1986 Proceedings
FALL JOINT COMPUTER CONFERENCE

November 2-6, 1986—INFOMART®—Dallas, Texas
Sponsored by **ACM and Computer Society of the IEEE**

Harold S. Stone, Proceedings Editor and Program Chairman
Stanley Winkler, Conference Chairman

Computer Society Order Number 743
Library of Congress Number 86-81582
IEEE Catalog Number 86CH2345-7
ACM Order Number 401860
ISBN 0-8186-0743-2

THE COMPUTER SOCIETY OF THE IEEE
Association for Computing Machinery
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC
COMPUTER SOCIETY PRESS

A COMPARISON OF GRID GENERATION TECHNIQUES

S. R. KENNON* - G. S. DULIKRAVICH**

* Graduate Research Assistant, Dept. of Aerospace Engineering and Engineering Mechanics, University of Texas at Austin, Austin, TX.
** Associate Professor, Dept. of Aerospace Engineering, Pennsylvania State University, University Park, PA.

Abstract

A survey of techniques for the numerical generation of computational grids is presented. These methods are shown to be a powerful addition to the numerical analyst's tools for simulation of complex physical processes. Computational grids are classified into their various forms and methods for generating each type of grid are discussed. In addition to the mathematical and numerical details of each method, examples of the application of each method are shown. A short bibliography of these methods is also given to assist the interested reader in finding more information about this important part of computational mechanics.

Introduction

This paper presents a survey of methods for the numerical generation of computational grids used in the simulation of physical problems governed by partial differential equations. Computational grids are required to represent a given region of space as a discrete collection of sub-regions. Within these sub-regions, the solution of the governing partial differential equation is assumed to be of a certain form representable on a computer. The discrete system can then be solved on a computer and the results analyzed. The problem of generating good quality grids is very difficult for complex regions and has been a pacing item in computational mechanics. As grid generation techniques have become more powerful, they have allowed analysts to simulate more and more complex and realistic physical problems. Moreover, with the advent of supercomputers, grid generation methods are now required for discretizing arbitrary three-dimensional regions such as the region surrounding an aircraft in flight, prior to solution of three-dimensional field equations in these domains. Researchers in a wide range of fields have developed some very sophisticated methods for computational grid generation. We begin this survey by first classifying different types of grids as either structured, quasi-structured or unstructured and discuss the different topologies of each. Next we discuss various popular methods for generating grids and give some discussion of their relative merits. Finally we note some special topics including grid improvement methods and adaptive grid generation.

Classification of Computational Grids

We define a computational grid in a domain Ω (subset of R^n) as a collection of M sub-domains Ω_i such that the intersection of Ω_i and Ω_j is empty for all i and j, $i \neq j$, i,j = 1,2,...M and the union of the Ω_i is "close" to Ω. Therefore, we exclude the possibility that any of the Ω_i overlap each other and we ensure that there are no 'gaps' in the grid (except possibly right at the boundary of Ω). Moreover, we do not allow the volume (area in 2-d or length in 1-d) of any of the Ω_i to vanish (see fig. 1). We will be primarily concerned with Ω_i that have boundaries $\partial\Omega_i$ that are representable by polynomials. For example, in a two-dimensional domain we can have straight sided quadrilaterals representable by linear functions of the nodal points. We define nodal points as special points on the boundary of Ω_i that are used to construct the polynomial representing the boundary of Ω_i (fig. 1b) and are normally the points at which the approximate numerical solution is evaluated.

Computational grids can be classified into three main groups: 1) structured, 2) quasi-structured, and 3) unstructured. Structured grids are those that can be logically associated with (mapped, or transformed to) another base grid that has a regular repeatable structure. The base (also called the canonical or computational) domain is constructed entirely of a repetition of the same sub-domain througout the domain. Thus if Ω^* is the base domain, then the base grid is the collection of Ω_i^* where each Ω_i^* has the same size and shape. Examples of different types of base grids are shown in figures 2 a) and b). One can see that a base grid can be defined very easily as a array of the nodal points and a simple rule that determines which nodal points belong to which sub-domains and which elements are connected to each other. Moreover, the discretization of a partial differential equation on the base grid assumes a very simple (essentially canonical) form. For this reason, structured grids have been the most popular grids to date, especially in the finite difference literature. One important aspect of quadrilateral structured grids is that the sides of the elements can be associated with generalized coordinate directions. Thus to generate a quadrilateral structured grid is tantamount to specifying a transformation function that maps the computational space with coordinates $\xi = (\xi_1, \xi_2, ..., \xi_n)$ into the physical domain Ω with coordinates $x = (x_1, x_2, ..., x_n)$. This transformation can be stated as $\xi = \xi(x)$. This fact is extensively used in finite difference methods where the governing equation in the physical coordinates x is transformed into the computational coordinates using the map $\xi(x)$ and then the equation is solved in the regular base domain Ω^*. This has the advantage that the finite difference equations are essentially the same at each nodal point and

CH2345-7/86/0000/0568$01.00

therefore are easy to code in a computer program. Finite element methods generalize this concept with the idea of a master element. The master element is the computational domain and consists of only one sub-domain that can be mapped to every element in the physical domain. The discretized governing equation assumes a canonical form on the master element and therefore, the solution procedure becomes easy to program on a computer.

We define a quasi-structured grid as a grid consisting of a union of structured grids. The union is required to discretize the domain as we require for sub-domains. These grids are used in cases where there does not exist a simple transformation that defines a structured grid (for example in multiply connected domains). Thus the domain to be discretized is broken into a few sub-domains that each form a structured grid. These grids are sometimes termed composite or zonal grids and are used extensively in finite difference methods.

Finally, we define unstructured grids as those that are neither structured nor quasi-structured. These types of grids have been used extensively in finite element methods, although the grids were mostly generated by hand. Only fairly recently have automatic unstructured grid generation methods become common.

Transformations and Computational Coordinates

Structured grid generation methods are based on the existence of a transformation from the physical domain Ω to the computational domain Ω^*. We now show how derivatives of quantities in the physical space transform into their respective derivatives in the computational space (we show this for a two-dimensional domain only). Assume we want to determine the derivatives $\partial/\partial x$ and $\partial/\partial y$ of some function $u(x,y)$ in terms of derivatives $\partial/\partial\xi$ and $\partial/\partial\eta$ in the computational coordinates (ξ,η). Using the chain rule of partial differentiation we find

$$\begin{aligned} \partial u/\partial x &= \partial\xi/\partial x\ \partial u/\partial\xi + \partial\eta/\partial x\ \partial u/\partial\eta \\ \partial u/\partial y &= \partial\xi/\partial y\ \partial u/\partial\xi + \partial\eta/\partial y\ \partial u/\partial\eta \end{aligned} \qquad (1)$$

This can be written as a matrix equation

$$\begin{bmatrix} \partial u/\partial x \\ \partial u/\partial y \end{bmatrix} = \begin{bmatrix} \partial\xi/\partial x & \partial\eta/\partial x \\ \partial\xi/\partial y & \partial\eta/\partial y \end{bmatrix} \begin{bmatrix} \partial u/\partial\xi \\ \partial u/\partial\eta \end{bmatrix} \qquad (2)$$

We can similarly use the chain rule on the derivatives $\partial u/\partial\xi$ and $\partial u/\partial\eta$ to obtain the relations

$$\begin{bmatrix} \partial u/\partial\xi \\ \partial u/\partial\eta \end{bmatrix} = \begin{bmatrix} \partial x/\partial\xi & \partial y/\partial\xi \\ \partial x/\partial\eta & \partial y/\partial\eta \end{bmatrix} \begin{bmatrix} \partial u/\partial x \\ \partial u/\partial y \end{bmatrix} \qquad (3)$$

The matrix can be inverted to obtain the relation

$$\begin{bmatrix} \partial u/\partial x \\ \partial u/\partial y \end{bmatrix} = \frac{1}{J} \begin{bmatrix} \partial y/\partial\eta & -\partial y/\partial\xi \\ -\partial x/\partial\eta & \partial x/\partial\xi \end{bmatrix} \begin{bmatrix} \partial u/\partial\xi \\ \partial u/\partial\eta \end{bmatrix} \qquad (4)$$

where J is the Jacobian and is given by $J(\xi,\eta) = \partial x/\partial\xi\ \partial y/\partial\eta - \partial y/\partial\xi\ \partial x/\partial\eta$. Equating coefficients of the two matrices in eq. 2 and 4 gives the following relationships:

$$\begin{aligned} \partial\xi/\partial x &= (\partial y/\partial\eta)/J & \partial\eta/\partial x &= (-\partial y/\partial\xi)/J \\ \partial\xi/\partial y &= (-\partial x/\partial\eta)/J & \partial\eta/\partial y &= (\partial x/\partial\xi)/J \end{aligned} \qquad (5)$$

These are the required relations that determine how derivatives of a function in physical space transform into derivatives in the computational space. We now discuss some techniques for the generation of structured grids.

Conformal Mapping Methods

Conformal mapping methods are based on the use of complex analytic conformal transformations. That is, the map from computational to physical space is accomplished with the use of complex valued analytic functions. If we let $z = x + iy$ $(i = \sqrt{-1})$ be the physical complex-valued coordinate and $\zeta = \xi + i\eta$ be the computational complex-valued coordinate then grids are determined by specifying analytic transformations from ζ to z:

$$z = F(\zeta) \qquad (6)$$

Since all analytic transformations are conformal (except at singular points) then these transformations will produce conformal maps. A conformal transformation is one that preserves both the magnitude and the sense of angles. Thus, if the base grid consists of mutually orthogonal lines, then the physical grid will also be orthogonal. This fact is an advantage for conformal mapping techniques because if the physical grid is orthogonal, the resulting discretization of the partial differential equation becomes greatly simplified, and truncation errors are reduced. However, it is most usual to use conformal mappings to map the physical space to a simpler non-orthogonal domain and then use simple algebraic transformations to map the intermediate domain to the computational grid. This makes the resulting physical grid non-orthogonal. An example of a nearly orthogonal grid generated using a conformal mapping is shown in fig. 3.

Conformal mapping methods (like the algebraic methods discussed below) are extremely efficient and thus were widely used in the past[1,2,3].

Algebraic Methods[1]

The algebraic grid generation method is based on interpolation of known values of the nodal values of the physical grid on the boundary $\partial\Omega$ throughout the domain. We begin with a short discussion of one-dimensional interpolation. Assume that we have a function $r(\xi)$ that we know the values of at some points ξ_i. We wish to interpolate this function with a simple interpolant that posesses certain desirable properties. We construct interpolants $\varphi_i(\xi)$ that have the property that $\varphi_i(\xi_j) = \delta_{ij}$ (where δ_{ij} is the Kronecker delta). Thus, we can write

$$r(\xi) = \Sigma_i r(\xi_i)\, \varphi_i(\xi) \qquad (7)$$

where we assume that ξ varies from 0 to 1. The functions $\varphi_i(\xi)$ can take many forms, but it is most usual to use polynomial interpolants such as the Lagrange or Hermite polynomials. It is also possible to use cubic splines, B-splines or other types of spline functions for the uni-directional interpolation. Now assuming that we know the values of the grid point coordinates on the boundary of the transformed space, that is we know

$$\begin{aligned} &r(\xi,\eta) \quad \text{for } \eta = 0 \text{ or } 1 \\ &r(\xi,\eta) \quad \text{for } \xi = 0 \text{ or } 1 \end{aligned} \qquad (8)$$

then we can construct two interpolants based on interpolating r in the ξ and η directions separately:

$$\begin{aligned} r(\xi,\eta) &= \Sigma_{i=1,2}\ r(\xi_i,\eta)\, \varphi_i(\xi) \\ r(\xi,\eta) &= \Sigma_{j=1,2}\ r(\xi,\eta_j)\, \psi_j(\eta) \end{aligned} \qquad (9)$$

These two functions exactly interpolate the values of r on the boundaries $\xi = 0$ or 1 and $\eta = 0$ or 1 respectively. We can add the two interpolants together, but the resulting function does not interpolate r on the boundaries. However, we can interpolate the discrepancies on the boundaries, and subtract off this error from the total interpolant. This results in the transfinite interpolant:

$$\begin{aligned} r(\xi,\eta) = \Sigma_{i=1,2}\ r(\xi_i,\eta)\, \varphi_i(\xi) + \Sigma_{j=1,2}\ r(\xi,\eta_j)\, \psi_j(\eta) \\ \Sigma_{i=1,2}\ \Sigma_{j=1,2}\ \varphi_i(\xi)\, \psi_j(\eta)\, r(\xi_i,\eta_j) \end{aligned} \qquad (10)$$

This function exactly interpolates the values of r on all four boundaries of the rectangular computational space. By uniformly dividing the computational space, and applying the interpolating function we can generate the grid in the physical space, a very efficient process. The algebraic method can be easily generalized to three-dimensional grids, and to generate grids posessing any desired order of smoothness by an appropriate choice of the basis functions. However, the user of this method does not have complete control over such grid quantities as orthogonality and clustering.

The Maximum Principle

Solutions of the Dirichlet problem

$$\begin{aligned} &\Delta u = 0 \ \text{ in } \Omega \\ &u = \underline{u} \quad \text{on } \partial\Omega \\ &(\Delta = \partial^2/\partial x^2 + \partial^2/\partial y^2) \end{aligned} \qquad (11)$$

exhibit a very special and useful property known as the maximum principle. This principle states that the solution achieves its maximum and minimum values on the boundary of the domain. This fact can be used to generate grids that are guaranteed not to be overlapped and in addition are very smooth. The smoothness of the grid results from the inherent smoothness of solutions to the Dirichlet problem. Methods that are based on the maximum principle are popular and include the elliptic grid generation methods of Winslow (ref) and Thompson *et. al.*[1] and the variational method of Brackbill and Saltzman[4].

Elliptic Grid Generation

The elliptic grid generation method[1] is based on the numerical solution of the following equations

$$\begin{aligned} \Delta\xi &= 0 \\ \Delta\eta &= 0 \end{aligned} \qquad (12)$$

Solutions to these equations will satisfy the maximum principle. Thus, the maximum and minimum values of ξ and η will be reached on the boundary of the physical space and thus there is no possibility that grid overlap can occur in the domain. To generate the grid, these equations are transformed using relations (5) above to relate the derivatives in the computational space to derivatives in physical space. The result is the following non-linear elliptic system of partial differential equations to be solved for the physical grid

$$\begin{aligned} \alpha\, x_{\xi\xi} - 2\,\beta\, x_{\xi\eta} + \gamma\, x_{\eta\eta} &= 0 \\ \alpha\, y_{\xi\xi} - 2\,\beta\, y_{\xi\eta} + \gamma\, y_{\eta\eta} &= 0 \end{aligned} \qquad (13)$$

where

$$\alpha = x_\eta^2 + y_\eta^2$$

$$\beta = x_\xi x_\eta + y_\xi y_\eta$$

$$\gamma = x_\xi^2 + y_\xi^2$$

These equations are discretized using finite differences and then solved using iterative techniques. The method can be modified so that interior grid point clustering can be controlled by the addition of non-homogeneous terms to the right-hand-side of eq. 12. This results in a coupled system of non-linear Poisson equations to be solved. An Example of the type of grid generated by this method is shown in fig. 4.

The Variational Grid Generation Method

The variational grid generation method[4] is based on some heuristic principles concerning what a good quality grid consists of. These principles are 1) the grid should be as smooth as possible, 2) orthogonal as possible, and 3) should cluster in specified regions of the physical domain. These principles can be stated mathematically by minimizing appropriate measures of the smoothness, orthogonality and clustering functions. The smoothness of the grid is measured by the following functional

$$I_s = \int_\Omega (\nabla\xi)^2 + (\nabla\eta)^2 \, dx \, dy \qquad (14)$$

The orthogonality of the grid is measured by the functional

$$I_o = \int_\Omega \nabla\xi \cdot \nabla\eta \, dx \, dy \qquad (15)$$

Finally, the grid clustering is measured by the functional

$$I_w = \int_\Omega w(\xi,\eta) \, J(\xi,\eta) \, dx \, dy \qquad (16)$$

Now we can form one functional that is a weighted average of all three quantities we wish to minimize:

$$I(\xi,\eta) = \lambda_s I_s + \lambda_o I_o + \lambda_w I_w \qquad (17)$$

where λ_s, λ_o, and λ_w are scalars that give different weighting to each functional. Minimizing $I(\xi,\eta)$ will produce an optimal grid with respect to the chosen grid quality measures. The ξ coordinates are transformed to the physical x coordinates using eq. 5 and then the Euler-Lagrange equations are formed. The resulting system is similar to the system of eq. 13 and is solved using finite difference discretization and iteration. One can see that where the weighting function is large, the grid cell size will be small, achieving the required clustering. Therefore, the method is particularly suited to generation of adaptive grids. The usual procedure is to choose the weight function as a measure of the rate of change of the solution to the equation we are trying to solve on the grid. One example of this adaption is shown in fig. 5. Note how the grid adapts to the steep gradients in the vicinity of the shock waves in the solution.

It is possible to directly discretize and minimize $I(\xi,\eta)$ before applying the Euler-Lagrange equations. This method was used in refs. 5 and 6 to generate structured and quasi-structured grids. Functionals analogous to I_s, I_o and I_w were developed to measure the grid quality. The grid points are solved for by minimizing the grid quality measures using a conjugate gradient optimization algorithm. An example grid generated by this procedure is shown in fig. 6. This method is also applicable to unstructured grids and is further discussed below.

Quasi-Structured Grids

Quasi-structured (zonal, composite or patched) grids have been used by many[1,2] to take advantage of the simplicity of finite difference methods while allowing the discretization of more general domains. The physical domain is first subdivided into a small number of regions, each of which can be separately associated with a rectangular computational domain. The grids are generated in each sub-domain and then patched together along common boundaries. These interfaces between the domains can be fixed or allowed to float along with the nodal points in the grid generation procedure. A method for generating composite grids based on the method of ref. 5 was developed in ref. 6. An example of using this procedure is shown in fig. 7. This is a composite grid consisting of two structured grids, one an O-type grid about a gas turbine engine blade and the second a C-type grid surrounding the O-type grid. The quasi-structured grid approach is a powerful tool; however, it is not as general as the unstructured grid methods.

Unstructured Grids

Unstructured grids afford the most geometric generality. In fact, one can discretize an arbitrary domain using an appropriate unstructured grid. For this reason, these types of grids have been receiving much attention recently, as solutions of field equations are required on more and more complex geometries. Unstructured grids can be generated by hand (or with the use of a digitizing tablet), but this method is tedious, time-consuming and has a large human error factor. Thus, we are led to automatic methods for unstructured grid generation. The most general automatic unstructured grid generation methods are those based on the n-simplex (triangle in 2-d and tetrahedra in 3-d). The procedure is to first specify the positions of the N nodal points which in this case are just the vertices of the triangles or tetrahedra. (The positions of the nodal points can, for example, be generated pseudo-randomly). Next, a scheme is used to connect the N points to form the triangular elements forming the grid. The points x_j, $j = 1,2,...N$, can be joined using purely heuristic or semi-optimal rules[7]. The optimality

of the grid can be measured by how similar each element is to an equilateral triangle (similarly in 3-d). It is striking that there exists a very simple rule for joining the vertices that ensures that the grid is optimal--the Delaunay criterion[8,9]. First we define the Voronoi polygon surrounding a point x_i as the set of points closer to x_i than any other point

$$V(x_i) = \{ x \mid d(x,x_i) \le d(x,x_j) \text{ for all } j=1,2,...,N\}$$

where $d(\cdot,\cdot)$ is the Euclidean metric ($d(x,y)$ = distance between x and y). Two points x_i and x_j are termed Voronoi neighbors if their respecive Voronoi polygons share a common edge. Thus, the Delaunay criterion states that node x_i is to be connected to node x_j if they are Voronoi neighbors. This procedure produces grids such as the one shown in fig. 8 for two airfoils in a wind tunnel. Note that the procedure has to be modified for non-convex domains such as this. In effect, the connections are checked to see if they cross any of the required boundary edges, and if so, one of the nodes is deleted from the search for Voronoi neighbors. This triangulation procedure can be made to take on the order of NlogN operations, and thus is very efficient.

Grid Improvement and Adaption Methods

In general, a given computational grid is not the optimal one for solving a problem. For example, it is usually required to have the grid clustered in some regions of the domain in which the solution to the problem is varying rapidly. Since the solution is not known *a priori*, we are led to methods that dynamically adapt the grid to the solution. This adaption can take either of two forms: 1) the main grid is held fixed while individual sub-domains are refined into smaller sub-domains (this is called adaptive refinement) or 2) the basic logical structure of the grid is held fixed and the grid points are allowed to move such that they become clustered in required regions (this is just called an adaptive grid). The effect of either of these types of adaptions is to make the grid spacing smaller in the required regions, and therefore increasing the accuracy of the numerical solution. An example of adaptive grid refinement[10] is shown in fig. 9. Note how the grid is refined in certain regions requiring resolution. An example of an adaptive grid is shown in fig 5. The grid points have moved and clustered in the required regions. The method of ref. 5 can be modified so that it is applicable to unstructured grids. Thus, unstructured adaptive grids could be generated analogously to structured adaptive grids. In conclusion, adaptive methods are one of the most promising techniques for improving the accuracy of numerical solutions to partial differential equations.

Conclusions

Numerical methods for the generation of computational grids have been presented. These methods have been shown to be very powerful tools for use in the simulation of complex physical processes governed by partial differential equations.

References

[1] Thompson, J. F., Z. U. A. Warsi, and C. W. Mastin. Numerical Grid Generation. Elsevier Science Publishing Co. New York. 1985.

[2] Ghia, U. and K. Ghia, eds. Advances in Grid Generation. American Society of Mechanical Engineers. 1983.

[3] Dulikravich, G. S., "CAS22-FORTRAN Program for Fast Design and Analysis of Shock-Free Airfoil Cascades Using Fictitious-Gas Concept," NASA CP-3507, January, 1982.

[4] Brackbill, J. U. and J. S. Saltzman, "Adaptive Zoning for Singular Problems in Two Dimensions," Journal of Computational Physics, **46**, 1982, pp. 342-368.

[5] Kennon, S. R. and G. S. Dulikravich, "A Posteriori Optimization of Computational Grids," AIAA paper no. 85-0483, Reno, Nevada, January, 1985.

[6] Kennon, S. R. and G. S. Dulikravich, "Composite Computational Grid Generation Using Optimization," First International Conference on Numerical Grid Generation in Computational Fluid Dynamics, Landshut, W. Germany, July 14-17, 1986.

[7] Nguyen, V. Ph., "Review of Techniques for Efficient Network Generation for Finite Element Analysis," VDI-Forschungsheft No. 2, 1982.

[8] Watson, D. F., "Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes," The Computer Journal, **24**, no. 2, 1981, pp 167-172.

[9] Bowyer, A., "Computing Dirichlet Tessellations," The Computer Journal, **24**, no. 2, 1981, pp. 162-166.

[10] Löhner, R., K. Morgan and O. C. Zienkiewicz, "An Adaptive Finite Element Procedure for Compressible High Speed Flows," CMAME, **51**, 1985, pp. 441-465.

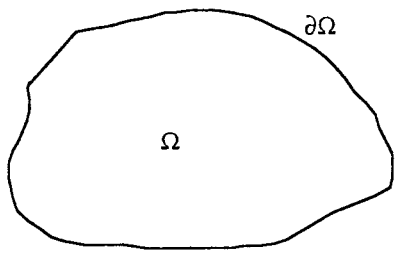

Fig. 1a A two-dimensional domain Ω

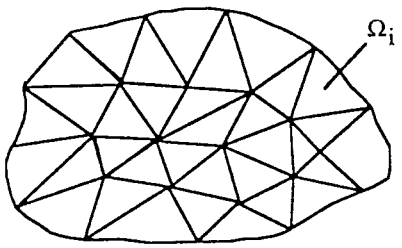

Fig. 1b The domain Ω showing its discretization into sub-domains Ω_i

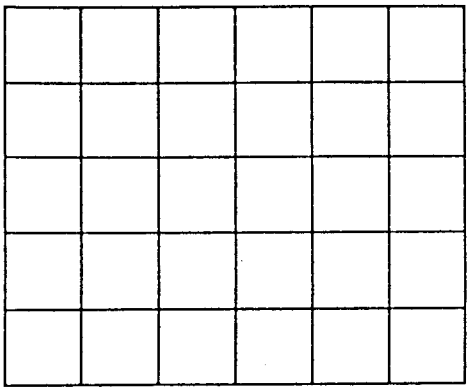

Fig. 2a A typical base domain showing discretization using a repetition of squares

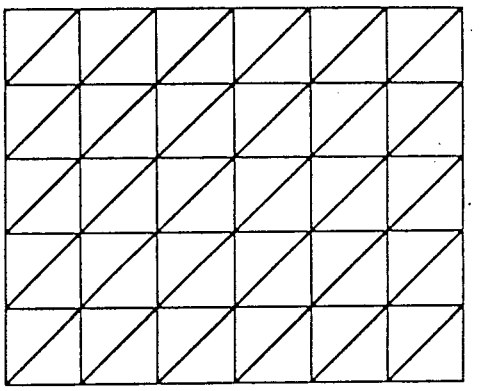

Fig. 2b A base domain consisting of a repetition of regular triangular sub-domains

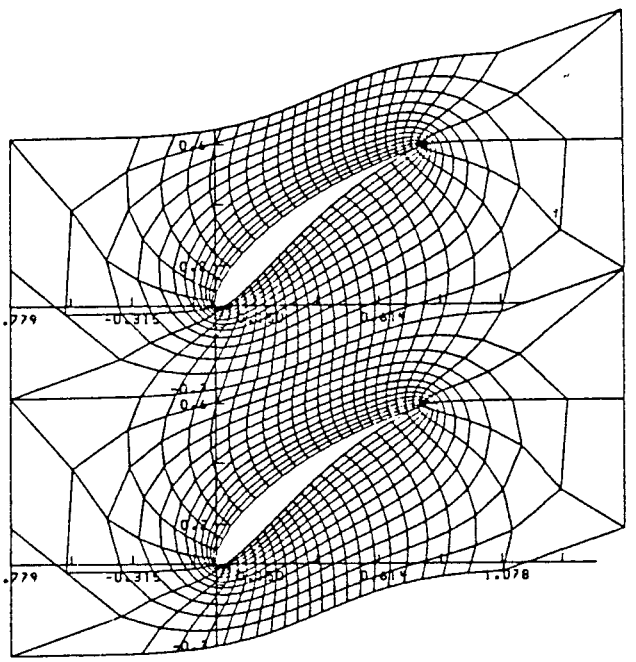

Fig. 3 Grid about a two-dimensional cascade generated using conformal mapping[3].

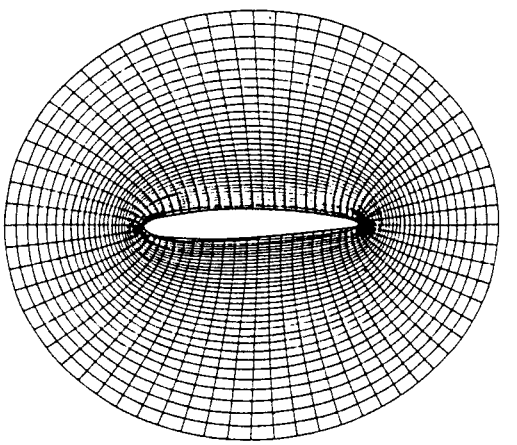

Fig. 4 Airfoil grid generated using the elliptic method[1].

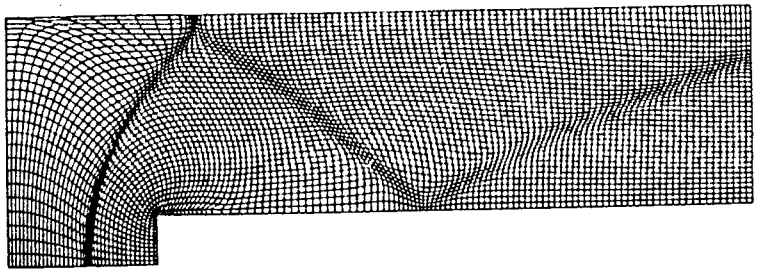

Fig. 5 Adaptive grid for a two-dimensional compressible flow problem generated using the variational method[4].

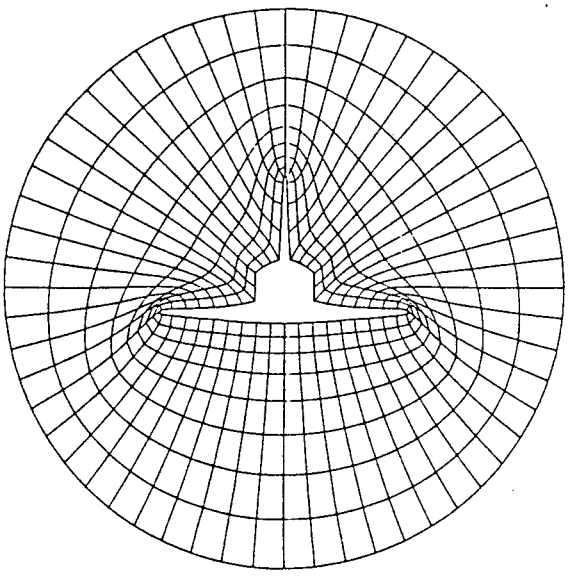

Fig. 6 Optimized grid for a space shuttle cross-section[5].

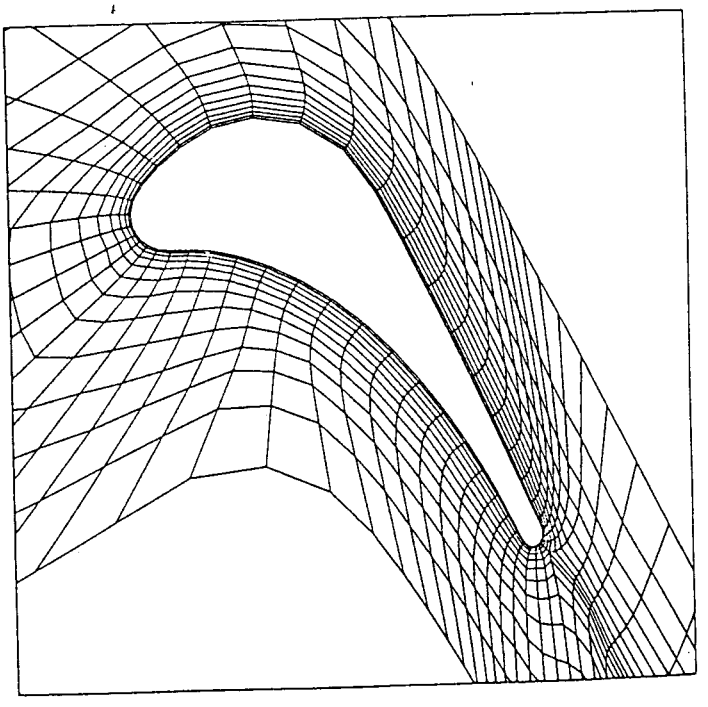

Fig. 7 Composite grid for a turbine blade[6].

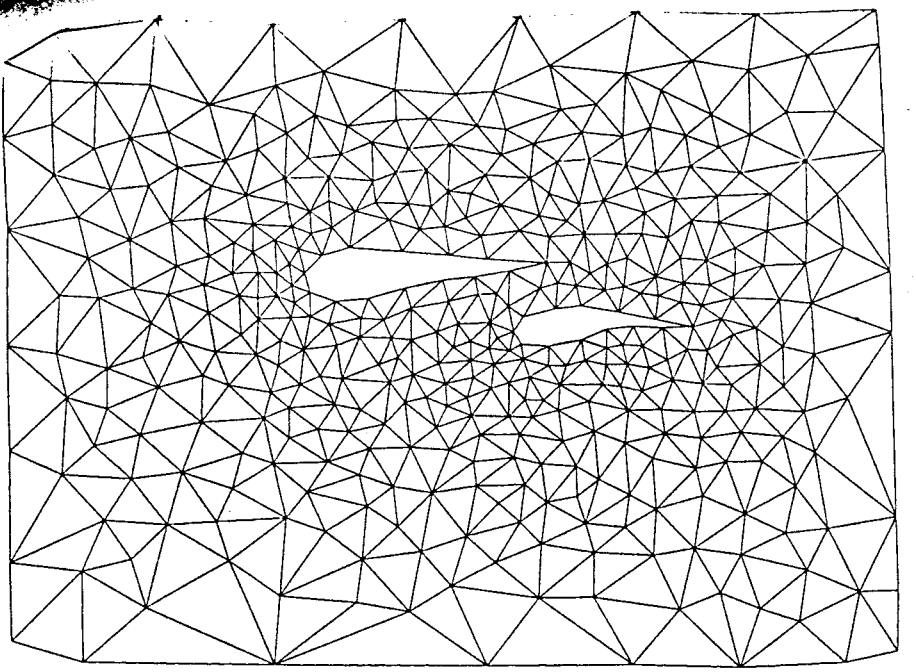

Fig. 8 Delaunay triangulation for two airfoils in a wind tunnel.

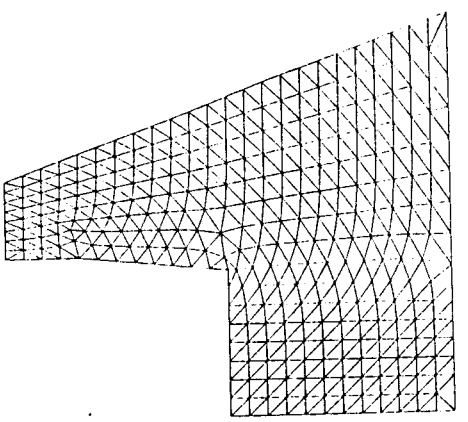

Fig. 9a Initial un-refined grid

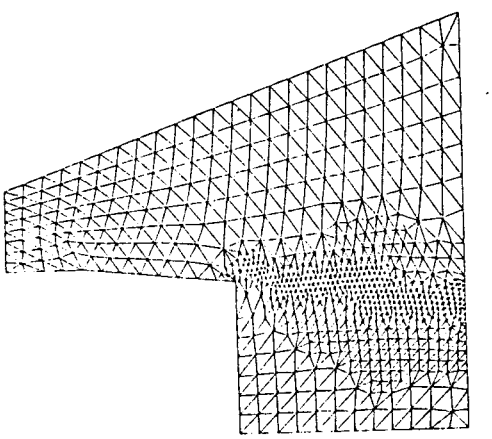

Fig. 9b Adaptively refined grid