilon_{\upsilon}}{\partial y}\right]^2 \right\} d\Omega \geq \int_{\Omega} \left\{ \left[\frac{\partial \varepsilon}{\partial x}\right]^2 + \left[\frac{\partial \varepsilon}{\partial y}\right]^2 \right\} d\Omega \tag{1.35}$$

Introducing the notation

$$\|\varepsilon\| = \int_{\Omega} \left\{ \left[\frac{\partial \varepsilon}{\partial x}\right]^2 + \left[\frac{\partial \varepsilon}{\partial y}\right]^2 \right\} d\Omega \qquad (1.36)$$

it can be seen that equation (1.35) can be written as

$$\|\varepsilon_v\| \geq \|\varepsilon\| \qquad (1.37)$$

Since the function $U_{(p)}$ was arbitrarily chosen from the set $\mathcal{T}_{(p)}$, we have thus demonstrated the optimality of the Galerkin approximation $T_{(p)}$ among the set of functions $\mathcal{T}_{(p)}$ according to the error measure defined in equation (1.36).

### *1.3 The Finite Element Method Applied to the Compressible Euler Equations*

With the essentials of the finite element approach briefly outlined above, we can now consider facing the more challenging problem of developing a finite element scheme for the solution of the two dimensional compressible Euler equations. The discretisation of the equations is to be accomplished using a mesh of linear triangular elements. A Galerkin approach will again be applied, but the hyperbolic character of the equation set means that optimality, in the sense of equation (1.37), can no longer be established. The approach to be followed will employ the time dependent form of the equations and steady state solutions will be computed by means of a false transient. A finite difference method will be used to advance the solution in time, which means that the variational formulation will be applied to the space dimensions only, in exactly the same form as above.

The two dimensional equations governing compressible inviscid flow are considered in the conservation form

$$\frac{\partial \underline{U}}{\partial t} + \frac{\partial \underline{E}}{\partial x} + \frac{\partial \underline{F}}{\partial y} = 0 \qquad (1.38)$$

where the unknown vector $\underline{U}$ and the flux vectors $\underline{E}$ and $\underline{F}$ are given by

$$\underline{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix} \qquad \underline{E} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho E + p)u \end{bmatrix} \qquad \underline{F} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho E + p)v \end{bmatrix} \qquad (1.39)$$

Here $\rho$, p and E denote the density, pressure and specific total energy of the fluid, while u and v are the components of the fluid velocity vector in the x and y directions respectively. The equation set is completed by the addition of the perfect gas equation of state

$$p = (\gamma - 1)\, \rho\, [E - \frac{1}{2}(u^2 + v^2)] \qquad (1.40)$$

where $\gamma$ is the ratio of the specific heats.

## Time-Stepping Scheme

To develop a time-stepping scheme for equation (1.38), we consider a Taylor expansion in time [13] in the form

$$\underline{U}^{n+1} = \underline{U}^{n} + \Delta t \left.\frac{\partial \underline{U}}{\partial t}\right|^{n} + \frac{\Delta t^2}{2} \left.\frac{\partial^2 \underline{U}}{\partial t^2}\right|^{n+\theta} \qquad (1.41)$$

where the superscript n denotes an evaluation at time $t = t_n$, the timestep $\Delta t = t_{n+1} - t_n$, and $t_{n+\theta} = t_n + \theta\, \Delta t$, $0 \le \theta \le 1$. This equation may be re-written, using equation (1.38), to give

$$\Delta \underline{U} = - \Delta t \left\{\frac{\partial \underline{E}}{\partial x} + \frac{\partial \underline{F}}{\partial y}\right\}^{n} - \frac{\Delta t^2}{2} \left\{\frac{\partial}{\partial x}\left(\frac{\partial \underline{E}}{\partial t}\right) + \frac{\partial}{\partial y}\left(\frac{\partial \underline{F}}{\partial t}\right)\right\}^{n+\theta} \qquad (1.42)$$

where $\Delta \underline{U} = \underline{U}^{n+1} - \underline{U}^{n}$. A solution algorithm can be produced by constructing a suitable linearisation for the second order terms on the right hand side of this equation. A straightforward linearisation [14] leads to the equation

$$\Delta \underline{U} - \frac{\theta \Delta t^2}{2} \left\{\frac{\partial}{\partial x} \left[\mathbf{A}^n \mathbf{A}^n \frac{\partial(\Delta \underline{U})}{\partial x} + \mathbf{A}^n \mathbf{B}^n \frac{\partial(\Delta \underline{U})}{\partial y}\right] + \frac{\partial}{\partial y} \left[\mathbf{B}^n \mathbf{A}^n \frac{\partial(\Delta \underline{U})}{\partial x} + \mathbf{B}^n \mathbf{B}^n \frac{\partial(\Delta \underline{U})}{\partial y}\right]\right\} \qquad (1.43)$$

$$= - \Delta t \left[\frac{\partial \underline{E}}{\partial x} + \frac{\partial \underline{F}}{\partial y}\right]^{n} + \frac{\Delta t^2}{2} \left\{\frac{\partial}{\partial x} \left[\mathbf{A}\left(\frac{\partial \underline{E}}{\partial x} + \frac{\partial \underline{F}}{\partial y}\right)\right] + \frac{\partial}{\partial y} \left[\mathbf{B}\left(\frac{\partial \underline{E}}{\partial x} + \frac{\partial \underline{F}}{\partial y}\right)\right]\right\}^{n}$$

where

$$\mathbf{A} = \frac{d\underline{E}}{d\underline{U}} \qquad \mathbf{B} = \frac{d\underline{F}}{d\underline{U}} \qquad (1.44)$$

and the approximations $\mathbf{A}^{n+1} \approx \mathbf{A}^{n}$ and $\mathbf{B}^{n+1} \approx \mathbf{B}^{n}$ have been made.

## Galerkin Finite Element Approximation

The problem of determining the $\Delta \underline{U}$ which satisfies equation (1.43), over a domain $\Omega$, can be put into a variational form by defining trial and weighting function sets as before. It will be convenient to assume that the boundary values of $\underline{U}$ on $\Gamma$ are independent of time and that these values are already satisfied by $\underline{U}^n$. Physical boundaries such as solid walls require a different treatment which will not be considered further here. Then we can define

$$\mathcal{T} = \{ \Delta\underline{U} \mid \Delta\underline{U} = 0 \text{ on } \Gamma \} \qquad \mathcal{W} = \{ W \mid W = 0 \text{ on } \Gamma \} \qquad (1.45)$$

and seek, for each n, $\Delta\underline{U}$ in $\mathcal{T}$ such that

$$\int_{\Omega} W \left[ \Delta\underline{U} - \frac{\theta\Delta t^2}{2} \left\{ \frac{\partial}{\partial x} \left[ \mathbf{A}^n\mathbf{A}^n \frac{\partial(\Delta\underline{U})}{\partial x} + \mathbf{A}^n\mathbf{B}^n \frac{\partial(\Delta\underline{U})}{\partial y} \right] \right.\right.$$
$$\left. + \frac{\partial}{\partial y} \left[ \mathbf{B}^n\mathbf{A}^n \frac{\partial(\Delta\underline{U})}{\partial x} + \mathbf{B}^n\mathbf{B}^n \frac{\partial(\Delta\underline{U})}{\partial y} \right] \right\} + \Delta t \left[ \frac{\partial\underline{E}}{\partial x} + \frac{\partial\underline{F}}{\partial y} \right]^n \qquad (1.46)$$
$$\left. - \frac{\Delta t^2}{2} \left\{ \frac{\partial}{\partial x} \left[ \mathbf{A} \left( \frac{\partial\underline{E}}{\partial x} + \frac{\partial\underline{F}}{\partial y} \right) \right] + \frac{\partial}{\partial y} \left[ \mathbf{B} \left( \frac{\partial\underline{E}}{\partial x} + \frac{\partial\underline{F}}{\partial y} \right) \right] \right\}^n \right] d\Omega = 0$$

for all W in $\mathcal{W}$. A weak variational formulation may be produced by using the divergence theorem. The variational statement then becomes : find, for each n, $\Delta\underline{U}$ in $\mathcal{T}$ such that

$$\int_{\Omega} [ W \Delta\underline{U} + \frac{\theta\Delta t^2}{2} \left\{ \frac{\partial W}{\partial x} \left[ \mathbf{A}^n\mathbf{A}^n \frac{\partial(\Delta\underline{U})}{\partial x} + \mathbf{A}^n\mathbf{B}^n \frac{\partial(\Delta\underline{U})}{\partial y} \right] \right.$$
$$\left. + \frac{\partial W}{\partial y} \left[ \mathbf{B}^n\mathbf{A}^n \frac{\partial(\Delta\underline{U})}{\partial x} + \mathbf{B}^n\mathbf{B}^n \frac{\partial(\Delta\underline{U})}{\partial y} \right] \right\} ] \, d\Omega \qquad (1.47)$$
$$- \Delta t \int_{\Omega} W \left[ \frac{\partial\underline{E}}{\partial x} + \frac{\partial\underline{F}}{\partial y} \right]^n d\Omega - \frac{\Delta t^2}{2} \int_{\Omega} \left\{ \frac{\partial W}{\partial x} \mathbf{A}^n + \frac{\partial W}{\partial y} \mathbf{B}^n \right\} \left\{ \frac{\partial\underline{E}}{\partial x} + \frac{\partial\underline{F}}{\partial y} \right\}^n d\Omega$$

We will assume that the spatial solution domain, Ω, has been discretised using 3 noded linear triangular elements, with the interior nodes numbered from 1 to p, and define the sets

$$\mathcal{T}_{(p)} = \{ \Delta\underline{U}_{(p)} \mid \Delta\underline{U}_{(p)} = N_1 \Delta\underline{U}_1 + N_2 \Delta\underline{U}_2 + \ldots\ldots + N_p \Delta\underline{U}_p ;\ \Delta\underline{U}_{(p)} = 0 \text{ on } \Gamma \}$$
$$\mathcal{W}_{(p)} = \{ W_{(p)} \mid W_{(p)} = a_1N_1 + a_2N_2 + \ldots\ldots + a_pN_p ;\ W_{(p)} = 0 \text{ on } \Gamma \} \qquad (1.48)$$

where $a_1, a_2, \ldots, a_p$ are constants. The Galerkin approximation determines the values of the nodal unknowns $\Delta\underline{U}_1, \Delta\underline{U}_2, \ldots\ldots, \Delta\underline{U}_p$ by inserting $\Delta U_{(p)}$ into the equation (1.47) and satisfying the resulting equation for each $W_{(p)}$ in $\mathcal{W}_{(p)}$. When $\Delta\underline{U}_{(p)}$ has been determined, the solution is updated according to $\underline{U}_{(p)}^{n+1} = \underline{U}_{p)}^{n} + \Delta\underline{U}_{(p)}$

## An Explicit Time-Stepping Scheme

If the value of θ is set equal to zero in equation (1.47), the Galerkin statement takes the form

$$\int_{\Omega} N_J \, \Delta \underline{U}_{(p)} \, d\Omega = - \Delta t \int_{\Omega} N_J \left[ \frac{\partial \underline{\mathbf{E}}_{(p)}}{\partial x} + \frac{\partial \underline{E}_{(p)}}{\partial y} \right]^n d\Omega$$

$$- \frac{\Delta t^2}{2} \int_{\Omega} \left\{ \frac{\partial N_J}{\partial x} \mathbf{A}_{(p)}{}^n + \frac{\partial N_J}{\partial y} \mathbf{B}_{(p)}{}^n \right\} \left\{ \frac{\partial \underline{\mathbf{E}}_{(p)}}{\partial x} + \frac{\partial \underline{E}_{(p)}}{\partial y} \right\}^n d\Omega = 0 \qquad (1.49)$$

for $J = 1, 2, \ldots\ldots, p$. Although $\underline{\mathbf{E}}_{(p)}$, $\underline{E}_{(p)}$, $\mathbf{A}_{(p)}$ and $\mathbf{B}_{(p)}$ at time level n could be expressed directly in terms of $\underline{U}_{(p)}{}^n$ and the right hand side of this equation evaluated by numerical integration, a convenient computational implementation is produced by linearly interpolating $\underline{\mathbf{E}}_{(p)}$ and $\underline{E}_{(p)}$ over each element and taking $\mathbf{A}_{(p)}$ and $\mathbf{B}_{(p)}$ to be element-wise constant. The terms in equation (1.49) may now be evaluated exactly, by assembling the contributions from individual elements, and the resulting set of equations written in the form [15]

$$\sum_{K=1}^{p} [\mathbf{M}]_{JK} \, \Delta \underline{U}_K = \underline{RHS}_J \qquad (1.50)$$

where

$$[\mathbf{M}]_{JK} = \int_{\Omega} N_J \, N_K \, d\Omega \qquad (1.51)$$

In the form of equation (1.50), the time-stepping scheme is implicit, as the consistent mass matrix **M** is not diagonal. In transient simulations, this equation is solved by explicit iteration [16]. For steady state problems, the simplification of replacing the consistent mass matrix by the standard lumped (diagonal) matrix [12], with non-zero diagonal entries $[\mathbf{M}_L]_J$, can be made, thus producing an explicit scheme. This explicit scheme is identical to the one step Lax-Wendroff method [10] when it is implemented on a mesh of linear elements in one dimension.

## Artificial Dissipation

For the successful simulation of flows with steep gradients, the explicit scheme described above needs the addition of an appropriate artificial viscosity model. The standard approach is to smooth the computed solution at the end of each time step before proceeding with the computation. The smoothing can be regarded as the application of an explicit diffusion with a suitably defined diffusion coefficient. A convenient method of achieving this effect follows from the observation that, on a mesh of linear elements, there is a relatively simple way of approximating a diffusion operator. At an interior node J on a mesh of linear elements in one dimension [17],

$$[\,(\mathbf{M}_L)^{-1}\sum_E(\mathbf{m}_E-\mathbf{m}_{LE})\,\underline{\mathbf{U}}_E\,]_J \approx \frac{1}{6}\,\frac{d}{dx}\left(h^2\frac{d\underline{U}}{dx}\right)\Bigg|_J \qquad (1.52)$$

where h(x) is the local mesh size. The summation in equation (1.52) represents the operation of assembly of element values and $\mathbf{m}_E$, $\mathbf{m}_{LE}$ and $\mathbf{U}_E$ denote the element mass matrix, element lumped mass matrix and element vector of nodal values of $\underline{U}$ respectively. By direct extension, on a general 2D mesh of linear elements, the smoothing in the case of the Euler equations is accomplished by replacing the computed nodal values $\underline{U}_J^{n+1}$ by smoothed values $\underline{U}_{SJ}^{n+1}$ according to

$$[\mathbf{M}_L]_J\,(\underline{U}_{SJ}^{n+1}-\underline{U}_J^{n+1}) = \sum_E k_E\,\{[\mathbf{m}_E]_{JK}-[\mathbf{m}_{LE}]_{JK}\}\,\underline{U}_{EK}^{n+1} \qquad (1.53)$$

Here $k_E$ is a pressure-switched artificial diffusion coefficient, which is computed as the mean of element nodal values $k_J$ defined according to

$$k_J = C_\upsilon\sum_E \frac{\{[\mathbf{m}_E]_{JK}-[\mathbf{m}_{LE}]_{JK}\}\,\underline{p}_{EK}}{|\{[\mathbf{m}_E]_{JK}-[\mathbf{m}_{LE}]_{JK}\}\,\underline{p}_{EK}\,|} \qquad (1.54)$$

where $C_\upsilon$ is a user specified constant, $\underline{p}_E$ is the vector of element nodal pressures and | . | denotes that absolute values of the element contributions are assembled. The coefficient $k_J$ takes values ranging from 0 to 1 and it can be shown that $k_J = 1$ when the pressure has a local extremum at node J. In equations (1.53) and (1.54) the summation extends over all elements E which belong to node J. The effectiveness of this artificial viscosity model, when it is applied in conjunction with the explicit time stepping procedure outlined above, is illustrated in figure 1.9 which shows the result of a computation of the steady flow past a circular cylinder, at a free stream Mach number of 3. Further examples, including a two step variant of the solution algorithm, can be found in a number of publications [18,19].

A High Resolution Extension

The easiest method of producing a scheme of higher resolution on a general unstructured grid is to apply the flux corrected transport (FCT) ideas of Boris and Book [20] and Zalesak [21]. In their notation, we identify the basic explicit scheme with no added artificial viscosity as the higher order solution, while the basic scheme plus the addition of a hefty amount of artificial viscosity is the lower order solution. The low order scheme must have the property that it produces monotonic solutions for the problem under investigation. By combining the two schemes, the objective is to remove as much of the artificial viscosity as possible, while still maintaining monotonicity. This is achieved by limiting the contributions made by the individual elements to the amount of added artificial viscosity. The first triangular grid implementation of these ideas was made by Parrott and Christie [22], in the context of a single transport equation, and applications to the Euler equations have also been made [23,24]. A demonstration of the effectiveness of the procedure is given in figure 1.10, which shows the results produced when the method is applied to the solution of transport in a rotational velocity field. In figure 1.10a we see the initial concentration, which takes the form of a circular cylinder with a cut-out. In

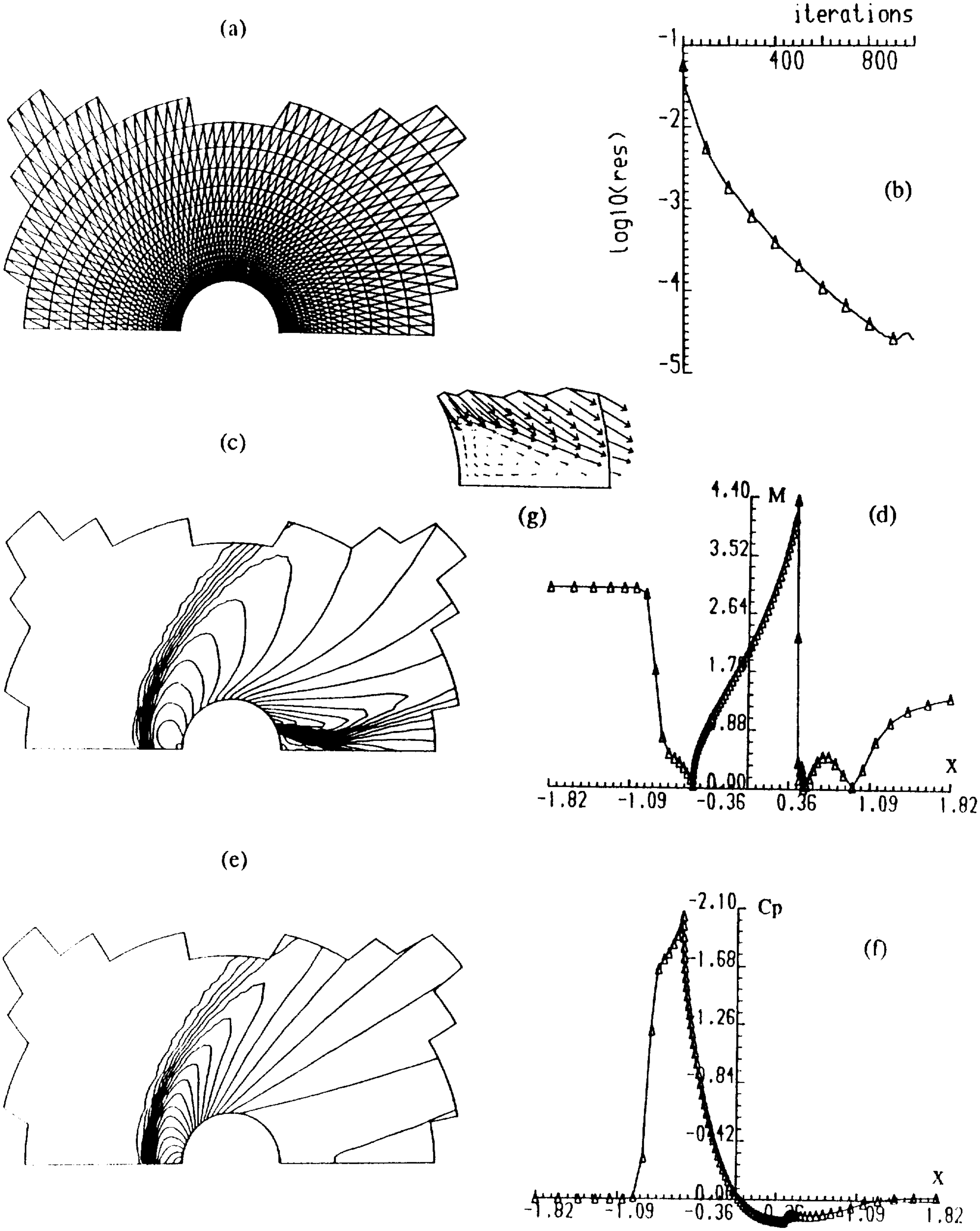


**Figure 1.9** External flow past a circular cylinder ($M_\infty = 3$)
(a) Enlargement of the mesh near the cylinder
(b) Convergence curve of the $L_2$ norm of the density residual
(c), (d) Contours and wall plots of the Mach number
(e), (f) Contours and wall plots of the pressure coefficient
(g) Flow detail behind the cylinder

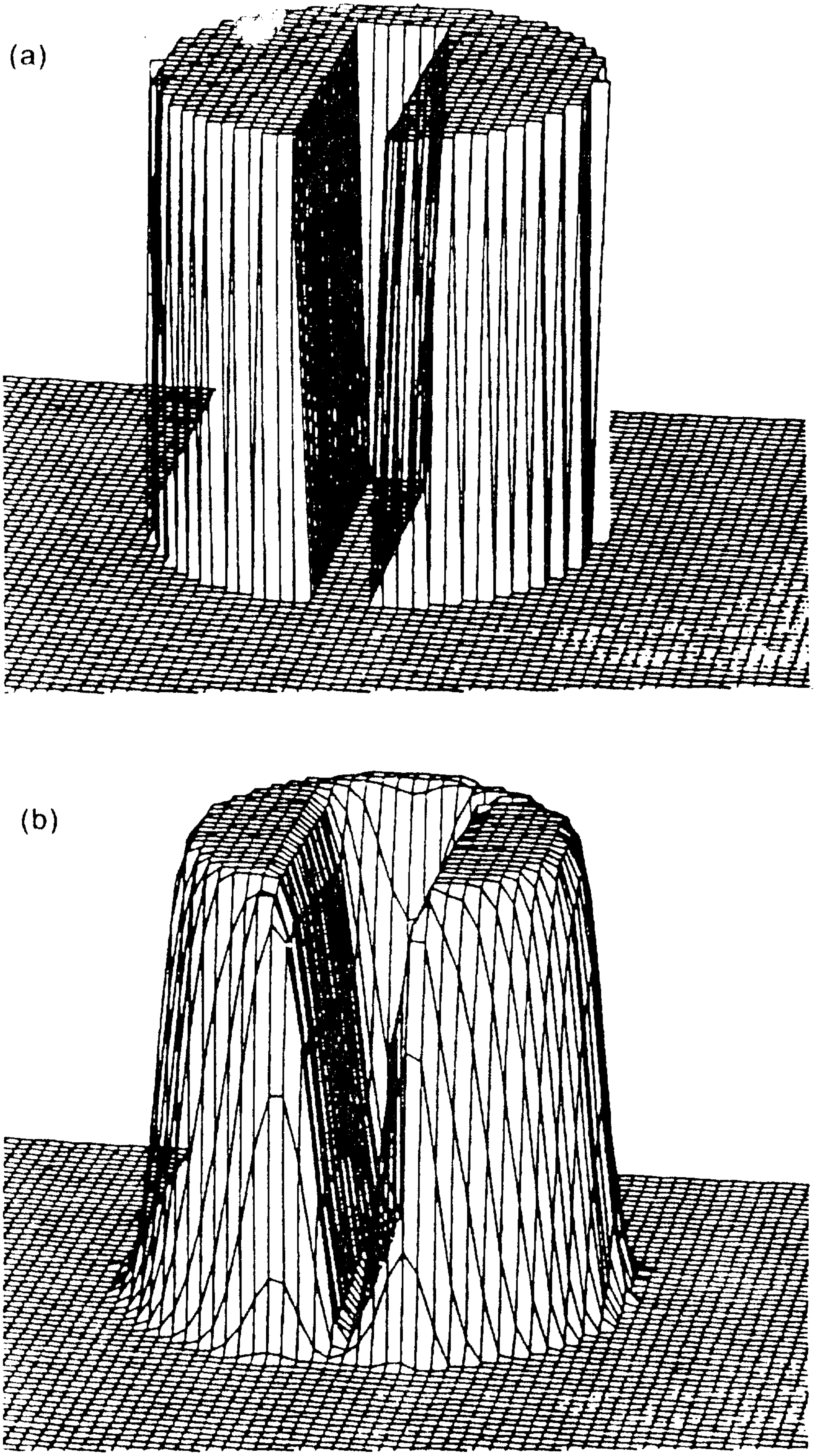


**Figure 1.10** FCT algorithm
(a) Initial solution
(b) Solution after 628 timesteps (1/4 revolution)

figure 1.10b we show the computed profile after 628 time steps (one quarter rotation) of the FCT algorithm. It is observed from the results that accuracy of this scheme compares well with more costly Riemann-solver based schemes.

### An Implicit Time-Stepping Scheme

If a value of $\theta$ other than zero in used in equation (1.47), the Galerkin statement leads to an implicit procedure [14] of the form

$$\sum_{K=1}^{p} [L]_{JK} \, \Delta \underline{U}_K = \underline{RHS}_J \tag{1.55}$$

where L can be expressed as

$$L = M + \theta K \tag{1.56}$$

For computational efficiency, in steady state simulations, the simplification of neglecting the cross derivative terms in the construction of K is generally made and this does not seriously affect the convergence behaviour of the method. Schemes of this type appear to possess certain desirable features e.g. it may be possible to apply them in the simulation of transonic flows without the necessity for introducing artificial diffusion [25]. As the identification of appropriate smoothing mechanisms for elements which are higher than linear in order is still an unsolved problem, this approach would then allow the possibility of producing an algorithm which could be implemented on quadratic elements. This would certainly have associated accuracy benefits on general grids.

However, although the direct solution of the equation system (1.55) can be contemplated for certain two dimensional flows, efficient iterative techniques are needed if an implicit approach is to be adopted for any realistic three dimensional simulation. Iterative strategies based upon the use of line relaxation have met with certain success within the context of finite difference methods on structured grids [10]. On such grids, the grid lines themselves can serve as appropriate lines for the relaxation process. To describe such a process, it is useful to introduce a renumbering operator $\vartheta$ associated to a line such that $\vartheta(\underline{U})$ denotes the vector of nodal unknowns re-arranged in the order implied by the line. The equation system (1.55) is then written as

$$\sum_{K=1}^{p} \vartheta([L]_{JK})\vartheta^{-1} \, \vartheta(\Delta \underline{U}_K) = \vartheta(\underline{RHS}_J) \tag{1.57}$$

which leads to a new equation system of the form

$$\sum_{J=1}^{p} [L^*]_{JK} \, \Delta \underline{U}_K^* = \underline{RHS}_J^* \tag{1.58}$$

If the matrix $\mathbf{L}^*$ is now decomposed as

$$\mathbf{L}^* = \mathbf{D}^* + (\mathbf{L}^* - \mathbf{D}^*) \qquad (1.59)$$

where $\mathbf{D}^*$ is the block tri-diagonal of $\mathbf{L}^*$, this would allow the use of the relaxation scheme

$$\sum_{K=1}^{p} [\mathbf{D}^*]_{JK} \;\; \Delta \underline{\mathbf{U}}_K^{*(r+1)} = \underline{\mathbf{RHS}}_J^* - \sum_{K=1}^{p} ([\mathbf{L}^*]_{JK} - [\mathbf{D}^*]_{JK}) \; \Delta \underline{\mathbf{U}}_K^{*(r)} \qquad (1.60)$$

where $\Delta \underline{\mathbf{U}}_K^{*(0)}$ is taken to be the last computed increment and r is the iteration count. A possible two dimensional implementation would therefore use the two families of mesh lines, with one iteration on each line every timestep. The block tri-diagonal matrices $\mathbf{D}^*$ can be factorised every timestep or there is the possibility of using the same factors for several iterations. A further refinement would be the incorporation of a line search minimisation.

On unstructured triangular grids, the identification of suitable lines, for use with a line relaxation scheme of this type, is not immediate. Recently [26], it has been shown that it is possible to employ a rather simple mesh processing algorithm to accomplish this task.

An Algorithm for Constructing Lines in an Unstructured Triangular Mesh

Here we describe a general procedure for constructing a line (or lines) through a general triangular mesh which passes through each node of the mesh once only. The elements, nodes and sides in the mesh are numbered. The algorithm to be described will then make use of the following connectivity arrays, which need to be determined before the process begins:

| | |
|---|---|
| IEP(IN=1:3;IE=1:NE) | The three nodes of element IE. |
| IEE(JE=1:3;IE=1:NE) | The three elements sharing the sides of element IE. |
| IES(IS=1:3;IE=1:NE) | The three sides of element IE. |

In addition, the following arrays will be used during the construction process:

| | |
|---|---|
| LPM(1:NP) | Node marker. |
| LEM(1:NE) | Element marker. |
| LSM(1:NS) | Side marker. |

and a list LSE, which is initially empty (NLSE=0), which contains the so-called active elements, in ascending order of a specified key variable . The elements in the list are kept in a heap data structure. Given a prescribed direction **n**, the recursive algorithm proceeds as follows:

```
1.- Set LPM(1:NP) = 0
    Set LEM(1:NE) = 0
    Set LSM(1:NS) = 0
```

2.- *Select* any element as a starting element IE.

3.- *Let* IPi, ISi and IEi for i = 1:3 be the three points, sides and surrounding elements of this element IE.

```
IF(LPM(IPi) $\neq$ 1 for any i = 1:3) THEN
   FOR i = 1:3 DO
        LSM(ISi) = LSM(ISi) + 1
        LPM(IPi) = 1
        IF(LEM(IEi) = 0) THEN
           LEM(IEi) = 1
           Insert element IEi into LSE according to KEYi
           NLSE = NLSE+1
        ENDIF
   ENDDO
ENDIF
```

4.- *Select* a new element IE

```
IF(NLSE $\neq$ 0) THEN
   NLSE = NLSE-1
   IE = LSE(1)
   GO TO 3
ELSE
   End process. The sides IS for which LSM(IS) = 1 form the line.
ENDIF
```

In the above KEYi is evaluated as the absolute value of the scalar product between $\underline{n}$ and the unit vector in the direction joining the centroids of elements IE and IEi.

The operation of this algorithm is illustrated by considering the mesh of 18 triangular elements with 16 nodes shown in figure 1.11a. In this example directionality is not taken into account, which means that elements are inserted into the list LSE without any directional preference. Starting from any element e.g. element 11, the side marker for sides 8-9, 9-11 and 11-8 is set to 1. At this stage, the line consists of these three sides. The node marker for nodes 8, 9 and 11 is set to one and the adjacent elements 10, 12 and 6 are marked and inserted into LSE. NLSE is now equal to 3. The first element in LSE, which is element 10, is selected and, as one of its points is not marked, the marker of its sides is incremented by one. This means that the common side with element 11, i.e side 11-8, is deleted from the current line and the sides 7-8 and 11-7 are added to the line. The marker for node 7 is set to one and the unmarked adjacent elements 3 and 9 are marked and inserted into LSE. The situation is then as shown in figure 1.11b. The procedure is continued, with the next stage being illustrated in figure 1.11c, until all the elements have been considered, i.e. NLSE is equal to zero. At this point, a line will have been constructed which passes through each node in the grid once and only once, as shown in figure 1.11d.

When the above algorithm is applied to the triangular mesh shown in figure 1.12a, it produces the lines shown in figures 1.12b and 1.12c, when two directions at right angles to each other are prescribed.

An example is given which illustrates the numerical performance of the fully implicit algorithm of equation (1.55) when the solution is obtained via the relaxation procedure of equation (1.60). The example consists of an inviscid flow

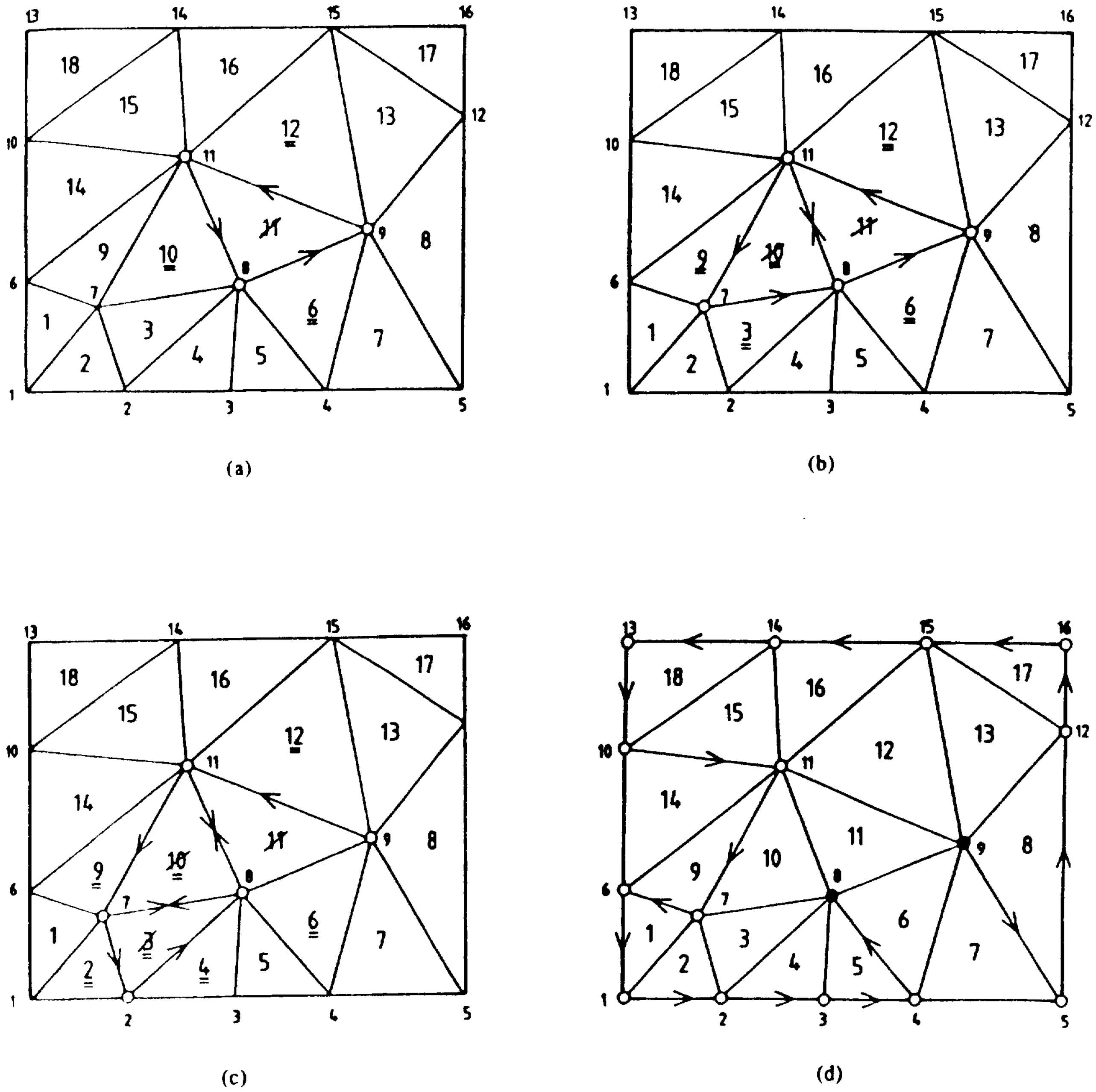


Figure 1.11 The construction of a relaxation line on an unstructured mesh showing (a) starting line (b) line after one stage (c) line after two stages (d) final line.

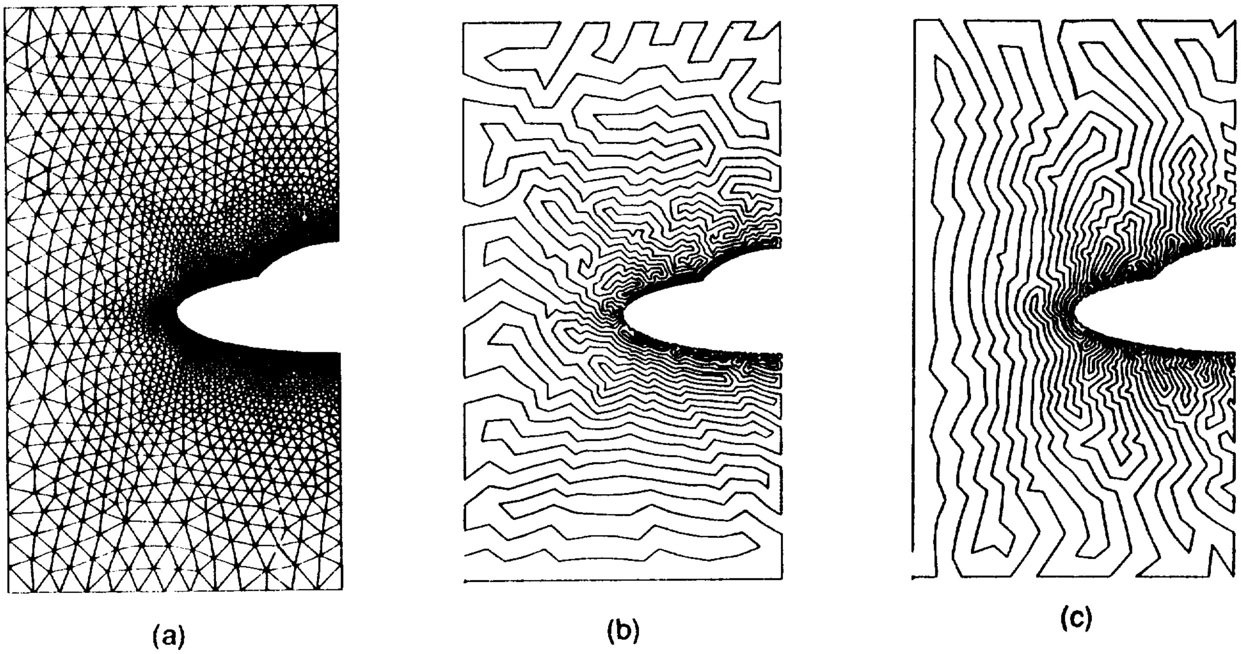


**Figure 1.12** Unstructured line relaxation (a) unstructured mesh (b) 'horizontal' line (c) 'vertical' line.

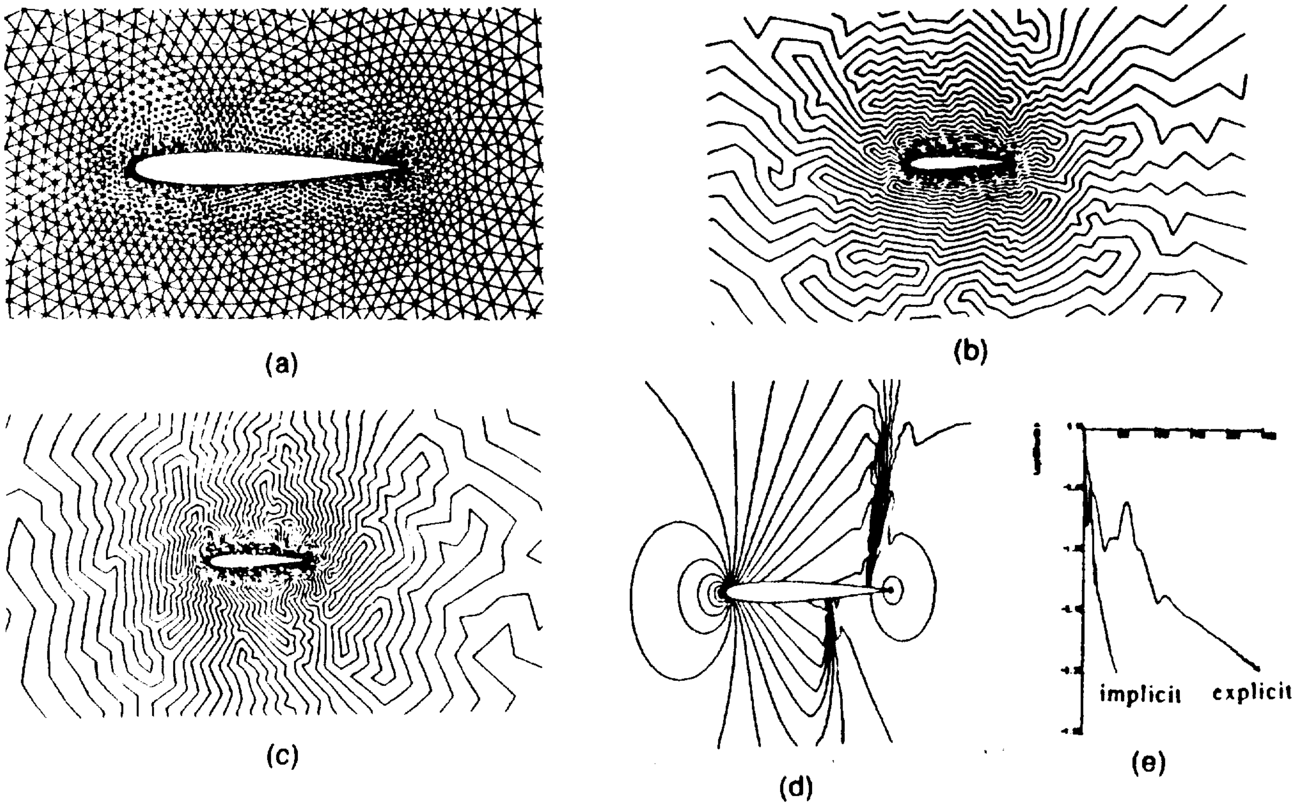


**Figure 1.13** Unstructured line relaxation (a) detail of the mesh (b) 'horizontal' line (c) 'vertical' line (d) Steady state pressure contours (e) convergence curve.

over a NACA0012 airfoil with a free stream Mach number of 0.85 and $1^{0}$ angle of attack of one degree. The unstructured triangular mesh employed consists of 8378 elements and 4292 points and a detail of the mesh is given in figure 1.13a. Details of the two lines used for the line relaxation procedure are shown in figures 1.13b and 1.13c. The computed steady state pressure contours are shown in figure 1.13d. The convergence behaviour of the implicit procedure is compared with that of the explicit approach of equation (1.50) in figure 1.13e.

# 2. GEOMETRY MODELLING

The problem of producing an unstructured mesh over a general computational domain will now be addressed. The boundary of the domain to be discretised needs to be represented in a suitable manner before the generation procedure can start. If the automatic discretisation of an arbitrary domain is to be achieved, the mathematical description of the domain topology ought to possess the greatest possible generality. The computer implementation of this description must provide means for automatically computing any geometrical quantity relevant to the generation procedure. Solid modelling provides [27] the most general up-to-date set of methods for the computational representation and analysis of general shapes matching the above requirements.

In this section we give a brief description of the geometry modelling strategy that we employ. More sophisticated representations giving more accurate definitions as well as easing the task of performing quick geometry modifications could be used [28].

In the planar two dimensional case, the boundary is represented by closed loops of orientated composite cubic spline curves [28]. For simply connected domains these boundary curves are orientated in a counterclockwise sense while for multiply-connected regions the exterior boundary curves are given a counter-clockwise orientation and all the interior boundary curves are orientated in a clockwise sense (figure 2.1).

In three dimensions, the domain to be discretised is viewed as a region bounded by surfaces which intersect along curves. The portions of these curves and surfaces needed to define the three dimensional domain of interest are called *curve* and *surface components* respectively. Figure 2.2 shows the decomposition of the boundary of a three dimensional domain into its surface and curve components. The approximate representation of the boundary components is accomplished by means of composite curves and surfaces [28]. In addition, boundary curves and surfaces are orientated (see figure 2.3). This is important in the generating process as it is used to define the location of the region that is to be discretised. The orientation of a boundary surface is defined by the direction of the inward normal. The orientation of the boundary curves is defined with respect to the boundary surfaces which contain them. Each boundary curve will be common to two boundary surfaces and will have opposite orientations with respect to each of them. An example of the approximated geometry for a surface component and its corresponding curve components is displayed in figure 2.4. It can be observed that the boundaries of the interpolated composite surface can be arbitrary and are not required to coincide with those of the surface component.

### *2.1 Curve Representation*

The parametric definition of a curve consists of a piecewise interpolation of cubic polynomials through an ordered set of data points. The order in which these points are given defines the orientation. In the Ferguson representation [29], each cubic polynomial is expressed, in terms of the position and tangent vectors at the two end points, as

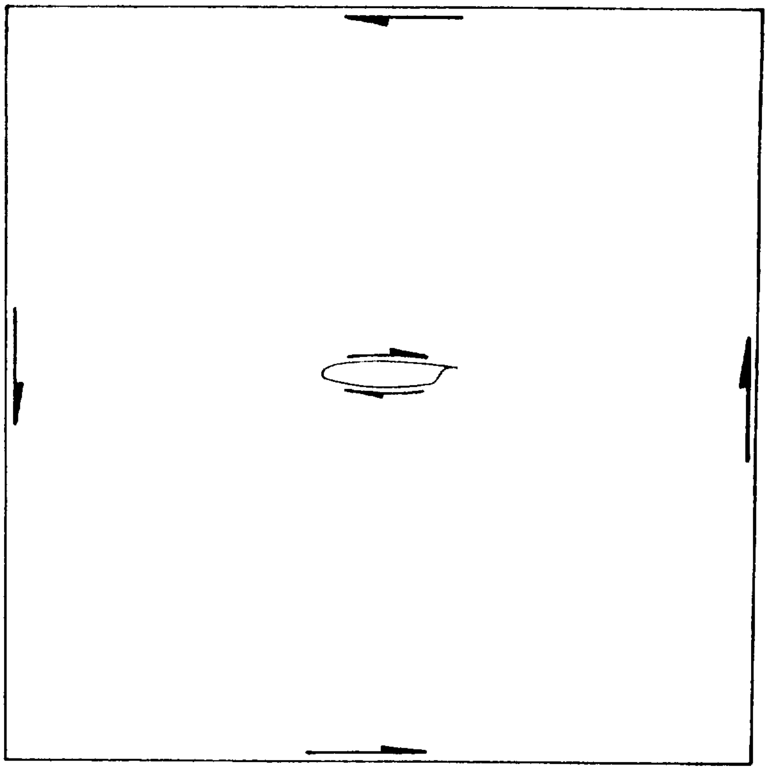


Figure 2.1 Boundary orientation for a two dimensional domain.

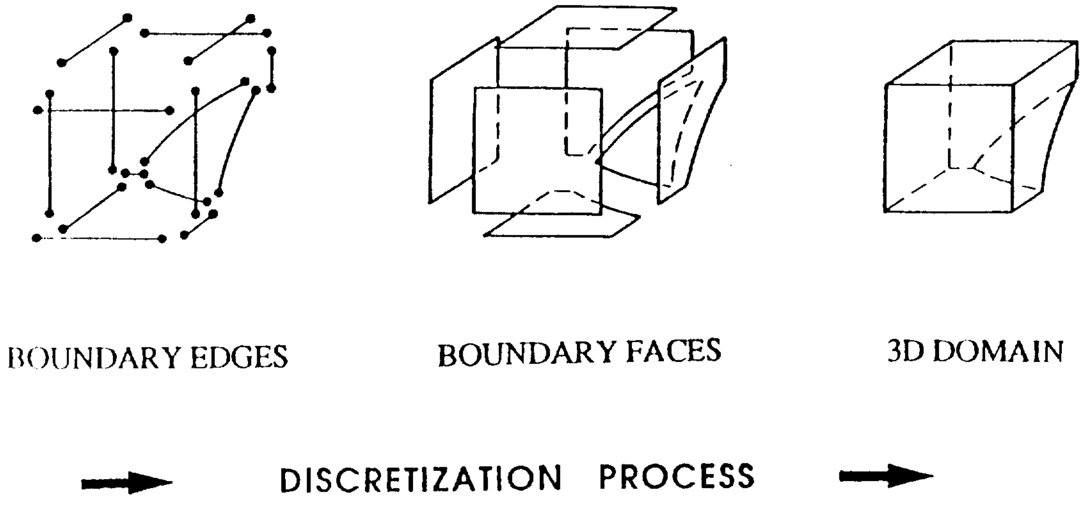


Figure 2.2 Decomposition of the boundary of a three dimensional domain into its surface and curve components.

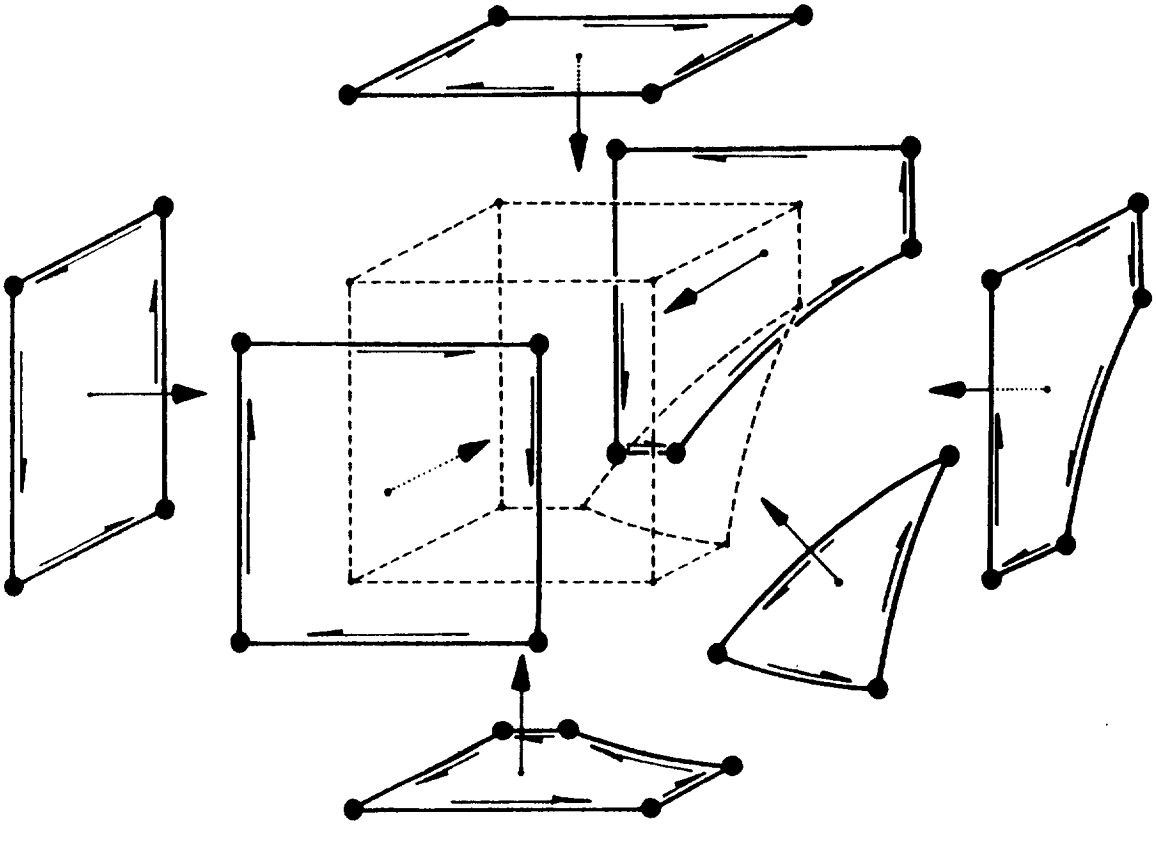


**Figure 2.3** Orientation of the boundary components in three dimensions.

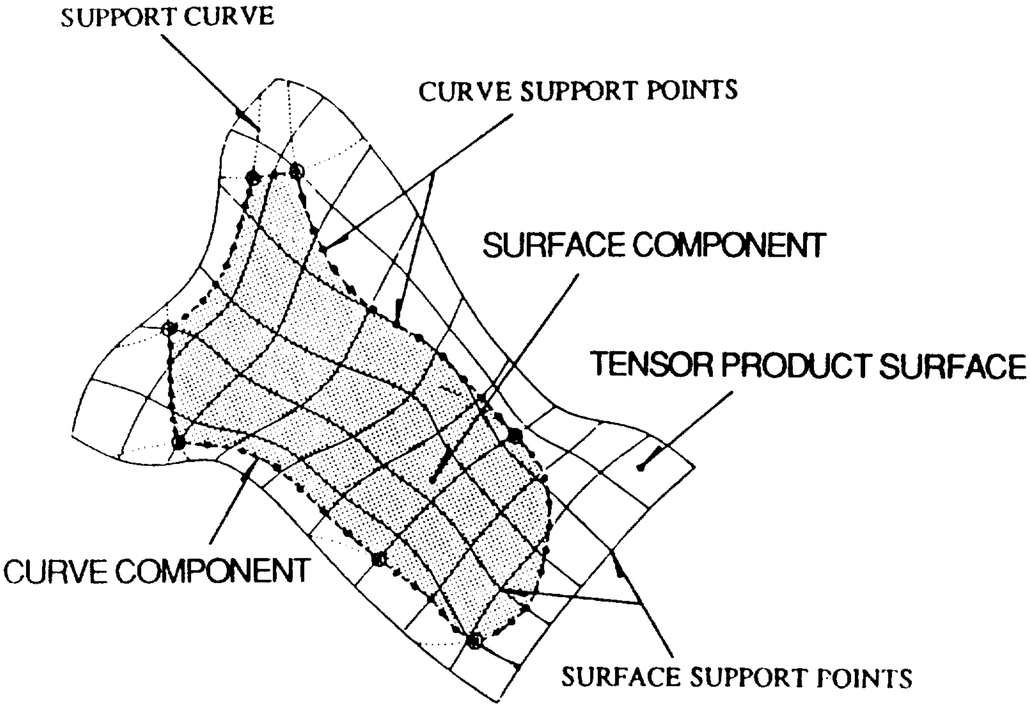


Figure 2.4 Approximated geometry for a surface component.

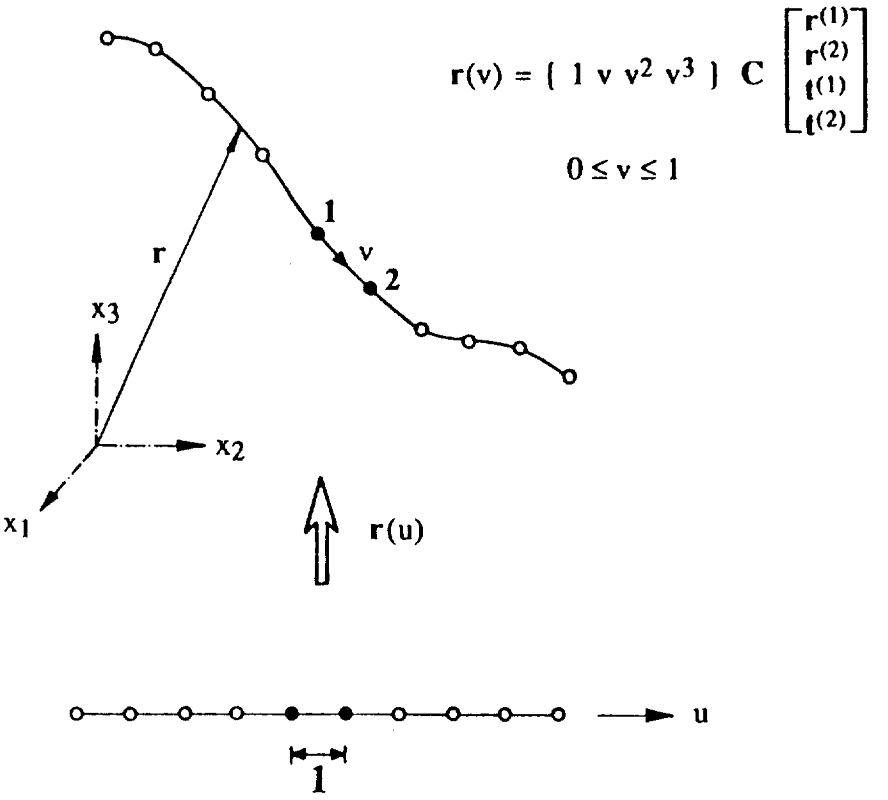

$r(v) = \{\ 1\ v\ v^2\ v^3\ \}\ C \begin{bmatrix} r^{(1)} \\ r^{(2)} \\ t^{(1)} \\ t^{(2)} \end{bmatrix}$
$0 \le v \le 1$
1
v
2
r
$x_3$
$x_2$
$x_1$
r(u)
u
1

Figure 2.5 Interpolation of a piecewise cubic spline through a set of data points.

$$\underline{r}(v) = \{\ 1\ v\ v^2\ v^3\ \}\ \mathbf{C} \begin{bmatrix} \underline{r}^{(1)} \\ \underline{r}^{(2)} \\ \underline{t}^{(1)} \\ \underline{t}^{(2)} \end{bmatrix} \qquad 0 \leq v \leq 1 \qquad (2.1)$$

where $\underline{r}^{(1)}$ and $\underline{r}^{(2)}$ are the coordinates of the end points of the segment, $\underline{t}^{(1)}$ and $\underline{t}^{(2)}$ are their respective tangents and **C** is a constant matrix given by

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & 1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \qquad (2.2)$$

The tangent to the curve is computed according to

$$\underline{t}(v) = \underline{r}'(v) = \frac{d\underline{r}(v)}{dv} \qquad (2.3)$$

The number of data points, and their spatial distribution, should be given in such a manner that the interpolated curve accurately approximates the intersection of the corresponding surface components. The interpolation problem, which is illustrated in figure 2.5, consists of fitting a parametric spline, defined in a piecewise manner, through a set of n points $\underline{r}_j$; j=1,...,n. At interior points, continuity of slopes is guaranteed for any choice of the tangent vectors, provided that a unique tangent vector is used for the definition of the two adjacent cubic segments. However, by employing a simple procedure [28], these vectors can be determined so that continuity of curvature is achieved throughout the interpolated curve. At the two end points, zero curvature is assumed. Note that the expressions given above are valid in two and three dimensions. The only difference in the two cases being the number of components of the vectors $\underline{r}$ and $\underline{t}$.

### *2.2 Surface Representation*

The mathematical representation of a surface is obtained by interpolating a composite surface, made up of quadrilateral patches, through a topologically rectangular set of data points $\underline{r}_{jk}$; j=1,...,m ; k=1,...n (see figure 2.6). Two families of parametric lines are obtained by interpolating spline curves, first through the points of constant j and then through the points of constant k. The procedure used for interpolating each spline curve is that described in the previous section. The mathematical expression for every quadrilateral surface patch is given, in terms of the four cubic curves that form its boundary and the twist vector at the four corner points, as

$$\underline{r}(v,w) = (1\ v\ v^2\ v^3\ )\ \mathbf{C} \begin{bmatrix} \underline{r}^{(1)} & \underline{r}^{(4)} & \underline{r}_{,w}^{(1)} & \underline{r}_{,w}^{(4)} \\ \underline{r}^{(2)} & \underline{r}^{(3)} & \underline{r}_{,w}^{(2)} & \underline{r}_{,w}^{(3)} \\ \underline{r}_{,v}^{(1)} & \underline{r}_{,v}^{(4)} & \underline{r}_{,vw}^{(1)} & \underline{r}_{,vw}^{(4)} \\ \underline{r}_{,v}^{(2)} & \underline{r}_{,v}^{(3)} & \underline{r}_{,vw}^{(2)} & \underline{r}_{,vw}^{(3)} \end{bmatrix} \mathbf{C}^T \begin{bmatrix} 1 \\ w \\ w^2 \\ w^3 \end{bmatrix} \qquad (2.4)$$

$$0 \leq v \leq 1;\ 0 \leq w \leq 1$$

$$r(v,w) = (1\ v\ v^2\ v^3)\ \mathbf{C} \begin{bmatrix} \mathbf{r}^{(1)} & \mathbf{r}^{(4)} & \mathbf{r}_{,w}^{(1)} & \mathbf{r}_{,w}^{(4)} \\ \mathbf{r}^{(2)} & \mathbf{r}^{(3)} & \mathbf{r}_{,w}^{(2)} & \mathbf{r}_{,w}^{(3)} \\ \mathbf{r}_{,v}^{(1)} & \mathbf{r}_{,v}^{(4)} & \mathbf{r}_{,vw}^{(1)} & \mathbf{r}_{,vw}^{(4)} \\ \mathbf{r}_{,v}^{(2)} & \mathbf{r}_{,v}^{(3)} & \mathbf{r}_{,vw}^{(2)} & \mathbf{r}_{,vw}^{(3)} \end{bmatrix} \mathbf{C}^t \begin{bmatrix} 1 \\ w \\ w^2 \\ w^3 \end{bmatrix}$$

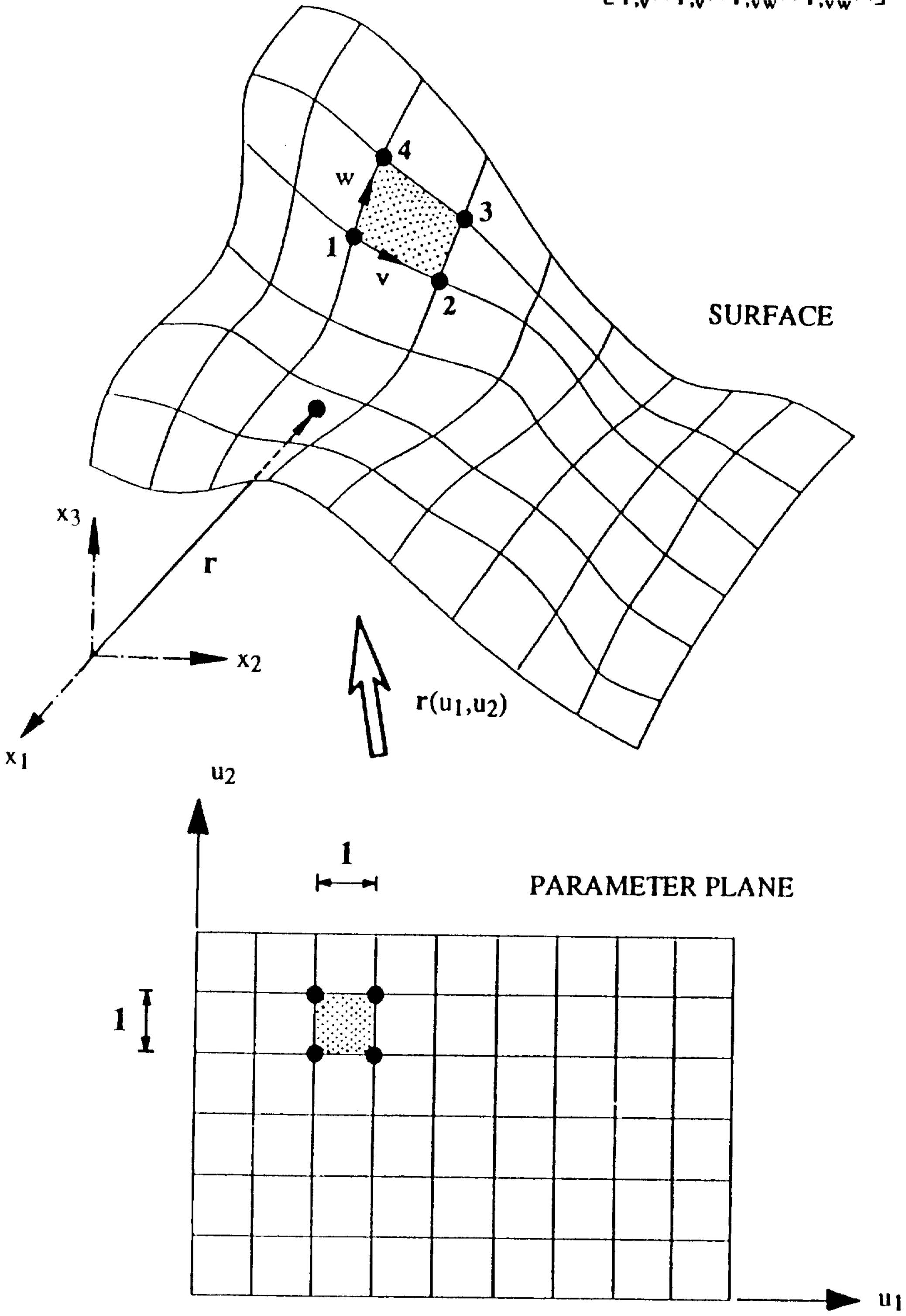


Figure 2.6 Interpolation of a composite surface through a set of data points.

where **C** is the matrix previously defined in (2.2), $\mathbf{C}^T$ is its transpose, and the comma denotes partial differentiation, i.e.,

$$\underline{r}_{,v} = \frac{\partial \underline{r}}{\partial v} \; ; \qquad \underline{r}_{w} = \frac{\partial \underline{r}}{\partial w} \; ; \qquad \underline{r}_{,vw} = \frac{\partial^2 \underline{r}}{\partial v \partial w} \tag{2.5}$$

Here the notation employed to denote the corner points of the patch is

$$\underline{r}^{(1)} = \underline{r}(0,0); \quad \underline{r}^{(2)} = \underline{r}(1,0); \quad \underline{r}^{(3)} = \underline{r}(1,1); \quad \underline{r}^{(4)} = \underline{r}(0,1) \tag{2.6}$$

This representation uses a Hermite interpolation between opposite boundaries of the patch [30]. The twist vectors $(\underline{r}_{,vw})_{jk}$ at the corner points are computed so that overall second order continuity is achieved on the interpolated surface. The implementation details of this algorithm can be found in [31]. For convenience, global parametric coordinates $u^1$ and $u^2$ are defined. For the patch (j,k) these coordinates are related to the local patch coordinates v and w according to $u^1 = v + j - 1$ and $u^2 = w + k - 1$. In this way, a global mapping $\underline{r}(u^1,u^2)$ is established between the rectangular region in the parametric plane defined by $0 \leq u^1 \leq p$ ; $0 \leq u^2 \leq q$ and the tensor product surface. The orientation of the surface is defined by specifying the outward normal which points towards the region to be discretised.

# 3. UNSTRUCTURED MESH GENERATION BY THE ADVANCING FRONT METHOD

## *3.1 The advancing front technique*

The algorithmic procedure to be described for the mesh generation is based upon the method originally proposed in [5] for two dimensions and then extended to three dimensions in [7,18]. The advocated approach is regarded as a generalization of the advancing front technique [32,33] with the distinctive feature that elements, i.e. triangles or tetrahedra, and points are generated simultaneously. This enables the generation of elements of variable size and stretching and differs from the approach followed in tetrahedral generators which are based upon Delaunay concepts [4,34], which generally connect grid points which have already been distributed in space.

The generation problem consists of subdividing an arbitrarily complex domain into a consistent assembly of elements. The consistency of the generated mesh is guaranteed if the generated elements cover the entire domain and the intersection between elements occurs only on common points, sides or triangular faces in the three dimensional case. The final mesh is constructed in a bottom-up manner. The process starts by discretising each boundary curve. Nodes are placed on the boundary curve components and then contiguous nodes are joined with straight line segments. In later stages of the generation process, these segments will become sides of some triangles. The length of these segments must therefore, be consistent with the desired local distribution of mesh size. This operation is repeated for each boundary curve in turn.

The next stage consists of generating triangular planar faces. For each two dimensional region or surface to be discretised, all the edges produced when discretising its boundary curves are assembled into the so called initial front. The relative orientation of the curve components with respect to the surface must be taken into account in order to give the correct orientation to the sides in the initial front. The front is a dynamic data structure which changes continuously during the generation process. At any given time, the front contains the set of all the sides which are currently available to form a triangular face. A side is selected from the front and a triangular element is generated. This may involve creating a new node or simply connecting to an existing one. After the triangle has been generated, the front is updated and the generation proceeds until the front is empty. Figure 3.1 illustrates the idea of the advancing front technique for a circular planar domain by showing the initial front and the form of the mesh at various stages during the generation process. The size and shape of the generated triangles must be consistent with the local desired size and shape of the final mesh. In the three dimensional case, these triangles will become faces of the tetrahedra to be generated later.

For the generation of tetrahedra the advancing front procedure is taken one step further. The front is now made up of the triangular faces which are available to form a tetrahedron. The initial front is obtained by assembling the triangulations of the boundary surfaces. Nodes and elements will be simultaneously created. When forming a new tetrahedron, the three nodes belonging to a triangular face from the front are connected either to an existing node or to a new node. After generating a tetrahedron, the front is updated. The generation procedure is completed when the number of triangles in the front is zero.

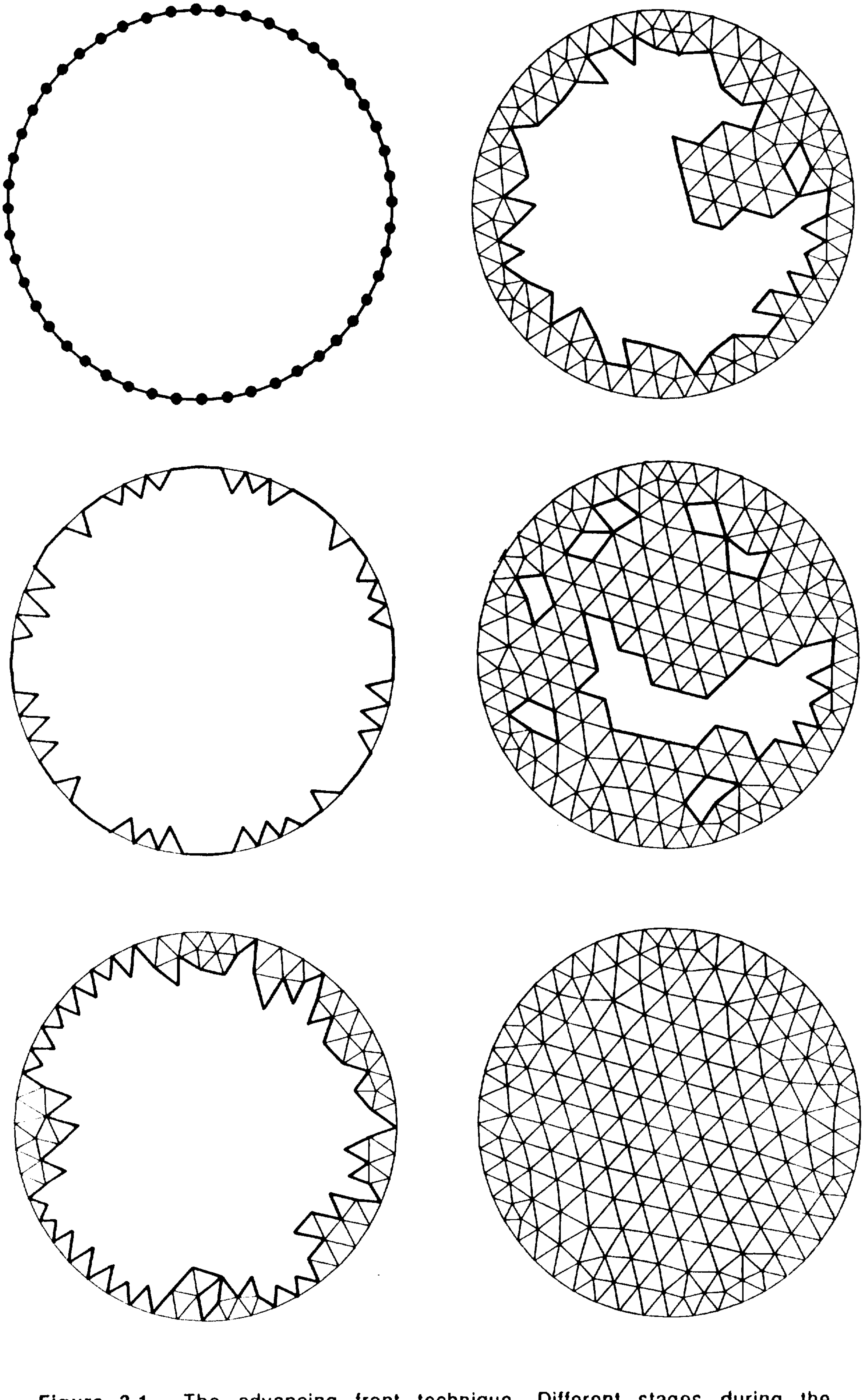


**Figure 3.1** The advancing front technique. Different stages during the triangulation process.

### *3.2 Characterisation of the mesh: mesh parameters*

The geometrical characteristics of a general mesh are locally defined in terms of certain *mesh parameters*. If N (=2 or 3), is the number of dimensions then, the parameters used are a set of N mutually orthogonal directions $\underline{\alpha}_i$; i=1, ... N, and N associated element sizes $\delta_i$; i=1, ... N (see figure 3.2). Thus, at a certain point, if all N element sizes are equal, the mesh in the vicinity of that point will consist of approximately equilateral elements. To aid the mesh generation procedure, a transformation T which is a function of $\underline{\alpha}_i$ and $\delta_i$ is defined. This transformation is represented by a symmetric N x N matrix and maps the physical space onto a space in which elements, in the neighbourhood of the point being considered, will be approximately equilateral with unit average size. This new space will be referred to as the normalised space. For a general mesh this transformation will be a function of position. The transformation **T** is the result of superimposing N scaling operations with factors $1/\delta_i$ in each $\underline{\alpha}_i$ direction. Thus

$$T\,(\underline{\alpha}_i\,,\delta_i\,) = \sum_{i=1}^{N} \frac{1}{\delta_i}\,\underline{\alpha}_i \otimes \underline{\alpha}_i \qquad (3.1)$$

where ⊗ denotes the tensor product of two vectors. The effect of this transformation in two dimensions is illustrated in figure 3.3 for the case of constant mesh parameters throughout the domain.

### *3.3 Mesh Control: The Background Mesh*

The inclusion of adequate mesh control is a key ingredient in ensuring the generation of a mesh of the desired form. Control over the characteristics is obtained by the specification of a spatial distribution of mesh parameters by means of a *background mesh*. The background mesh is used for interpolation purposes only and is made up of triangles in two dimensions and tetrahedra in three dimensions. Values of $\underline{\alpha}_i$ and $\delta_i$, and hence T, are defined at the nodes of the background mesh. At any point within an element of the background grid, the transformation **T** is computed by linearly interpolating its components from the element nodal values. The background mesh employed must cover the region to be discretised (see figure 3.4). In the generation of an initial mesh for the analysis of a particular problem, the background mesh will usually consist of a small number of elements. The generation of the background mesh can in this case be accomplished without resorting to sophisticated procedures e.g. a background mesh consisting of a single element can be used to impose the requirement of linear or constant spacing and stretching through the computational domain. The generation process is always carried out in the normalised space. The transformation T is repeatedly used to transform regions in the physical space into regions in the normalised space. In this way the process is greatly simplified, as the desired size for a side, triangle or tetrahedra in this space is always unity. After the element has been generated, the coordinates of the newly created point, if any, are transformed back to the physical space using the inverse transformation. The effect of prescribing a variable mesh spacing and stretching is illustrated in figure 3.5 for a rectangular domain and using a background grid consisting of two triangular elements.

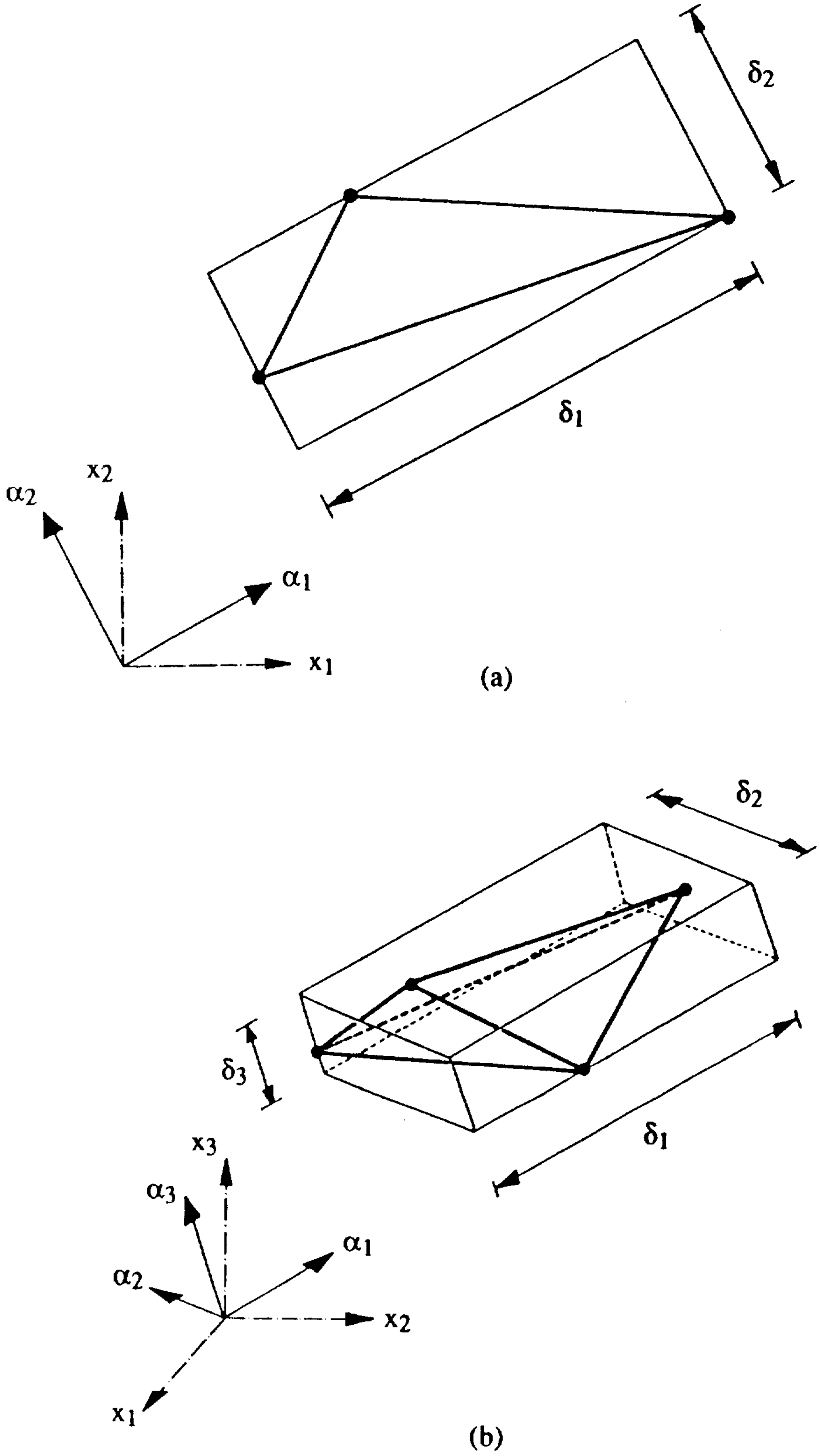


**Figure 3.2** Characterisation of the mesh.

Mesh parameters: (a) In two dimensions.
(b) In three dimensions.

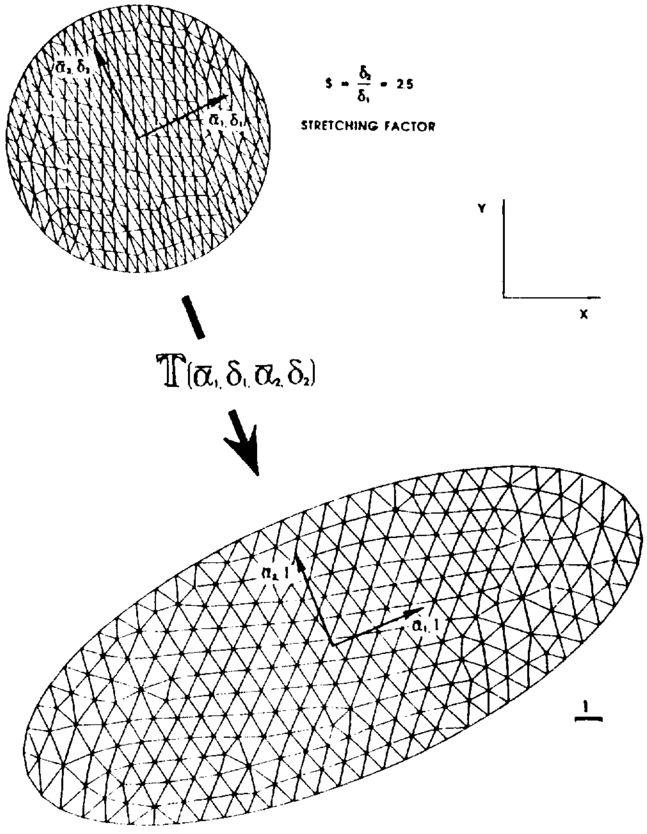

$\alpha_2, \delta_2$
$\alpha_1, \delta_1$
$s = \frac{\delta_2}{\delta_1} = 25$
STRETCHING FACTOR
Y
X
$\mathbb{T}(\alpha_1, \delta_1, \alpha_2, \delta_2)$
$\alpha_2, 1$
$\alpha_1, 1$
1

Figure 3.3 The effect of the transformation T for a constant distribution of the mesh parameters.

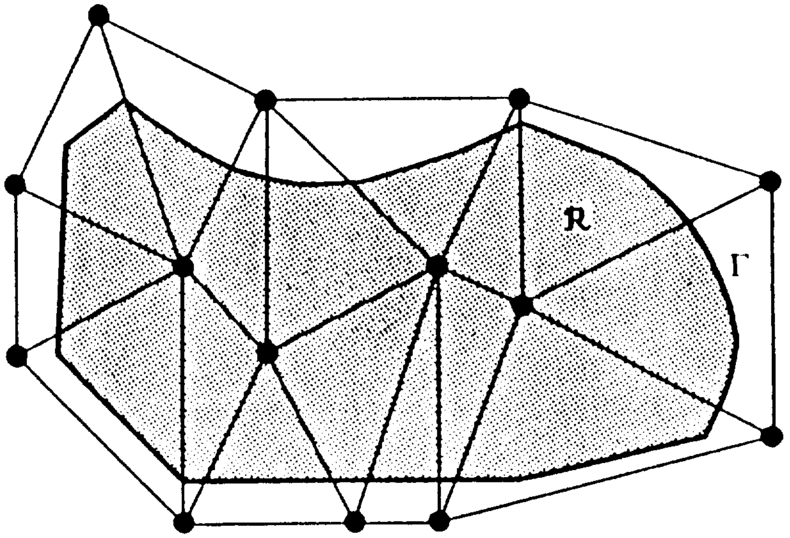


Figure 3.4 Specification of a spatial distribution of mesh parameters. Background mesh.

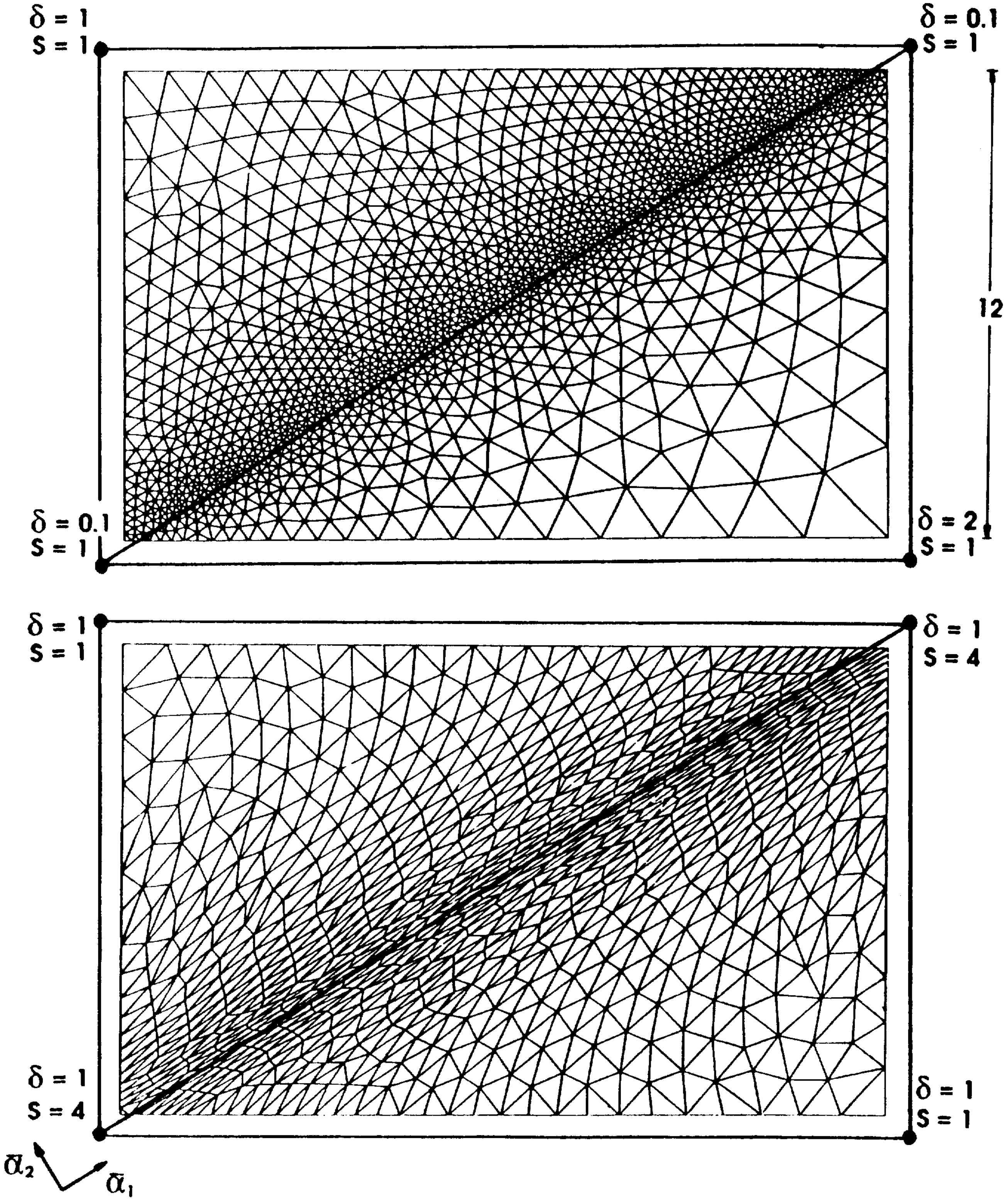


Figure 3.5 Variable mesh spacing and stretching. Meshes generated for a rectangular domain using a background mesh consisting of two triangles.

### *3.4 Curve Discretisation*

The discretisation of the boundary curve components is achieved by positioning nodes along the curve according to a spacing dictated by the local value of the mesh parameters. Consecutive points are joined by straight lines to form sides. In order to determine the position and number of nodes to be created on each curve component, the following steps are followed:

i) Subdivide recursively each cubic segment into smaller cubic segments until their length is smaller than a certain prescribed value. A safe choice for this value is the minimum spacing specified in the background mesh but often, considerably larger values can be taken. The length of each cubic segment is computed numerically. When subdividing a cubic segment, the position and tangent vectors corresponding to the new data points can be found directly from the original definition of the segment.

ii) For all the data points $\hat{\underline{r}}_j$; j=1,...,n (i.e. those used to define the curve and those created to satisfy the maximum length criterion), interpolate from the background mesh the coefficients of the transformation $T_j$ and transform the position and tangent vectors i.e. $\hat{\underline{r}}_j = T_j\,\underline{r}_j$ and $\hat{\underline{t}}_j = T_j\,\underline{t}_j$. The new position and tangent vectors $\hat{\underline{r}}_j$, $\hat{\underline{t}}_j$; j=1,...,n, define a spline curve which can be interpreted as the image of the original curve component in the normalised space. It must be noted that because of the approximate nature of this procedure, the new curve will in general have discontinuities of curvature even though the curvature of the original curve varies continuously.

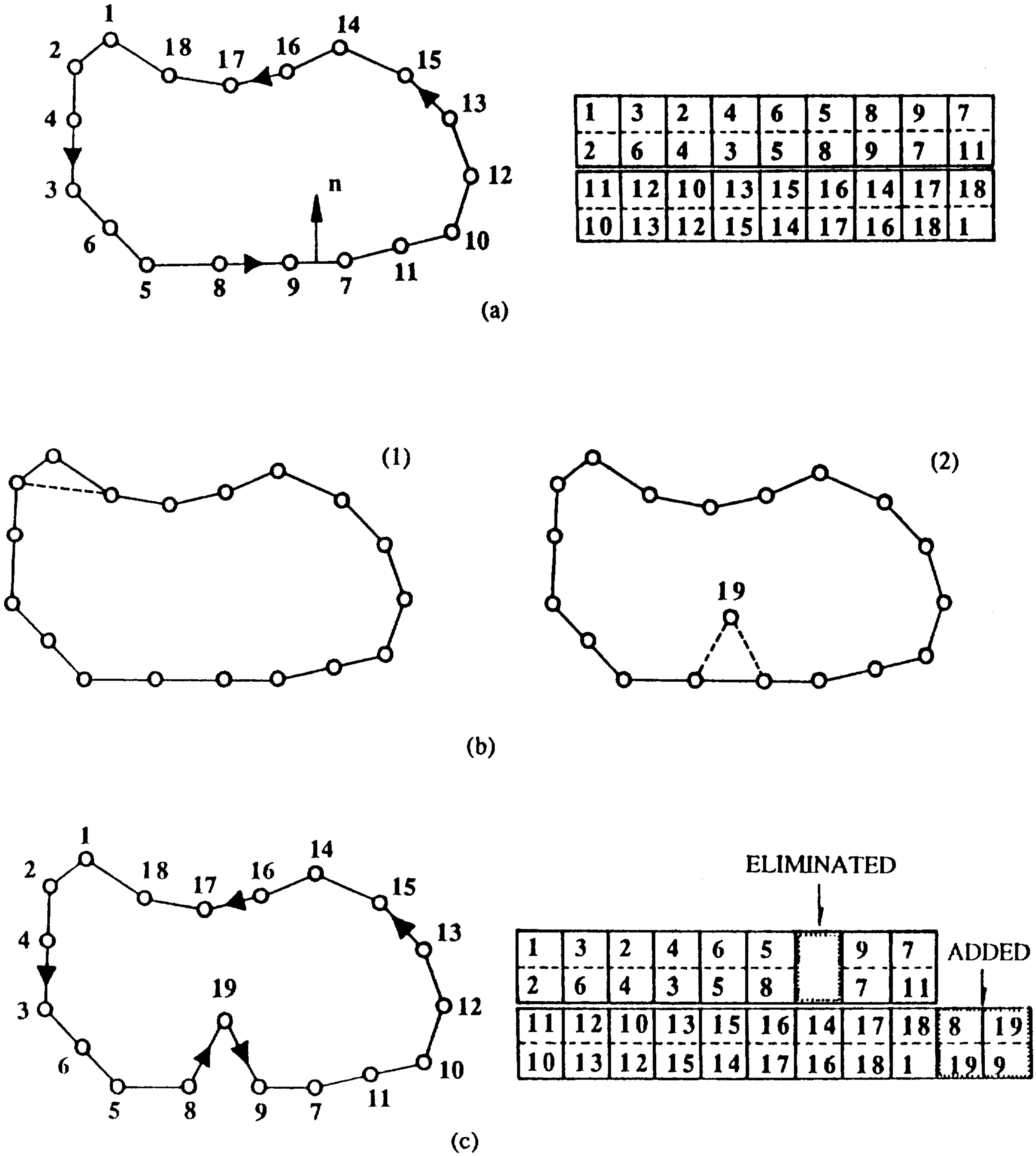
iii) Compute the length of the curve in the normalised space and subdivide it into segments of approximately unit length. For each newly created point, calculate the cubic segment in which it is contained and its parametric coordinate. This information is used to determine the coordinates of the new nodes in the physical space, using the curve component definition.

### *3.5 Triangle generation in two dimensional domains.*

The triangle generation algorithm utilises the concept of a generation front. At the start of the process the front consists of the sequence of straight line segments which connect consecutive boundary nodes. During the generation process, any straight line segment which is available to form an element side is termed active, whereas any segment which is no longer active is removed from the front. Thus while the domain boundary will remain unchanged, the generation front changes continuously and needs to be updated whenever a new element is formed. This updating process is illustrated in figure 3.6.

In the process of generating a new triangle the following steps are involved (figure 3.7):

i) Select a side AB of the front to be used as a base for the triangle to be generated. Here, the criterion is to choose the shortest side. This is especially advantageous when generating irregular meshes.



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 2 | 4 | 6 | 5 | 8 | 9 | 7 |
| 2 | 6 | 4 | 3 | 5 | 8 | 9 | 7 | 11 |
| 11 | 12 | 10 | 13 | 15 | 16 | 14 | 17 | 18 |
| 10 | 13 | 12 | 15 | 14 | 17 | 16 | 18 | 1 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 2 | 4 | 6 | 5 | | 9 | 7 | | |
| 2 | 6 | 4 | 3 | 5 | 8 | | 7 | 11 | | |
| 11 | 12 | 10 | 13 | 15 | 16 | 14 | 17 | 18 | 8 | 19 |
| 10 | 13 | 12 | 15 | 14 | 17 | 16 | 18 | 1 | 19 | 9 |

Figure 3.6 Front updating procedure in two dimensions.
(a) Initial generation front.
(b) Creating a new element (1) No new point is created
(2) A new point (19) is created
(c) Front updating for the case b.2.

ii) Interpolate from the background grid the transformation **T** at the centre of the side $\underline{M}$ and apply it to the nodes in the front which are relevant to the triangulation. In our implementation we define the relevant points to be all those which lie inside the circle of centre $\underline{M}$ and radius three times the length of the side being considered. Let $\hat{\underline{A}}$, $\hat{\underline{B}}$ and $\hat{\underline{M}}$ denote the positions in the normalised space of the points $\underline{A}$, $\underline{B}$ and $\underline{M}$ respectively.

iii) Determine, in the normalised space, the ideal position $\hat{\underline{P}}_1$ for the vertex of the triangular element. The point $\hat{\underline{P}}_1$ is located on the line perpendicular to the side that passes through the point $\hat{\underline{M}}$ and at a distance $\delta_1$ from the points $\hat{\underline{A}}$ and $\hat{\underline{B}}$. The direction in which $\hat{\underline{P}}_1$ is generated is determined by the orientation of the side. The value $\delta_1$ is chosen according to:

$$\delta_1 = \begin{cases} 1 & \text{if} \quad 0.55 \cdot L < 1 < 2 \cdot L \\ 0.55 \cdot L & \text{if} \quad 0.55 \cdot L < 1 \\ 2 \cdot L & \text{if} \quad 1 > 2 \cdot L \end{cases} \tag{3.2}$$

where L is the distance between points $\hat{\underline{A}}$ and $\hat{\underline{B}}$. Only in situations where the side AB happens to have characteristics very different from those specified by the background mesh will the value of $\delta_1$ be different from unity. However, the above inequalities must be taken into account to ensure geometrical compatibility. Expression (3.2) is purely empirical and different inequalities could be devised to serve the same purpose.

iv) Select other possible candidates for the vertex and order them in a list. Two types of points are considered viz. (a) all the nodes $\hat{\underline{Q}}_1, \hat{\underline{Q}}_2$ ... in the current generation front which are, in the normalised space, interior to a circle with centre $\hat{\underline{P}}_1$ and radius $r = \delta_1$, and (b) the set of points $\hat{\underline{P}}_1, \ldots, \hat{\underline{P}}_5$ generated along the height $\hat{\underline{P}}_1\hat{\underline{M}}$. For each point $\hat{\underline{Q}}_i$, construct the circle with centre $\hat{\underline{Q}}_i$, on the line defined by points $\hat{\underline{P}}_1$ and $\hat{\underline{M}}$ and which passes through the points $\hat{\underline{Q}}_i$, $\hat{\underline{A}}$ and $\hat{\underline{B}}$. The position of the centres $\hat{Q}_i$, of these circles on the line $\hat{\underline{P}}_1\hat{\underline{M}}$ defines an ordering of the the $\hat{\underline{Q}}_i$ points. A list is created which contains all the $\hat{\underline{Q}}_i$ points with the furthest point from $\hat{\underline{P}}_1$ appearing at the head of list. The points $\hat{\underline{P}}_1, \ldots, \hat{\underline{P}}_5$ are added at the end of this list.

v) Select the best connecting point. This is the first point in the ordered list which gives a consistent triangle. Consistency is guaranteed by ensuring that none of the newly created sides intersects with any of the existing sides in the front.

vi) Finally, if a new node is created, its coordinates in the physical space are obtained by using the inverse transformation $\mathbf{T}^{-1}$.

vii) Store the new triangle and update the front by adding/removing the relevant sides.

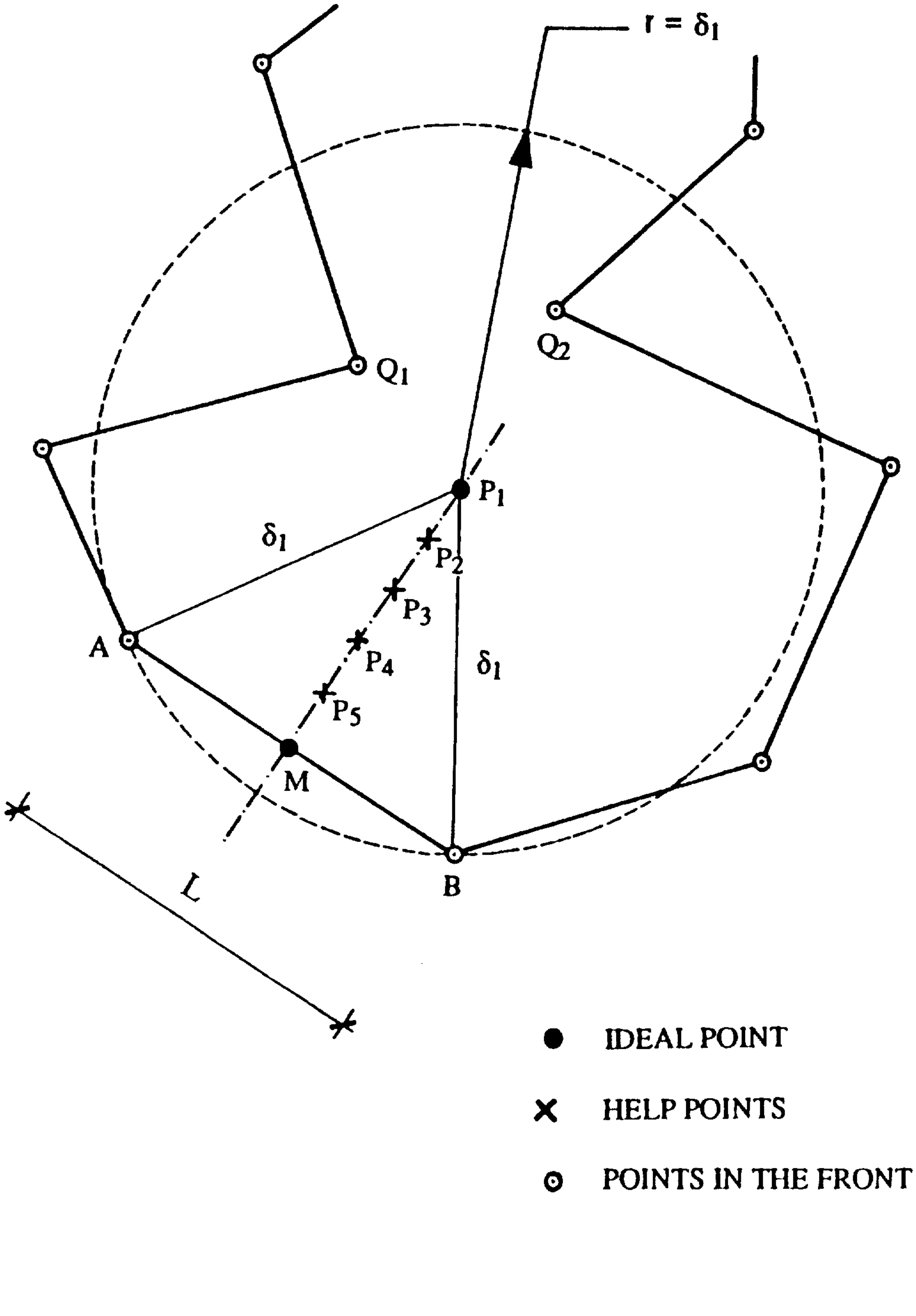


**Figure 3.7** Generation of a new triangle.

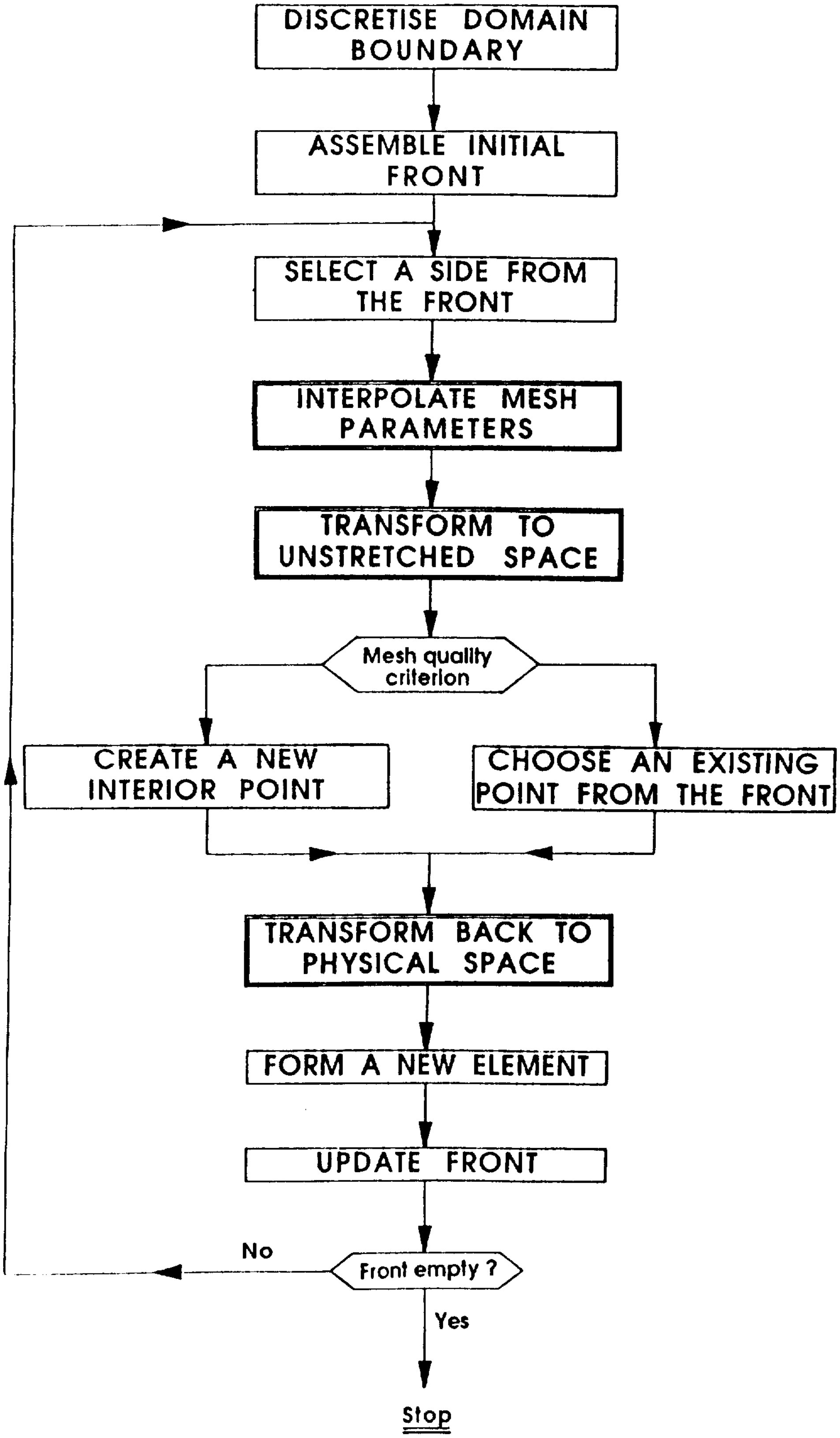

DISCRETISE DOMAIN BOUNDARY
ASSEMBLE INITIAL FRONT
SELECT A SIDE FROM THE FRONT
INTERPOLATE MESH PARAMETERS
TRANSFORM TO UNSTRETCHED SPACE
Mesh quality criterion
CREATE A NEW INTERIOR POINT
CHOOSE AN EXISTING POINT FROM THE FRONT
TRANSFORM BACK TO PHYSICAL SPACE
FORM A NEW ELEMENT
UPDATE FRONT
No
Front empty ?
Yes
Stop

Figure 3.8 Mesh generation procedure using the advancing front technique. Double line boxes are necessary to include the effects of variable mesh size and stretching.

This mesh generation procedure is schematically presented in the diagram shown in figure 3.8

Mesh quality enhancement

In order to enhance the quality of the generated mesh, two post-processing procedures are applied. These procedures, which are local in nature, do not alter the total number of points or elements in the mesh.

- Diagonal swapping.- This changes the connectivities among nodes in the mesh without altering their position. This process requires a loop over all the element sides excluding those sides on the boundary. For each side AB (figure 3.9) common to the triangles ABC and ADB one considers the possibility of swapping AB by CD, thus replacing the two triangles ABC and ADB by the triangles ADC and BCD. The swapping is performed if a prescribed regularity criterion is satisfied better by the new configuration than by the existing one. In our implementation, the swapping operation is performed if the minimum angle occuring in the new configuration is larger than in the original one.

- Mesh smoothing.- This alters the positions of the interior nodes without changing the topology of the mesh. The element sides are considered as springs of stiffness proportional to the length of the side. The nodes are moved until the spring system is in equilibrium. The equilibrium positions are found by iteration. Each iteration amounts to performing a loop over the interior points and moving their coordinates to coincide with those of the centroid of the neighbouring points. Usually three to five iterations are performed.

The combined application of these two post-processing algorithms is found to be very effective in improving the smoothness and regularity of the generated meshes.

### 3.6 Surface Discretisation.

The method followed for the triangulation of the surface components is an extension of the mesh generation procedure for planar domains described above. The discretisation of each surface component is accomplished by generating a two dimensional mesh of triangles in the parametric plane $(u^1,u^2)$ and then using the mapping $\underline{r}(u^1,u^2)$ defined in section 2.2. This mapping establishes a one to one correspondence between the boundary surface component and a region on the parametric plane $(u^1,u^2)$ (figure 3.10). Thus, a consistent triangular mesh in the parametric plane will be transformed, by the mapping $\underline{r}(u^1,u^2)$, into a valid triangulation of the surface component. The construction of the triangular mesh in the parameter plane $(u^1,u^2)$ using the two dimensional mesh generator, requires the determination of an appropriate spatial distribution of the two dimensional mesh parameters. These consist of a set of two mutually orthogonal directions $\underline{\alpha}_i^*$; i=1, 2, and two associated element sizes $\delta_i^*$; i=1, 2.

The two dimensional mesh parameters in the $(u^1,u^2)$ plane can be evaluated from the spatial distribution of the three dimensional mesh parameters and the distortion and stretching introduced by the mapping. To illustrate this process, consider a point $\underline{P}^*$ in the parametric plane of coordinates $(u^{1P},u^{2P})$ where the values of the mesh parameters $\delta_i^*$, $\underline{\alpha}_i^*$; i=1,2 are to be computed. Its image on the surface will be the point $\underline{P} = \underline{r}(u^{1P},u^{2P})$. The transformation between the physical space and the normalised space at this point $T_P$ can be obtained by direct interpolation from the

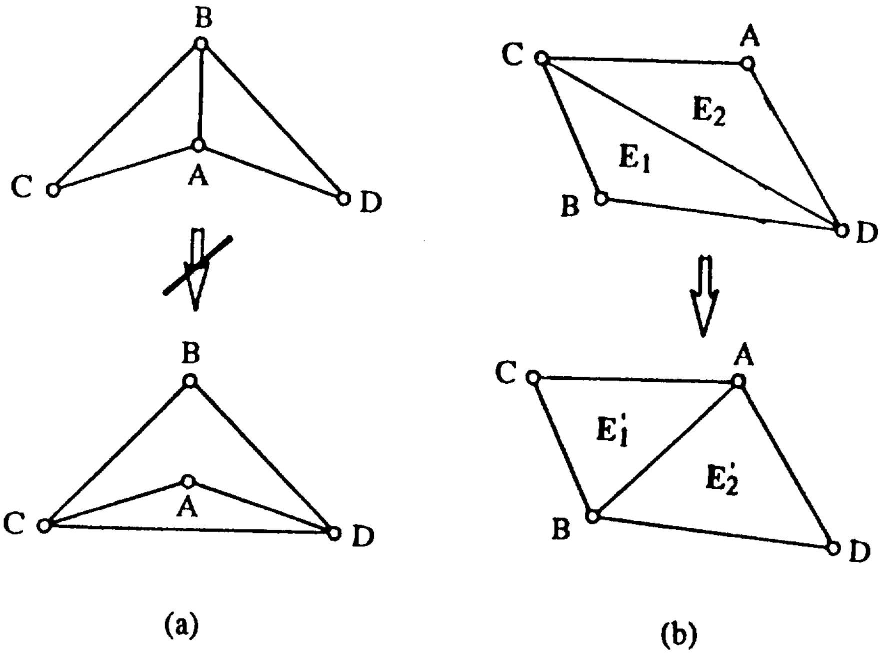


Figure 3.9 Diagonal swapping procedure: (a) non-admissible, (b) admissible.

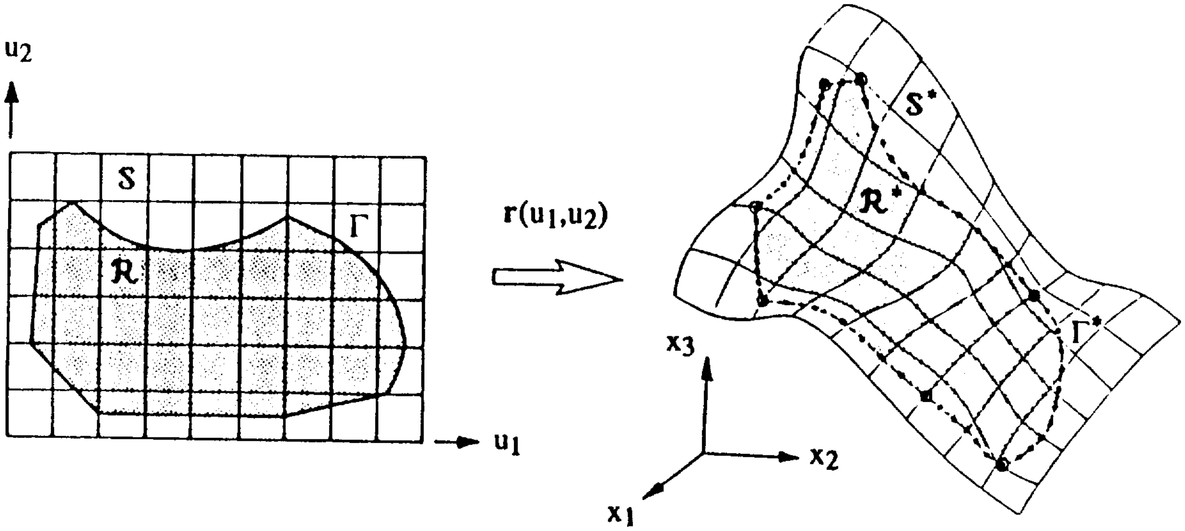


Figure 3.10 Mapping of a surface component onto a two dimensional domain.

background mesh. A new mapping, valid in the neighbourhood of point P, can now be defined between the parametric plane $(u^1,u^2)$ and the normalised space as

$$\underline{R}(u^1,u^2) = T_P\,\underline{r}(u^1,u^2) \qquad (3.3)$$

A curve in the parametric plane passing through point $\underline{P}^*$ and with unit tangent vector $\underline{\beta} = (\beta^1,\beta^2)$ at this point, is transformed by the above mapping into a curve in the normalised space passing through the point $T_P\,\underline{P}$. The arc length parameters *ds* and $d\zeta$, along the original and transformed curves respectively, are related by the expression [35]

$$(d\zeta)^2 = \left\{ \sum_{i;j=1}^{2} \frac{\partial \underline{R}}{\partial u^i}\,\frac{\partial \underline{R}}{\partial u^j}\,\beta^i\,\beta^j \right\} (ds)^2 \qquad (3\ 4)$$

Assuming that this relation between the arc length parameters also holds for the spacings, we can compute the spacing $\delta_\beta$ along the direction $\underline{\beta}$ in the parameter plane as

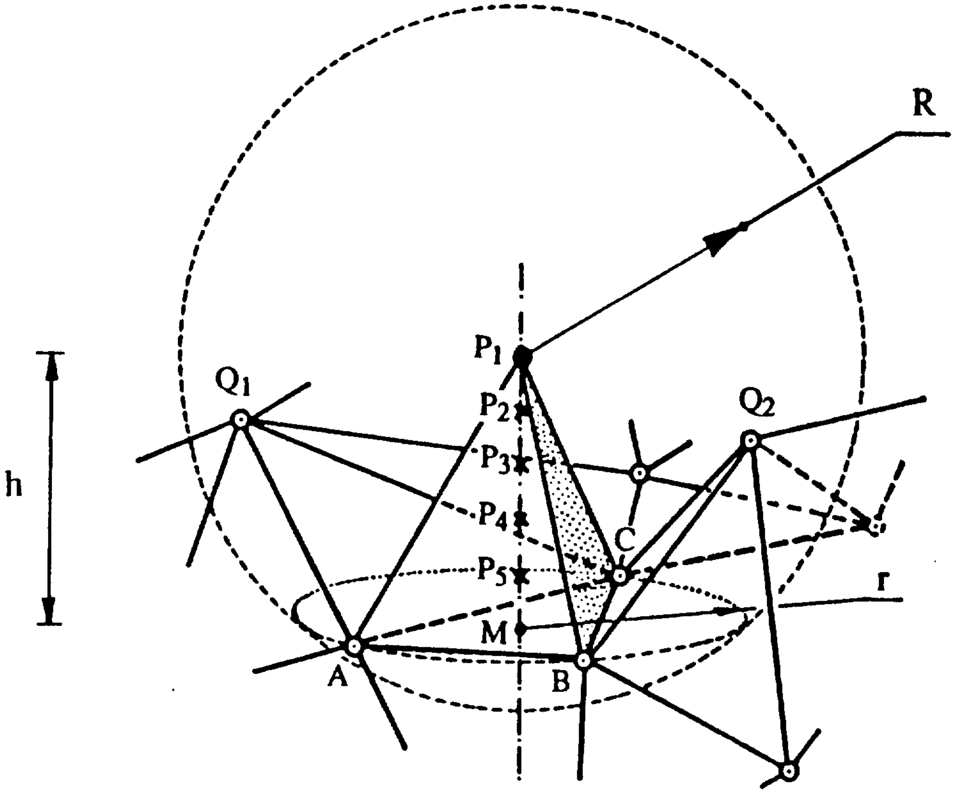
$$\delta_\beta = \sqrt{\sum_{i;j=1}^{2} \frac{\partial \underline{R}}{\partial u^i}\,\frac{\partial \underline{R}}{\partial u^j}\,\beta^i\,\beta^j} \qquad (3.5)$$

The two dimensional mesh parameters $\underline{\alpha}_i^*, \delta_i^*; i=1,\ 2$ are determined from the directions in which $\delta_\beta$ attains an extremum. This reduces to finding the eigenvalues and eigenvectors of a symmetric 2 x 2 matrix.

To form the initial front, the $(u^1,u^2)$ coordinates of the nodes already generated on the boundary curve components have to be computed. As the mapping $\underline{r}(u^1,u^2)$ cannot be inverted analytically, the coordinates $(u^1,u^2)$ of such points are found numerically by using a direct iteration procedure [31].

### *3.7 Generation of Tetrahedra.*

The starting point for the discretisation of the three dimensional domain into tetrahedra is the formation of an initial generation front. The initial front is the set of oriented triangles which constitutes the discretised boundary of the domain and is formed by assembling the discretised boundary surface components. The order in which the nodes of these triangles are given defines the orientation, which is the same as that of the corresponding boundary surface component. The algorithm for generating tetrahedra is analogous to that described above for the generation of triangles (see figure 3.8). However, in the three dimensional case the range of possible options at each stage is much wider and the number of geometrical operations involved increases considerably. Thus, the ability of the method to produce a mesh and the efficiency of its implementation relies heavily upon the type of strategy selected. The generation of a generic tetrahedral element involves the following steps (figure 3.11):



● IDEAL POINT

X HELP POINTS

⊙ POINTS IN THE FRONT

**Figure 3.11** Generation of a tetrahedral element.

i) Select a triangular face ABC from the front to be a base for the tetrahedron to be generated. In principle, any face could be chosen, but we have found it to be advantageous in practice to consider the smallest faces first. For this purpose, the size of the face is defined in terms of the size of its shortest height.

ii) Interpolate from the background grid the transformation **T** at the centroid of the face $\underline{M}$ and apply it to the nodes in the front which are relevant to the triangulation. In our implementation, we define the relevant points to be those which lie inside the sphere of centre $\underline{M}$ with radius equal to three times the value of the maximum dimension of the face being considered. Let $\hat{\underline{A}}$, $\hat{\underline{B}}$, $\hat{\underline{C}}$ and $\hat{\underline{M}}$ denote the positions in the normalised space of the points $\underline{A}$, $\underline{B}$, $\underline{C}$ and $\underline{M}$ respectively.

iii) Determine, in the transformed space, the ideal position $\hat{\underline{P}}_1$ for the vertex of the tetrahedral element. The point $\hat{\underline{P}}_1$ lies on the line which passes through the point $\hat{\underline{M}}$ and which is perpendicular to the face. The direction in which $\hat{\underline{P}}_1$ is generated is determined by the orientation of the face. The location of $\hat{\underline{P}}_1$ is computed so that the average length of the three newly created sides which join point $\hat{\underline{P}}_1$ with points $\hat{\underline{A}}$, $\hat{\underline{B}}$ and $\hat{\underline{C}}$ is unity. For faces whose size in the parametric plane is very different from unity, this step may have to be modified, as in expression (3.2), to ensure geometrical compatibility. However, such cases rarely occur in practice. Let $\delta_1$ be the maximum of the distances between point $\hat{\underline{P}}_1$ and points $\hat{\underline{A}}$, $\hat{\underline{B}}$ and $\hat{\underline{C}}$.

iv) Select other possible candidates for the vertex and order them in a list. Two types of points are considered viz. (a) all the nodes $\hat{\underline{Q}}_1, \hat{\underline{Q}}_2$, ... in the current generation front which are, in the normalised space, interior to a sphere with centre $\hat{\underline{M}}$ and radius $r = \delta_1$, and (b) a new set of points $\hat{\underline{P}}_1, \ldots, \hat{\underline{P}}_5$ generated along the height $\hat{\underline{P}}_1\hat{\underline{M}}$. Consider the set of points $\hat{\underline{A}}$, $\hat{\underline{B}}$ and $\hat{\underline{C}}$ and denote by $\hat{\underline{D}}$ the member of this set which is furthest away from $\hat{\underline{M}}$. For each point $\underline{Q}_i$, construct the sphere with centre $\hat{\underline{O}}_i$ on the line defined by points $\hat{\underline{P}}_1$ and $\hat{\underline{M}}$ and which passes through point $\underline{Q}_i$ and $\underline{D}$. The position of the centres $\hat{\underline{O}}_i$ of these spheres on the line $\hat{\underline{P}}_1\hat{\underline{M}}$ defines an ordering of the the $\hat{\underline{Q}}_i$ points with the furthest point from $\hat{\underline{P}}_1$ appearing at the head of list. The points $\hat{\underline{P}}_1, \ldots, \hat{\underline{P}}_5$ are added at the end of this list.

v) Select the best connecting point. This is the first point in the ordered list which gives a consistent tetrahedron. Consistency is guaranteed by ensuring that none of the newly created sides intersects with any of the existing faces in the front, and that none of the existing sides in the front intersect with any of the newly created faces.

vi) If a new node is created, its coordinates in the physical space are obtained by using the inverse transformation $\mathbf{T}^{-1}$.

vii) Store the new triangle and update the front by adding/removing the necessary sides.

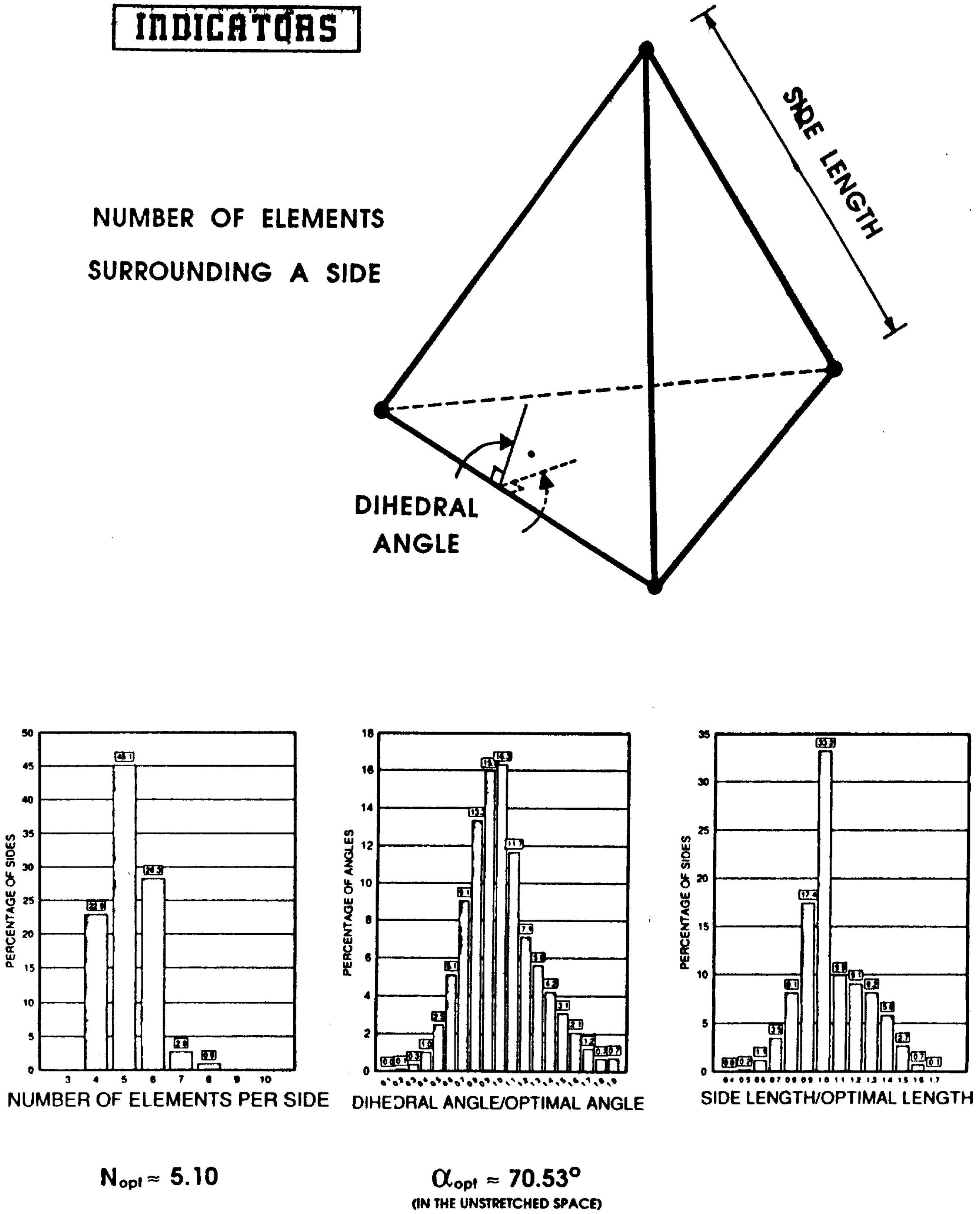


**Figure 3.12** Mesh quality statistics.

### *3.8 Mesh quality assessment.*

Any discussion of mesh quality should be intimately related to the form of the solution we are trying to represent on that mesh. Two factors need to be considered here:

i) Determination of the characteristics of the optimal mesh for the problem at hand. This introduces the concept of adaptivity and this aspect is considered in section 5.

ii) Assessment on how well the generated mesh meets the requirements specified by the mesh parameters. This assessment can be made by examining the generated mesh and determining the statistical distribution of certain indicators. For example in figure 3.12 we have chosen as indicators the number of elements around a side, the magnitude of the element dihedral angles and the length of the side. These indicators are compared with optimal values i.e. those of a regular tetrahedron which has the exact dimensions specified by the mesh parameters.

### *3.9 Application examples*

#### Multi-component airfoil

A two dimensional discretisation of the domain around a four component airfoil in landing configuration is shown in figure 3.13. The background mesh employed for the generation, consisting of a few elements only, has been superposed on the generated mesh. The mesh in the vicinity of the airfoils is nearly of constant size and varies rapidly away from the airfoils. Very little distortion in the triangles is observed even though large variations in mesh size occur.

#### Generic fighter configuration

In computational aerodynamics, a problem of current interest is the prediction of the inviscid flowfield about complete aircraft configurations. The problem considered here is the simulation of the flow past a generic fighter with canard, 70-20 cranked delta wing, vertical fin and engine inlet. This same configuration has been studied previously using an algebraic grid generation approach [36]. Due to the symmetry of the problem only half of the fighter is modelled. Figure 3.14(a) shows the geometry definition of the computational domain. The background mesh employed is illustrated in figure 3.14(b). The curve components, defined in terms of cubic splines and the discretisation of these components is displayed in figure 3.14(c). The individual surface components are described by tensor product surfaces and the surface discretisation is illustrated in figure 3.14(d). An intermediate stage during the tetrahedra generation process is displayed in figure 3.14(e). The final mesh consisted of 76,522 tetrahedra and included a full simulation of the engine inlet.

#### Space shuttle configuration

The geometry considered is that of HERMES like space shuttle. In figure 3.15(a) the surface definition for half of the model which contained 13 surface components and 29 curve components is shown. Two views of the triangulated surface are displayed in figure 3.15(b). The triangulation consisted of 5,776 elements of nearly constant

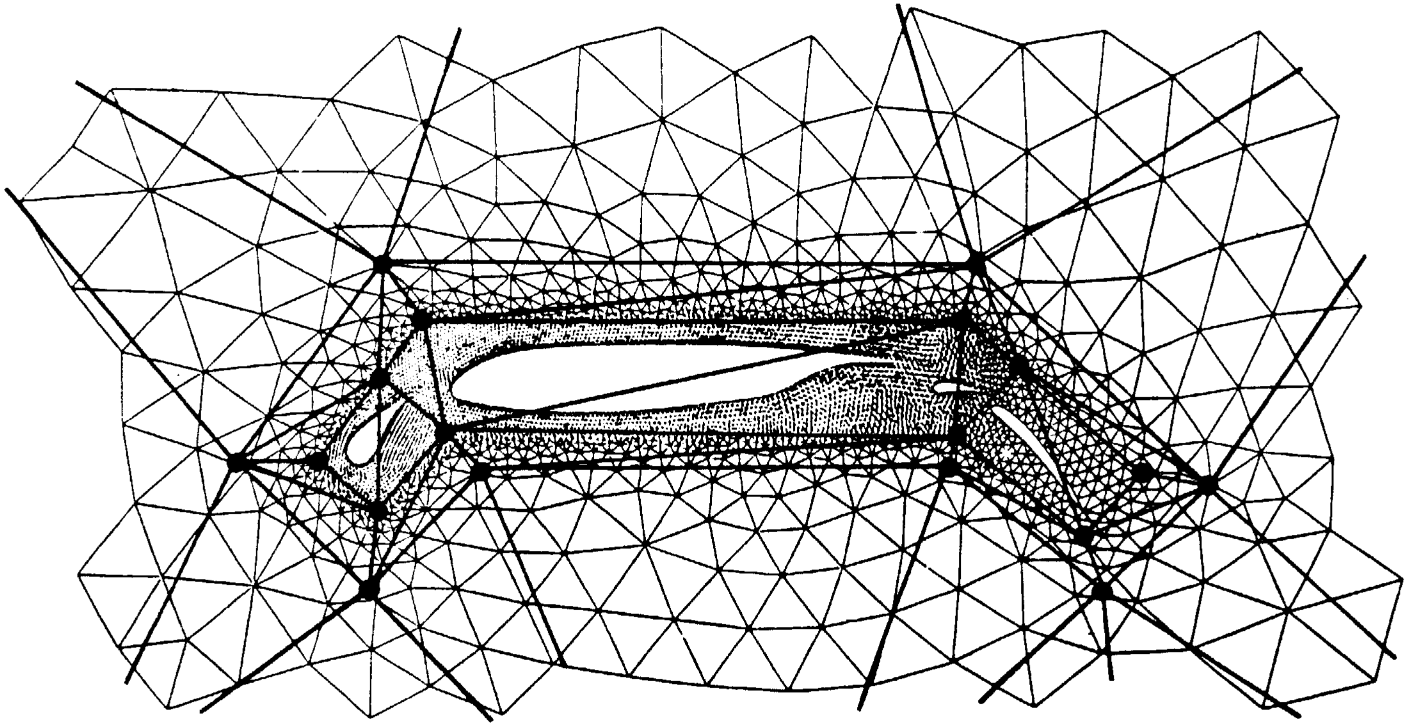


Figure 3.13 Multi-component airfoil configuration: (a) detail of the generated mesh near the airfoil and the triangular background mesh employed for the generation.

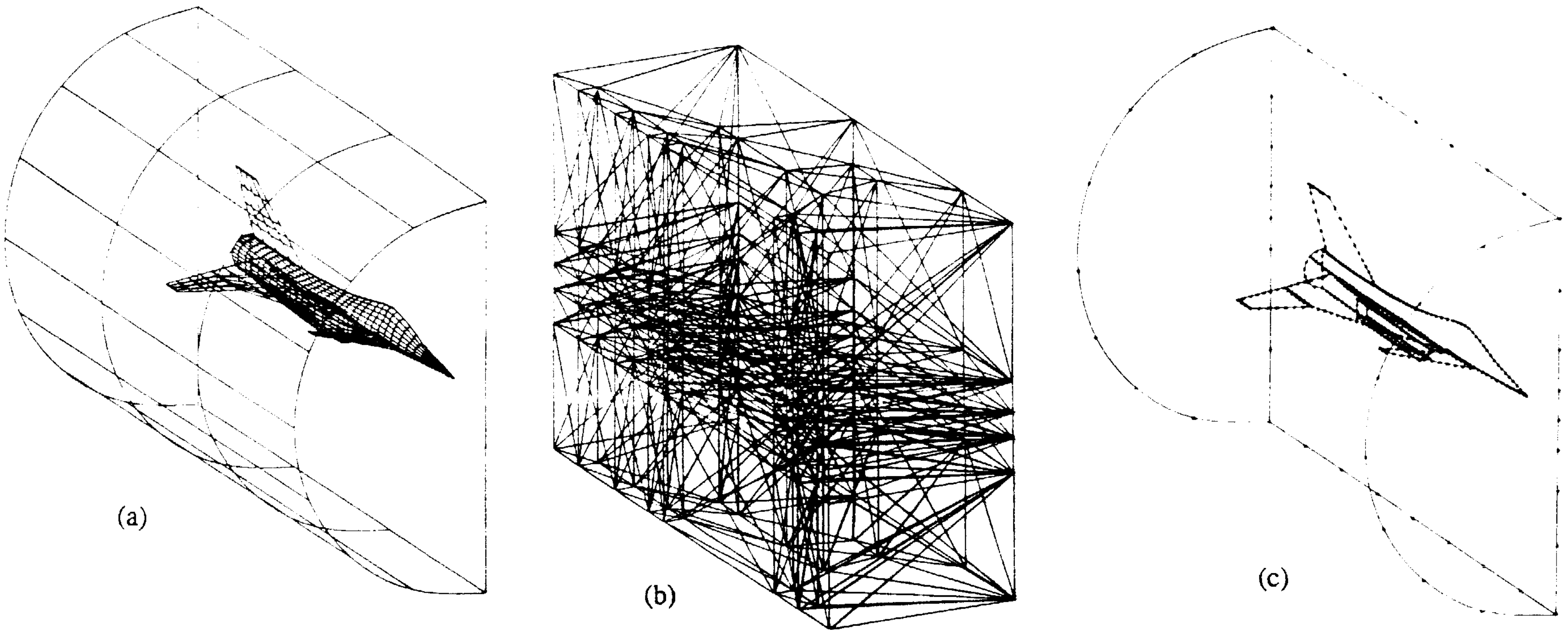


Figure 3.14 Mesh generation for a complete aircraft configuration: (a) computational domain and geometry definition, (b) background mesh and (c) curve components representation and generated points.

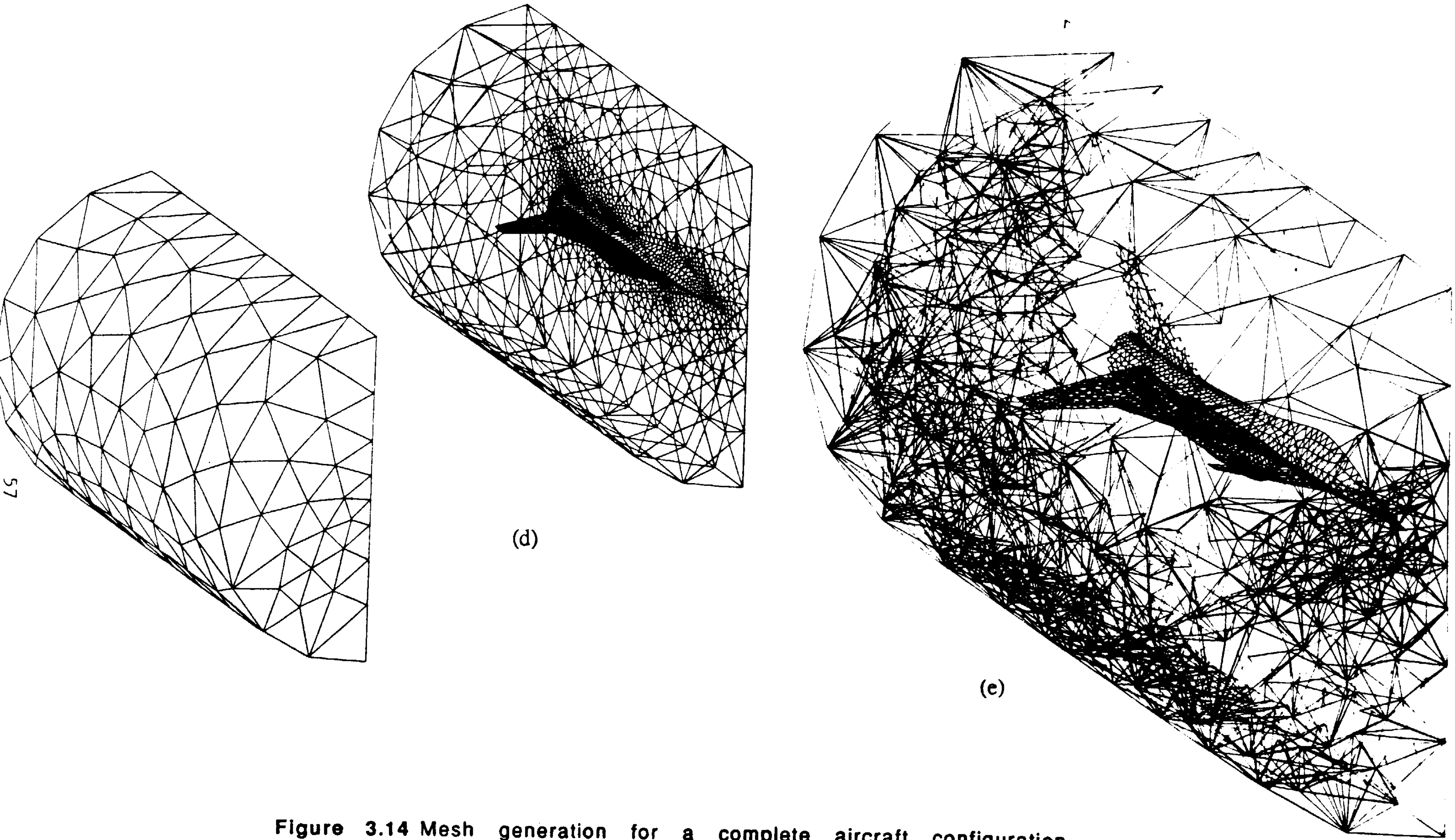


**Figure 3.14** Mesh generation for a complete aircraft configuration (continuation): (d) surface discretisation and (e) partial view of the tetrahedral mesh.

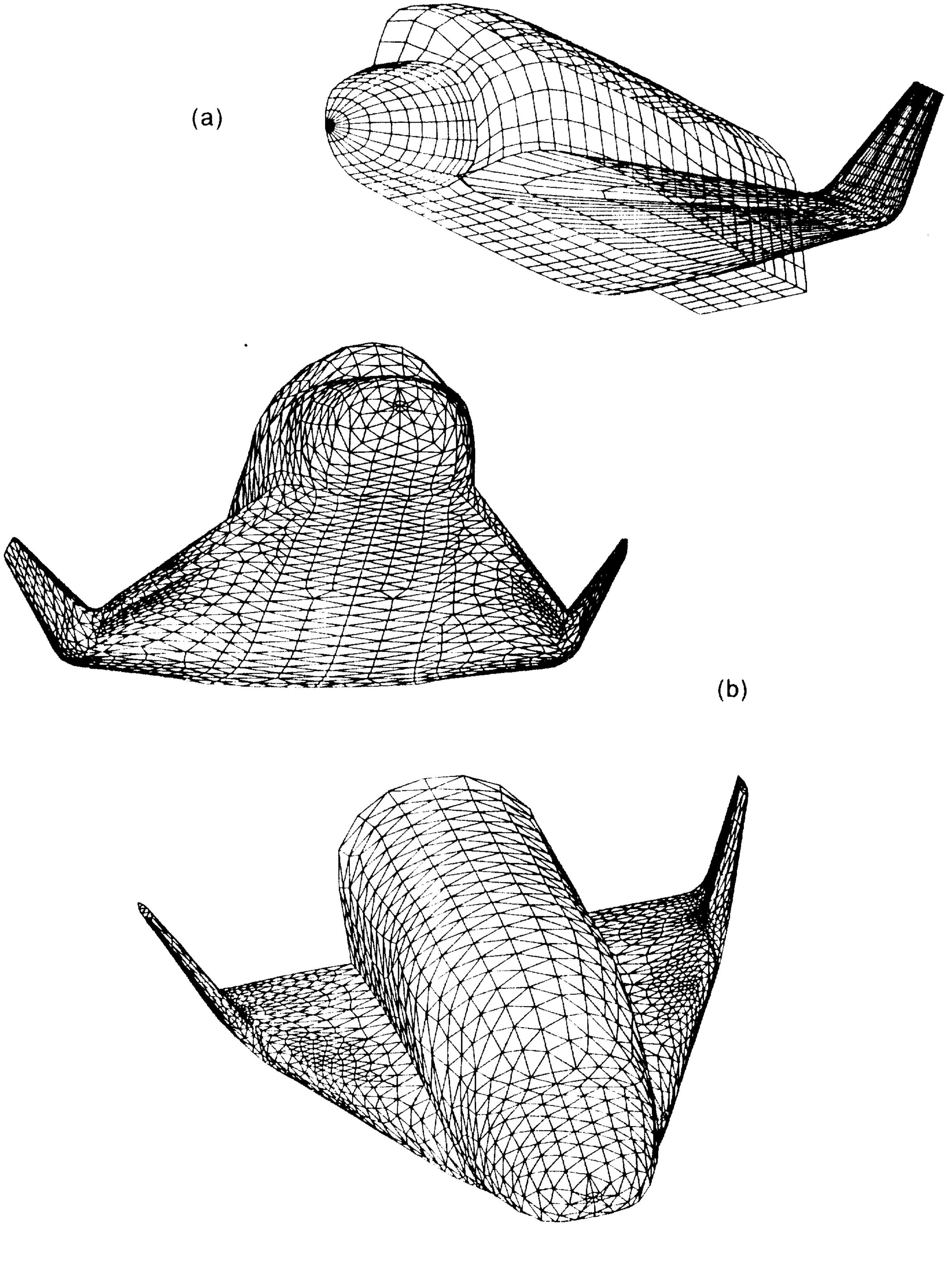


**Figure 3.15** Space shuttle configuration
(a) surface definition
(b) surface triangulation.

size. The three dimensional domain was filled in with 87,896 tetrahedra of increasing size away from the body.

### Boeing 747 in landing configuration

The problem considered in this section is the subsonic flow over a Boeing 747 aircraft. The leading edge slats and trailing flaps in the wing are deployed in a landing configuration. The flaps are detached from the wing. This means that the generator is required to mesh the region between wing and flaps. For this reason the generation of structured grids for this type of configuration presents severe problems. The geometrical definition of the aircraft by means of tensor product surfaces is shown in figure 3.16(a). Due to the symmetry of the geometry only half aeroplane is considered in the computational domain. The boundary definition consists of 29 surface components and 57 curve components. The outer boundary is a circular cylinder of radius equal to 30 times the mean chord of the wing. The discretisation of the surface has 13,030 triangles and the computational domain has been discretised using 194,307 tetrahedra. An attempt has been made to generate a mesh which has nearly uniform element size in the vicinity of the aeroplane and increases rapidly away from it. An inviscid flow computation was performed for a free stream Mach number of 0.3 and an angle of attack of $5^0$. Figure 3.16(b) shows two views of the mesh and the computed pressure contours on the surface of the complete aircraft. The computed solution was obtained after 700 timesteps of the explicit algorithm described in section 1.3. This results were not fully converged and are only preliminary.

### F-18 fighter configuration

Here the flow past an F-18 fighter aircraft, including the modelling of the engine effects is considered. The free stream Mach for the computation is 0.9 and the angle of attack is $3^0$. The simulation of the effects is accomplished by specifying the Mach number at the engine inlet to be 0.4. The flow conditions at the outlet are those corresponding to a jet pressure ratio of 3 in the engine. Due to the symmetry of the geometry only half of the domain is considered. The geometry of the boundary of the computational domain is defined by means of 37 surface components and 87 curve components. The spline definition of the aircraft surface and the engine ducts is represented in figure 3.17(a). The outer boundary is a rectangular box which is situated at an approximate distance of 15 wing chords from the aircraft. The initial generation contains 30,743 triangular faces. The triangulation of the surface of the full F-18 is depicted in figures 3.17(b), in which the engine ducts are shown detached, and 3.17(c). The element size is almost uniform in the neighbourhood of the aircraft and increases away from it. The generated mesh for half of the domain consists of 451,641 tetrahedral elements and 84,827 points. The solution on this mesh was computed using 3,500 explicit timesteps. It required a total CPU time of 6 hours on a single processor of a Cray 2. The pressure solution on the surface of the aircraft is shown in figure 3.17(d). Some of the features expected in such a flow such as, for instance, the canopy and wing shocks can be appreciated.

### Falcon aeroplane

In this section we consider the flow past a two engine Falcon aeroplane. The geometry definition which required 24 surface components and 52 curve components, is displayed in figure 3.18(a). The pylons and nacelles are included in the model and flow through the interior of the nacelles has been allowed in the numerical simulation. No engine effects are considered. In this example an attempt has been

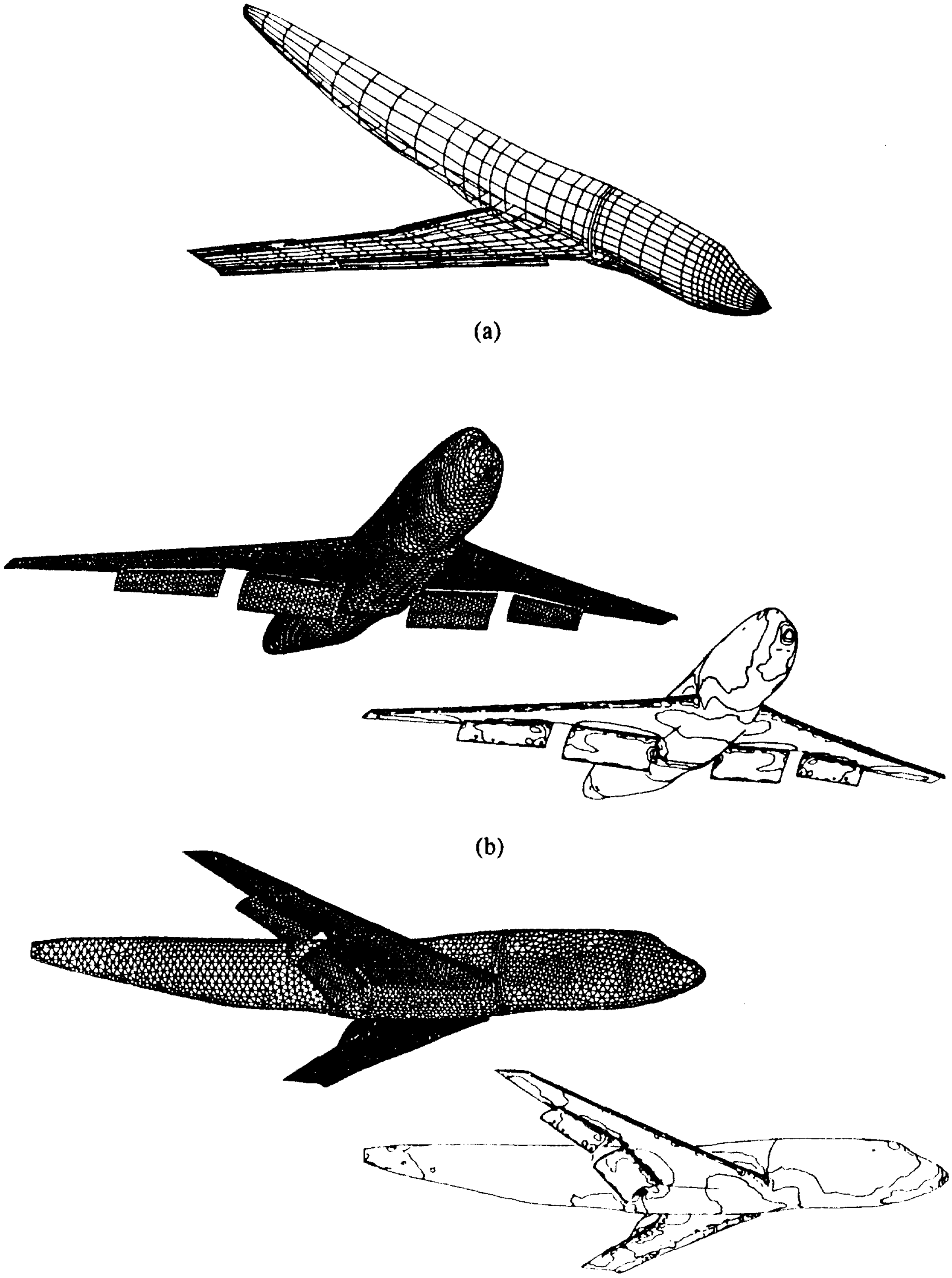


Figure 3.16 Boeing 747 in landing configuration ($M_\infty$=0.3, $\alpha$=5$^0$)
(a) Geometry definition - aircraft surface patches
(b) Two views of the mesh and pressure contours on the surface.

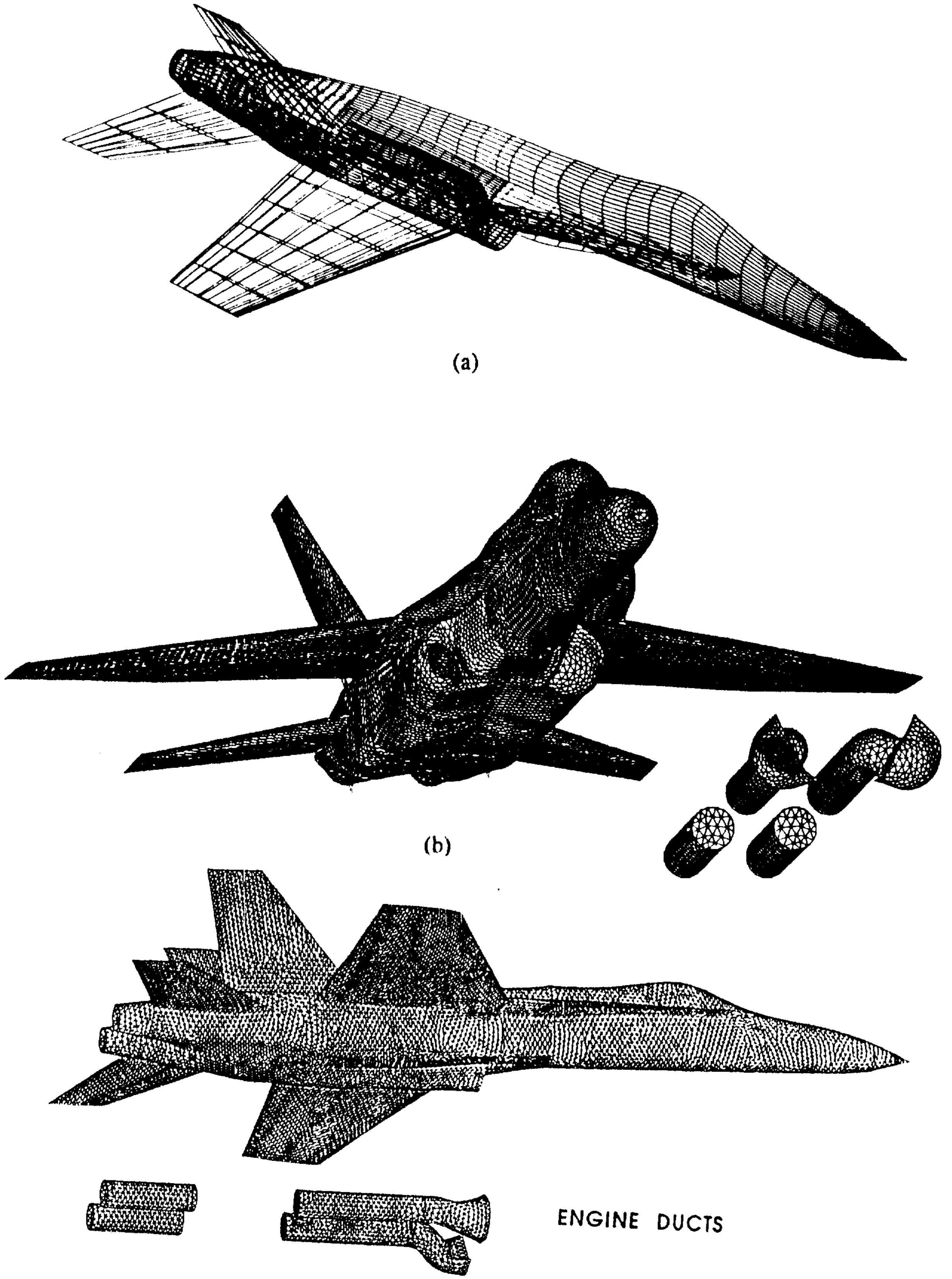
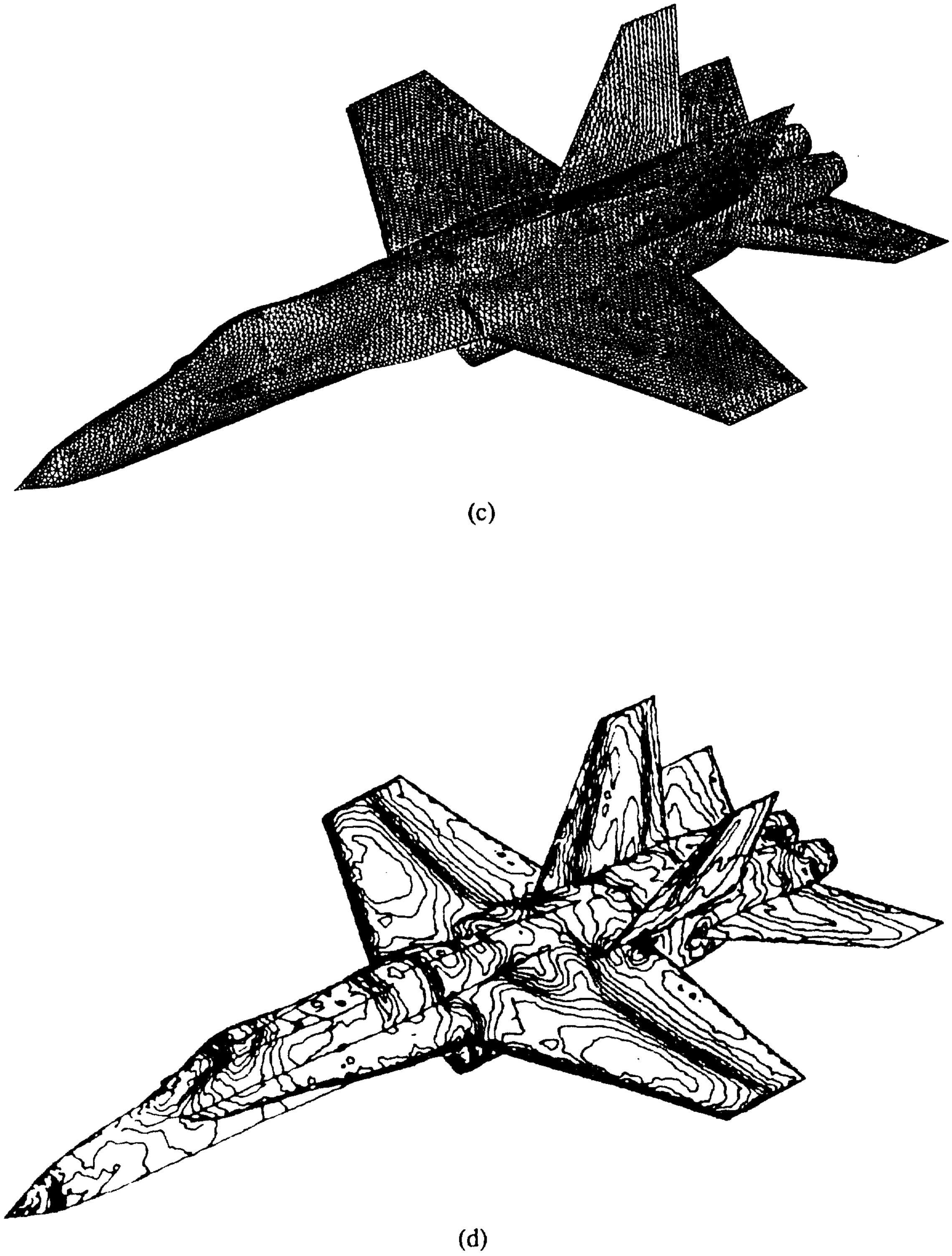


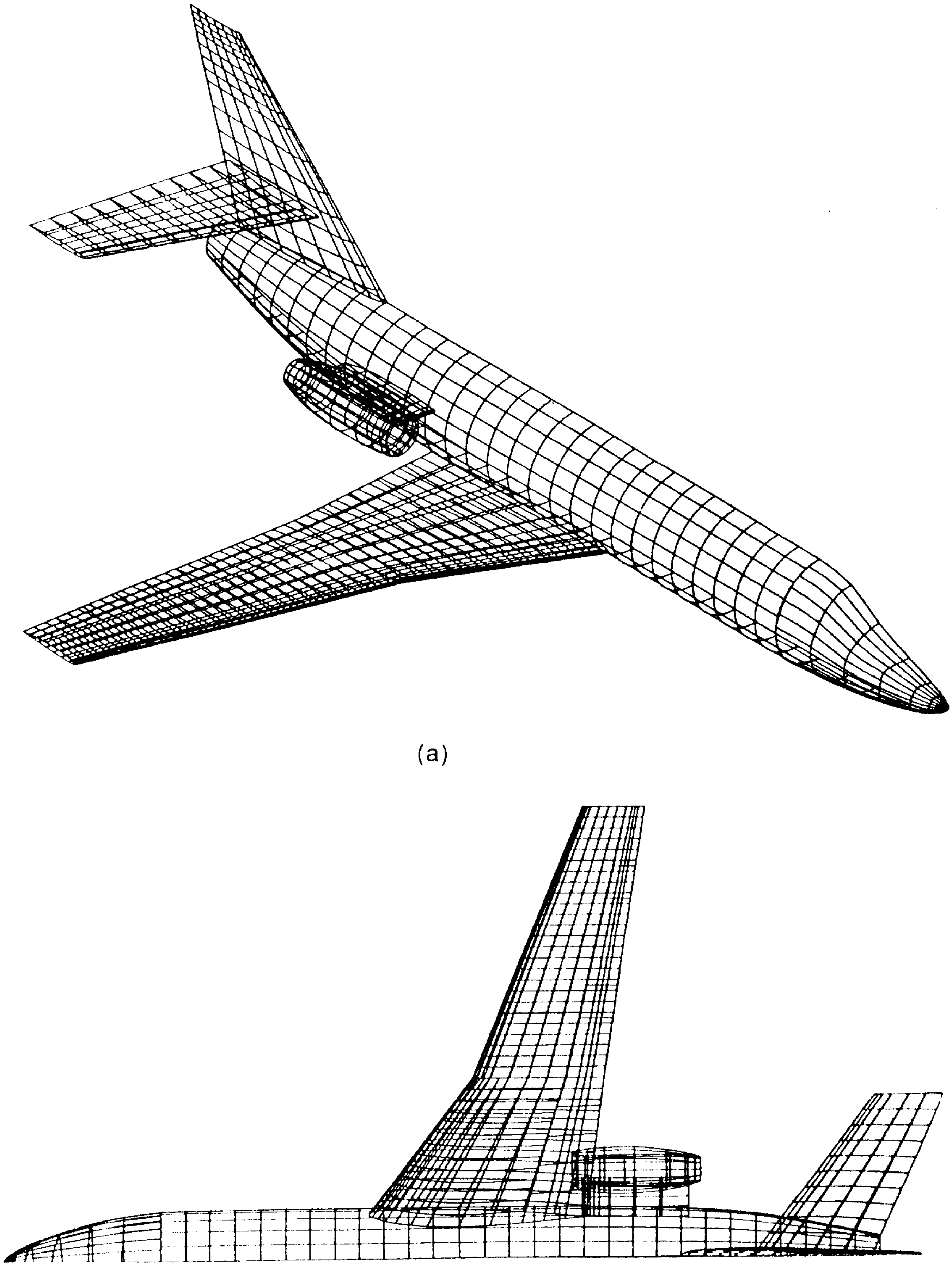
**Figure** 3.17 Flow past an F-18 fighter configuration ($M_\infty=0.9$, $\alpha=3^o$)
(a) Geometry definition - aircraft surface patches
(b) Surface triangulation showing engine ducts



(c)

(d)

**Figure 3.17** Flow past an F-18 fighter configuration ($M_\infty=0.9$, $\alpha=3^o$) (continuation)
(c) Triangulation of the F-18 surface
(d) Computed pressure solution



(a)

**Figure 3.18** Two engine falcon aircraft ($M_\infty$=0.85, $\alpha$=2$^0$)
(a) Geometry definition.

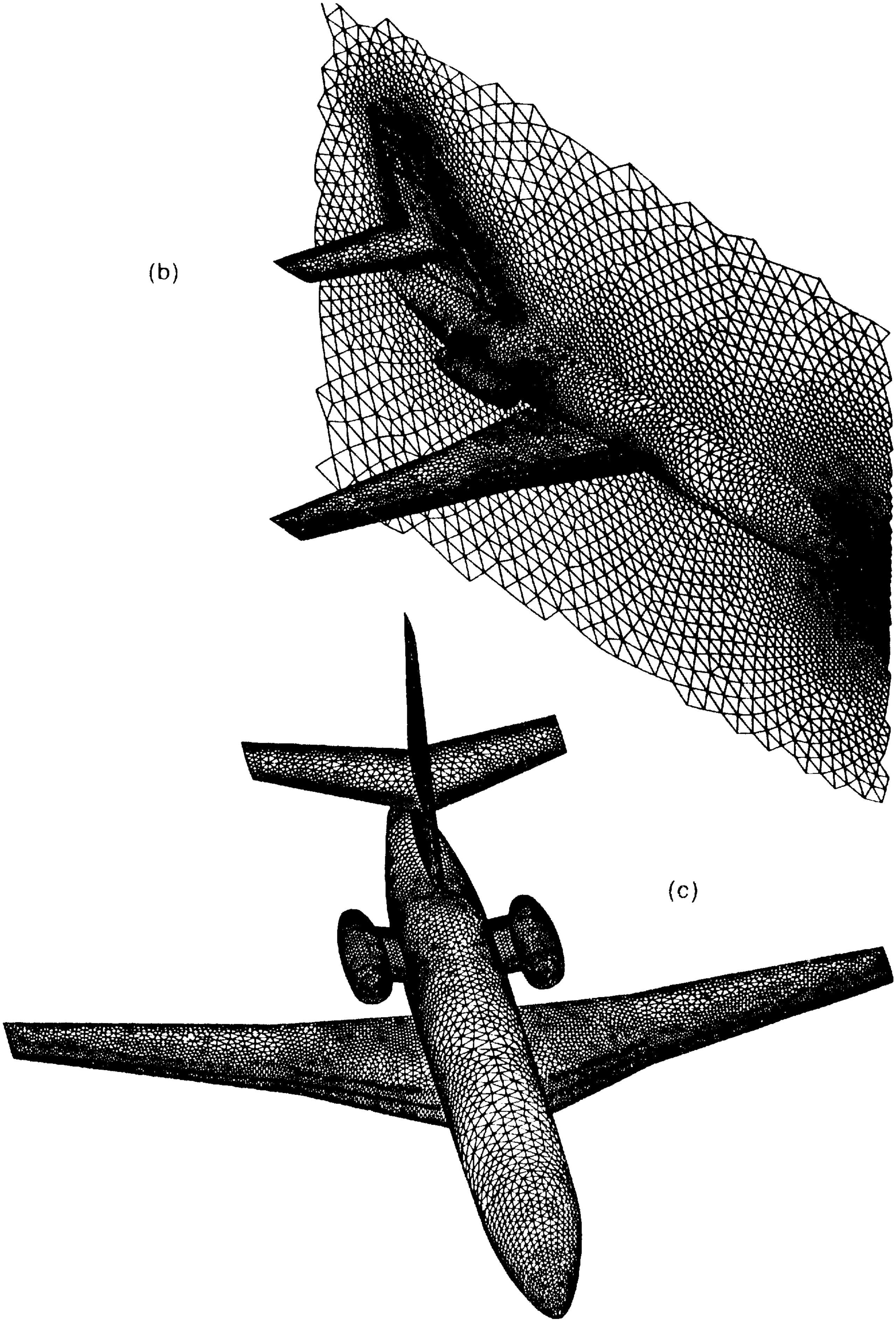


**Figure 3.18** Two engine falcon aircraft ($M_\infty$=0.85, $\alpha$=2$^0$) (continuation)
(b) Triangulation of the aeroplane surface and plane of symmetry
(c) Surface triangulation for the complete aeroplane

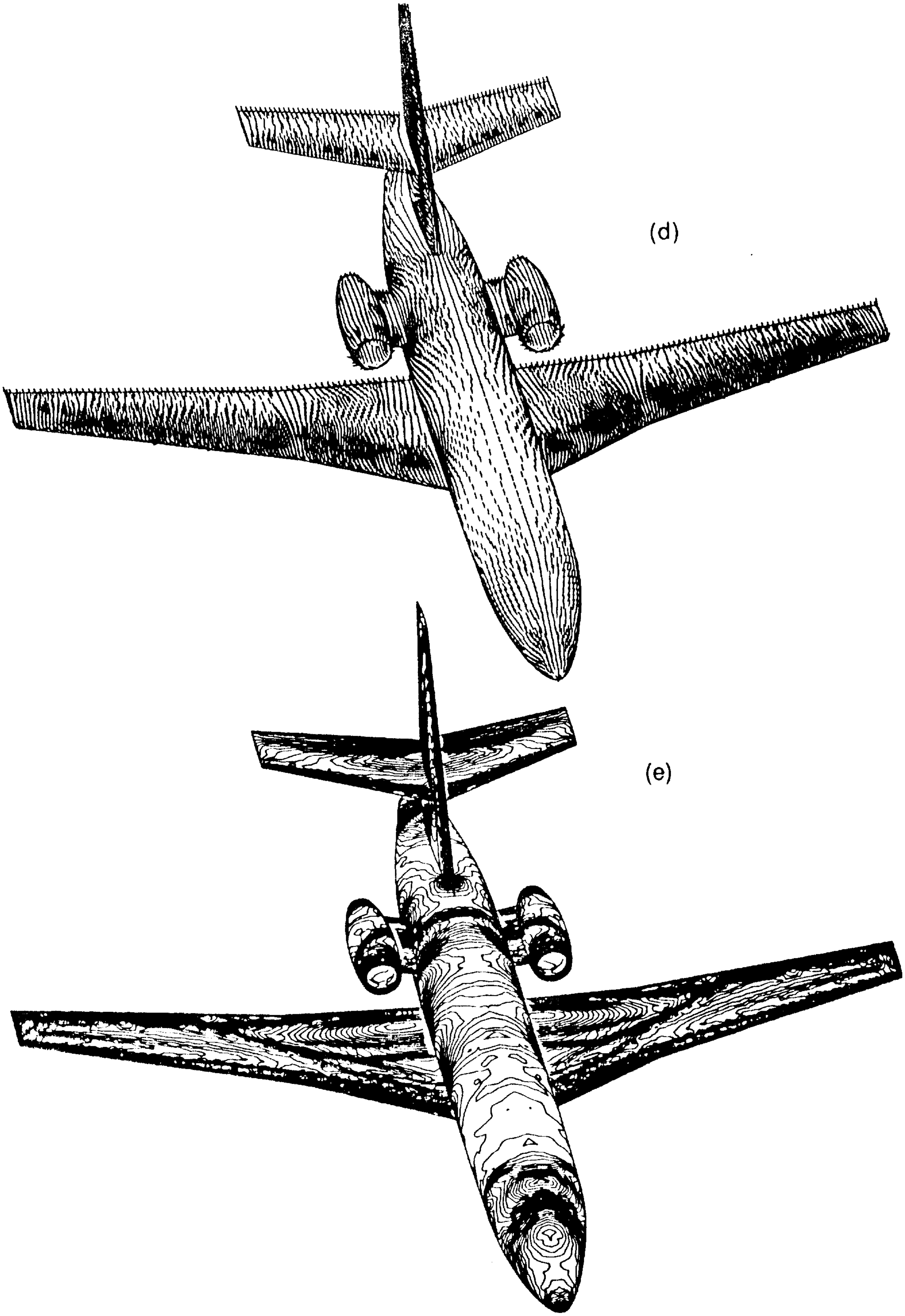


**Figure 3.18** Two engine falcon aircraft ($M_\infty=0.85$, $\alpha=2^0$) (continuation)
(d) Computed velocity vectors on the surface
(e) Computed surface pressure contours

made to produce a mesh which has increased resolution in those regions where high gradients in the solution can be expected e.g. leading/trailing edges, nose etc. This can be appreciated in figure 3.18(b) which shows the generated triangulation on the surface of the aeroplane and on the plane of symmetry. A view of the surface mesh, which contains 30,628 triangles for the complete aeroplane, is displayed in figure 3.18(c). The volume surrounding the aeroplane was discretised using 720,859 tetrahedra and 133,080 points. The generation of the mesh was performed on a Cray YMP machine and required approximately two hours of CPU time using a single processor. A flow simulation of steady state flight at a free stream Mach number of 0.85 and an angle of attack of $2^0$ was performed. The flow simulation needed about 8 and a half hours of CPU time on a Cray 2 machine. The velocity vectors on the surface of the complete aeroplane are displayed in figure 3.18(d). The pressure contours on the surface of the aircraft are depicted in figure 3.18(e). The solution is rather oscillatory. This may indicate that the solution is not fully converged yet. The formation of a lambda shock pattern on the wing and the shock structure on the pylons and nacelles can be observed.

# 4. DATA STRUCTURES

this section has been written in collaboration with

**J. Bonet**
Institute for Numerical Methods in Engineering
University College Swansea
SWANSEA SA2 8PP, UK

From the previous section it is apparent that a successful implementation of the presented algorithm will require the use of data structures which enable certain sorting and searching operations to be performed efficiently. For instance, the generation front will require a data structure which allows for the efficient insertion/deletion of sides/faces and which also allows for the efficient identification of the sides/faces which intersect with a prescribed region in space.

The problem of determining the members of a set of n points which lie inside a prescribed subregion of an N dimensional space is known as *geometric searching*. Several algorithms have been proposed [37-40] which solve the above or equivalent problems with a computational expense proportional to log(n). The problem complexity increases considerably when, instead of considering points, one deals with finite size objects such line segments, triangles or tetrahedra. A common problem encountered here, namely *geometric intersection*, consists of finding the objects which overlap a certain subregion of the space being considered. Algorithms for solving this problem in two dimensions exist [41] and have been applied in determining the intersection between geometrical objects in the plane. To our knowledge, the only algorithm capable of solving this problem in three dimensions is based on the use of the alternate digital tree [42]. The particular application which motivated the development of this data structure was the implementation of the mesh generation algorithm described in the previous section.

In what follows, we shall describe an algorithm and associated data structure, called the alternating digital tree (ADT), which allows for the efficient solution of the geometric searching problem. It naturally offers the possibility of inserting and removing points and optimally searching for the points contained inside a given region. It is applicable to any number of dimensions, and is a natural extension of the so called digital tree search technique which is exhaustively in [43] for one dimensional problems. A procedure which allows treatment of any geometrical object in an N dimensional space as a point in a 2N dimensional space will be introduced; thereby allowing the proposed technique to be employed for the solution of geometric intersection problems.

### *4.1 Binary tree structures*

Binary trees provide the basis for several searching algorithms, including the one to be presented here. It is therefore necessary to introduce some basic concepts and terminology related to binary tree structures. More detailed expositions can be found in [41,44].

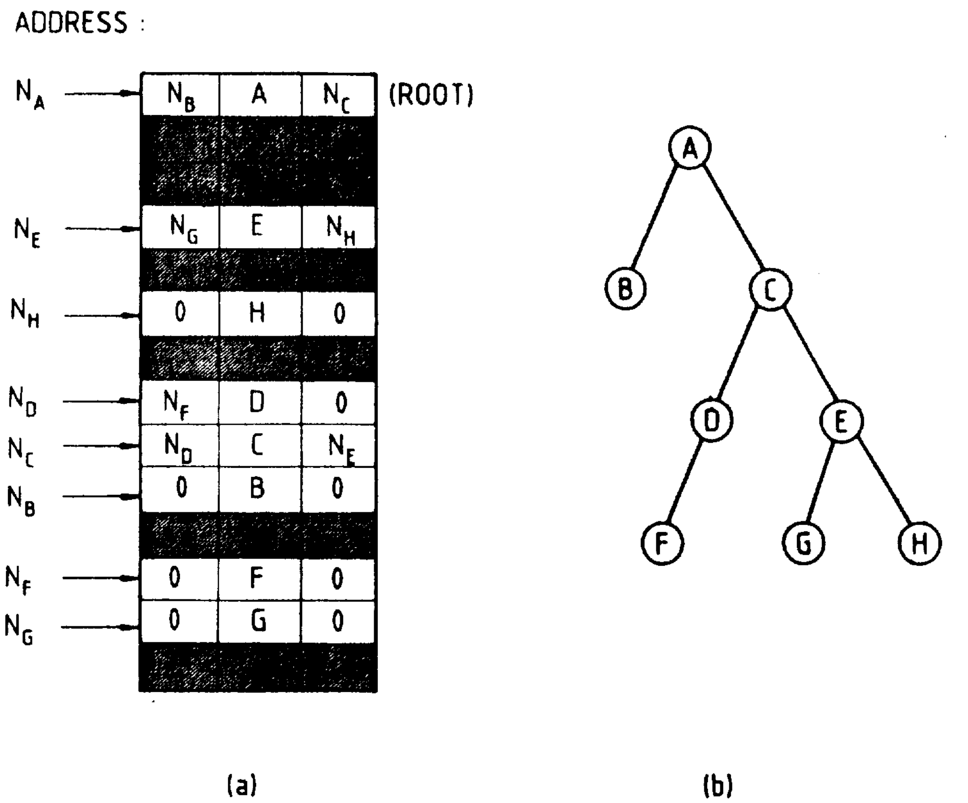


**Figure 4.1** A simple binary tree and its storage in computer memory.

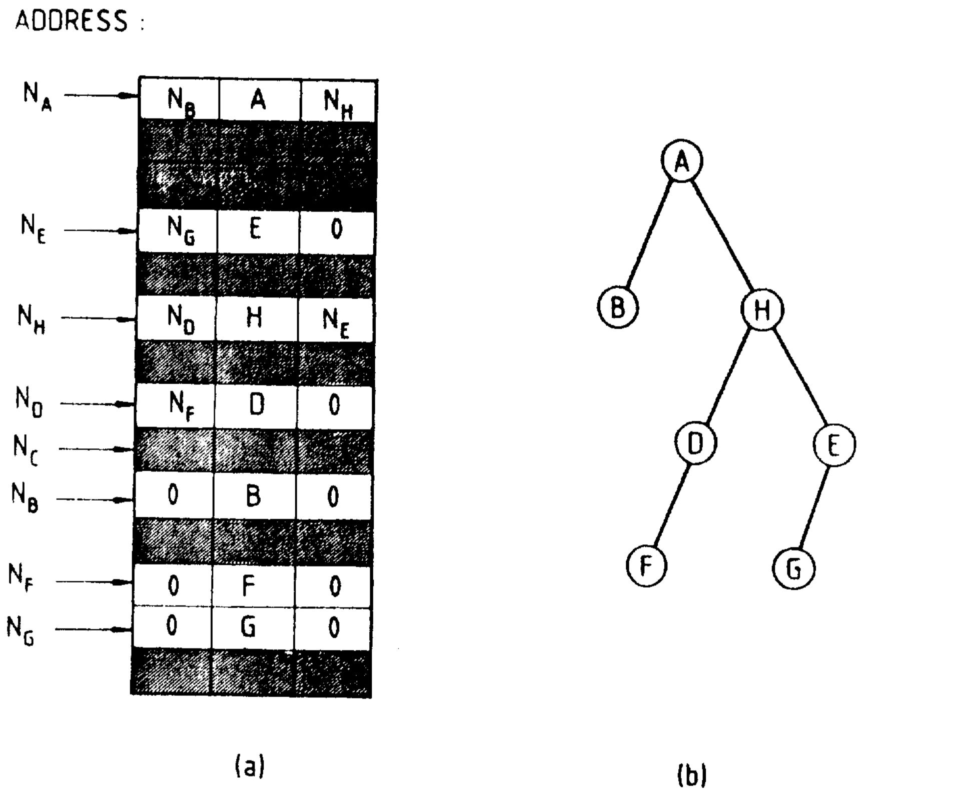


**Figure 4.2** Deletion process.

## Definition and Terminology

Tree structures provide a systematic way of storing a collection of data items which enables not only a quick access to the information stored, but also frequent insertions and deletions of items. This degree of flexibility requires the storage of data items in non-sequential locations of the computer memory. As figure 4.1a illustrates, to achieve this, each data item is extended by the addition of two integer values, known as the *left* and *right links*, and stored in what is known as a *node* of the tree. Each added link can either be equal to zero or equal to the position in memory where another node of the tree can be found. Hence, from one node of the tree it is possible to reach at most two other nodes. Moreover, in order to ensure that every node can be reached, these links must be such that for each node except one, known as the root, there is one and only one link pointing at it. This definition establishes a hierarchy of nodes: the root at the top level of the hierarchy points at 0, 1 or 2 nodes at the next level; each of these in turn points at other 0, 1 or 2 nodes at the next level of the hierarchy; and so forth. This hierarchical structure inspires the graphical representation shown in figure 4.1b for a simple tree comprising only eight nodes {A, B, C, D, E, F, G, H}.

Genealogical terms are normally used to describe the relative position of nodes in a tree: when a node points at a second node, the former is called the father of the latter, and this the son of the former node. A node without sons, that is, with both links blank, is called a terminal node, and the only node without a father is the root (node A in figure 4.1b). Given a node, the set of nodes formed by itself together with all its descendants constitutes a subtree of the main tree. For instance, in figure 4.1b the trees {C, D, E, F, G, H} and {E, G, H} are subtrees of the main tree rooted at C and E respectively.

## Tree Traversal

To retrieve information stored in a given node requires knowledge of its location in memory, which is kept by its father. Hence, a node in the tree can only be examined or visited if all its ancestors are visited first. However, it is possible to systematically examine each node in such a way that every node is visited exactly once. Such an operation is known as traversing the tree and provides the basis for the searching methods discussed below. Although several algorithms can be found in the literature to traverse a binary tree [44], attention will be centred here on the so-called preorder traversal method. This technique is embodied in the following three steps:

1. *Visit the root of the current subtree*
2. *If the left link of the root is not zero then traverse the left subtree.*
3. *If the right link of the root is not zero then traverse the right subtree.*

The procedure determined by these three steps is clearly recursive, that is, steps 2 and 3 invoke again the algorithm which they define. In order to illustrate this process, consider again the tree shown in figure 4.1b; for this tree, the repeated application of the above algorithm yields the following sequence:

```
1. Traverse the tree {A, B, C, D, E, F, G, H}
        1.1. Visit A
```

```
1.2. Traverse the tree {B}
     1.2.1. Visit B
     1.2.2. Skip
     1.2.3. Skip
1.3. Traverse the tree {C, D, E, F, G, H}
     1.3.1. Visit C
     1.3.2. Traverse the tree {D, F}
            1.3.2.1. Visit D
            1.3.2.2. Traverse the tree {F}
                     1.3.2.2.1 Visit F
                     1.3.2.2.2 Skip
                     1.3.2.2.3 Skip
            1.3.2.3 Skip
     1.3.3. Traverse the tree {E, G, H}
            1.3.3.1. Visit E
            1.3.3.2. Traverse the tree {G}
                     1.3.3.2.1 Visit G
                     1.3.3.2.2 Skip
                     1.3.3.2.3 Skip
            1.3.3.3. Traverse the tree {H}
                     1.3.3.3.1 Visit H
                     1.3.3.3.2 Skip
                     1.3.3.3.3 Skip
```

Thus, the nodes of the tree in figure 4.1b in preorder are A, B, C, D, F, E, G and H.

We notice in the above algorithm that, before moving on to traverse the left subtree - step 2 in the previous algorithm - it is necessary to store the value of the right link, that is, the address of the right son, in order to enable the subsequent traversal of the right subtree. Moreover, whilst traversing the left subtree it is likely that additional right links will have to be stored. In fact, a list containing the addresses of all right subtrees encountered along the way which are yet to be traversed, must be kept and has to be continuously updated as follows. After visiting each node, the right link, if different from zero, is added to the list and if the left link is not zero the left subtree is traversed. When a zero left link is encountered, the last right link inserted in the list is retrieved, as well as removed, from the list and the subtree rooted at this address is traversed.

This type of list, in which items are inserted one by one and extracted, also one at a time, in the reverse order, is known as a stack [44]. A stack consists of a linear array, or vector, together with an integer variable to record the number of items in the array. This variable, being initially zero, is increased by one every time an item is added to the stack and decreased by one when an item is extracted from it.

With the help of a stack, any recursive algorithm can be implemented without the need to use recursive routines. For instance, a non-recursive implementation of the traversal algorithm given above can be symbolically expressed as:

```
0.a Set root_address = address of the root node
0.b Set stack_size = 0
   1.  Visit the node stored at root_address
   2.  If right_link $\neq$ 0 then:
```

```
            Set stack_size = stack_size +1
            Set stack(stack_size) = right_link
        endif
3a.     If left_link ≠ 0 then:
            Set root_address = left_link
            go to 1
        endif
3b.     If left_link = 0 then:
            If stack_size ≠ 0 then:
                Set root_address = stack(stack_size)
                Set stack_size = stack_size - 1
                go to 1
            endif
        endif
        If stack_size = 0            →      terminate the process
```

### Inserting and Deleting nodes

In order to add a new data item to a binary tree, a node containing the new item of information must be created and stored in a convenient memory location. The left and right links of this node are set to zero. If the current tree is empty, the new node becomes the root of the tree, otherwise the node must be inserted or linked to the existing tree. To achieve this, the tree is followed downwards, starting from the root and jumping from father to son, until a blank link is found. This link is then set to the memory position of the new node. When moving down the tree, a criterion must be provided at each node to chose between the left or right branches. This criterion determines the final position in the tree of the new node and, consequently, the shape of the tree itself.

Deleting a node from a binary tree is a straightforward operation if the undesired node is a terminal node; changing to zero the corresponding link of its father effectively 'prunes' the node from the tree and renders the memory occupied by it available for future uses. In the case of an intermediate node, the process becomes slightly more complicated since a gap can not be left in the tree. To overcome this problem, the unwanted node is replaced by a terminal node chosen from among its descendants. This operation can be carried out by modifying the links to suit the new structure of the tree and without moving the nodes from their memory positions. Figures 4.2a and 4.2b illustrate the deletion of node C from the tree shown in figures 4.1a and 4.1b and its replacement by node H.

If the application at hand demands frequent deletion and insertion, a memory book keeping system is necessary for the efficient implementation of tree structures. This is required so that new nodes can be placed in the memory space released by the deletion of previous nodes. This problem can be solved by using a linked list structure to record all the available memory spaces. A linked list is a data structure that differs from the binary tree data structure described above in that every node has always only one link pointing at another node, and every node has always one link pointing at it. There are two exceptions, which are the *head* and the *end* nodes. The head is a node with no link pointing at it - the address of which is kept separately - and the end is a node with a blank link.

As shown in figure 4.3 the two data structures, binary tree and linked list, are updated simultaneously. Initially, the available memory is partitioned into cells of the correct size to store tree nodes. These cells, which contain no relevant

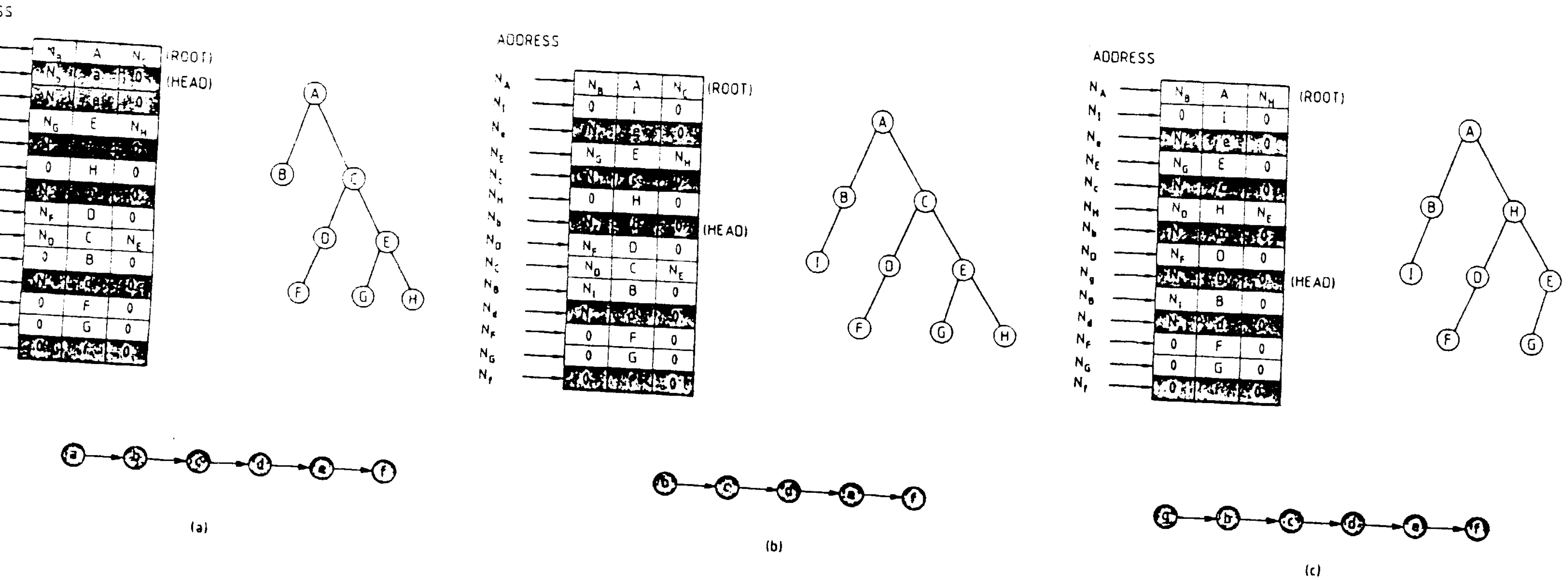


Figure 4.3 Binary tree/linked list configurations: (a) tree shown in figure 4.1, (b) after inserting node I using storage released by node and (c) after deleting node C.

information other than a single link, are then joined together to form a linked list. Every time a node needs to be inserted into the tree, the memory space required by this new tree node is generated by removing a node from the list (see figure 4.3b). Similarly, when a node is deleted from the tree it is added to the list (see figure 4.3c). Inserting and deleting nodes in the list always takes place at the head. To insert a node into the list, the link of the new node is set equal to the address of the head and the inserted node becomes the new head of the list. The deletion of the head node can be done by simply allowing its link to be the new head.

## *4.2 The Alternating Digital Tree*

Consider a set of n points in a N dimensional space ($R^N$ ) and assume for simplicity that the coordinate values of their position vectors $\{\underline{x}_1, \underline{x}_2 \ldots \underline{x}_n\}$, after adequate scaling, vary within the interval [0,1). The aim of geometric searching algorithms is to select from this set those points that lie inside a given subregion of the space. To facilitate their representation, only rectangular - or 'hyper-rectangular' - regions will be considered, thereby allowing their definition in terms of the scaled coordinates of the lower and upper vertices as $(\underline{a}, \underline{b})$.

Comparing the coordinates of each point k with the vertex coordinates of a given subregion to check whether the condition $a_i \leq x^k_i \leq b_i$ is satisfied for i = 1, 2 ... N, would render the cost of the searching operation proportional to the number of points n. This computational expense, however, can be substantially reduced by storing the points in a binary tree, in such a way that the structure of the tree reflects the positions of the points in space. There exist several well known algorithms that will accomplish this effect for one dimensional problems; the most popular are the binary search tree and digital tree methods [41,43]. Binary search trees have been extended to N dimensional problems in [45], but the resulting tree structure, known ad N-d *trees*, do not allow the efficient deletion of nodes. The algorithm presented here is a natural extension of the one dimensional digital tree algorithm and overcomes the difficulties encountered in N-d trees.

### Definition and node insertion

Broadly speaking, an alternating digital tree can be defined as a binary tree in which a set of n points are stored following certain geometrical criteria. These criteria are based on the similarities arising between the hierarchical and parental structure of a binary tree and a recursive bisection process: each node in the tree has two sons, likewise a bisection process divides a given region into two smaller subregions. Consequently, it is possible to establish an association between tree nodes and subregions of the unit hypercube as follows: the root represents the unit hypercube itself; this region is now bisected across the $x^1$ axis and the region for which $0 \leq x^1 < 0.5$ is assigned to the left son and the region for which $0.5 \leq x^1 < 1$ is assigned to the right son; at each of these nodes the process is repeated across the $x^2$ direction as shown in figure 4.4. In a two dimensional space this process can be repeated indefinitely by chosing $x^1$ and $x^2$ directions in alternating order; similarly, in a general N dimensional space, the process can be continued by choosing directions $x^1$, $x^2$, ... $x^N$ in cyclic order.

Generally, if a node k at the hierarchy level m - the root being level 0 - represents a region $(\underline{c}_k, \underline{d}_k)$, the subregions associated to its left and right sons, $(\underline{c}_{kl}, \underline{d}_{kl})$ and $(\underline{c}_{kr}, \underline{d}_{kr})$ result from the bisection of $(\underline{c}_k, \underline{d}_k)$ by a plane normal to the j-th coordinate axis, where j is chosen cyclically from the N space directions as:

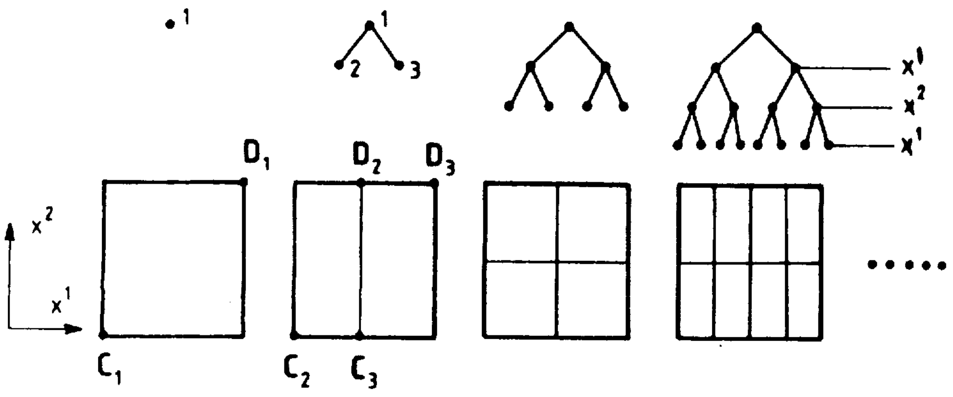
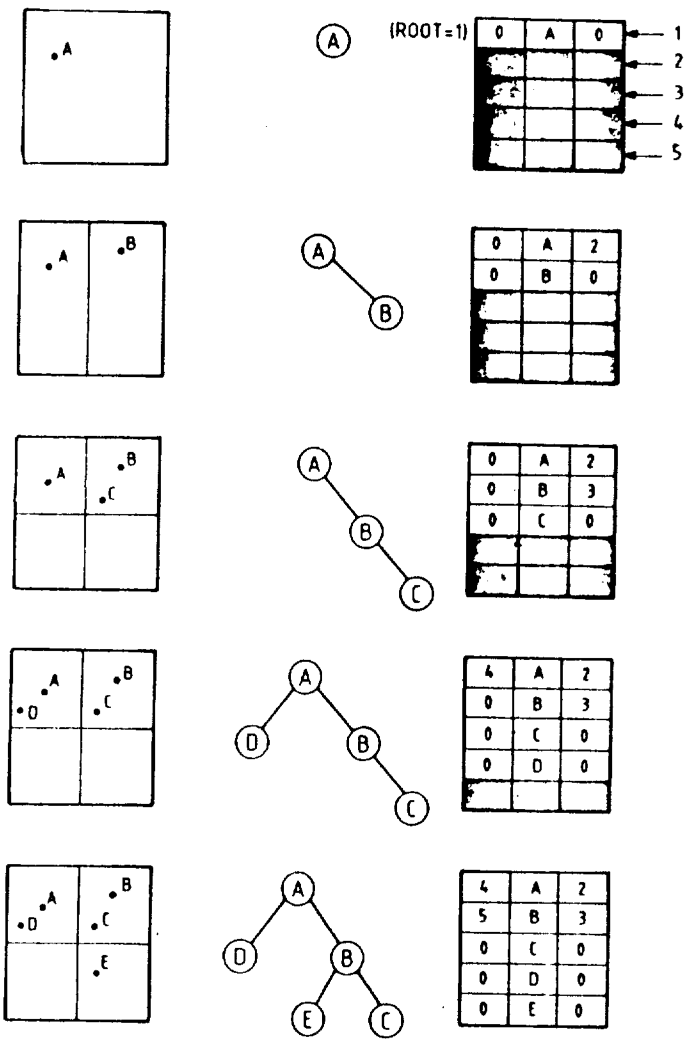


Figure 4.4 Relation between a binary tree and a bisection process.



| | | |
|---|---|---|
| 0 | A | 0 |
| | | |
| | | |
| | | |
| | | |

| | | |
|---|---|---|
| 0 | A | 2 |
| 0 | B | 0 |
| | | |
| | | |
| | | |

| | | |
|---|---|---|
| 0 | A | 2 |
| 0 | B | 3 |
| 0 | C | 0 |
| | | |
| | | |

| | | |
|---|---|---|
| 4 | A | 2 |
| 0 | B | 3 |
| 0 | C | 0 |
| 0 | D | 0 |
| | | |

| | | |
|---|---|---|
| 4 | A | 2 |
| 5 | B | 3 |
| 0 | C | 0 |
| 0 | D | 0 |
| 0 | E | 0 |

Figure 4.5 Building an ADT by successive insertion.

$$j = 1 + \mathrm{mod}(m,N) \qquad (4.1)$$

and mod(m,N) denotes the remainder of the quotient of m over N. Hence ($\underline{c}_{kl}$, $\underline{d}_{kl}$) and ($\underline{c}_{kr}$, $\underline{d}_{kr}$) are obtained as:

$$c_{kl}^i = c_k^i,\ d_{kl}^i = d_k^i \quad \text{for } i \neq j \text{ and} \quad c_{kl}^j = c_k^j,\ d_{kl}^j = \frac{1}{2}(c_k^j + d_k^j) \qquad (4.2a)$$

$$c_{kr}^i = c_k^i,\ d_{kr}^i = d_k^i \quad \text{for } i \neq j \text{ and} \quad c_{kl}^j = \frac{1}{2}(c_k^j + d_k^j),\ d_{kl}^j = d_k^j \qquad (4.2b)$$

This correlation between nodes and subdivisions of the unit hypercube allows an ADT to be further defined by imposing that each point in the tree should lie inside the region corresponding to the node where it is stored. Consequently, if node k of an ADT structure contains a point with coordinates $\underline{x}_k$, the following condition must be satisfied:

$$c_k^i \leq x_k^i < d_k^i \qquad \text{for } i = 1,2 \ldots N \qquad (4.3)$$

Due to this additional requirement there exists only one possible way in which a new point can be inserted in the tree. As discussed in the previous section the tree is followed downwards until an unfilled position where the node can be placed is found. During this process, however, left or right branches are now chosen according to whether the new point lies inside the region related to the left or right sons, thereby ensuring that condition (4.3) is satisfied.

Given a predetermined set of n points, an ADT structure can be built by placing anyone point at the root and then inserting the remaining points in consecutive order according to the algorithm described above. This is illustrated in figure 4.5 for a set of 5 points {A,B,C,D,E}. The shape of the tree obtained in this way depends mainly on the spatial distribution of the points and somewhat on the order in which the points were inserted. The cost of operations like node insertion/deletion and geometric searching depends strongly on the shape of the tree; generally poor performances are to be expected from highly degenerated trees (see figure 4.6), whereas well balanced trees (see figure 4.7), as those obtained for fairly uniform distributions of points, will result in substantial reductions of the searching cost. In these cases the average number of levels in the tree, and therefore the average cost of inserting a new point, becomes proportional to log(n); clearly a considerable cost if compared with the cost of storing the points in a sequential list, but fully justifiable in view of the reduction in searching costs that ADT structures will provide.

### Geometric searching

Consider now a set of points stored in an ADT structure. The fact that condition (4.3) is satisfied by every point provides the key to the efficient solution of a geometric searching problem. To illustrate this, note first that the recursive structure of the bisection process described above implies that the region related to a given node k contains all the subregions related to nodes descending from k; consequently, all points stored in these nodes must also lie inside the region represented by node k. For instance, all points in the ADT structure are stored in nodes descended from the root

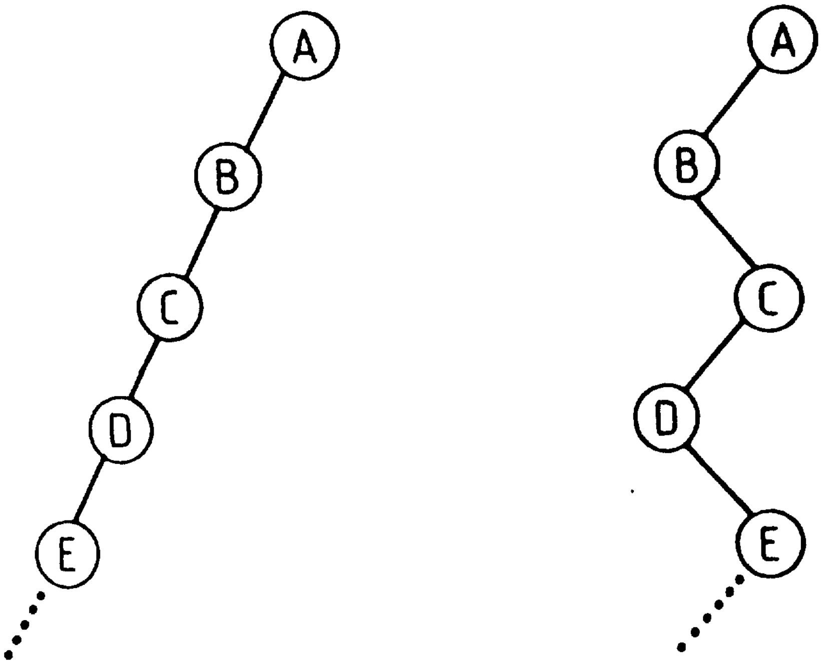


Figure 4.6 Degenerated trees.

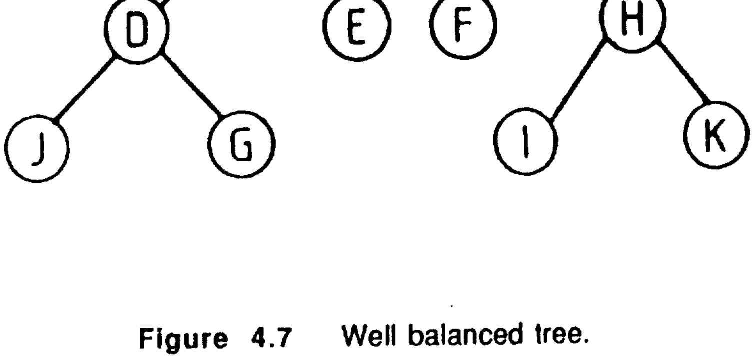


Figure 4.7 Well balanced tree.

and, clearly, all of them lie inside the unit hypercube - the region associated with the root. Analogously, the complete set of points stored in any subtree is inside the region represented by the root of the subtree.

This feature can be effectively used to reduce the cost of a geometric searching process by checking, at any node k, the intersection between the searching range $(\underline{a}, \underline{b})$ and the region represented by node k, namely $(\underline{c}_k, \underline{d}_k)$. If these two regions fail to overlap, then the complete set of points stored in the subtree rooted at k can be disregarded from the search, thus avoiding the need to examine the coordinates of every single point.

Consequently, a systematic procedure to select the points that lie inside a given searching range $(\underline{a}, \underline{b})$ can be derived from the traversal algorithm previously presented. Now the generic operation 'visit the root' can be re-interpreted as checking whether the point stored in the root falls inside the searching range. Additionally, the left and right subtrees need to be traversed only if the regions associated with their respective root nodes intersect with the range. Accordingly, a geometric searching algorithm emerges in a recursive form as:

1. *Check whether the coordinates of the node stored in the root, say* $\underline{x}_k$, *are inside* $(\underline{a}, \underline{b})$ *i.e. check whether* $a^i \le x_k^i < b^i$ *for* i = 1,2 ... N.
2. *If the left link of the root is not zero and the region* $(\underline{c}_{kl}, \underline{d}_{kl})$ *overlaps with* $(\underline{a}, \underline{b})$ *i.e. if* $d_{kl}^i \ge a^i$ *and* $c_{kl}^i \le b^i$ *for* i = 1,2 ... N, *search the left subtree.*
3. *If the right link of the root is not zero and the region* $(\underline{c}_{kr}, \underline{d}_{kr})$ *overlaps with* $(\underline{a}, \underline{b})$ *i.e. if* $d_{kr}^i \ge a^i$ *and* $c_{kr}^i \le b^i$ *for* i = 1,2 ... N, *search the right subtree.*

In order to illustrate this process, consider the set of points and the searching range shown in figure 8a and the corresponding alternating digital tree depicted in figure 8b. For this simple example, the algorithm given above results in the following sequence of steps:

Search the tree {A,B,C,D,E,F,G,H}:

```
1. Check if $a^i \le x_A^i \le b^i$ for i = 1,2
2. Since $d_B^i \ge a^i$ and $c_B^i \le b^i$ search the tree {B,C,D,E}:
    2.1. Check if $a^i \le x_B^i \le b^i$
    2.2. Since $d_C^i \ge a^i$ and $c_C^i \le b^i$ search the tree {C,E}:
        2.2.1. Check if $a^i \le x_C^i \le b^i$
        2.2.2. Skip (left link is zero)
        2.2.3. Skip ($c_E^1 > b^1$)
    2.3. Skip ($c_D^2 > b^2$)
3. Skip ($c_F^1 > b^1$)
```

Again a 'non-recursive' implementation of this algorithm can be achieved using a stack in a very similar way to that previously described for the traversal algorithm.

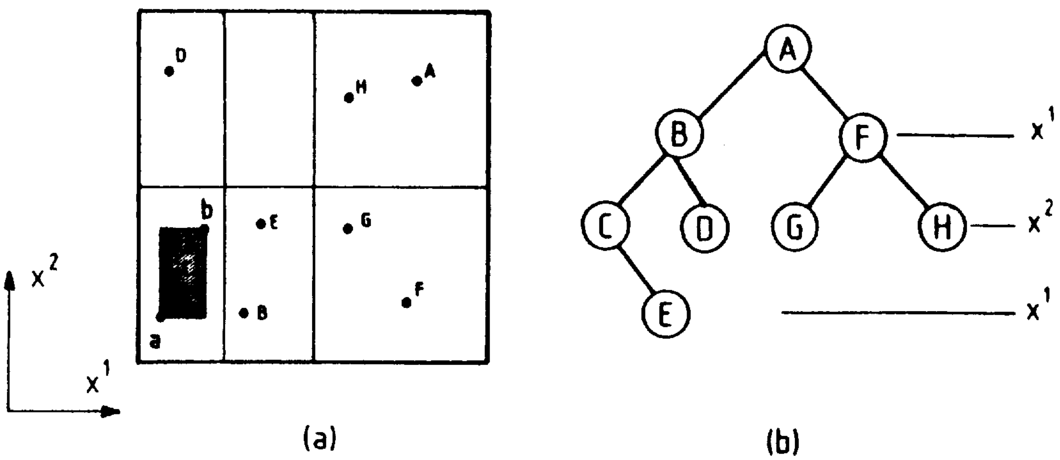


**Figure 4.8** Searching problem in R².

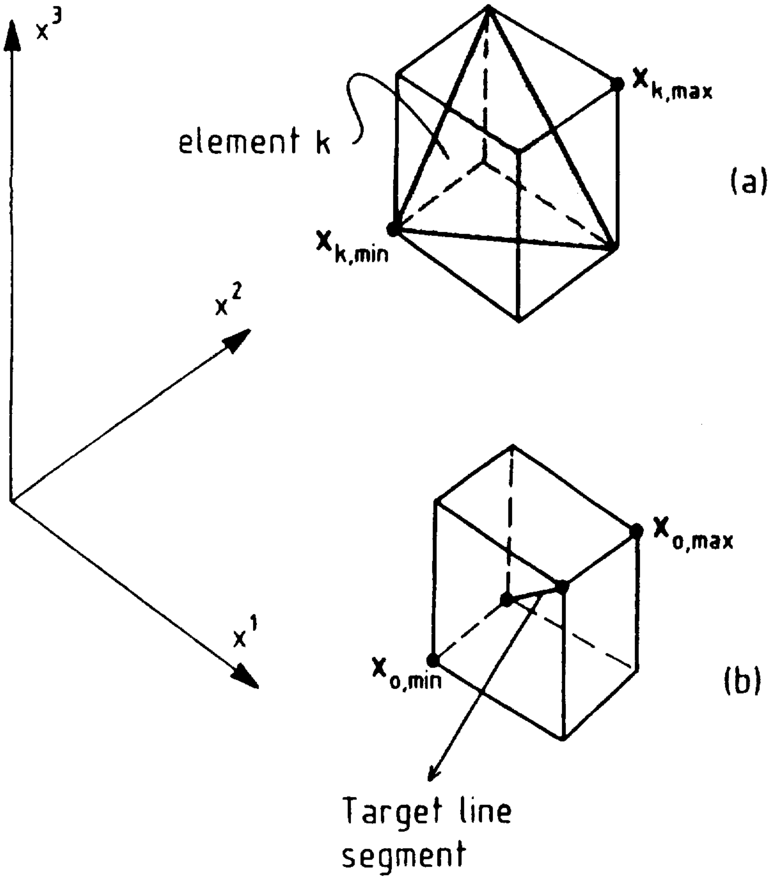


**Figure 4.9** Definition of coordinate limits for triangular elements and straight line segments.

Note that, with this technique, only the coordinates of points A,B and C are actually examined, the rest being immediately disregarded in view of their position in the tree. In general, only those points stored in nodes with associated regions overlapping ($\underline{a}$, $\underline{b}$) will be checked during the searching process.

### *4.3 Geometric Intersection*

Geometrical intersection problems can be found in many applications; for instance, a common problem that may emerge in contact algorithms [46], hidden line removal applications or in the advancing front mesh generation algorithm presented in section 3, is to determine from a set of three noded triangular elements those which intersect with a given line segment. Similar problems, involving other geometrical objects, are encountered in a wide range of geometrical applications. In general, a geometric intersection problem consists of finding from a set of geometrical objects those which intersect with a given object. If every one-to-one intersection is investigated, the solution of these problems can become very expensive, especially when complex objects such as curves or surfaces are involved. Fortunately, many of these one-to-one intersections can be quickly discarded by means of a simple comparison between the coordinate limits of every given pair of objects. For instance, a triangle with x-coordinate varying from 0.5 and 0.7 cannot intersect with a segment with x-coordinate ranging from 0.1 to 0.3. Generally, the intersection between two objects in the N dimensional Euclidean space, requires each of the N pairs of coordinate ranges to overlap. Consider for instance the intersection problem between triangular facets and a target straight line segment in $\mathbb{R}^3$; then, if ($\underline{x}_{k,min}$, $\underline{x}_{k,max}$) are the coordinate limits of element k and ($\underline{x}_{0,min}$, $\underline{x}_{0,max}$) are the lower and upper limits of the target segment (see figure 4.9), an important step towards the solution of a geometric intersection problem is to select those which satisfy the inequality:

$$\left.\begin{array}{l} x_{k,min}{}^{i} \leq x_{0,max}{}^{i} \\ \\ x_{k,max}{}^{i} \geq x_{0,min}{}^{i} \end{array}\right. \qquad \text{for } i = 1,2 \ldots N \qquad (4.4)$$

The cost of checking condition (4.4) for every element grows proportionally to n, and for very numerous sets may become prohibitive. This cost, however, can be substantially reduced by using a simple device whereby the process of selecting those elements which satisfy condition (4.4) can be interpreted as a geometric searching problem. Additionally, since the number of elements that satisfy condition (4.4) will normally be much smaller than n, the cost of determining which of these intersects with the target segment becomes affordable.

In order to interpret condition (4.4) as a geometric searching problem, it is first convenient to assume that all the elements to be considered lie inside a unit hypercube - a requirement that can be easily satisfied through adequate scaling of the coordinate values. Consequently, condition (4.4) can be re-written as:

$$\begin{array}{c} 0 \leq x_{k,min}{}^{1} \leq x_{0,max}{}^{1} \\ \cdot \\ \cdot \\ \cdot \\ 0 \leq x_{k,min}{}^{N} \leq x_{0,max}{}^{N} \end{array}$$

$$\begin{aligned} x_{0,min}^{1} &\le x_{k,max}^{1} \le 1 \\ &\vdots \\ x_{0,min}^{N} &\le x_{k,max}^{N} \le 1 \end{aligned} \qquad (4.5)$$

Consider now a given object k in $\mathbb{R}^N$ with coordinate limits $\underline{x}_{k,min}$, and $\underline{x}_{k,max}$; combining this two sets of coordinate values, it is possible to view an object k in $\mathbb{R}^N$ as a point in $\mathbb{R}^{2N}$ with coordinates $\underline{x}_k^i$ for i = 1,2 ... 2N defined as (see figure 4.10):

$$\underline{x}_k = [\, x_{k,min}^{1}, \dots x_{k,min}^{N}, x_{k,max}^{1}, \dots x_{k,max}^{N}]^{T} \qquad (4.6)$$

Using this representation of a given object k, condition (4.5) becomes simply:

$$a^i \le x_k^i \le b^i \qquad \text{for } i = 1,2 \dots 2N \qquad (4.7)$$

where $\underline{a}$ and $\underline{b}$ can be interpreted as the lower and upper vertices of a 'hyper-rectangular' region in $\mathbb{R}^{2N}$ and, recalling (4.5), their components can be obtained in terms of the coordinate limits of the target object (see figure 4.11) as:

$$\underline{a} = [\, 0, \dots 0, x_{0,max}^{1}, \dots x_{0,max}^{N}]^{T} \qquad (4.8a)$$

$$\underline{b} = [\, x_{0,min}^{1}, \dots x_{0,min}^{N}, 1, \dots 1]^{T} \qquad (4.8b)$$

Consequently, the problem of finding which objects in $\mathbb{R}^N$ satisfy condition (4.4) becomes equivalent to a geometric searching problem in $\mathbb{R}^{2N}$ i.e. obtaining the points $\underline{x}_k$ which lie inside the region limited by $\underline{a}$ and $\underline{b}$. Once this subgroup of elements has been selected, the intersection of each one of them with the target object must be checked to complete the solution of the geometric intersection problem.

### *4.4. The use of the ADT for mesh generation*

It is obvious from the advancing front algorithm described in section 3 that operations such as searching for the points inside a certain region of the space and determining intersections between geometrical objects - in this case sides and faces - will be performed very frequently. The complexity of the problem is increased by the fact that the set of faces forming the generation front changes continuously as new faces need to be inserted and deleted during the process. Clearly, for meshes consisting of a large number of elements the cost of performing this operations can be very important.

A successful implementation of the above algorithms has been accomplished by making extensive use of the ADT data structure. For instance, the algorithm of section 3.5 for tetrahedra generation employs two tree structures; one for the faces in the front and the other for the sides defined by the intersection between each pair of faces in the front (see figure 3.11). This combination allows a high degree of flexibility and the operations of insertion, deletion, geometric searching and

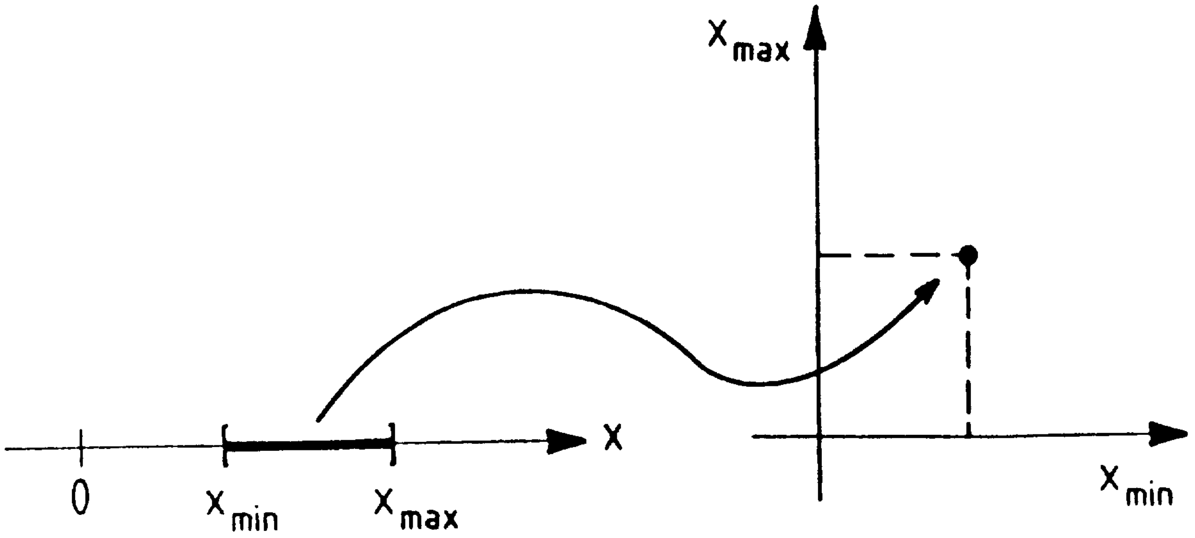


**Figure 4.10** Representation of a region in $R^1$ as a point in $R^2$.

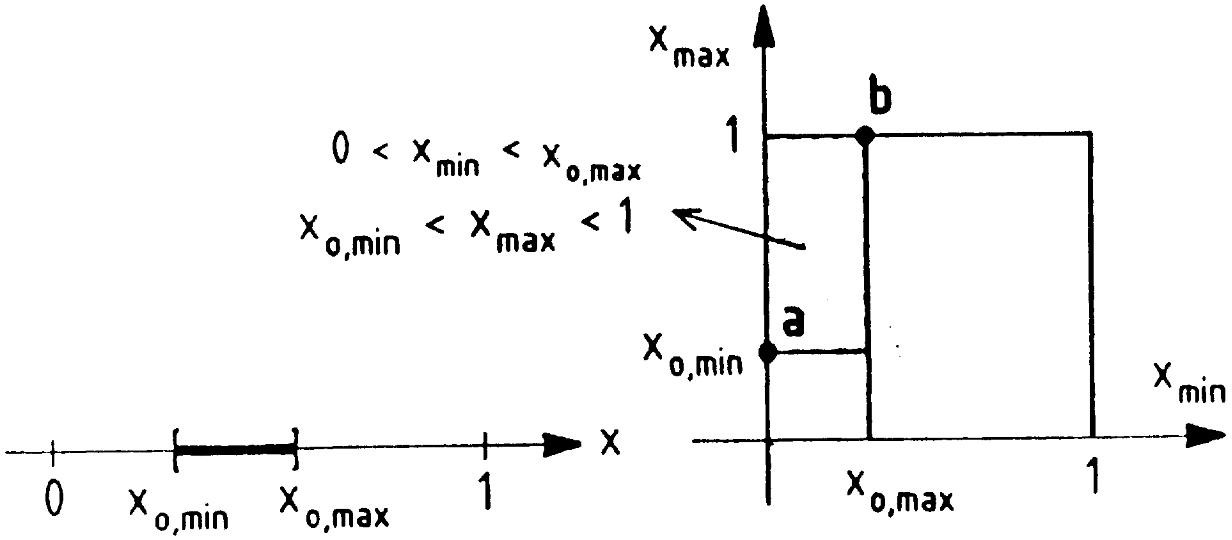


**Figure 4.11** Intersection problem in $R^1$ as a searching problem in $R^2$.

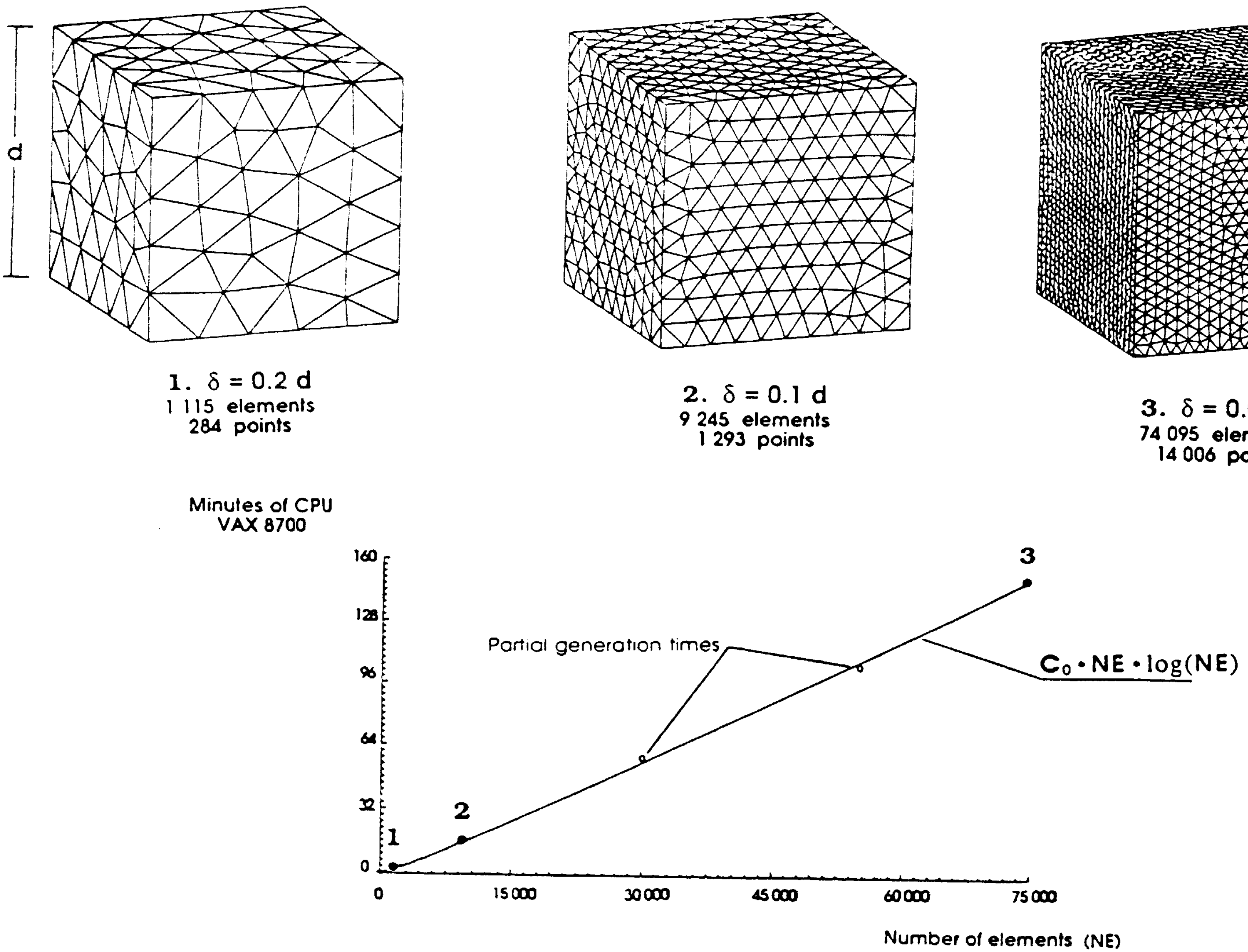


Figure 4.12 Mesh generation CPU times.

geometric intersection can be performed optimally. The overall computational performance of the algorithm is demonstrated by generating tetrahedral meshes, using the above method, for a unit cube (see figure 4.12). Different numbers of elements have been obtained by varying the mesh size. In figure 4.12 the computer time required on a VAX 8700 machine has been plotted against the number NE of elements generated. It can be observed that a typical NE*log(NE) behaviour is attained. Using this approach meshes containing up to one million elements have been generated and no degradation in the performance has been detected.

## 5. ADAPTIVITY FOR STEADY STATE PROBLEMS

The procedures described above allow for the computation of an initial approximation to the steady state solution of a given problem. This approximation can generally be improved by adapting the mesh in some manner. Here, we follow the approach of using the computed solution to predict the desired characteristics (i. e. element size and shape) for a new, adapted mesh. The ultimate aim of the adaptation procedure is to predict the characteristics of the optimal mesh. This can be defined as the mesh in which the number of degrees of freedom required to achieve a specified level of accuracy is a minimum. Alternatively, it can be interpreted as the mesh in which a given number of degrees of freedom are distributed in such a manner that the highest possible solution accuracy is achieved. In practical situations however, there are several factors which make the achievement of such optimal meshes extremely difficult. Some of these factors are:

i) The concept of optimality is intimately linked to that of accuracy, which is not uniquely defined. Hence optimality of a mesh needs to be defined with respect to a given norm or measure of the error. An additional inconvenience related to the measure of accuracy, in the present context, arises from the fact that we are attempting to solve a coupled set of non linear partial differential equations and, therefore, a rigorous measure of the error should involve all the relevant variables.

ii) For linear elliptic operators, as we have shown in Section 1, Galerkin finite element algorithms are readily derived which guarantee that the approximation obtained is the most accurate amongst all the possible approximations within the trial space of functions. Here, accuracy is defined with respect to a norm implied by the operator itself (the energy norm). For the Euler equations, however, such an energy norm does not exist and no numerical schemes are known which possess this optimality property.

iii) This best approximation property means that the error of the computed solution, measured in the energy norm, is bounded above by that of the exact interpolant. i.e. the approximation in the space of current trial functions which has exact nodal values. Using results of interpolation theory [47], it is then possible to produce rigorous bounds on the error of the numerical approximation. These results are based on certain regularity assumptions on the solution, which for the Euler equations will be invalid in the vicinity of discontinuities in the flow.

iv) Finally, the error estimates produced are based on the computed solution. As this is only an approximate solution, such error estimates will only be as good as the computed solution. This means that, even in the best situation, the optimal mesh will only be achieved in the asymptotic limit. i.e. when the solution is so good that the computed error becomes very reliable.

In view of these observations and limitations, we have made an attempt to develop a heuristic adaptive strategy. This strategy uses error estimates which are based upon concepts from interpolation theory. The possible presence of discontinuities in the solution is taken into account and, in addition, the procedure provides information about any directionality which may be present in the solution. The advantages of using directional error indicators become apparent when we consider the nature of the solutions to be computed involving flows with shocks, contact discontinuities etc.

Such features can be most economically represented on meshes which are stretched in appropriate directions. Although, these error estimates have no associated mathematical rigour, considerable success has been achieved with their use in practical situations.

The computed error, estimated from the current solution, is transformed into a spatial distribution of 'optimal' mesh spacings which are interpolated using the current mesh. The current mesh is then modified with the objective of meeting these 'optimal' distribution of mesh characteristics as closely as possible. Three alternative procedures will be discussed here for performing the mesh adaption. The resulting mesh is employed to produce a new solution and this procedure can repeated several times until the user is satisfied with the quality of the computed solution.

### *5.1 Error indicator in 1D.*

The development of a method for error indication is considerably simplified if we restrict consideration to problems involving a single scalar variable. For this reason, when solving the Euler equations, a key variable is identified and then the mesh adaptation is based on an error analysis for that variable alone. The choice of the best variable to use as a key variable remains an open question, but the the Mach number has been adopted for the computations reported in these notes.

Consider first the one dimensional situation in which the exact values of the key variable $\sigma$ are approximated by a piecewise linear function $\hat{\sigma}$. The error E is then defined as

$$E = \sigma(x^1) - \hat{\sigma}(x^1) \qquad (5.1)$$

We note here that if the exact solution is a linear function of $x^1$ then the error will vanish. This is because our approximation has been obtained using piecewise linear finite element shape functions. Moreover, if the exact solution is not linear, but is smooth, then it can be represented, to any order of precision, using polynomial shape functions.

To a first order of approximation, the error E can be evaluated as the difference between a quadratic finite element solution $\tilde{\sigma}$ and the linear computed solution. To obtain a piecewise quadratic approximation one could obviously solve a new problem using quadratic shape functions. This procedure however, although possible, is not advisable as it would be even more costly than the original computation. An alternative approach for estimating a quadratic approximation from the linear finite element solution is therefore employed. Assuming that the nodal values of the quadratic and linear approximations coincide i.e. the nodal values of E are zero, a quadratic solution can be constructed on each element, once the value of the second derivative is known. Thus the variation of the error E within an element e can be expressed as

$$E_e = \frac{1}{2}\zeta(h_e - \zeta)\left.\frac{d^2\tilde{\sigma}}{dx^{12}}\right|_e \qquad (5.2)$$

where $\zeta$ denotes a local element coordinate and $h_e$ denotes the element length. A procedure for estimating the second derivative of a piecewise linear function is described below.

The root mean square value $E_e^{RMS}$ of this error over the element can be computed as

$$E_e^{RMS} = \left\{ \int_0^{h_e} \frac{E_e^2}{h_e} \, d\zeta \right\}^{1/2} = \frac{1}{\sqrt{120}} h_e^2 \left| \frac{d^2\hat{\sigma}}{{dx^1}^2} \right|_e \tag{5.3}$$

where | . | stands for absolute value.

We define the 'optimal' mesh, for a given degree of accuracy, as the mesh in which this root mean square error is equal over each element. In the present context, this requirement may be regarded as being somewhat arbitrary. However, it has been shown [48] that the requirement of equidistribution of the error leads to optimal results when applied to certain elliptic problems. This requirement is therefore written as

$$h_e^2 \left| \frac{d^2\hat{\sigma}}{{dx^1}^2} \right| = C \tag{5.4}$$

where C denotes a positive constant.

Finally, the requirement of equation (5.4) suggests that the 'optimal' spacing $\delta$ on the new adapted mesh should be computed according to

$$\delta^2 \left| \frac{d^2\hat{\sigma}}{{dx^1}^2} \right| = C \tag{5.5}$$

*5.2 Recovery of the second derivatives.*

The first derivative of the computed solution on a mesh of linear elements will be piecewise constant and discontinuous across elements. Therefore, straightforward differentiation of $\hat{\sigma}$ leads to a second derivative which is zero inside each element and is not defined at the nodes. However, by using a recovery process, based upon a variational or weighted residual statement [12], it is possible to compute nodal values of the second derivatives from element values of the first derivatives of $\hat{\sigma}$.

To illustrate this process, consider a one dimensional domain $0 < x^1 < L$ which has been discretised into (n-1) linear two noded finite elements. The piecewise linear distribution of the computed solution $\hat{\sigma}$ is expressed as

$$\hat{\sigma} = \sum_{J=1}^{n} N_J \hat{\sigma}_J \tag{5.6}$$

where $N_J$ is the standard linear finite element shape function [12] associated to node J. Similarly, a piecewise linear approximation to the distribution of the second derivative, which we seek to determine, can be written as

$$\frac{d^2\hat{\sigma}}{dx^{12}} = \sum_{J=1}^{n} N_J \left.\frac{d^2\hat{\sigma}}{dx^{12}}\right|_J \qquad (5.7)$$

The nodal values of the second derivative may be computed from the approximate variational requirement that

$$\sum_{J=1}^{n} \left( \int_\Omega N_J N_K \, d\Omega \right) \left.\frac{d^2\hat{\sigma}}{dx^{12}}\right|_J = - \int_\Omega \left( \sum_{J=1}^{n} \frac{dN_J}{dx^1} \hat{\sigma}_J \right) \frac{dN_K}{dx^1} d\Omega$$
$$- \left(\frac{d\hat{\sigma}}{dx^1} N_K\right)_{x^1=0} + \left(\frac{d\hat{\sigma}}{dx^1} N_K\right)_{x^1=L} \qquad K= 1, ...,n \qquad (5.8)$$

The values of the derivatives at the two end points can be inserted, if known, or can be taken to be equal to the constant value of the derivative in the adjacent elements. The resulting set of algebraic equations can be solved, in a few iterations, by using a Jacobi procedure [16] or alternatively, the consistent mass matrix appearing on the left hand side of equation (5.7) can be lumped, thus yielding a diagonal system of equations. Numerical results obtained to date do not indicate any significant differences in the meshes produced by using these two approaches.

### *5.3 Extension to multidimensions*

Following the process described above, nodal values of the second derivative can be obtained from the approximate solution on the current mesh. The use of expression (5.5) then yields directly a nodal value of the 'optimal' spacing for the new mesh.

Expression (5.5) can be directly extended to the N dimensional case by writing the quadratic form

$$\delta_\beta^2 \left( \sum_{i;j=1}^{N} m^{ij} \beta^i \beta^j \right) = C \qquad (5.9)$$

where $\beta$ is an arbitrary unit vector, $\delta_\beta$ is the spacing along the direction of $\beta$, and $m^{ij}$ are the components of a NxN symmetric matrix of second derivatives

$$m^{ij} = \frac{\partial^2\hat{\sigma}}{\partial x^i \partial x^j} \qquad (5.10)$$

These derivatives are computed, at each node of the current mesh, by using the N dimensional equivalent of the procedure presented in the previous section. The meaning of equation (5.9) is graphically illustrated in figure 5.1 which shows how the value of the spacing in the $\beta$ direction can be obtained as the distance from the origin to the point of intersection of the vector $\beta$ with the surface of an ellipsoid. The directions and lengths of the axes of the ellipsoid are the principal directions and eigenvalues of the matrix **m** respectively.

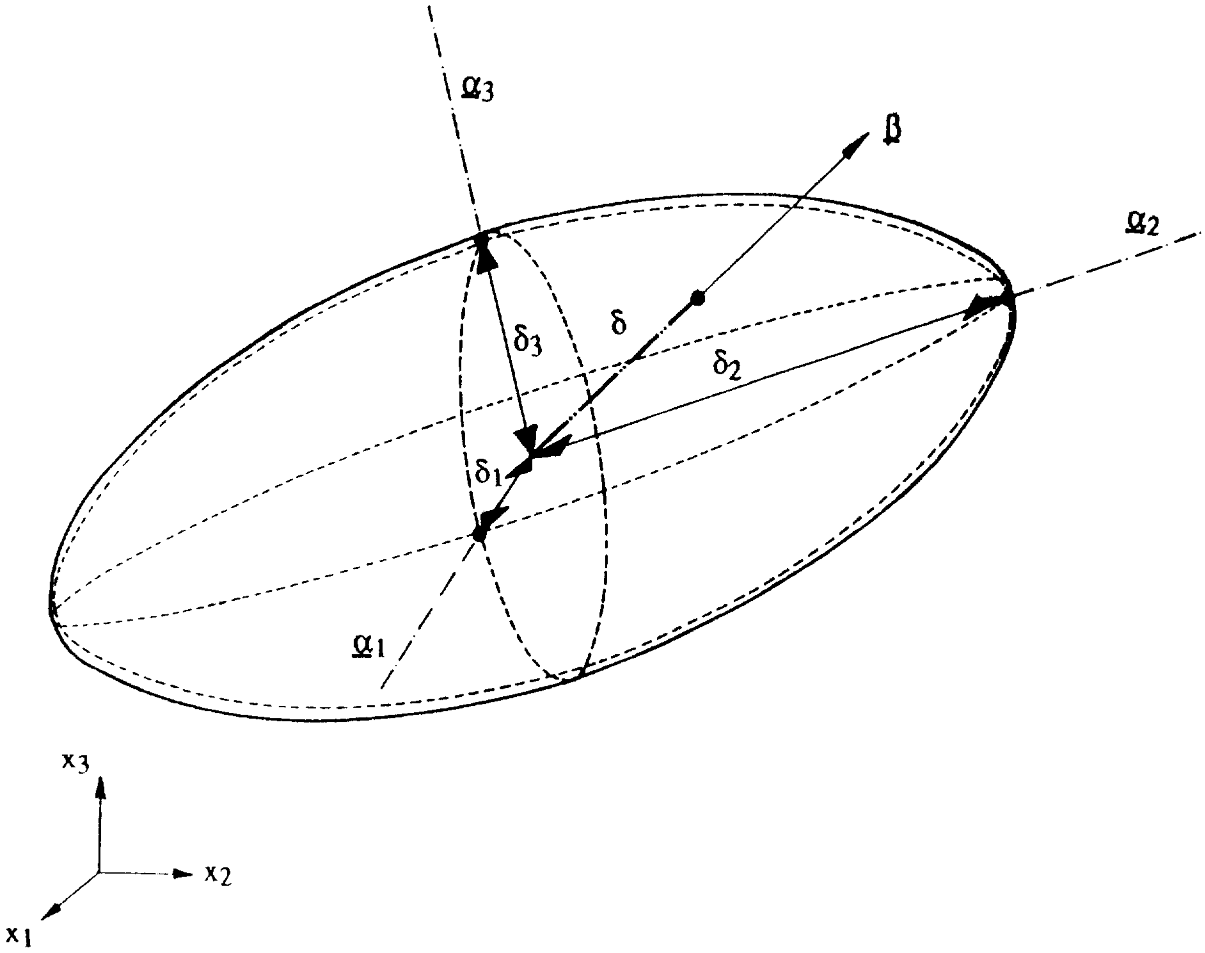

$\underset{\sim}{\alpha}_3$
$\underset{\sim}{\beta}$
$\underset{\sim}{\alpha}_2$
$\delta$
$\delta_3$
$\delta_2$
$\delta_1$
$\underset{\sim}{\alpha}_1$
$x_3$
$x_2$
$x_1$

**Figure 5.1** Determination of the value of the spacing $\delta$ along the $\underset{\sim}{\beta}$ direction.

Several alternative procedures exist for modifying an existing mesh in such a way that the requirement expressed by equation (5.8) is more closely satisfied. Three such methods will be described here. In the first procedure, called *mesh enrichment*, the nodes of the current mesh are kept fixed but some new nodes/elements are created. In the second procedure, referred to as *mesh movement*, the total number of elements and nodes remains fixed but their position is altered. Finally, in the *adaptive remeshing* algorithm, the mesh adaption is accomplished by completely regenerating a new mesh using the mesh generation algorithm presented in section 3.

### *5.4 Mesh enrichment*

In order to adapt a mesh using mesh enrichment, a sweep over all the sides in the mesh is made and the 'optimal' spacing in the direction of each side is computed according to expression (5.9). For each side, the matrix **m** is taken to be the average of its value at the two nodes of the side. The enrichment procedure consists of introducing an additional node for each side for which the calculated spacing is less than the length of the side. For interior sides, this additional node is placed at the mid-point of the side, whereas for boundary sides, it is necessary to refer to the boundary definition and to ensure that the new node is placed on the true boundary. When any side is subdivided in this manner, the elements associated with that side will also need to be subdivided in order to preserve the consistency of the final mesh. Figure 5.2 illustrates the three possible ways in which this element subdivision might have to be performed in two dimensions. The number of sides to be refined depends on the choice of the constant C in equation 5.9. To avoid excessive refinement in the vicinity of discontinuities, a minimum threshold value for the computed spacing can be used. When the mesh enrichment procedure has been completed, the values of the unknowns at the new nodes are linearly interpolated from the original mesh and the solution algorithm is re-started. This procedure has been successfully implemented in two and three dimensions and several impressive demonstrations of the power of this technique have been made. [8,19,49,50].

The application of the enrichment procedure in the solution of a two dimensional example is illustrated in figure 5.3. The problem solved is a Mach 8.15 flow past a double ellipse configuration at $30^0$ angle of attack. The initial mesh and two adaptively enriched meshes are shown together with the computed Mach number solutions. The application of the enrichment algorithm in three dimensions is shown in figure 5.4. The inviscid flow past a $30^0$ wedge is solved. The free stream Mach number is 3. This is a two dimensional problem computed on a three dimensional mesh. Two views of the initial mesh and solution are shown. A single application of the enrichment algorithm produces the mesh and solution which are also displayed in figure 5.4.

It can be observed, from the examples presented, how the quality of the solution is significantly improved by the application of. the enrichment procedure. The main drawback of the approach is that the number of elements increases considerably following each application of the procedure. This means that, in the simulation of practical three dimensional problems, only a small number of such adaptations can be contemplated.

### *5.5 Mesh movement*

For the mesh movement alogrithm, the element sides are considered as springs of prescribed stiffness and the nodes are moved until the spring system is in

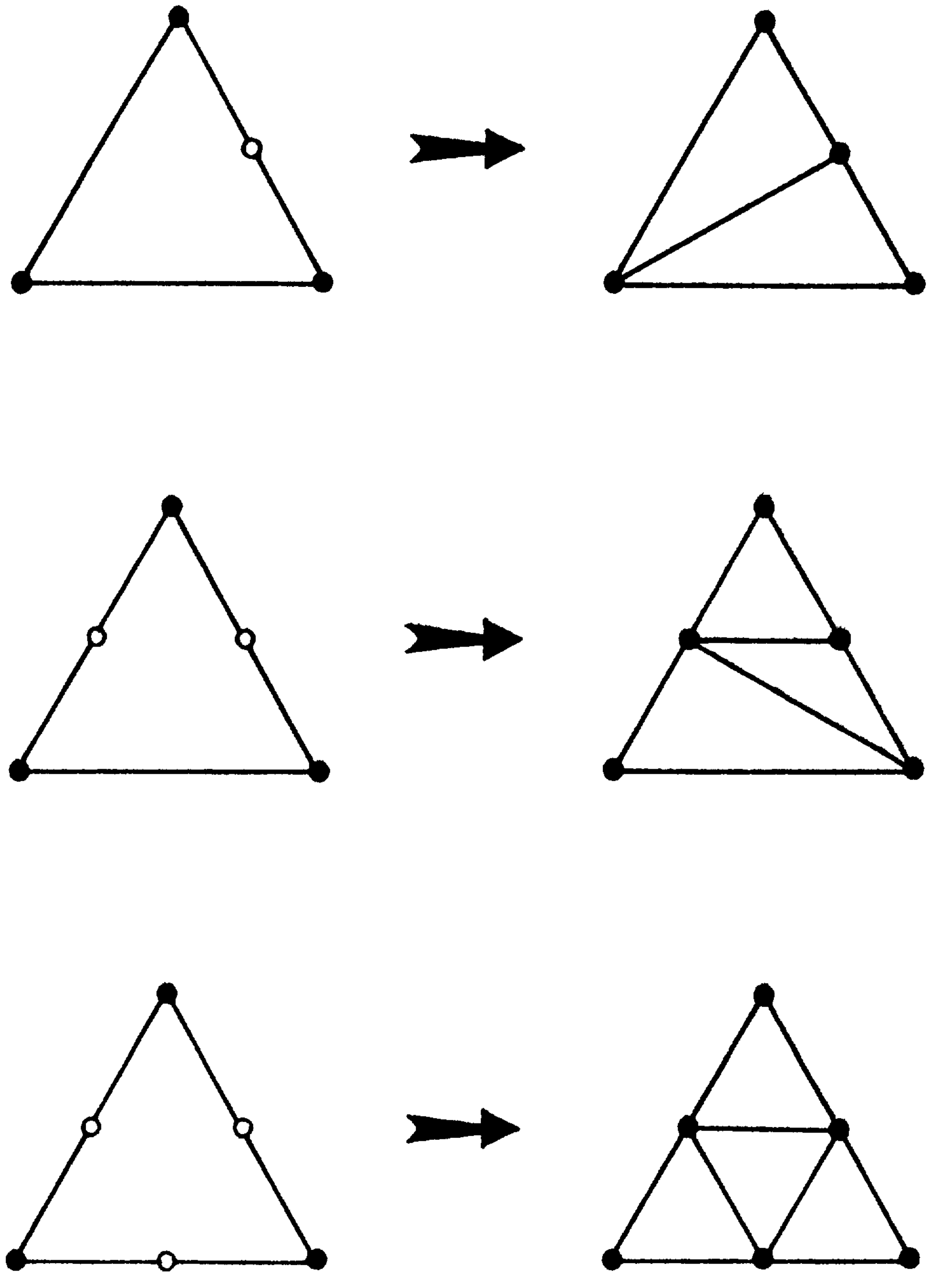


**Figure 5.2** Mesh enrichment: three possible cases of refinement for a triangle

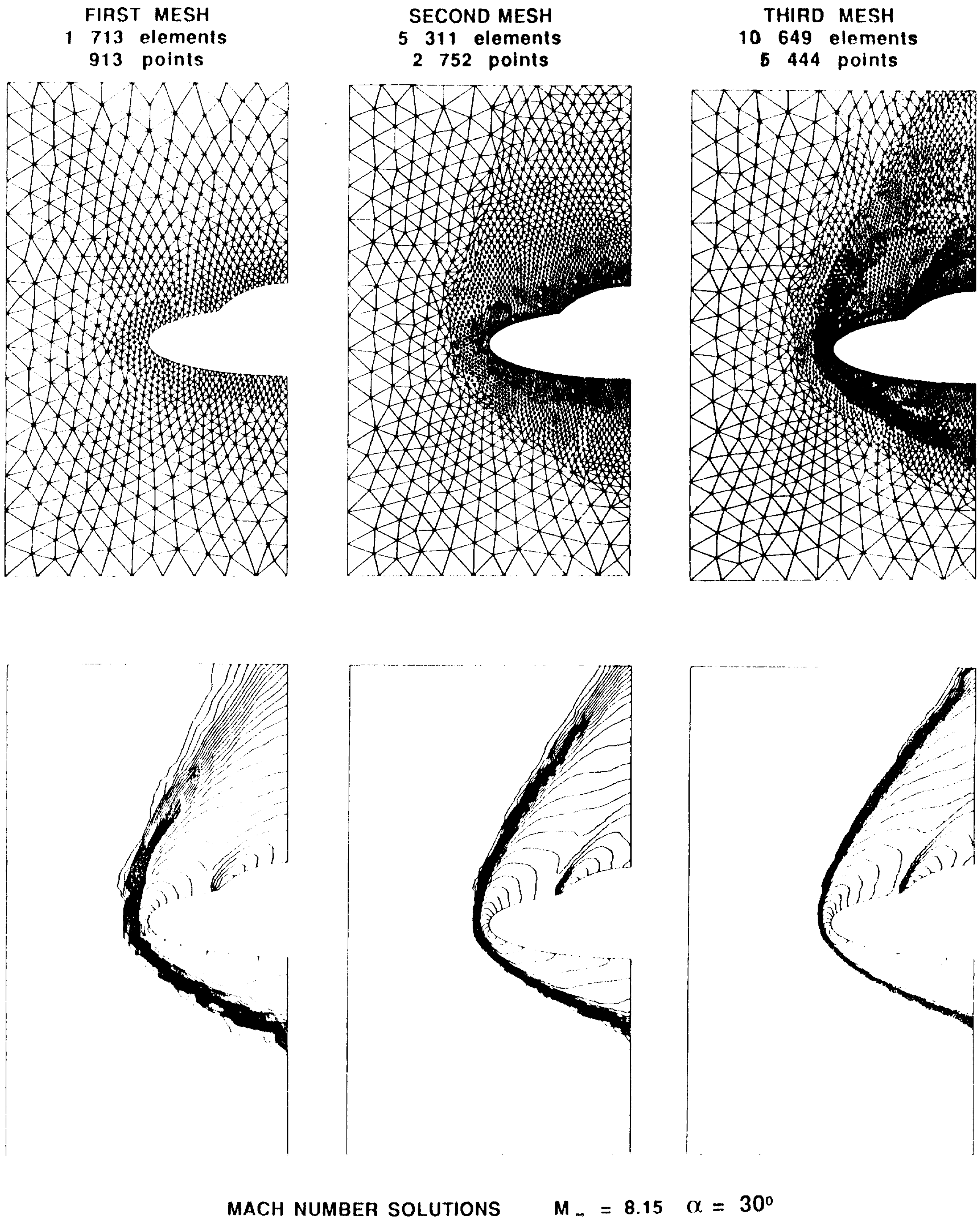


Figure 5.3 Supersonic flow past a double ellipse configuration. Sequence of meshes and solutions obtained using adaptive enrichment.

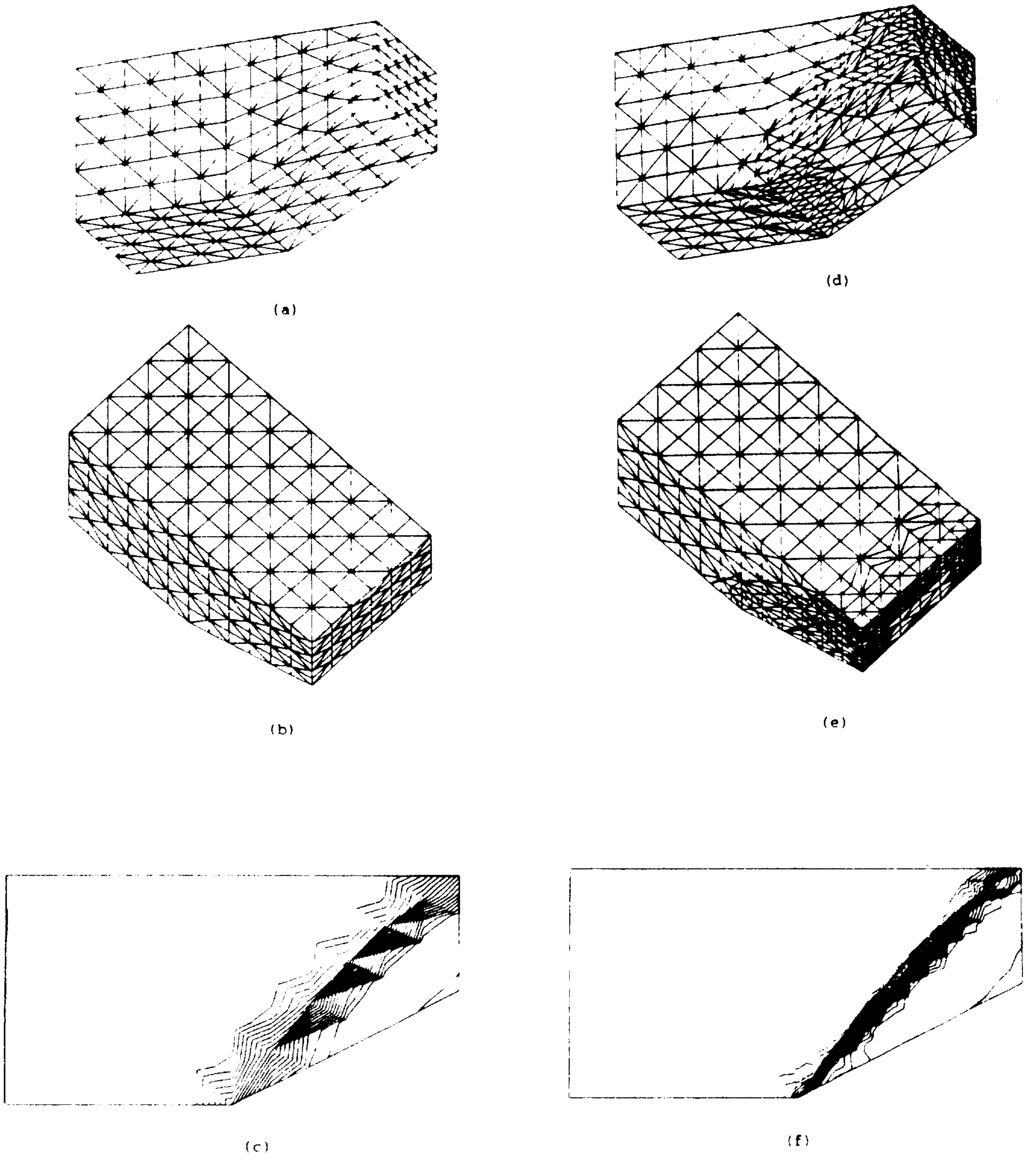


Figure 5.4 Mach 3 inviscid flow past a 30$^0$ wedge in 3D: (a), (b) two views of the initial mesh and (c) the compuited density contours; (d), (e) two views of the enriched mesh and (f) the final density contours.

equilibrium. Consider two adjacent nodes J and K as shown in figure 5.5. The force $\underline{f}_{JK}$ exerted by the spring connecting these two nodes can be taken to be

$$\underline{f}_{JK} = C_{JK} \; ( \underline{r}_J - \underline{r}_K ) \qquad (5.11)$$

where $C_{JK}$ is the stiffness of the spring and $\underline{r}_J$ and $\underline{r}_K$ are the position vectors of nodes J and K respectively. Assuming that

$$h = | \underline{r}_J - \underline{r}_K | \qquad (5.12)$$

the adaptation requirement of equation (5.9) will be satisfied if the spring stiffnesses are defined as

$$C_{JK} = h \left| \sum_{i;j=1}^{N} m^{ij} \, n_{JK}{}^{i} n_{JK}{}^{j} \right| \qquad (5.13)$$

Here $\underline{n}_{JK}$ is the unit vector in the direction of the side joining nodes J and K. For equilibrium, the sum of spring forces at each node should be equal to zero. The assembled system can be brought into equilibrium by simple iteration. In each iteration, a loop is performed over all the interior nodes and new nodal coordinates are calculated according to the expression

$$\underline{r}_J^{NEW} = \frac{\sum_{K=1}^{S_J} C_{JK} \; \underline{r}_K}{\sum_{K=1}^{S_J} \underline{r}_K} \qquad (5.14)$$

where the summation extends over the number of nodes, $S_J$, which surround node J. Sufficient convergence is normally achieved after three to five passes through this procedure.

This technique will not necessarily produce meshes of better quality, as badly formed elements can appear in regions (such as shocks) in which the spring coefficients $C_{JK}$ vary rapidly over a short distance. To avoid this problem, the definition of the value of $C_{JK}$ given in equation is (5.13) can be replaced by an expression of the form

$$C_{JK}{}^{MOD} = 1 + \frac{A \; C_{JK}}{B \; + \; C_{JK}} \qquad (5.15)$$

This can be regarded as a blending function definition for the spring stiffnesses and it has been constructed so as to ensure that, with a suitable choice for the constants A and B, excessively small or excessively large element sizes are avoided. This, in turn means that meshes of acceptable quality will be produced. More sophisticated procedures for controlling the quality of the mesh during movement can also be devised [51] and mesh movement algorithms have been successfully used in two and three dimensional flow simulations on both structured and unstructured meshes [15,51,52].

The mesh movement algorithm described has been applied to the problem of flow past a double ellipse configuration which has been treated previously. Figure 5.6 shows the solutions produced following two mesh adaptations. It can be seen that the improvement obtained after the second adaptation is minor. This is because the algorithm does not allow for the creation of new nodes and so the quality of the final solution is very much dependent on the topology of the initial mesh. This is a major drawback of the mesh movement strategy. A possible remedy to this problem is to combine mesh enrichment and mesh movement procedures. This is demonstrated in figure 5.7 which shows the application of the movement procedure to the final enriched mesh of figure 5.3.

### *5.6 Adaptive remeshing*

The basic idea of the adaptive remeshing technique is to use the computed solution to provide information on the spatial distribution of the mesh parameters. This information will be used by the mesh generator described in section 3 to generate a completely new adapted mesh for the problem under investigation.

The 'optimal' values for the mesh parameters are calculated at each node of the current mesh. The directions $\boldsymbol{\alpha}_i$; i=1, ..., N, are taken to be the principal directions of the matrix **m**. The corresponding mesh spacings are computed from the eigenvalues $e_i$; i=1, ..., N, as

$$\delta_i = \sqrt{\frac{C}{e_i}} \qquad \text{for} \quad i=1, \ldots, N \qquad (5.16)$$

The spatial distribution of the mesh parameters is defined when a value is specified for the constant C. The total number of elements in the adapted mesh will depend upon the choice of this constant. For smooth regions of the flow, this constant will determine the value of the root mean square error in the key variable that we are willing to accept. Therefore this constant should be decreased each time a new mesh adaption is performed. On the other hand, solutions of the Euler equations are known to exhibit discontinuities. At such discontinuities, the root mean square error will always remain large and therefore a different strategy is needed in the vicinity of such features.

In the practical implementation of the present method, two threshold values for the computed spatial distribution of spacing are used: a minimum spacing $\delta_{min}$ and a maximum spacing $\delta_{max}$, so that

$$\delta_{min} \leq \delta_i \leq \delta_{max} \qquad \text{for} \quad i=1, \ldots, N \qquad (5.17)$$

The reason for defining the maximum value $\delta_{max}$ is to account for the possibility of a vanishing eigenvalue in (5.16) which would render that expression meaningless. The value of $\delta_{max}$ is chosen as the spacing which will be used in the regions where the flow is uniform (the far field, for instance). On the other hand, maximum values of the second derivatives occur near the discontinuities (if any) of the flow where the error indicator will demand that smaller elements are required. By imposing a minimum value $\delta_{min}$ for the mesh size, we attempt to avoid an excessive concentration of elements near discontinuities. As the flow algorithm is known to spread discontinuities over a fixed number of elements (i.e. two or three), $\delta_{min}$ is therefore set to a value that is considered appropriate to ensure that discontinuities

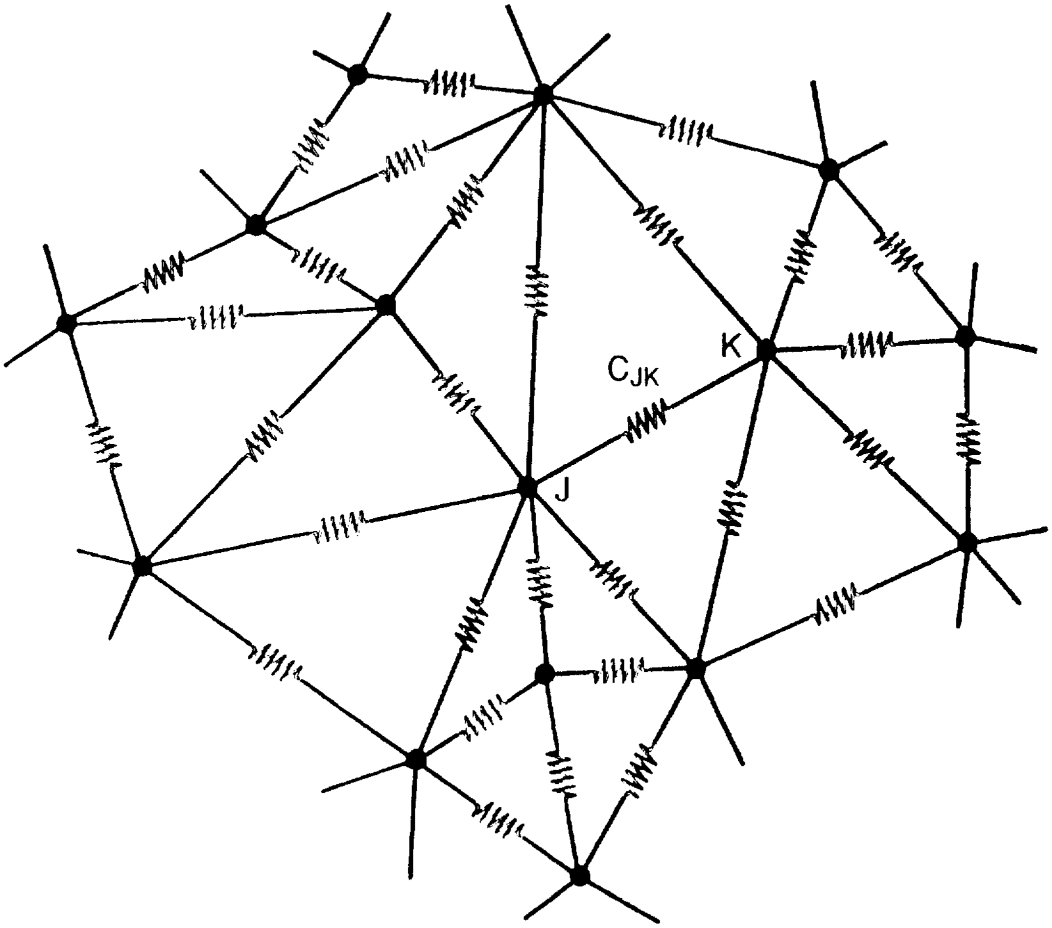


**Figure 5.5** Mesh movement algorithm. Element sides are replaced by springs.

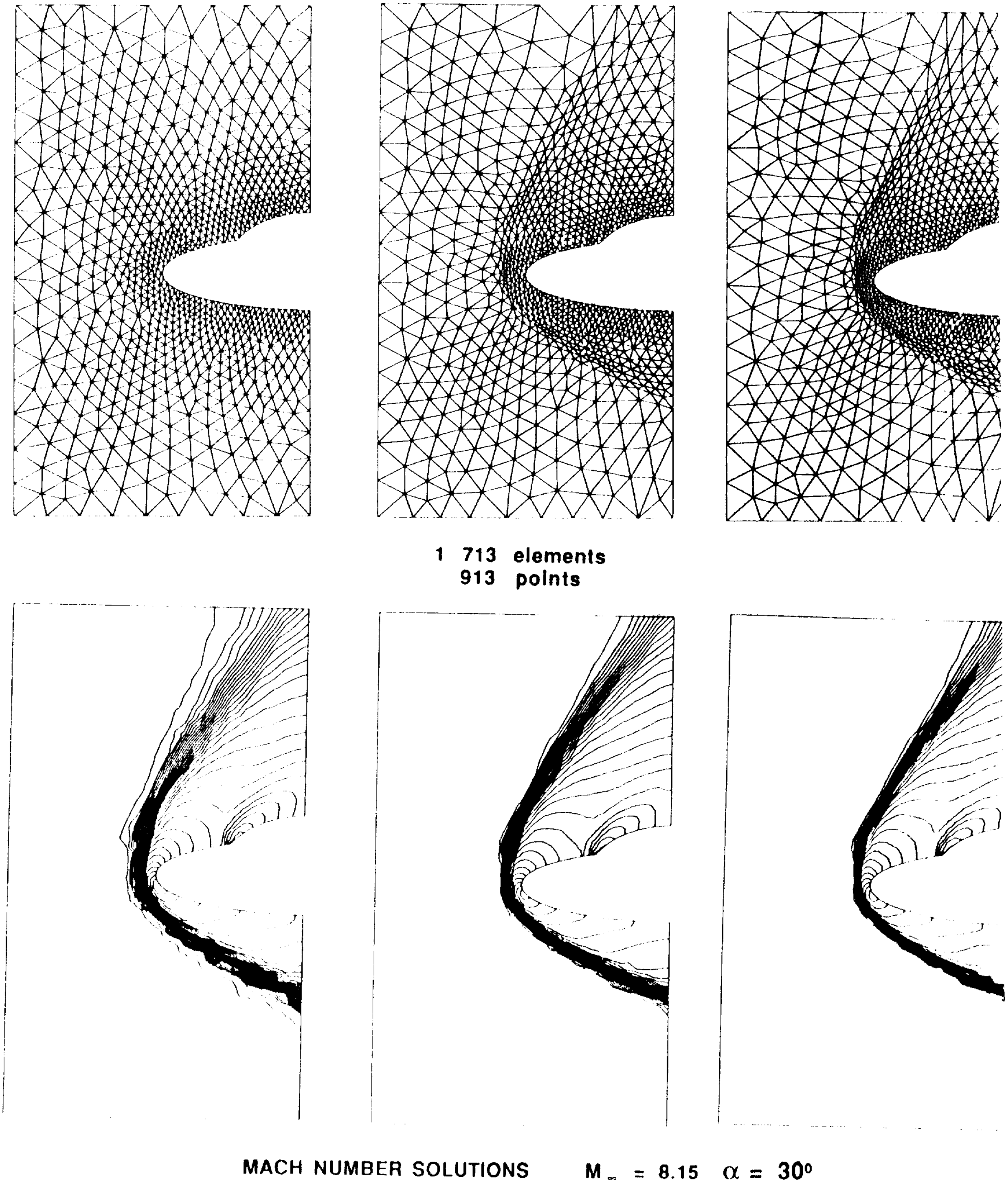


Figure 5.6 Supersonic flow past a double ellipse configuration. Sequence of meshes and solutions obtained using adaptive movement.

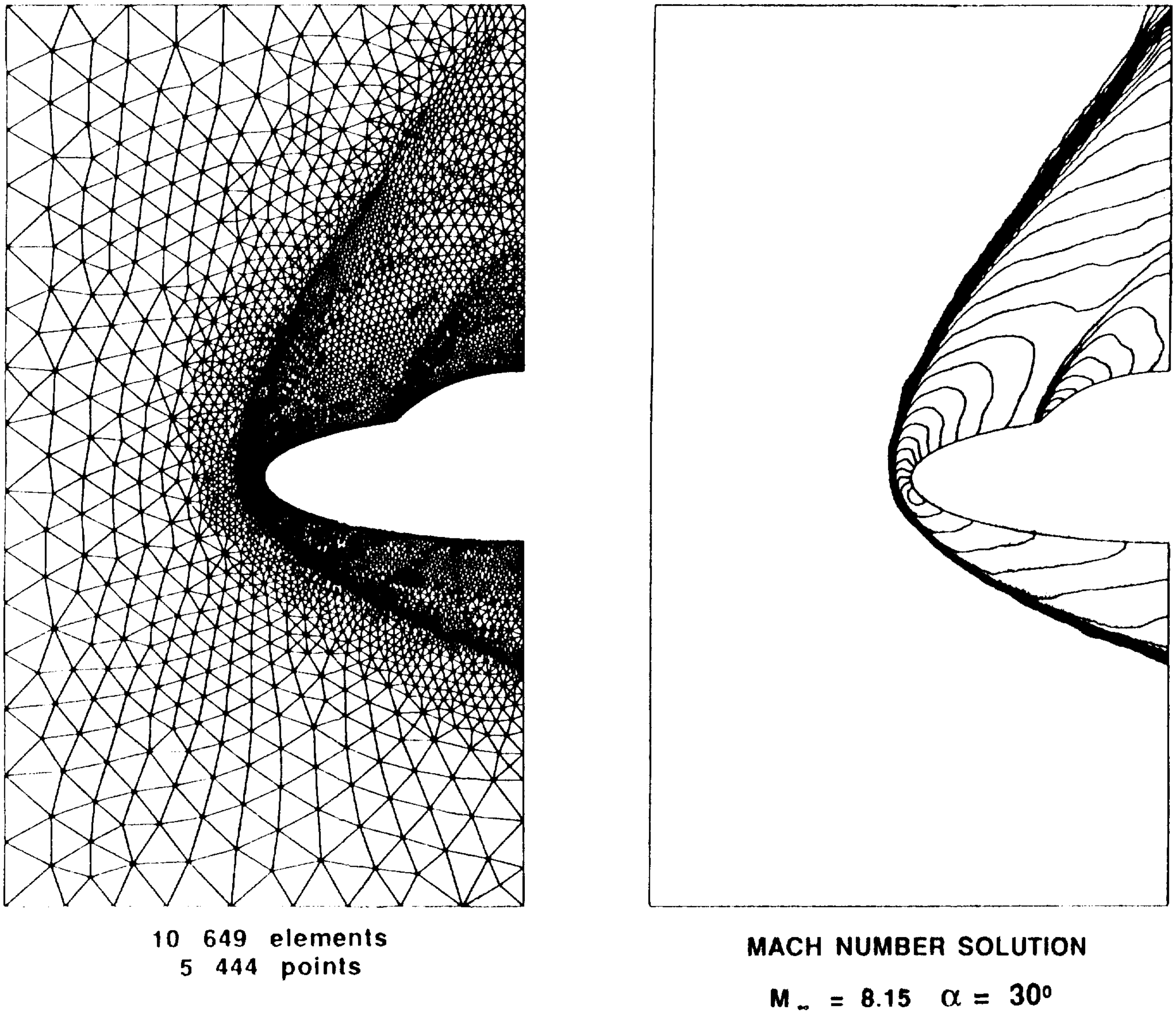


Figure 5.7 Supersonic flow past a double ellipse configuration. Mesh and solution obtained after applying the adaptive movement procedure to a previously enriched mesh.

are represented to a required accuracy. This treatment also accounts for the presence of shocks of different strength in which, since the numerical values of the second derivative are different, equation (5.16) will assign them different mesh spacings (e.g. larger spacings in the vicinity of weaker shocks).

The total number of elements generated in the new mesh will now depend on the values selected for C, $\delta_{max}$, and $\delta_{min}$. However, it turns out that this number is mainly determined by the choice of the constant C, which is somehow arbitrary. The criterion employed here is to select a value that produces a computationally affordable number of elements.

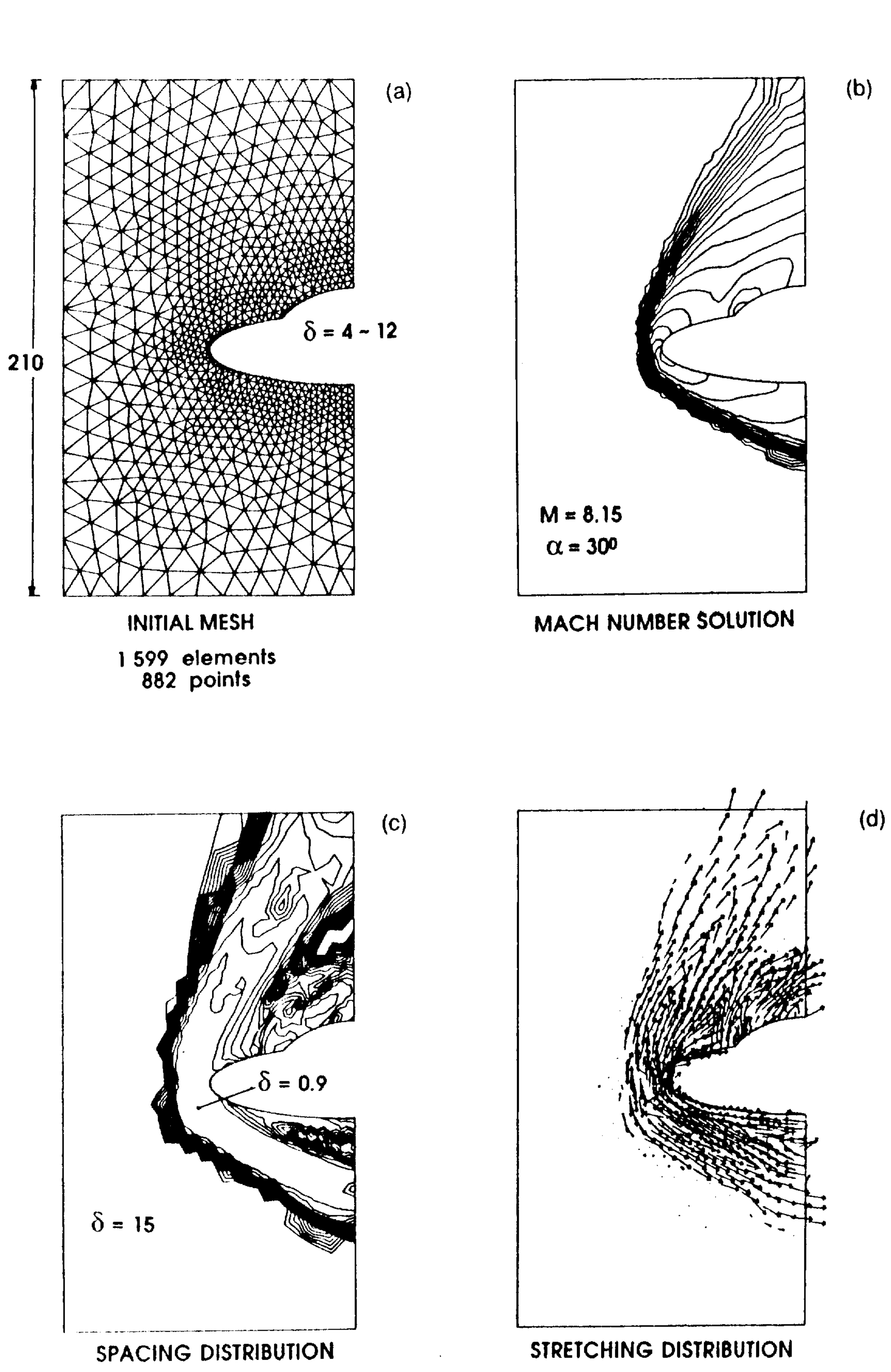
The adaptive remeshing strategy presented in this section is illustrated in figure 5.8 by showing the various stages during the adaptation process. Figure 5.8(a) shows the initial mesh employed for the computation of the supersonic flow past a double ellipse configuration. The Mach number contours of the solution obtained on the inital mesh are shown in figure 5.8(b). The flow conditions are a free stream Mach number of 8.15 and an angle of attack of $30^0$. The application of expression 5.16 to the solution obtained produces the distribution of spacing and stretching displayed in figures 5.8(c) and 5.8(d) respectively. In figure 5.8(c) the contours corresponding to the value of the minimum spacing occuring in any direction is shown, whereas in figure 5.9(d) the value of and the direction of stretching is displayed in the form of a vector field. The magnitude of the vector represents the amount of stretching i.e. ratio between maximum and minimum spacings, and the direction of the vector indicates the direction along which the spacing is maximum. In this example expression 5.17 has been applied to the computed spacings with values of $\delta_{max} = 15$ and $\delta_{min} = 0.9$. Figures 5.8(e) - 5.8(h) show various stages during the regeneration process. It can be observed how small elements are generated first as discussed in section 3.5. The completed mesh is shown in figure 5.8(i) and the solution computed on this adapted mesh is shown in figure 5.8(j). It can be observed how a very significant improvement in the solution is obtained using, in this case, a single adaptation.

Estimating the number of elements to be generated.

The regeneration process uses the current mesh as the background mesh. Such a background mesh clearly represents accurately the geometry of the computational domain. In this case, the number of elements to be generated, denoted by $N_e$, can be estimated as follows. Once the values of C, $\delta_{max}$, and $\delta_{min}$ have been selected, the spatial distribution of mesh parameters $\delta_i$, $\underline{\alpha}_i$; i=1, ..., N is computed. For each element of the background mesh, the values of the transformation **T** is computed at the centroid. The transformation is applied to the nodes of the element and its volume $V_e$ in the normalised space is computed. The number of elements $N_e$ is assumed to be proportional to the total volume in the unstretched space, i.e.

$$N_e = \chi \sum_{e=1}^{N_b} V_e \qquad (5.18)$$

where $N_b$ is the number of elements in the background mesh and $\chi$ is a constant. The value of $\chi$ is calculated as a statistical average of the values obtained for several generated meshes. The calculated value is $\chi \approx 9$. This procedure gives estimates of the value of $N_e$ with an error of less than 20%, which is accurate enough for most practical purposes. If the estimated value of $N_e$ is either too big or too small, then



INITIAL MESH

1 599 elements
882 points

MACH NUMBER SOLUTION

SPACING DISTRIBUTION

STRETCHING DISTRIBUTION

**Figure 5.8** Illustration of the adaptive remeshing procedure

(a) Initial mesh

(b) Mach number solution obtained on the initial mesh ($M_\infty$=8.15, α= $30^0$)

(c) Computed distibution of minimum spacings ($\delta_{min}$ = 9, $\delta_{max}$ = 15)

(d) Computed values and directions for the stretching

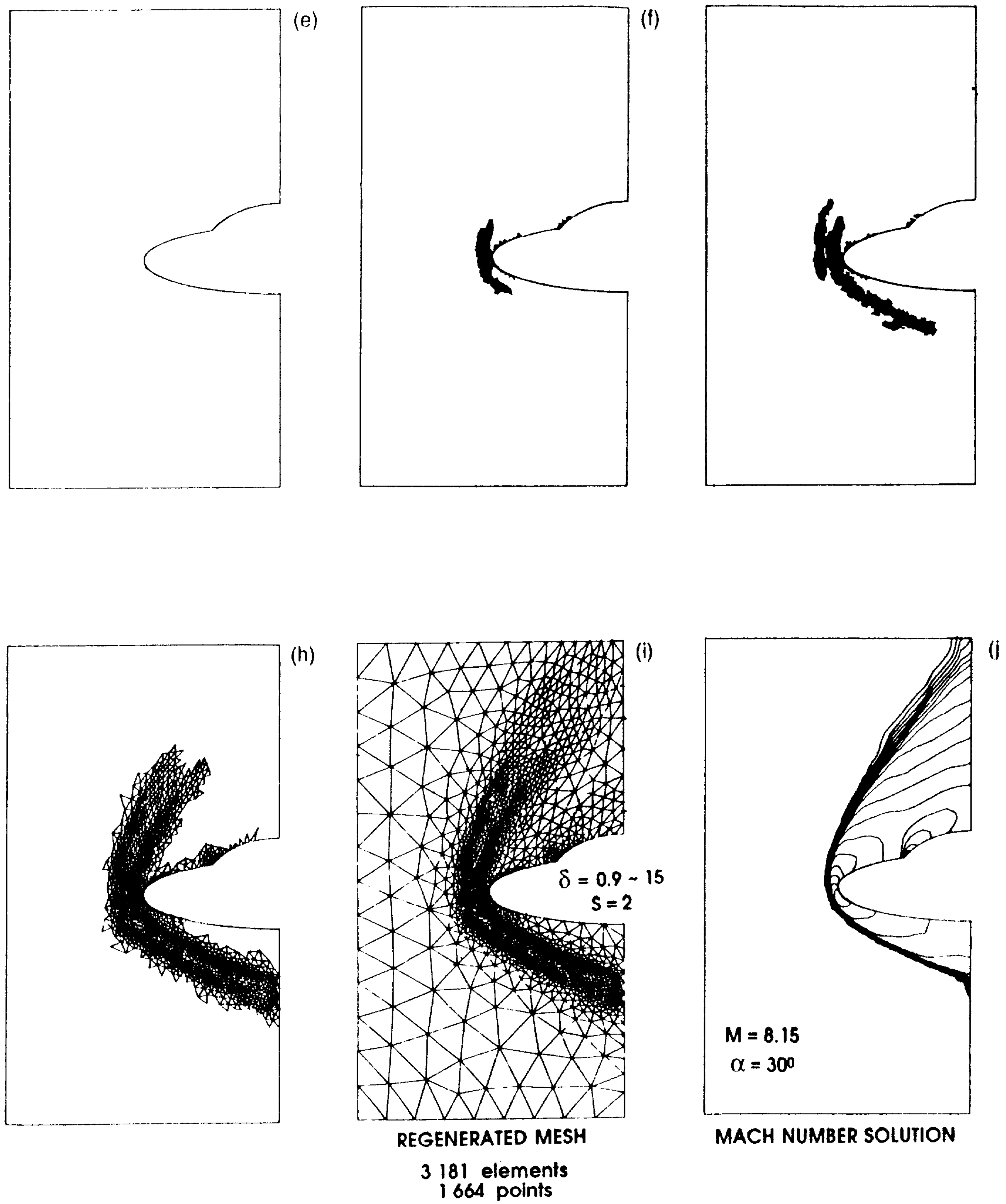


**Figure 5.8** Illustration of the adaptive remeshing procedure (continuation)
(e) - (h) different stages during the regeneration proccess
(i) Final regenerated mesh
(j) Solution obtained on adaptively regenerated mesh ($M_\infty$=8.15, α= $30^0$)

the value of C is reduced or increased and the process repeated until the value of C produces a number of elements which is regarded as being computationally acceptable.

### Application examples

Double ellipse.- The adaptive remeshing procedure is applied twice to the problem of flow past a double ellipse. The flow conditions are those previously considered for this configuration. The inital and two adapted meshes and the solutions for Mach number are shown in figure 5.9. The characteristics of the meshes employed are displayed in table 5.1.

| Mesh | Elements | Points | $\delta_{min}$ |
|---|---|---|---|
| 1 | 2027 | 1110 | 4.0 |
| 2 | 3557 | 1864 | 0.9 |
| 3 | 6403 | 3294 | 0.25 |

**Table 5.1** Double ellipse ($M_\infty$=8.15, $\alpha$=30°): mesh characteristics.

It is observed how the application of the adaptive procedure, when compared to the enrichment strategy, allows for a larger increase in the resolution at the expense of a smaller increase on total number of elements. On the other hand the remeshing procedure does not suffer from the limitations inherent in the mesh movement algorithm.

Shock interaction on a swept cylinder.- This is a problem of practical interest because its implications to the design of hypersonic vehicles [53]. The experimental apparatus and the computational domain adopted are shown diagramatically in figure 5.10(a). The numerical simulation has been carried out for a sweep angle of 15° on a cylinder of diameter D equal to 3 inches and length L equal to 9 inches. The undisturbed free stream Mach number is 8.03. The fluid which has been turned by the shock generator enters the computational domain with a Mach number of 5.26. The initial mesh and those obtained after two adaptive remeshings and the density contours distribution are shown in figure 5.10(b). The characteristics of the meshes are shown in table 5.2.

| Mesh | Elements | Points | $\delta_{min}$ | $\delta_{max}$ |
|---|---|---|---|---|
| 1 | 51 190 | 10 041 | 1.0 | 1.0 |
| 2 | 100 071 | 18 660 | 0.5 | 3.0 |
| 3 | 171 800 | 31 083 | 0.18 | 3.0 |

**Table 5.2** 3D Shock interaction on a swept cylinder: mesh characteristics.

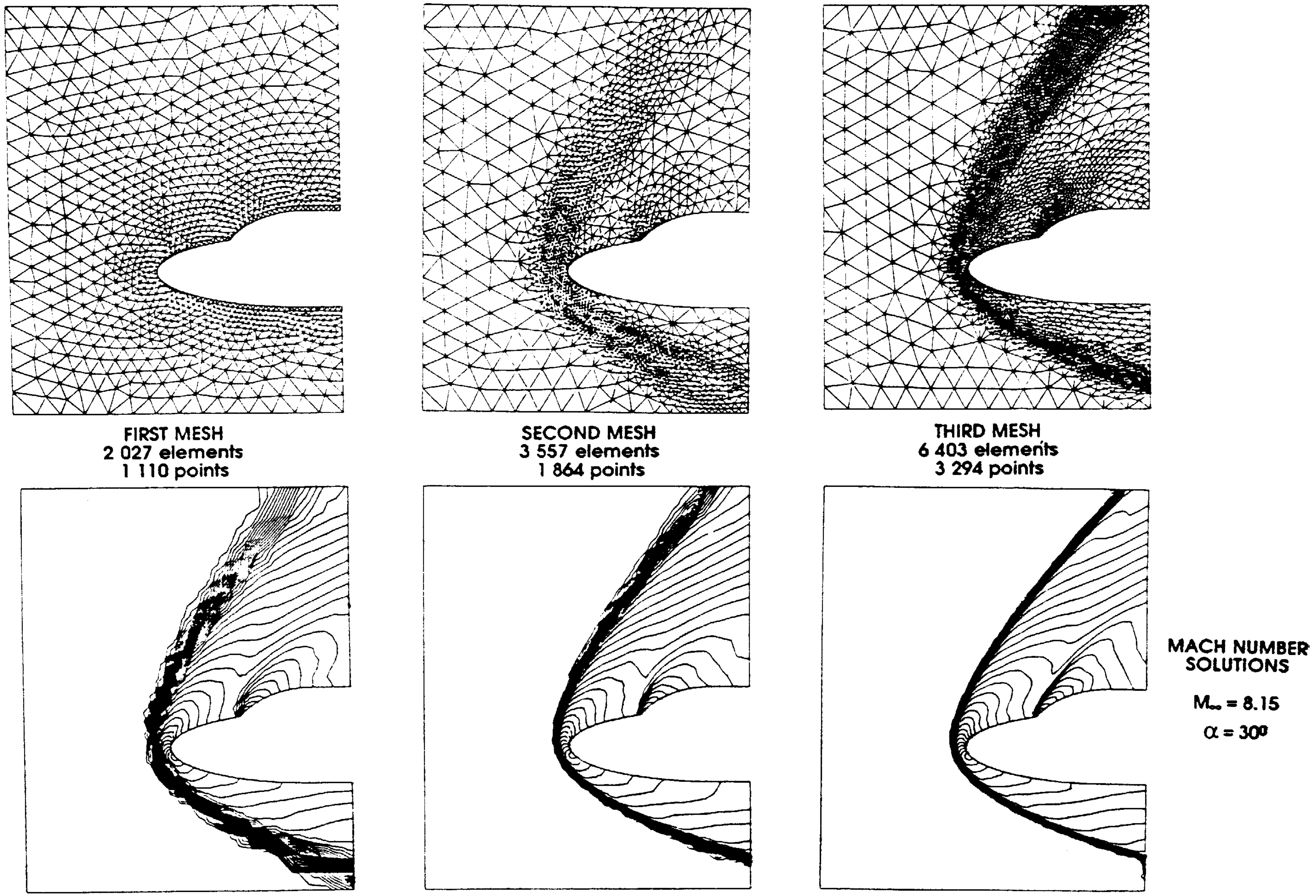


Figure 5.9 Supersonic flow past a double ellipse configuration. Sequence of meshes and solutions obtained using adaptive remeshing.

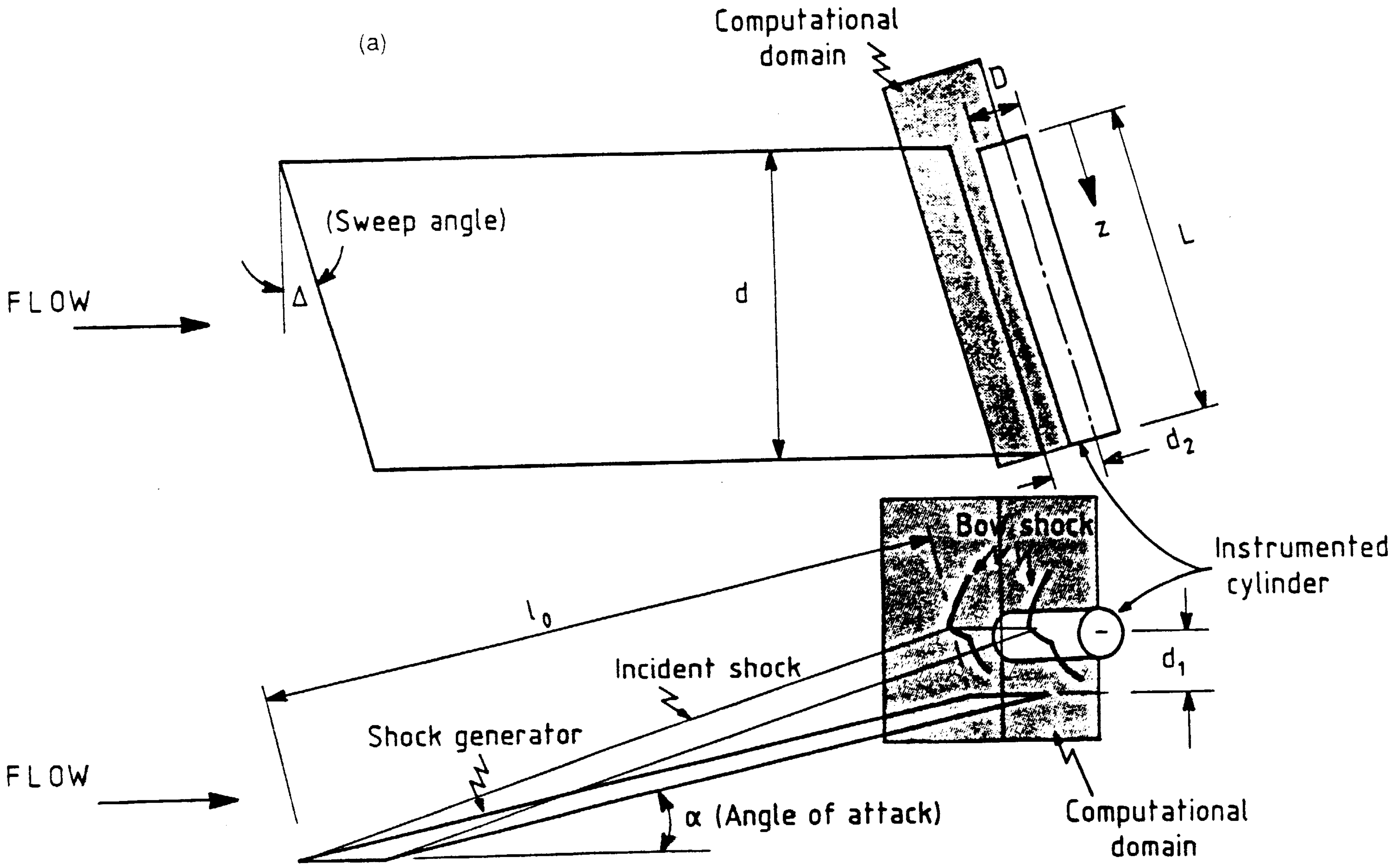


Figure 5.10 Shock interaction on a swept cylinder ($M_\infty$=8.03, Δ=15$^0$)
(a) Sketch of the experimental apparatus and computational domain

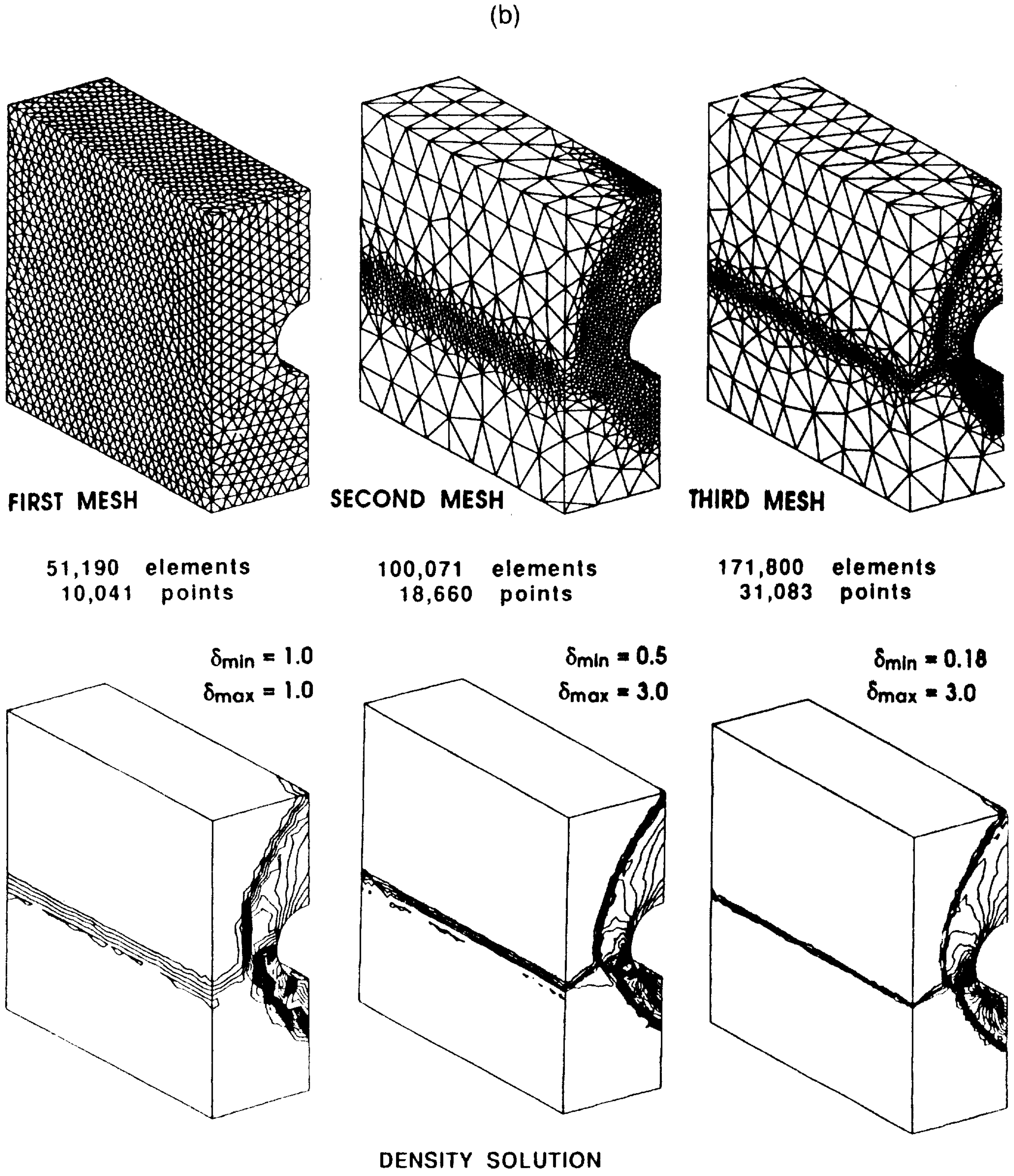
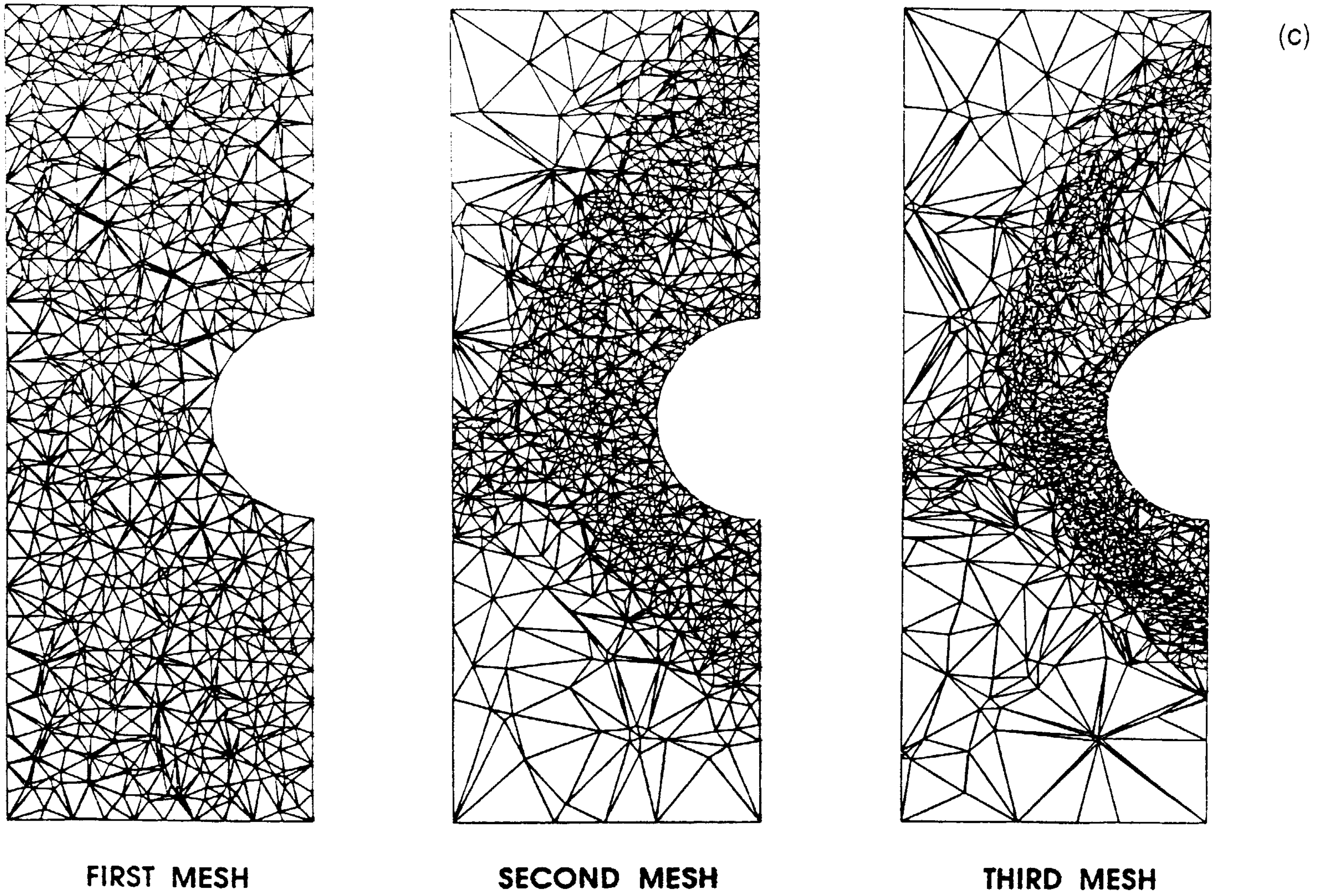


Figure 5.10 Shock interaction on a swept cylinder ($M_\infty$=8.03, $\Delta$=15$^0$) (continuation)
(b) Sequence of meshes and density solutions obtained using adaptive remeshing



FIRST MESH SECOND MESH THIRD MESH

Figure 5.10 Shock interaction on a swept cylinder ($M_\infty$=8.03, $\Delta$=15$^0$) (continuation)
(c) Cross sections though the 3D meshes half way along the cylinder

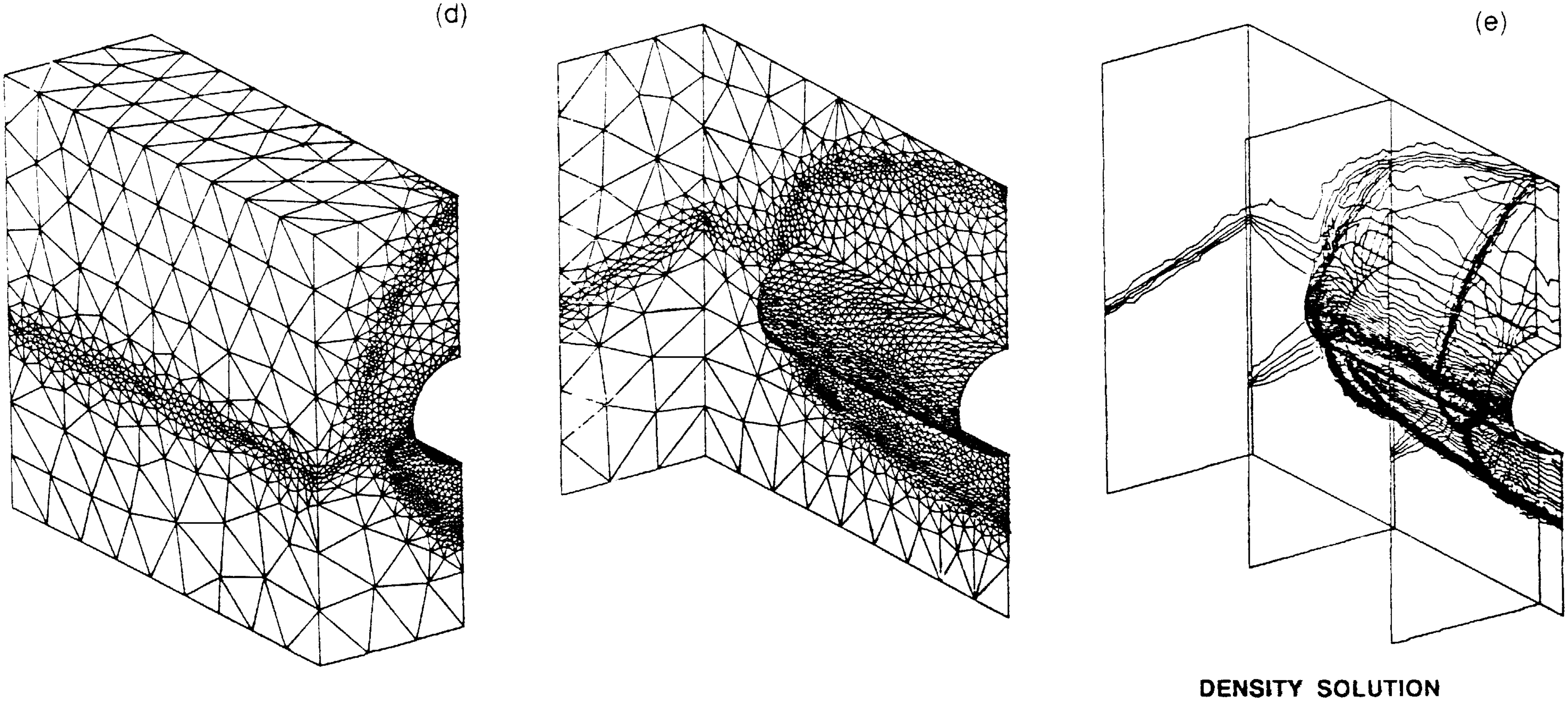


**Figure 5.10** Shock interaction on a swept cylinder ($M_\infty$=8.03, $\Delta$=15$^0$) (continuation)
(d) Third adapted mesh
(e) Detail of the shock interaction on the third adapted mesh

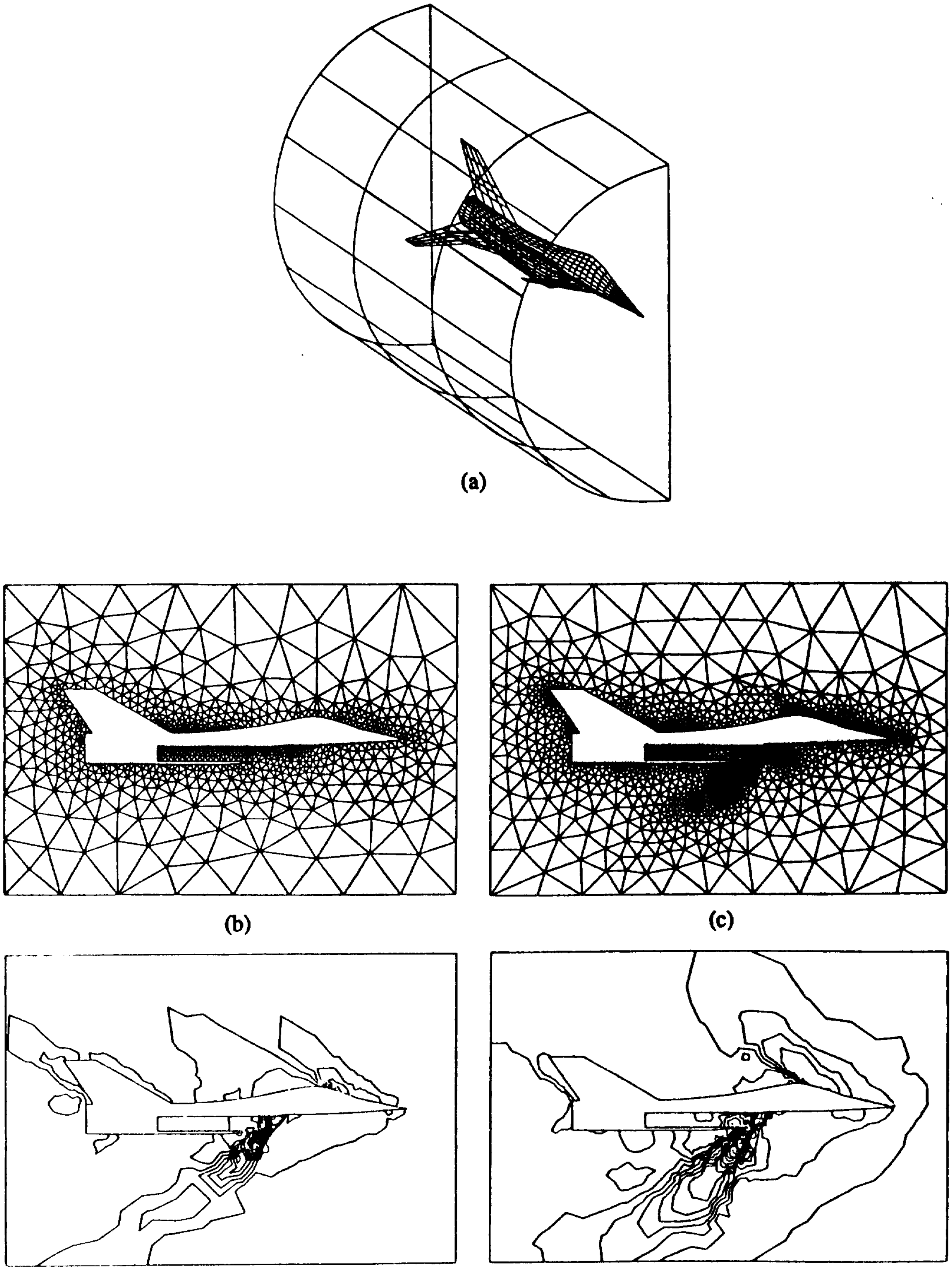
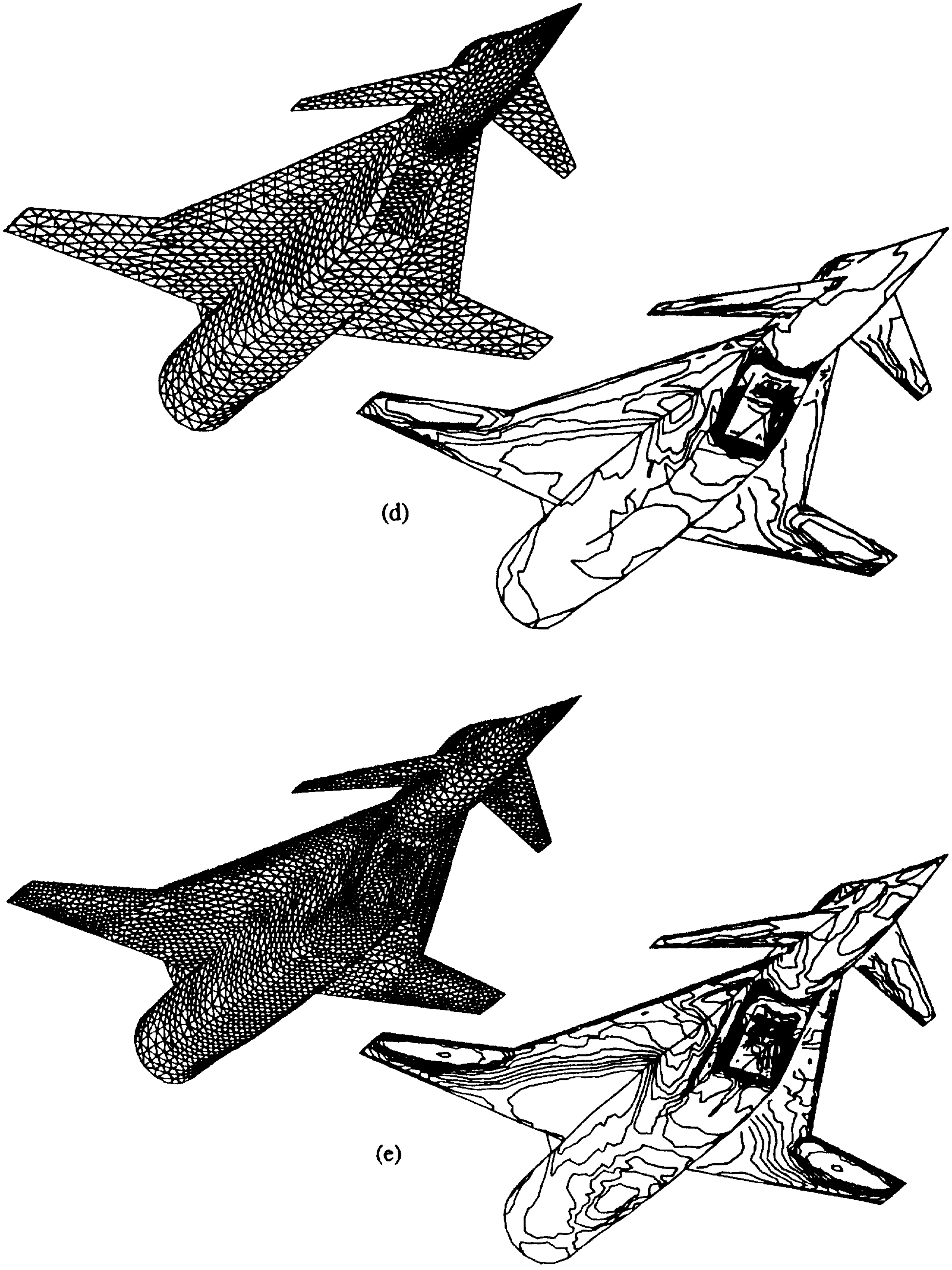


**Figure 5.11** Generic fighter configuration ($M_\infty=2$, $\alpha=3.79^0$)

(a) Geometry definition - aircraft surface and outer boundary.

(b) Initial mesh and computed pressure solution in the symmetry plane.

(c) Second mesh and computed pressure solution in the symmetry plane.



(d)

(e)

**Figure 5.11** Generic fighter configuration ($M_\infty$=2, $\alpha$=3.79$^0$)(continuation)

(d) Initial mesh and computed pressure solution.

(e) Second mesh and computed pressure solution.

The potential advantages of the adaptive remeshing procedure are clearly illustrated in this three dimensional example. The final adapted mesh has a resolution of more than five times that of the inital mesh whereas the total number of degrees of freedom increases by only a factor of 3.4. The effects of the three dimensional adaptation are best shown in figure 5.10(c) which shows the cross section through the meshes half way along the cylinder. Two views of the three dimensional mesh for the final adaptation together with the solution obtained are shown in figures 5.10(d) and 5.10(e) respectively.

Generic fighter configuration.- This example concerns the simulation of the flow past a generic fighter configuration. The generation of the initial mesh for that problem has been described in section 3.9. The flow conditions considered correspond to a free stream Mach number of 2 at an angle of attack of $3.79^0$. The engine inlet is modelled by prescribing a Mach number of 0.3 within the engine. At the outlet supersonic flow conditions are assumed. Because of the symmetry of the problem only half of the domain is modelled. The spline definition of the geometry is shown in figure 5.11(a) and consists of 23 surface components and 53 curve components. Two meshes have been employed in an initial demonstration of adaptive remeshing applied to full aircraft configuration. The inital mesh contains 76,522 tetrahedral elements with 4,128 triangular faces on the boundary. A preliminary first solution was computed using 1,500 iterations of the basic explicit scheme. A second mesh was adaptively generated using the Mach number as the key variable in the error analysis. The new mesh is formed by 70,125 tetrahedra with 7,262 triangles on the boundary. It is interesting to notice that the number of elements in the two meshes is approximately the same whereas the number of faces on the surface has increased. Moreover, the minimum spacing on the adapted mesh is 3.5 times smaller than the one on the initial mesh, thus indicating also an increase in the mesh resolution. The solution on the new mesh was obtained after 2,000 iterations. The meshes and computed solutions at the plane of symmetry are shown in figures 5.11(b) for the initial mesh and 5.11(c) for the adapted mesh. The effect of the adaptation in the vicinity of the engine inlet can be observed. The mesh and solution on the surface of the aircraft is shown in figure 5.11(d) for the initial mesh and in figure 5.11(e) for the adapted mesh. In this case the adaptation is very mild and is hardly noticeable. The main reason for this is that the resolution on the initial grid is rather poor and some important flow features are not properly captured.

## 6.TRANSIENT FLOWS

this section has been written in collaboration with

**J. Probert and O. Hassan**
Computational Dynamics Research
Innovation Centre, University College Swansea
SWANSEA SA2 8PP, UK

### *6.1 Transient flows*

Solutions of the Euler equations are smooth over large areas of the computational domain and exhibit large gradients in localised parts of the flow. In transient simulations, these localised regions will generally move through the computational domain and may sweep across very large areas e.g. the case of flows involving propagating shocks. This means that, unless adaptivity is used, a globally fine mesh will be necessary to provide the required resolution. Thus the use of adaptivity, with the possibility for local mesh refinement and coarsening, offers the potential for considerable computational savings. We have already seen that only a few mesh adaptations are generally needed to obtain a satisfactory solution to a steady problem, but we can expect that mesh adaptation will have to be performed several thousand times in a transient flow simulation. Thus any potential computational savings which appear to be offered by the use of adaptivity in this case will only be realised if the adaptation of the mesh can be performed in an efficient manner. Successful implementations of adaptivity to the solution of transient problems have already been made within the context of both structured [54] and unstructured meshes [55,56].

### *6.2 Mesh enrichment*

An obvious method of achieving mesh adaptation for transient flow simulation is the extension of the mesh enrichment ideas introduced above for the solution of steady state problems. An extremely successful implementation on unstructured triangular meshes has been made by Löhner [55]. In his method, the grid is automatically refined and de-refined as necessary according to the results of an error indicating process. An example [56] of the application of this procedure to shock-bubble interaction problem is shown in figure 6.1. This problem involves the interaction between a weak shock, travelling at a Mach number of 1.29 in air, and a bubble of heavier material (freon). From the figure it can be seen how the shock speed inside the bubble decreases, owing to the higher density of the freon, whereas the outer shock bends over. The inner shock focuses at the right hand end of the bubble, producing a significant over-pressure and intiating a small circular blast wave.

This method has also recently been applied to three dimensional flow simulations [57], but much less resolution can now achieved, as only a limited number of refinement levels can be afforded computationally.

### *6.3 Transient flows involving moving bodies*

The complexity involved in transient flow simulation increases if one considers problems in which certain boundaries of the computational domain are allowed to

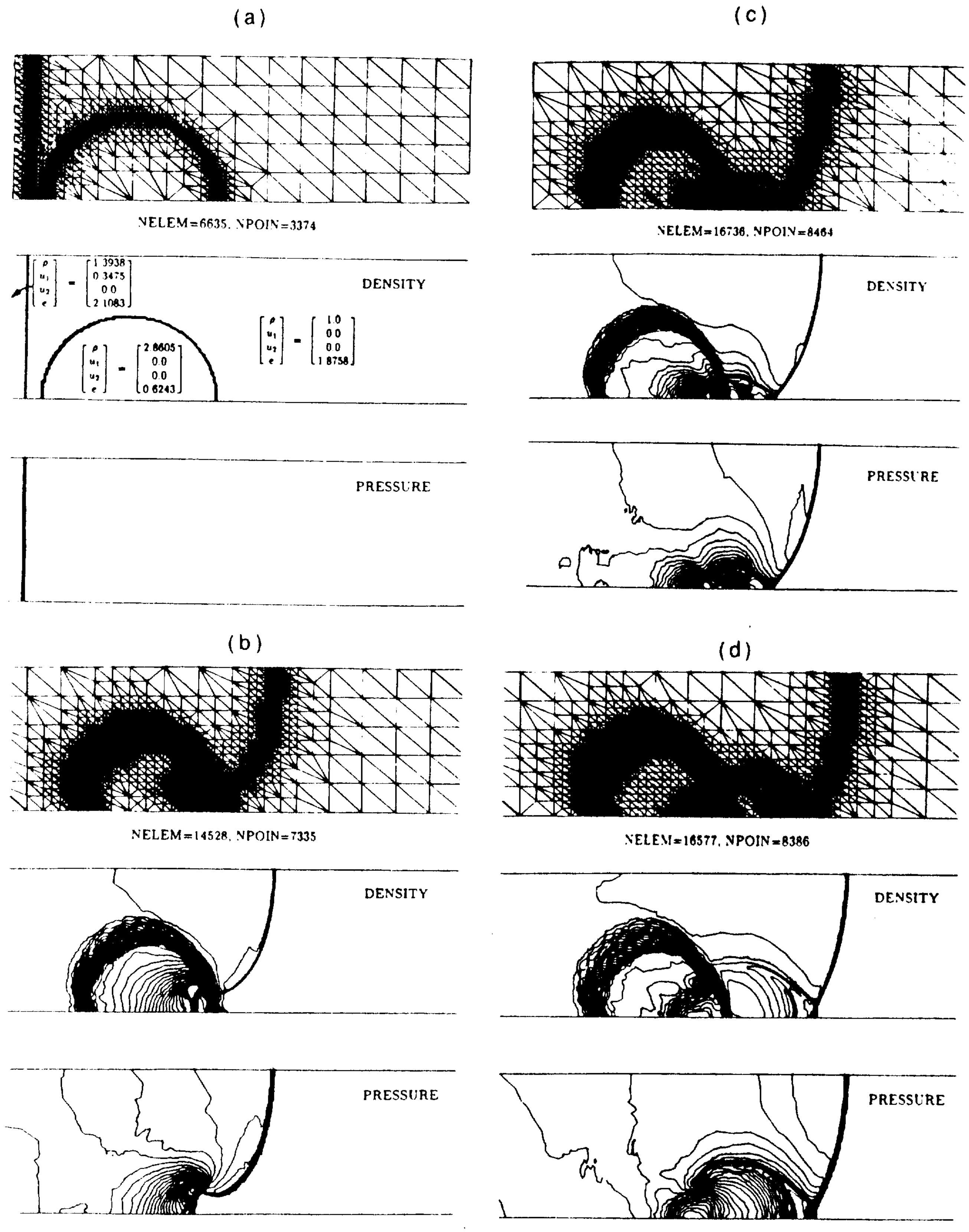


**Figure 6.1** Shock-bubble interaction problem using transient adaptation based on enrichment (from [56])

(a) Initial mesh and solution contours

(b) Mesh and solution at t=0.6

(c) Mesh and solution at t=0.7

(d) Mesh and solution at t=0.8

move, so that the geometry of the domain changes with time. This means that the mesh must be modified during the computation in order to accommodate these geometrical changes. One approach which has proved to be successful for tackling such problems is the chimera approach, in which each individual geometry component can have its own associated structured mesh which can move independently of the other meshes. Three dimensional viscous simulations involving moving bodies have already been produced by this method [58]. Unstructured meshes have been applied to the solution of inviscid two dimensional transient flows involving moving bodies [59,60], using a method which is an extension of the remeshing procedures presented in section 5.6 and this is the approach that will be presented here.

We restrict our consideration to two dimensional inviscid flows and note that the basic variational statement for the problem will need to be modified to account for the fact that the spatial domain $\Omega$ is varying with time. Suppose that we have the solution $\underline{U}_n$ at a certain time level $t_n$. We attempt to satisfy the compressible Euler equations (1.38) over the space-time domain $D = (\Omega(t),\ t_n \leq t \leq t_{n+1})$. To express this problem in a variational form we need to introduce suitable trial and weighting function sets. We assume, for the purposes of this discussion, that the conditions on the boundary $\Gamma$ of $\Omega$ can be expressed in the form

$$\underline{U} = \underline{0} \qquad\qquad \text{on } \Gamma(t) \tag{6.1}$$

Although such conditions are somewhat unrealistic, the actual boundary conditions which would need to be applied in the simulation of a given problem can be readily incorporated by making appropriate modifications to the following analysis. We may define

$$\begin{aligned} \mathcal{T} &= \{\, \underline{U} \mid \underline{U} = \underline{0} \text{ on } \Gamma;\ \underline{U} = \underline{U}_n \text{ on } \Omega \text{ at } t = t_n \} \\ \mathcal{W} &= \{\, W \mid W = 0 \text{ on } \Gamma \} \end{aligned} \tag{6.2}$$

and a variational formulation of the problem can be stated as : find $\underline{U}$ in $\mathcal{T}$ such that

$$\int_{t_n}^{t_{n+1}} \int_{\Omega} W \left[ \frac{\partial \underline{U}}{\partial t} + \frac{\partial \underline{E}}{\partial x} + \frac{\partial \underline{F}}{\partial y} \right] d\Omega \, dt = \underline{0} \tag{6.3}$$

for every W in $\mathcal{W}$. We will assume that the spatial domain $\Omega$ has been discretised using 3 noded linear triangular elements, with interior nodes numbered from 1 to p and introduce the sets

$$\begin{aligned} \mathcal{T}_{(p)} &= \{\, \underline{U}_{(p)} \mid \underline{U}_{(p)} = M_1 \underline{U}_1 + M_2 \underline{U}_2 + \ldots\ldots + M_p \underline{U}_p;\ \underline{U}_{(p)} = \underline{0} \text{ on } \Gamma;\ \underline{U}_{(p)} = \underline{U}_n \text{ on } \Omega \text{ at } t = t_n \} \\ \mathcal{W}_{(p)} &= \{ W_{(p)} \mid W_{(p)} = a_1 M_1 + a_2 M_2 + \ldots + a_p M_p\ ;\ W_{(p)} = 0 \text{ on } \Gamma \} \end{aligned} \tag{6.4}$$

In the approach which is to be followed here, certain nodes in the mesh will be fixed, while others will move with a prescribed velocity. The shape functions $M_J$ are linear functions of space and time which satisfy

$$M_J(\underline{x},t_n) = N_J^n(\underline{x}) \qquad\qquad M_J(\underline{x},t_{n+1}) = N_J^{n+1}(\underline{x}) \qquad (6.5)$$

where $N_J^n$ is the standard finite element shape function, defined in section 1.2, associated to node J at time $t_n$. Working with the function sets defined in equation (6.4), the Galerkin approximation statement takes the form : find $\underline{U}_{(p)}$ in $\mathcal{T}_{(p)}$ such that

$$\int_{t_n}^{t_{n+1}} \int_{\Omega} M_J \left[ \frac{\partial \underline{U}_{(p)}}{\partial t} + \frac{\partial \underline{E}_{(p)}}{\partial x} + \frac{\partial \underline{F}_{(p)}}{\partial y} \right] d\Omega \, dt = \underline{0} \qquad (6.6)$$

for J = 1,2,....,p. Considering the first term appearing in this integral, it is possible to show that

$$\int_{\Omega} M_J \frac{\partial \underline{U}_{(p)}}{\partial t} d\Omega = \frac{d}{dt} \int_{\Omega} M_J \underline{U}_{(p)} \, d\Omega - \int_{\Gamma} \underline{v}.\underline{n} \, M_J \underline{U}_{(p)} \, d\Gamma - \int_{\Omega} \frac{\partial M_J}{\partial t} \underline{U}_{(p)} \, d\Omega \qquad (6.7)$$

where $\underline{v}$ denotes the velocity of the moving nodes. With $\underline{v}_{(p)} = (v_x, v_y)$ interpolated linearly between the nodal values of $\underline{v}$, an observer moving with the mesh will not detect any change in the shape functions i.e.

$$\frac{DM_J}{Dt} = \frac{\partial M_J}{\partial t} + v_x \frac{\partial M_J}{\partial x} + v_y \frac{\partial M_J}{\partial y} = 0 \qquad (6.8)$$

where D/Dt denotes differentiation following the moving mesh and so

$$\int_{\Gamma} \underline{v}.\underline{n} \, M_J \underline{U}_{(p)} \, d\Gamma + \int_{\Omega} \frac{\partial M_J}{\partial t} \underline{U}_{(p)} \, d\Omega = \int_{\Omega} M_J \left[ \frac{\partial v_x \underline{U}_{(p)}}{\partial x} + \frac{\partial v_y \underline{U}_{(p)}}{\partial y} \right] d\Omega \qquad (6.9)$$

Finally, combining equations (6.6), (6.7) and (6.9), the Galerkin approximation satisfies

$$\int_{\Omega_{n+1}} M_J \underline{U}_{(p)} \, d\Omega - \int_{\Omega_n} M_J \underline{U}_{(p)} \, d\Omega = - \int_{t_n}^{t_{n+1}} \int_{\Omega} M_J \left[ \frac{\partial \underline{E}_{(p)}^{+}}{\partial x} + \frac{\partial \underline{F}_{(p)}^{+}}{\partial y} \right] d\Omega \; dt \qquad (6.10)$$

where

$$\underline{E}_{(p)}{}^{+} = \underline{E}_{(p)} - v_x \, \underline{U}_{(p)} \qquad\qquad \underline{E}_{(p)}{}^{+} = \underline{E}_{(p)} - v_y \, \underline{U}_{(p)} \qquad (6.11)$$

Inserting the assumed form for $\underline{U}_{(p)}$ from equation (6.4),

$$[M\ \underline{U}]_J{}^{n+1} - [M\ \underline{U}]_J{}^{n} = -\int_{t_n}^{t_{n+1}} \int_{\Omega} M_J \left[ \frac{\partial \underline{E}_{(p)}{}^{+}}{\partial x} + \frac{\partial \underline{E}_{(p)}{}^{+}}{\partial y} \right] d\Omega \ dt \qquad (6.12)$$

The integral appearing here can be evaluated by first employing one point integration in time (at $t = t_{n+1/2}$) and then using a two-step approximation [18,19]. Artificial viscosity will again be needed with a scheme of this type and the resolution of the resulting scheme may be improved by the use of the FCT ideas mentioned in Section 1.3.

### *6.4 Adaptive remeshing for transient flows involving moving bodies*

The method described above, whereby a grid may be adapted by remeshing, is a natural approach to follow for the simulation of flows involving moving bodies. It will be assumed that the motion of any moving boundary is prescribed and the objective is to determine the resulting flow field. The description of an algorithm which can be devised to advance the solution of equation (6.11) in time can be written as follows:

1. Generate an initial mesh to represent the computational domain and to adequately resolve the initial solution.

2. Start the timestep loop

2.1 Advance the solution one timestep
2.2 Update the coordinates of the points on the moving boundaries
2.3 Use an error indicator to examine the current solution and define an 'optimal' distribution of mesh spacing and stretching
2.4 Compare the current mesh with the 'optimal' mesh. Delete the elements whose size and shape is too different from the optimal
2.5 Triangulate the regions where elements have been deleted according to the new distribution of mesh parameters
2.6 Determine, by interpolation, the flow variables at the new nodes

End the timestep loop

It is apparent that the crucial phase in this process is the mesh adaptation in steps 2.3-2.5. The mechanics of this process is illustrated diagramatically in figure 6.2. The success of the procedure depends upon the reliability of the error indicator which is employed. The indicator of equation (5.9) has again been used for this application.

Application examples

1D Shock propagation.- The first example considered consists of a two dimensional simulation of the transient development of the flow in shock tube. The purpose of this

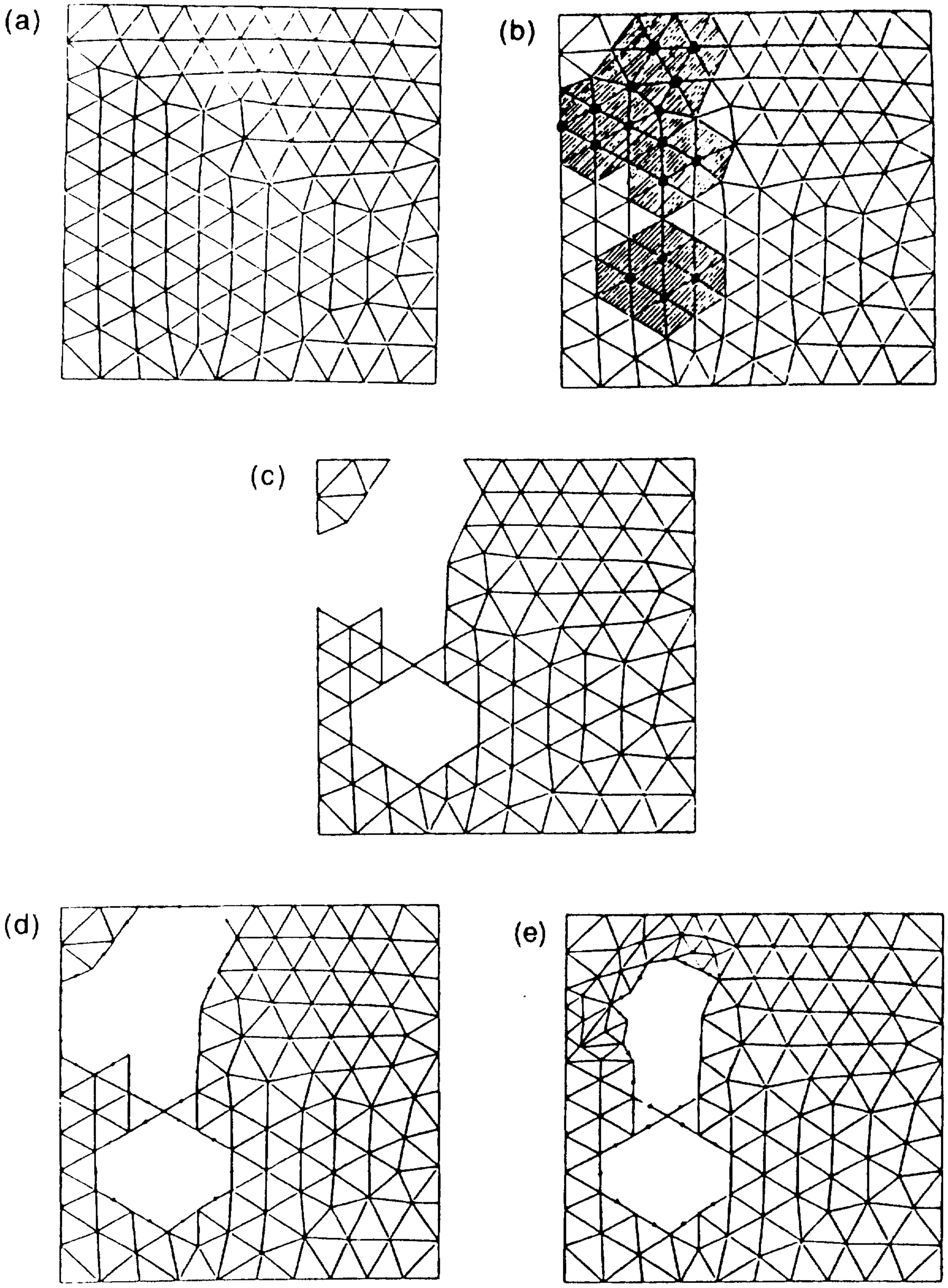


**Figure 6.2** Illustration of the adaptive remeshing procedure applied to transient problems

(a) Initial mesh

(b) Marked unwanted nodes and elements

(c) Elements are removed from the mesh

(d) Boundary sides are generated according to the new spacing distribution to form a closed loop arround each hole

(e) Triangulation of the holes using the advancing front and the new distribution of spacings

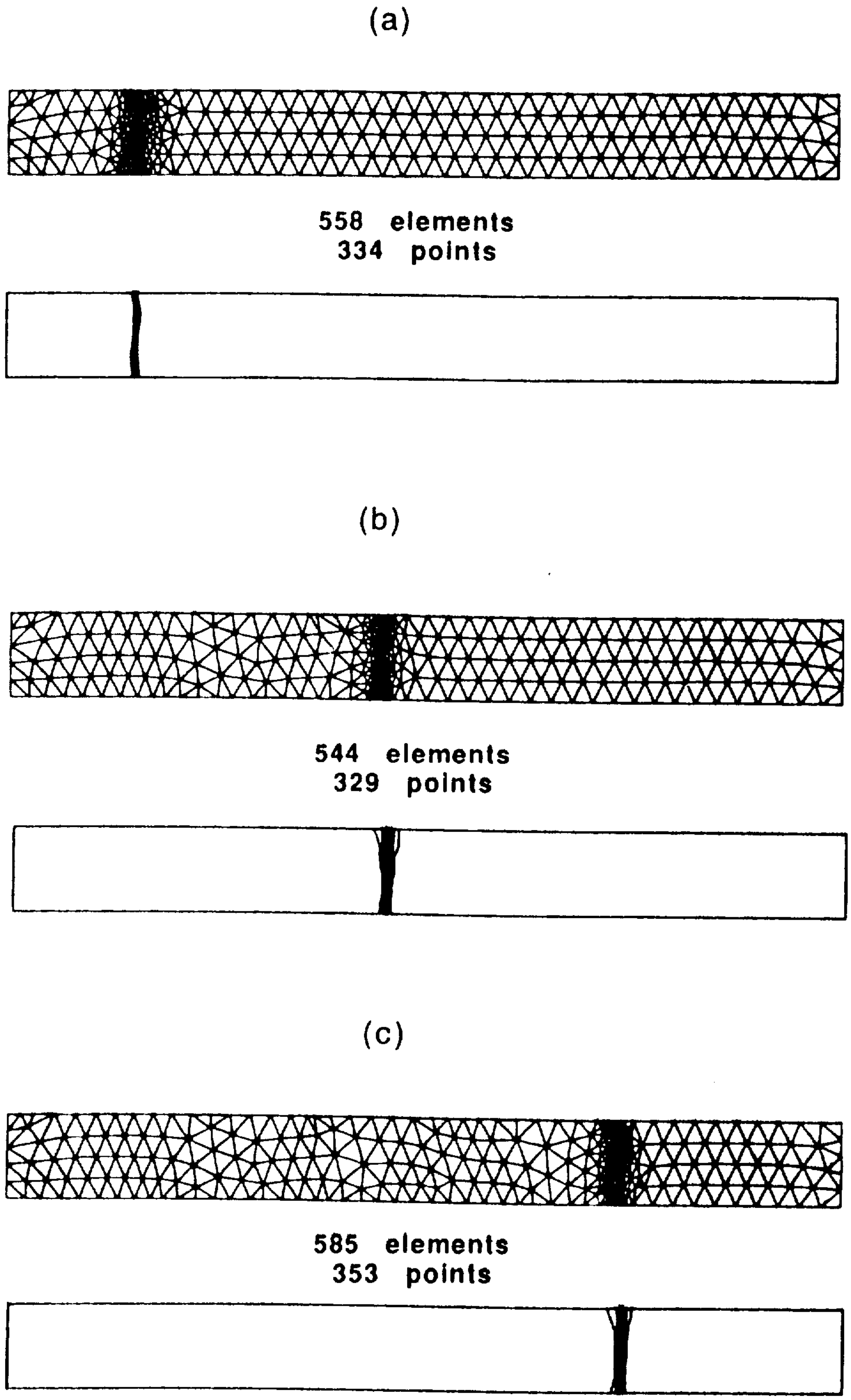


**Figure 6.3** Shock tube example using adaptively refined meshes.
(a) Adapted mesh and density contours at t=0
(b) Adapted mesh and density contours at t=4
(c) Adapted mesh and density contours at t=8

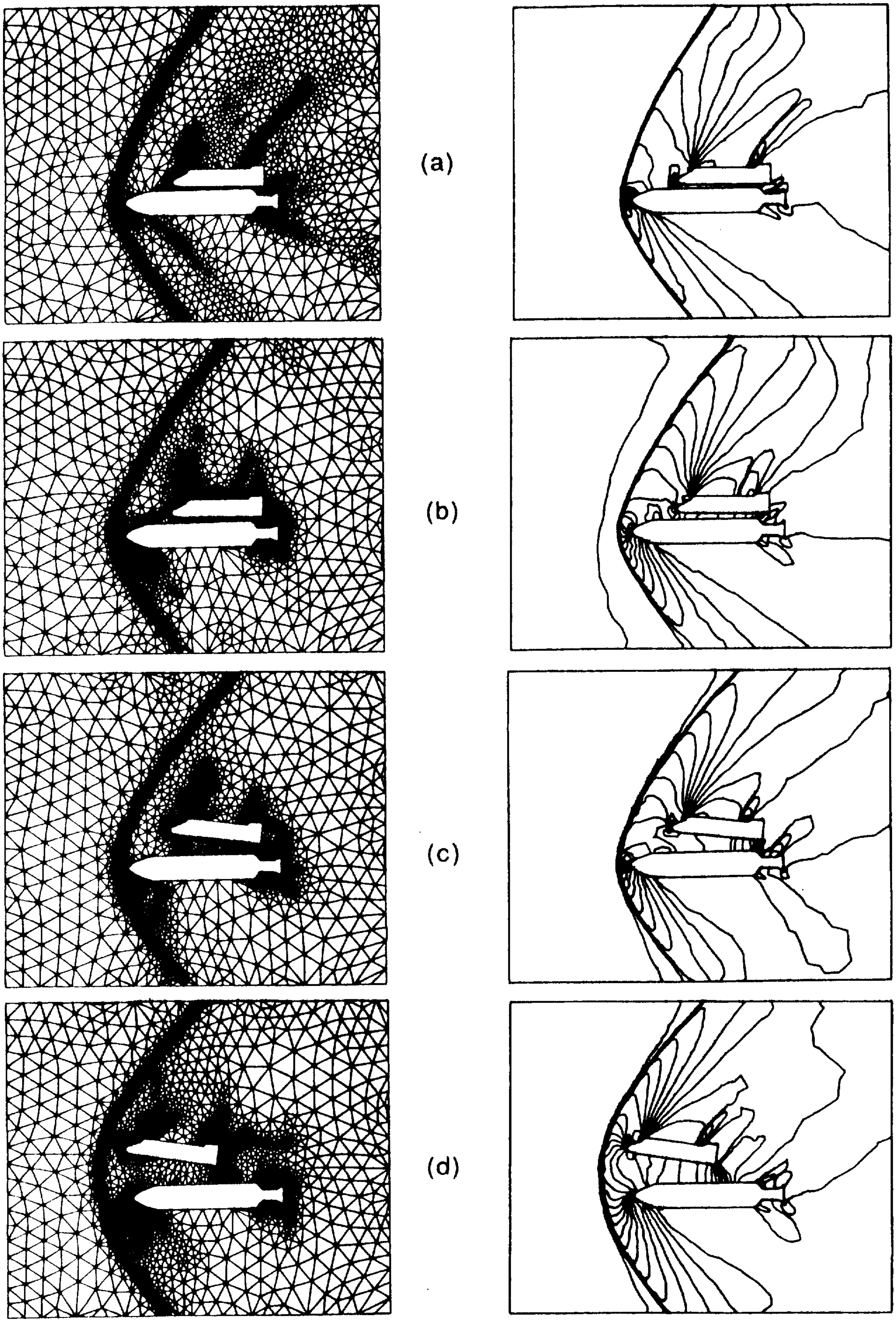


**Figure 6.4** Space shuttle vehicle and booster rocket simulation ($M_\infty=2$, $\alpha=-4^0$). The shuttle moves upwards and away from the rocket with a prescribed motion. Sequence of meshes and pressure solutions obtained during the transient simulation starting from a steady state solution. The meshes consist of: (a) 7,870 elements and 4,130 points (b) 7,377 elements and 3,867 points (c) 6,847 elements and 3,580 points and (d) 8,459 elements and 4,379

# ACKNOWLEDGEMENTS

The authors would like to acknowledge Drs. Javier Bonet, Luca Formaggia, Oubay Hassan, Jane Probert and Professor O. C. Zienkiewicz whose collaboration was essential for the development of the work presented here. Our work in this area has benefited greatly from the support over several years of the Aerothermal Loads Branch, NASA Langley Research Center, under grants NAGW-478 and NAGW-1809 and we are pleased to acknowledge this contribution. We also acknowledge the assistance of the U. K. Science and Engineering Research Council for providing time on the Cray computer at the Rutherford Appleton Laboratory; Avions Marcel Dassault for partially funding this work and providing the geometry definitions of the Hermes and the Falcon aeroplane; NASA Ames for the support provided under grant NCCW-008 and Cray UK for providing computer time which allowed us to perform the computation of the flow past the Falcon plane. The discussions and comments of Dr. N. P. Weatherill and the assistance of S. Soltani in providing the adaptive unstructured mesh of figure 1.1 are also acknowledged.

## REFERENCES

[1] N. P. Weatherill, 'Mesh generation in computational fluid dynamics', *von Karman Institute for Fluid Dynamics*, Lecture Series 1989-04, Brussels, 1989.

[2] J. F. Thompson, Z. U. A. Warsi and C. W. Mastin, 'Numerical grid generation - foundations and applications', *North-Holland*, 1985.

[3] S. Allwright, 'Multiblock topology specification and grid generation for complete aircraft configurations', in *Applications of Mesh Generation to Complex 3-D Configurations*, AGARD Conference Proceedings No. 464, 11.1-11.11, 1990.

[4] T. J. Baker, 'Unstructured mesh generation by a generalized Delaunay algorithm', in *Applications of Mesh Generation to Complex 3-D Configurations*, AGARD Conference Proceedings No. 464, 20.1-20.10, 1990.

[5] J. Peraire, M. Vahdati, K. Morgan and O. C. Zienkiewicz, 'Adaptive remeshing for compressible flow computations', *J. Comp. Phys. 72*, 449-466, 1987.

[6] A. Jameson, T. J. Baker and N. P. Weatherill, 'Calculation of inviscid transonic flow over a complete aircraft', *AIAA Paper 86-0102*, 1986.

[7] J. Peraire, J. Peiro, L. Formaggia, K. Morgan and O. C. Zienkiewicz, 'Finite element Euler computations in three dimensions', *Int. J. Num. Meth. in Engn.*, 26, 1988.

[8] R. Löhner, K. Morgan and O. C. Zienkiewicz, 'Adaptive grid refinement for the compressible Euler equations', in *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, Edited by I. Babuska et al, Wiley, 281-297, 1986.

[9] L. Formaggia, J. Peraire, K. Morgan and J. Peiro, 'Implementation of a 3D explicit Euler solver on a CRAY computer', in *Proc. 4th Int. Symposium on Science and Engineering on CRAY Supercomputers*, Minneapolis, 45-65, 1988.

[10] C. Hirsch, 'Numerical computation of internal and external flows, Volume 2', *Wiley*, 1990.

[11] C. Johnson, 'Numerical solutions of partial differential equations by the finite element method', *Cambridge University Press*, 1987.

[12] O. C. Zienkiewicz and K. Morgan, 'Finite Elements and Approximation', *Wiley*, 1983.

[13] J. Donéa, 'A Taylor-Galerkin method for convective transport problems', *Int. J. Num. Meth. Engng. 20*, 101-119, 1984.

[14] O. Hassan, K. Morgan and J. Peraire, 'An adaptive implicit/explicit finite element scheme for compressible viscous high speed flows', *AIAA Paper 89-0363*, 1989.

[15] K. Morgan and J. Peraire, 'Finite element methods for compressible flows', *von Karman Institute for Fluid Dynamics*, Lecture Series *1987-04*, Brussels 1987.

[16] J. Donéa and S. Giuliani, 'A simple method to generate high-order accurate convection operators for explicit schemes based on linear finite elements', *Int. J. Num. Meth. Fluids 1*, 63-79, 1981.

[17] K. Morgan and J. Peraire, 'An introduction to finite element methods for computational fluid dynamics', *Institute for Numerical Methods in Engineering Report C/R/608/88*, University College of Swansea 1988.

[18] J. Peraire, K. Morgan and J. Peiro, 'Unstructured finite element mesh generation and adaptive procedures for CFD', in *Applications of Mesh Generation to Complex 3-D Configurations*, AGARD Conference Proceedings No. 464, 18.1-18.12, 1990.

[19] R. Löhner, K. Morgan, J. Peraire and o. C. Zienkiewicz, 'Finite element methods for high speed flows', *AIAA Paper 85-1531-CP*, 1985.

[20] J. P. Boris and D. L. Book, 'Flux corrected transport I : SHASTA, a fluid transport algorithm that works', *J. Comp. Phys. 11*, 8-69, 1973.

[21] S. T. Zalesak, 'Fully multidimensional flux corrected transport algorithm for fluids', *J. Comp. Phys. 31*, 335-362, 1979.

[22] A. K. Parrott and M. K. Christie, 'FCT applied to the 2D finite element solution of tracer transport by single phase flow in a porous medium', in *Numerical Methods for Fluid Dynamics*, Edited by K. W. Morton and M. J. Baines, Oxford University Press, 27-53, 1986.

[23] R. Löhner, K. Morgan, J. Peraire and M. Vahdati, 'Finite element flux corrected transport (FEM-FCT) for the Euler and Navier-Stokes equations', in *Finite Elements in Fluids, Volume 7*, Edited by R. H. Gallagher et al, 105-121, 1988.

[24] K. Morgan, J. Peraire and R. Löhner, 'Adaptive finite element flux corrected transport techniques for CFD', in *Finite Elements - Theory and Application*, Edited by D. L. Dwoyer et al., Springer-Verlag, 165-175, 1988.

[25] A. Lerat and J. Sides, 'Efficient solution of the steady euler equations with a centered implicit scheme', in *Numerical Methods for Fluid Dynamics III*, K.W Morton and M. J. Baines ed., 65-86, Clarendon Press, Oxford, 1988.

[26] O. Hassan, K. Morgan and J. Peraire, 'An implicit finite element method for high speed flows', *AIAA Paper-90-0402*, 1990

[27] A. A. G. Requicha and H. B. Voelcher,' Solid modelling: A Historical Summary and contemporary assessment', *IEEE Computer Graphics and Applications, 3*, n. 2, 9-24, 1982.

[28] I. D. Faux and M. J. Pratt, Computational Geometry for Design and Manufacture, *Ellis Horwood*, Chichester, 1981.

[29] J. C. Ferguson, 'Multivariate curve interpolation', *J. ACM 11*, 221 ,1964.

[30] S. A. Coons, 'Surfaces for Computed Aided Design of Space Forms', *Report MAC-TR-41, Project MAC*, M.I.T., 1967.

[31] J. Peiro,'A finite element procedure for the solution of the Euler equations using unstructured meshes', *University of Wales Ph.D. Thesis*, C/Ph/126/89, 1989.

[32] A. J. George, 'Computer implementation of the finite element method', *Ph. D. Thesis, Stanford University*, STAN-CS-71-208, 1971.

[33] S. H. Lo, 'A new mesh generation scheme for arbitrary planar domains', *Int. J. Numer. Methods Engng.* 21, 1403-1426 (1985).

[34] J. C. Cavendish, D. A. Field and W. H. Frey, 'An approach to automatic three dimensional finite element mesh generation', *Int. J. Num. Meth. Engng.* 21, 329-348,1985.

[35] J. J. Stoker, Differential geometry, *Wiley Interscience*, New York, 1969.

[36] L.-E. Eriksson, R. E. Smith, M. R. Wiese and N. Farr, ' Grid generation and inviscid flow computation about cranked-winged airplane geometries', *AIAA Paper 87-1125*,1987.

[37] J.L. Bentley and J.H. Friedman, 'Data structures for range searching', *Computing Surveys, 11*, 4, 1979.

[38] M.I. Shamos and D. Hoey, 'Geometric intersection problems', in *17th Annual Symposium on Foundations of Computer Science*, IEEE, 1976.

[39] 'Fundamental algorithms for computer graphics', edited by R. A. Earnshaw, *NATO ASI Series F, vol. 17*, Springer Verlag 1985.

[40] J.Boris, 'A vectorised algorithm for determining the nearest neighbours', *J. Comp. Phys., 66*, 1-20, 1986.

[41] R. Sedgewick, 'Algorithms', *Addison Wesley*, Reading, 1988.

[42] J. Bonet and J. Peraire, "An alternating digital tree (ADT) algorithm for geometric searching and intersection problems", *Int. J. Num. Meth. Engng.*, in press (1990).

[43] D. Knuth, 'The art of computer programming - Sorting and searching', vol. 3, *Addison Wesley*, 1973.

[44] D. Knuth, 'The art of computer programming - Fundamental algorithms', vol. 1, *Addison Wesley*, 1969.

[45] J.L. Bentley, ' Multidimensional binary search trees used for associative searching', *Communications of the ACM, 18*, 1, 1975.

[46] J. Bonet, 'Finite element analysis of thin sheet superplastic forming processes', *University of Wales Ph.D. Thesis*, C/PhD/128/89.

[47] P. G. Ciarlet and P. A. Raviart, 'General Lagrange and Hermite interpolation in $R^n$ with applications in finite element methods', *Arch. Rat. Mech. Anal., 46*, 177-199, 1972.

[48] J. T. Oden, 'Grid optimisation and adaptive meshes for finite element methods, *University of Texas at Austin Notes*, 1983.

[49] B. Palmerio, V. Billey, A. Dervieux and J. Periaux, 'Self-adaptive mesh refinements and finite element methods for solving the Euler equations', in *Numerical Methods for Fluid Dynamics II*, K.W Morton and M. J. Baines ed., 369-388, Clarendon Press, Oxford, 1985.

[50] J. Peraire, K. Morgan, J. Peiro and O. C. Zienkiewicz, 'An adaptive ainite element method for high speed flows', *AIAA Paper 87-0558*, 1987.

[51] B. Palmerio and A. Dervieux, '2D and 3D unstructured mesh adaption relying on physical analogy', in Proc. of the *Second International Conference on Numerical Grid Generation in Computational Fluid Mechanics*, Miami Beach, Florida, 1988.

[52] K. Nakahashi and G.S. Deiwert, 'A practical adaptive-grid method for complex fluid flow problems', *Lecture Notes in Physics, vol. 218*, 422-426, Springer Verlag, 1985.

[53] C. Glass, A. R. Wieting and M. Holden, 'Effect of leading edge sweep on shock-shock interference heating at Mach 8', *AIAA Paper 89-0271*, 1989

[54] P. Woodward and P. Colella, 'The numerical simulation of two dimensional fluid flow with strong shocks', *J. Comp. Phys. 54*, 115-173 , 1984.

[55] R. Löhner, 'An adaptive finite element scheme for transient problems in CFD' *Comp. Meth. in Appl. Mech and Engn. 61*, 323-338, 1987.

[56] R. Löhner, K. Morgan, J. Peraire and M. Vahdati, 'Finite element flux corrected transport for the Euler and Navier-Stokes equations', *Int. J. Num. Meth. in Fluids 7*, 1093-1109, 1987.

[57] R. Löhner and J. D. Baum, 'Numerical simulation of shock interaction with complex geometry three-dimensional structures using a new adaptive h-refinement scheme on unstructured grids', *AIAA Paper 90-0700*, 1990.

[58] P. G. Bunning, I. T. Chiu, S. Obayashi, Y. M. Rizk and J. L. Steger, 'Numerical simulation of the integrated space shuttle vehicle in ascent', *AIAA Paper 88-4359-CP*, 1988.

[59] L. Formaggia, J. Peraire and K. Morgan, 'Simulation of a store separation using the finite element method', *Appl. Math. Modelling* 12, 175-181, 1988.

[60] E. J. Probert, 'Finite element methods for transients compressible flows', *University of Wales Ph.D. Thesis*, C/Ph/123/89, 1989.