NASA Contractor Report 182090

A PACKAGE FOR 3-D UNSTRUCTURED GRID GENERATION, FINITE-ELEMENT FLOW SOLUTION AND FLOW FIELD VISUALIZATION

```
(NASA-CR-182090)  A PACKAGE FOR 3-D                        N90-29309
UNSTRUCTURED GRID GENERATION, FINITE-ELEMENT
FLOW SOLUTION AND FLOW FIELD VISUALIZATION
Final Report  (Vigyan Research Associates)                 Unclas
104 p                                     CSCL 01A G3/02   0309061
```

Paresh Parikh, Shahyar Pirzadeh, and Rainald Löhner

VIGYAN, INC.
Hampton, Virginia

Contract NAS1-18670
September 1990

NASA
National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

TABLE OF CONTENTS

A PACKAGE FOR 3-D UNSTRUCTURED GRID GENERATION, FINITE-ELEMENT FLOW SOLUTION AND FLOW FIELD VISUALIZATION

ABSTRACT

A set of computer programs for 3-D unstructured grid generation, fluid flow calculations and flow field visualization has been developed. The grid generation program, called **VGRID3D**, generates grids over complex configurations using the advancing front method. In this method, the point and element generation is accomplished simultaneously. **VPLOT3D** is an interactive, menu-driven pre- and post-processor graphics program for interpolation and display of unstructured grid data. The flow solver, **VFLOW3D** is an Euler equation solver based on an explicit, two-step, Taylor-Galerkin algorithm which uses the Flux Corrected Transport (FCT) concept for a wriggle-free solution. Using these programs, increasingly complex 3-D configurations of interest to aerospace community have been gridded, such as the Space Transportation System comprised of the space-shuttle orbiter, the solid-rocket boosters and the external tank. Flow solutions have been obtained on various configurations in subsonic, transonic and supersonic flow regimes.

ACKNOWLEDGEMENT

The authors would like to express their sincere appreciation to Mr. M. D. Salas, Fluid Mechanics Division, NASA Langley Research Center for his unending support of this work. We are also grateful to Mr. Clyde Gumbert, Computational Aerodynamics Branch, NASA Langley Research Center for sharing his expertise on various matters related to graphics.

Thanks are also due to the following persons/companies for supplying the initial surface definitions from which input data to the current grid generator were extracted. The Boeing Commercial Airplane Company, Seattle for the Boeing 747 configuration, Dr. R. E. Smith, Computer Applications Branch, NASA Langley Research Center for a generic fighter configuration, McDonnel Douglas Company for an F-18 fighter configuration and Dr. C. P. Li, NASA Johnson Space Center for the Space Transportation System configuration.

This work was supported by NASA under an SBIR contract entitled "Generation of Unstructured Grids in Three-Dimensions" (NAS1-18670). Mr. Clyde Gumbert was the Technical Monitor.

LIST OF FIGURES

1. INTRODUCTION

One of the greatest concerns in computational fluid dynamics is the generation of suitable grids. Even though considerable effort has been devoted towards development of robust, user-friendly grid generators, the process of generating 3-D grids around complex geometries remains a formidable challenge. With the availability of large supercomputers, it is now possible to calculate flowfields around complex configurations in a matter of hours. However, the grid generation, using structured grid methods, still makes up a large portion of a typical computational cycle for a complex configuration.

Over the past few years, an alternative grid generation technique, unstructured grids, has received a lot of attention. Besides their inherent capability of handling complex configurations with ease, unstructured grids are apt to efficiently incorporate adaptive refinement and moving boundaries and offer better control over the grid size and point clustering. It has become convincingly clear that to handle complex 3-D configurations of aerospace interest, the use of unstructured grids is a viable, if not the only, solution.

During the phase I of this study, a technique of generating 3-D unstructured grids using the advancing front method was explored. In this method, the introduction of new points and elements is carried out simultaneously. An advantage of this method, over others that generate grids by connecting an already existing point distribution, is that it does not require a separate library of modules for generation of points in advance, thus giving a better control over the grid spacing. Due to their inherent lack of directionality, unstructured grids and their corresponding flow solutions are diffucult to visualize. Thus the need for a post-processing program became immediately apparent. An initial attempt was made to develop such a post-processor program during the phase I study.

The current (phase II) work was begun with a broad aim of maturing the unstructured grid technique as much as possible. The grid generator needed to be more robust and efficient, the graphics post-processor program needed to be expanded and a preliminary version of an Euler equation solver based on the unstructured grids was to be developed. Some pre-processor programs were also needed to extract data suitable for the grid generator from other widely used data structures.

These requirements, and more, have been accomplished during the phase II study. As a result, the following three computer programs have been developed.

VGRID3D is a program for the generation of 3-D unstructured grids around complex configurations using the advancing front method [1]. During the course of this work, many increasingly complex configurations were gridded using VGRID3D; the surface grids on two of them, a Space Transportation System (STS) configuration and a Boeing 747 with flow through engines, are shown in Figures 1 and 2, respectively.

VPLOT3D is a workstation-based, interactive pre- and post-processor graphics program for unstructured grid data manipulation and display. It has the ability to display a variety of scalar and vector quantities on user defined planes and surfaces as well as field quantities such as particle traces [2]. A sample picture generated using VPLOT3D is shown in Figure 3.

VFLOW3D is an Euler equation solver based on an explicit, two-step Taylor-Galerkin algorithm using the Flux Corrected Transport methodology [3]. The program has options for solution adaptive grid refinement and capability of handling moving grids. Although all the options have not been fully and thoroughly tested, VFLOW3D has been shown to give reasonable results for steady, transonic cases. As an example, surface pressure contours on the STS configurations are shown in Figure 4.

The present report is organized into 6 chapters. In Chapter 2, details of the procedure for generation of grids using the advancing front method are described. Chapter 3 is a user's manual for VGRID3D. In Chapter 4, a description of the capabilities of the post-processing program, VPLOT3D, is provided along with some comments on its use. Chapter 5 describes the flow solution methodology and includes a sample calculation compared with experimental data. Finally, some concluding remarks are presented in

Chapter 6.

In summary, a set of programs for unstructured grid construction, fluid flow calculations and flow field visualization has been developed.

2. UNSTRUCTURED GRID GENERATION

A wide variety of algorithms has been devised for the generation of unstructured grids around complex geometries over the recent years. Among the different techniques are Watson's algorithm for Voronoi tesselation [4-9], the modified octree method [10], and advancing front technique [11-13]. In the Voronoi/Delaunay family of grid generation techniques, grid points are first distributed throughout the computational domain and then connected to form triangular/tetrahedral grid cells. The criterion for generation of grids, in this method, is that the circumsphere of any grid element should not contain any other point in the grid. While the Voronoi algorithm [9] has shown to be fast and reliable in generating meshes, the incorporation of directional refinement in this technique appears difficult unless the reconnection of points based on the purely geometrical Delaunay criterion is substituted by some other criterion that incorporates directionality into the algorithm.

In the advancing front technique, new points are introduced while the domain of interest is being filled with triangles/tetrahedra. After obtaining an initial front consisting of linear (2-D) or triangular (3-D) faces over the surface of the configuration of interest, elements are generated by successively removing faces from the current front and adding new points and faces (advancing the front) until the whole region is filled with grid elements. This method is advocated here because it does not require a separate library of modules to introduce points throughout the domain in advance, and it has the flexibility of regenerating the grid adaptively as well as directional refinement in a simple manner. The advancing front method and various aspects of the present grid generator are discussed in detail in the following sections.

2.1 OUTLINE OF THE TECHNIQUE

The grid generation process, in the present work, is composed of the following main steps:

(i) The boundaries of the domain to be gridded are divided into a number of lines (2-D) or surface patches (3-D). These lines or surface patches define the configuration of interest as well as the far-field boundaries.

(ii) A background grid is set up to define the local grid characteristics such as grid point spacings, grid element stretching ratios, and stretching directions.

(iii) The lines or surface patches of step (i) are further subdivided into a number (dictated by the background grid) of straight line segments (2-D) or triangles (3-D) to form the first front.

(iv) The front is advanced by introducing new points and faces to complete the grid.

(v) The completed grid may optionally be refined by direct splitting of each grid cell (h-refinement).

(vi) The final grid can be smoothed by removing 'distorted' and 'small' elements and moving grid points around.

Steps (i) and (ii) are accomplished interactively using graphic codes developed on Silicon Graphics IRIS 4D series workstations. Steps (iii) - (vi) are automatically performed by the 3-D grid generator using the input data obtained from steps (i) and (ii).

2.2 DEFINITION OF SURFACES

As mentioned earlier, the advancing front algorithm requires an initial front (surface mesh) in order to start the triangulation/tetrahedrization of the domain. This, in turn, means that the surfaces representing the domain to be gridded need to be defined. Several approaches have been proposed in the literature [14]:

(a) *Boolean Operations on Solids* - In this approach, the domain to be gridded is constructed from primitives (box, sphere, cylinder, etc.) The user combines these primitives through Boolean operations (union, intersection, exclusion, etc.) to represent the domain to be gridded. The surface is then obtained in a post-processing operation.

(b) *Boolean Operations on Surfaces* - Here, only the surface of the domain to be gridded is defined in terms of independent surface patches . The surface patches are then combined using Boolean operations to yield the final surface of the domain to be gridded.

Both approaches have their advantages. The first approach is more compatible with many current CAD-CAM systems in use, thus allowing transfer of the object data directly from design to analysis. The second approach requires less manual input. Moreover, since the desired surface of the domain is obtained immediately, and not in a post-processor step, the software involved is less complex. In order to be compatible with the surface definitions currently in use in the aerospace and automobile industries, the second approach is adopted here. Surface patches are easily generated from available panel representations or sets of cross-section shapes which describe the configuration using an interactive graphics code. To minimize the manual effort involved in defining a surface, a hierarchical data structure is adopted. There are three levels of data: points, lines and surfaces. Lines are obtained by joining the points, and surfaces by joining lines. The line types implemented are:

(a) straight line segments defined by two points,

(b) parabolic line segments defined by three points, and

(c) cubic spline segments defined by four or more points.

The surface types implemented are:

(a) planar surfaces defined by three or more arbitrary line segments, all lying in one plane,

(b) triangular isoparametric parabolic surfaces [15] defined by three parabolic line segments,

(c) rectangular isoparametric serendipity surfaces [15] defined by four parabolic line segments,

(d) triangular Barnhill-Gregory-Nielson patches [16] defined by three arbitrary line segments, and

(e) bilinear transfinite Coon's patches [16] defined by four arbitrary line segments.

The line and surface types are not, of course, limited to those listed here; other desirable types such as quintic and exponential splines and bicubic Coon's patches as well as other C^1 and C^2 continuous surfaces can be implemented as required.

2.3 BACKGROUND GRID

The local characteristics of the grid to be generated are defined by a set of parameters distributed throughout the domain. These parameters control the point distribution, element stretching, and the stretching direction of the final grid. The control parameters are stored at the nodes of a mesh referred to as the background grid. The background grid is made of a few tetrahedral elements which completely encloses the domain to be gridded. This mesh is not required to conform to any of the surfaces of the configuration. It is used as a source of information and for interpolation purpose only.

There are three methods in use for constructing three-dimensional background grids:

(a) in the first method, a 2-D triangular grid plane (usually aligned with the plane of symmetry) is duplicated a number of times along a desired axis. The corresponding nodal points, in pairs of planes, are then connected to form tetrahedra. At the nodes of this grid, local grid parameters are specified. An interactive graphics code has been developed to facilitate the construction of a 2-D triangular plane with the desired point distribution. Once the plane of triangles is prepared, it is duplicated at a specified number of locations along the axis perpendicular to the original plane, and points are connected automatically. The positions of the grid points may be modified interactively, and the corresponding grid parameters can be changed as desired.

(b) For certain configurations such as axisymmetric geometries, a 2-D plane of triangles is duplicated as it is rotated about the axis of symmetry. Points are then connected as in the previous method.

(c) In the third method, an arbitrary number of points are distributed interactively in the 3-D space. This set of points along with the outer faces are used to form tetrahedra with the advancing front method.

While the third method is probably the most desirable one for its generality and flexibility, it requires the highest amount of manual input. The first method has been used extensively in the present work.

2.4 GENERATION OF THE INITIAL FRONT

The generation of the initial front for a three-dimensional configuration is carried out in two main steps:

(i) the line segments connecting surface patches are divided into straight line segments, called 'sides';

(ii) the surface segments are triangulated using the two-dimensional version of the advancing front method with the 'sides' serving as the initial front.

The size of each 'side' or triangle is determined from the backgroung grid element into which the side or triangle falls. This is done by a simple linear interpolation of the spatial distribution parameters stored at the four nodes of the element.

The division of line segments into 'sides' is performed through the following steps:

(i) given the current location specified by the position vector $\overline{x}_0$, use the background grid and the tangential vector of the line to determine the vector $\overline{dx}_0$ which tentatively defines the next 'side';

(ii) determine another vector $\overline{dx}_1$ using the background grid and the tangential vector of the line at the mid-point position $\overline{x}_1 = \overline{x}_0 + 0.5\,\overline{dx}_0$;

(iii) if $\overline{dx}_0 \not\approx \overline{dx}_1$, set $\overline{dx}_0 = 0.5\,(\overline{dx}_0 + \overline{dx}_1)$ and go to (ii);

(iv) add the new 'side', reset $\overline{x}_0$ to the new position, and go to (i).

In order to avoid an unusually short 'side' left at the end of a line segment, the length of the remaining (undivided) portion of the line segment is checked at each iteration. If the remaining length is within a specified limit, the end point of the line segment is used to form the last 'side'. Once all the line segments forming surface patches are divided into 'sides', each surface patch is then triangulated independently using a 2-D version of the advancing front grid generator. Since this section of the program operates in a 2-dimensional frame of reference, mappings between the 3-D and 2-D worlds are required. Hence, a 3-D surface patch is first mapped to a 2-D segment, then triangulated using a 2-D grid generator, and finally mapped back to the original 3-D shape.

The mappings are such that the shape and size of the 3-D surfaces are approximately maintained in the 2-D frame and are performed in two steps as illustrated in Figure 5:

(i) mapping of 3-D surface patches to unit 2-D triangles or squares ($x, y, z \rightarrow \xi, \eta$),

(ii) stretching and shearing of unit 2-D triangular or square shapes to approximate the shape of 3-D surface patches in a 2-D domain ($\xi, \eta \rightarrow \xi'', \eta''$).

The mappings for various surface types are described below.

2.4.1 Planar Surface Segment

With the notation defined in Figure 6, we select two arbitrary vectors $\overline{a}$ and $\overline{b}$ that lie in the plane. The normal to the plane is given by $\overline{n} = \overline{a} \times \overline{b}$, and a vector normal to both $\overline{a}$ and $\overline{n}$ is computed as $\overline{c} = \overline{n} \times \overline{a}$. Defining the covariant basis vectors $\overline{x}_1$ and $\overline{x}_2$ of the plane as

$$\overline{x}_1 = \frac{\overline{a}}{|\overline{a}|} \quad , \quad \overline{x}_2 = \frac{\overline{c}}{|\overline{c}|} \quad , \tag{2.1}$$

any vector lying in the plane can be written as

$$\overline{x} = \overline{x}_0 + \xi''\overline{x}_1 + \eta''\overline{x}_2 \quad , \tag{2.2}$$

where the ξ'' and η'' values are given by

$$\xi'' = \overline{x}_1 \cdot (\overline{x} - \overline{x}_0) \quad , \quad \eta'' = \overline{x}_2 \cdot (\overline{x} - \overline{x}_0) \quad . \tag{2.3}$$

2.4.2 Triangular Isoparametric Parabolic Surface Segment

The triangular isoparametric parabolic surface segment is most easily described in terms of the six points defining it. The situation is shown in Figure 7. Any vector lying on the surface may be written as

$$\overline{x} = \sum_{i=1}^{6} N^i \overline{x}_i \quad , \tag{2.4}$$

where the shape-functions N^i are given by [15]:

$$
\begin{aligned}
N^1 &= (1-\xi-\eta)(1-2\xi-2\eta)\\
N^2 &= 4\xi(1-\xi-\eta)\\
N^3 &= \xi(2\xi-1)\\
N^4 &= 4\xi\eta\\
N^5 &= \eta(2\eta-1)\\
N^6 &= 4\eta(1-\xi-\eta)
\end{aligned}
\tag{2.5}
$$

2.4.3 Rectangular Isoparametric Serendipity Surface Segment

This surface segment is again most easily described in terms of the eight points defining it. The situation is shown in Figure 8. Any vector lying on the surface may be written as

$$\overline{x} = \sum_{i=1}^{8} N^i \overline{x}_i \quad ,$$

where the shape-functions N^i are given by [15]:

$$
\begin{aligned}
N^1 &= (1-\xi)(1-\eta)(1-2\xi-2\eta)\\
N^2 &= 4\xi(1-\xi)(1-\eta)\\
N^3 &= -\xi(1-\eta)(1-2\xi+2\eta)\\
N^4 &= 4\xi\eta(1-\eta)\\
N^5 &= -\xi\eta(3-2\xi-2\eta)\\
N^6 &= 4\xi\eta(1-\xi)\\
N^7 &= -\eta(1-\xi)(1+2\xi-2\eta)\\
N^8 &= 4\eta(1-\xi)(1-\eta)
\end{aligned}
\tag{2.6}
$$

2.4.4 Triangular Barnhill-Gregory-Nielson Patch [16]

This surface segment is described in terms of the three line-functions along its edges. The situation is shown in Figure 9. Any vector lying on the surface may be written as

$$\overline{x} = \xi\overline{F}_2(\eta) + \eta\overline{F}_2(1-\xi) + \overline{F}_1(\xi) + \overline{F}_3(1-\eta) - \eta\left[\overline{F}_1(\xi) + \overline{F}_3(\xi)\right]$$
$$-\xi\left[\overline{F}_1(1-\eta) + \overline{F}_3(1-\eta)\right] - (1-\xi-\eta)\overline{F}_1(0). \qquad (2.7)$$

We observe that not all the sides are treated equally by this formula. The $\overline{F}_2$-line plays a different role than the other two lines defining the patch. This means that some care has to be taken when using this surface segment.

2.4.5 Rectangular Bilinear Coon's Patch [16]

This surface segment is described in terms of the four line-functions along its edges. The situation is shown in Figure 10. Any vector lying on the surface may be written as

$$\overline{x} = (1-\eta)\overline{F}_1(\xi) + \xi\overline{F}_2(\eta) + \eta\overline{F}_3(\xi) + (1-\xi)\overline{F}_4(\eta)$$
$$-\left[(1-\xi)(1-\eta)\overline{F}_1(0) + \xi(1-\eta)\overline{F}_2(0) + \xi\eta\overline{F}_2(1) + (1-\xi)\eta\overline{F}_3(0)\right]. \qquad (2.8)$$

2.4.6 Stretching and Shearing of 2-D Unit Segments ($\xi, \eta \rightarrow \xi'', \eta''$)

In order to obtain a reasonable triangulation of curved surfaces, the unit triangle or square is stretched and sheared in 2-D so as to approximate the 3-D surface segment. The easiest way to accomplish this for a rectangular surface segment is to use a bilinear isoparametric mapping between the unit square and the 2-D surface approximation. This case is shown in Figure 5. However, it is not straightforward to transform back and forth with the bilinear isoparametric mapping. We therefore set the η'' value of points D and C to be the same. The mapping is then given by:

$$\xi'' = x_B\xi + x_D\eta + (-x_B + x_C - x_D)\xi\eta \quad , \quad \eta'' = y_D\eta \quad , \tag{2.9}$$

or

$$\eta = \frac{\eta''}{y_D} \quad , \quad \xi = \frac{\xi'' - x_D\eta}{x_B + (-x_B + x_C - x_D)\eta} . \tag{2.10}$$

The distances A-B, A-D and D-C are obtained from the line information given, and the angle at the origin (point A) is approximated from the 3-D surface segment. For triangular surface segments, point C is omitted.

2.5 ADVANCING THE FRONT

Once the initial front (surface mesh) is constructed, the field grid is generated by introducing new points in the domain. This involves forming tetrahedral elements by connecting the new and/or existing front points and redefining the current front. Front advancement consists of the following steps:

(i) Find the next face (triangle) to be removed from the front. In order to avoid large elements crossing over regions of small elements, the face forming the next smallest element is usually selected as the next face to be deleted from the list of faces.

(ii) For the face to be removed, determine a 'best point' as the fourth node of the next tetrahedral element to be formed. This point is positioned in such a way that its projection onto the plane of the face is located either at the mid-point or at the circumcenter of the face. Its distance from the face is determined using the information interpolated from the nodes of the background grid element into which the face falls.

(iii) Determine whether an existing point on the current front could be used instead of the 'best point'. This involves finding all close points to the face to be removed.

(iv) Find all close faces to the face to be removed. This information is needed later for face-crossing check.

(v) Filter those close points which are either not on the front or in the 'wrong' side of the face.

(vi) Filter unneeded close faces.

(vii) Order the remaining close points and the 'best point' in a list according to an appropriate criterion. Points are ordered in this list from the most desirable element making point at the top to the least appropriate one at the bottom of the list. The criteria used for the ordering are discussed in a subsequent section.

(viii) Starting from the top of the list, check whether the new element formed by a point would cross any existing face. If it does not cross, take that point for forming the next element; if it does cross, try the next point in the list.

(ix) Once an appropriate point is selected, add the new point (if any) , face(s), and element to their respective lists. Also delete from the list the face(s) which has (have) been removed from the front due to formation of the new element.

(x) For the new face(s), find the generation parameters from the background grid element(s) enclosing the face(s). This information is subsequently used for determining the next face to be removed from the front (step i) and calculating the corresponding 'best point' position (step ii).

(xi) Repeat steps (i) through (x) until no face in the front is left, i.e., the entire computational domain is gridded.

A considerable portion of the total computational time, in any unstructured grid technique, is associated with various search operations as no simple (i, j, k)-addressing of grid points, faces, and elements exists, like the one in structured grids. These operations, as mentioned before, involve (a) finding the next face to be removed from the front, (b) finding the closest points to a given point in the field, (c) finding the faces adjacent to a given point, and (d) finding the background grid element which encloses a given point. These operations could potentially lead to an unsatisfactory efficiency of $O(N^{1.5})$ or even $O(N^2)$ for the algorithm where N is the total number of points, faces, or elements. The objective here is to design efficient data structures for performing the search operations which would improve the overall efficiency of the algorithm. The data structures and the front advancing steps are described in the following sub-sections.

2.5.1 Heap List for Face-Removal Search

Heap lists are well known binary tree data structures in computer science [17,18]. The ordering of the tree is accomplished by requiring that the key characteristic of any father (root) be smaller than those of the two sons (branches). In the present application, the key characteristic is the spacing parameter assigned to each face in the front. Hence, at any time, the face making the next smallest element is positioned at the top of the list with the remaining faces arranged in an increasing order of their spacing parameters through branches and sub-branches. An example of a tree ordered in this manner is given in Figure 11(a), where a possible tree for the letters of the word 'EXAMPLE' is shown. The letters have been arranged according to their place in the alphabet, thus the letter A has the smallest key characteristic. We need to devise ways to efficiently add or delete entries without altering the orderly arrangement. In the example, letter E first enters the tree followed by X and A. Since A has a lower key than E, their positions are exchanged. Similarly, the other letters are added to the end of the tree one by one, and the ordering is re-established, if necessary, by comparing and replacing father and son pairs, starting from the bottom of the tree moving upwards.

Algorithmically, the addition of a new face to the front and re-arranging the list are performed as follows. We denote by IPSON and IPFATH the positions (ranks) of the son and the father, respectively, in the heap list LHEAP(1:NHEAP) where NHEAP is the total number of entries in the list. Accordingly, we denote by IFSON and IFFATH the identification numbers (face numbers) of the son and the father, respectively. Hence, IFSON=LHEAP(IPSON) and IFFATH =LHEAP(IPFATH). For the two sons of a father with position IPFATH, we have IPSON1=2×IPFATH and IPSON2=2×IPFATH+1. We also denote by RFACE(1:NFACE) the spacing parameter associated with a face where NFACE is the total number of faces in the front and is equal to NHEAP. When adding a new face IFNEW to the front, we

(i) increase NHEAP by one: NHEAP=NHEAP+1;

(ii) place IFNEW at the end of the list: LHEAP(NHEAP)=IFNEW;

(iii) set the position of IFNEW in the heap list to IPSON=NHEAP;

(iv) set the position of the father in the list to IPFATH=IPSON/2 (integer division);

(v) set the face numbers associated with the positions of the father and son to IFSON=LHEAP(IPSON) and IFFATH=LHEAP(IPFATH), respectively;

(vi) exchange father and son if RFACE(IFSON)<RFACE(IFFATH):
   - interchange the faces stored in LHEAP:
     LHEAP(IPFATH)=IFSON, LHEAP(IPSON)=IFFATH;
   - set IPSON=IPFATH;
   - go back to step (iv) to compare IFNEW with its new father if
     IPSON $\neq 1$ (top of the list);

(vii) stop.

With this procedure, the face with the smallest associated spacing parameter, LHEAP(1), is always positioned at the top of the list, ready for removal from the front.

When the face at the top is removed from the list, it is replaced with the one at the bottom of the tree, i.e., LHEAP(NHEAP). The ordering is then re-established by comparing and replacing father and son pairs, starting from the top moving downward. In the example shown in Figure 11(b), letter A at the top is removed, and E from the bottom of the tree takes its place. Letter E has also the lowest level among the remaining letters, thus is out next. Then L is moved to the top; however, since it has a higher level than one of its sons (E), they exchange their positions, and so forth. Finally, all letters are removed with the desired ordering.

Algorithmically, removing a face from the front is achieved by re-establishing the list as follows:

(i) take out the face at the top of the list: IFOUT=LHEAP(1);

(ii) place the face stored at the end of the heap list (IFEND) at the top and decrease NHEAP by one:
LHEAP(1)=LHEAP(NHEAP)=IFEND, NHEAP=NHEAP-1;

(iii) set the position of this face in the heap list to IPFATH=1;

(iv) set the positions of the two sons IPSON1 and IPSON2 in the list to
IPSON1=2×IPFATH and IPSON2=IPSON1+1;

(v) set the faces associated with the positions of father and sons to:
IFSON1=LHEAP(IPSON1),
IFSON2=LHEAP(IPSON2), and
IFFATH=LHEAP(IPFATH);

(vi) determine which son (if any) needs to be exchanged:
if RFACE(IFSON1) > RFACE(IFSON2) > RFACE(IFFATH),
set IPEXCH=0;
if RFACE(IFSON2) > RFACE(IFSON1) > RFACE(IFFATH),
set IPEXCH=0;
if RFACE(IFSON1) > RFACE(IFFATH) > RFACE(IFSON2),
set IPEXCH=IPSON2;

```
     if RFACE(IFSON2) > RFACE(IFFATH) > RFACE(IFSON1),
     set IPEXCH=IPSON1;
     if RFACE(IFFATH) > RFACE(IFSON1) > RFACE(IFSON2),
     set IPEXCH=IPSON2;
     if RFACE(IFFATH) > RFACE(IFSON2) > RFACE(IFSON1),
     set IPEXCH=IPSON1;
(vii) exchange father and son if IPEXCH ≠ 0:
     - interchange the faces stored in LHEAP:
       LHEAP(IPFATH)=IFSON1 and LHEAP(IPSON1)=IFFATH if
       IPEXCH=IPSON1, or
       LHEAP(IPFATH)=IFSON2 and LHEAP(IPSON2)=IFFATH if
       IPEXCH=IPSON2;
     - set IPFATH=IPEXCH;
     - go back to step (iv) to compare IFEND with its new sons if
       2 × IPFATH+1 ≤ NHEAP;
(viii) stop.
```

With this procedure, the face with the smallest associated spacing parameter, LHEAP(1), would again be positioned at the top of the list. It can be shown that both the insertion and deletion of a face into and from the heap list will take $O[log_2(NHEAP)]$ operations on the average.

2.5.2 Quad/Octrees for Point Search

An important and costly aspect of the unstructured grid generation is locating points which are arbitrarily scattered in the computational domain. To improve the efficiency, we use quadtrees and octrees as effective data structures for this search operation. Octrees and quadtrees are extensively used to accelerate the search for nearest neighbors in graphics algorithms [14], battle management [19], particle simulations [20], and grid generation [10]. Samet [21] has given an extensive survey on the subject. In Reference [10], the main role of octrees is to define the objects to be gridded. In this work, quadtrees and octrees are used for two- and three-dimensional grid generations, respectively, to provide an $O(logN)$-efficient search algorithm for arbitrary point distribution.

The idea is to divide the domain into regions: square quadrants for two-dimensional and cubic octants for three-dimensional domains . Each quadrant contains a maximum of four points, whereas there can be at most eight points in each octant. If a fifth point falls into a quad, the quad is divided into four subregions. The old points are relocated into their respective new quads based on their coordinates, and the new point is introduced to one of the new quads according to its position. If it happens that all five points fall again into the same quad, the subdivision process continues until an unfilled quad if found to house the new point. Similarly, an octant breaks into eight suboctants once a ninth point is introduced to it. The points are then relocated according to their coordinates as in the 2-D case. The process is illustrated in Figure 12 where a newly introduced point E falls into the quad IQ. As IQ already contains the four points A, B, C, and D, the quad is subdivided into four: NQUAD+1, NQUAD+2, NQUAD+3, and NQUAD+4. Points A, B, C, and D are relocated to their respective new quads, and point E is added to NQUAD+2.

The information required for each quad is stored in two arrays LQUAD(1:7, NQUAD) and RQUAD(1:4,NQUAD), where NQUAD is the total number of quads. For a typical quad IQ, we have the following information.

LQUAD(7 ,IQ) $= 0$:	the quad IQ is empty
> 0 :	number of points in IQ
< 0 :	IQ is full and already divided

LQUAD(6 ,IQ) : the parent quad from which IQ resulted

LQUAD(5 ,IQ)	: the position in the parent quad from which IQ resulted
LQUAD(1:4,IQ)	: the point numbers in IQ for LQUAD(7,IQ)>0
	: the quads into which IQ was subdivided for LQUAD(7,IQ)<0
RQUAD(1:4,IQ)	: minimum and maximum values of x and y coordinates for the quad IQ.

Figure 12 also shows the entries in the LQUAD array, as well as the associated tree-structure. A quadtree obtained for the point distribution of an unstructured grid around a simple two-dimensional configuration is shown in Figure 13. The data structure described for two-dimensional domains can easily be extended to 3-D cases with the information stored in two arrays: LOCTA(1:11,NOCTA) and ROCTA (1:8,NOCTA) where NOCTA is the total number of octants.

Once a quadtree (octree) is constructed for an arbitrary point distribution, 'close points' in the neighborhood of a specified point, such as the mid-point of a face, are found as follows. First, a search region (a square for 2-D, a cube for 3-D) around the point in question is defined. Then, going down the levels of the quadtree (octree), we disregard, at the highest possible level, those quads (octants) which completely lie outside the search area and collect into a list all points in the remaining quads (octants). The points which fall inside the search area are then selected from this list based on their coordinates. With a quadtree, it takes $O(log_4 N)$ operations to find all points inside a search region, whereas with an octree, the number of operations is $O(log_8 N)$.

2.5.3 Linked List for Adjacent Face/Element Search

In the process of advancing the front, the quality or 'correctness' of the grid is checked before a new element is introduced. The criterion for this test is that the new element must not cross any existing face of the front. Hence, the prior knowledge of existing faces in the vicinity of the face to be removed from the front is essential. This is accomplished by finding the adjacent faces to the already known close points. The data structure used for this operation is called 'linked list'. Linked lists are often used to relate data of different nature and different number of linked items. For unstructured grids, one can efficiently relate points to faces or elements using linked lists as the number of faces and elements connected to a point vary from point to point. A point-element link information is required for the background grid element search.

The information for constructing a linked list structure is stored in two arrays: LPOIN(1:NPOIN) and LFAPO(1:NENTR,NFAPO) where NPOIN is the total number of points, NFAPO is the maximum number of storage locations for storing adjacent faces or elements, and NENTR is a specified maximum number of entries (number of faces or elements) in each storage location. Then for each point IPOIN, we have

LPOIN(IPOIN)	:	the location IFAPO in the array LFAPO where the storage of the faces (elements) surrounding point IPOIN starts;
LFAPO(NENTR,IFAPO)	> 0 :	the number of faces (elements) surrounding IPOIN stored in location IFAPO;
	$= 0$:	the storage location IFAPO is empty;
	< 0 :	the location IFAPO is full and storage of more faces (elements) around IPOIN are continued in location JFAPO=|LFAPO(NENTR,IFAPO)| of the array LFAPO;
LFAPO(1:NENTR-1,IFAPO)	$= 0$:	a zero (empty) entry in the storage location

IFAPO;

> 0 : a face (element) number surrounding IPOIN.

In a two-dimensional grid, each point on the front is surrounded by two faces (line segments), thus NENTR is specified as 3. For 3-D cases, the number of faces (triangles) around a point on the front varies from point to point, and a value of 6 or 7 may be specified for NENTR. For element search, NENTR is typically 10.

An example of a linked list structure and the addition of a new face to the list are shown in Figure 14 where a NENTR of 3 has been specified. There are two faces F1 and F2, in this example, connected to the point IPOIN initially. The IPOINth value of the LPOIN array refers to the IFAPOth location of the LFAPO array where the information is stored. The third entry of this storage location indicates that there are two faces surrounding IPOIN, and the first and second entries give the faces F1 and F2, respectively. Suppose another face, F3, is subsequently connected to IPOIN. Then, the third entry of the storage location IFAPO becomes negative [-(NFAPO+1)], indicating that besides F1 and F2, there exists an additional face adjacent to IPOIN which is stored at location NFAPO+1.

2.5.4 Filtering Close Points and Faces

Among the close points found, there are three groups of points which are not suitable for forming elements and need to be filtered out of the list before further considerations. These are:

(a) points which are not on the front. These are referred to as 'inactive' points. A single array, LACTP(1:NPOIN), determines whether a point is active or inactive. If LACTP(IPOIN) is equal to one, the point IPOIN is active; if it is zero, IPOIN is inactive and must be taken out.

(b) Points in the 'wrong' side of the face to be removed from the front which would make elements with zero or negative volumes. The position vectors ($\overline{v}$) of these points, with respect to a node of the face to be removed, make zero or negative scalar products with the face outward unit normal ($\overline{v}_n$), Figure 15, i.e.,

$$\overline{v} \cdot \overline{v}_n \leq 0. \tag{2.11}$$

(c) Points which are in the 'wrong' side of the faces adjacent to the face to be removed from the front, Figure 16. These points would form elements which cross the adjacent faces. The non-positive scalar product argument, Eqn. (2.11), applies to the adjacent faces for filtering this group of points .

After filtering the points, we also take out those faces which are in the 'wrong' side of the adjacent faces to eliminate unnecessary face-crossing checks.

2.5.5 Ordering the Candidate Points

The process of decision making as to which point, if any, from the remaining close points be selected to form the next element has an important effect on the generation and the quality of the final grid. A poor judgement of putting a new point instead of selecting an existing well-positioned close point, for example, could result in a locally condensed and distorted element grouping. This may even cause a failure, later on, when an attempt is made to put new elements on an entangled front formed by such a grouping. While seemingly simple to the eye judgement, the process of intelligently choosing the right point has proven to be non-trivial to program, especially in three-dimensions. Such a choice should ideally guarantee the decency of not only the element being currently formed but the future neighboring elements as well.

In the present work, we attempt to order the remaining close points as well as the new 'best point' in a list from the most to the least desirable element-forming point. Then, starting from the top of the list, we check points for face-crossing and pick the first one encountered which passes the test. Now it remains to define a criterion which can be used to measure the degree of appropriateness of the points for comparison purpose and preparing the list. In proposing such a criterion, we should, at least, take into consideration the distance and skewness of a point in relation to the face to be removed as a measure of distortion for the element being formed. In a two-dimensional case, the angle between the two sides of an element (triangle) at the point in question (apex angle) is an appropriate criterion. As the distance between a point and a face becomes larger, or its position becomes more skewed with respect to the face, the corresponding apex angle gets smaller, Figure 17. Thus, we order points in the list from the largest to the smallest apex angle. This criterion has proven to work well for all two-dimensional triangular grids generated so far.

In a three-dimensional case, there are surfaces instead of lines for sides of an element. While only three angles affect the shape of a triangular element, the number of influencing angles are one order of magnitude higher in a tetrahedron as there are angles between adjacent edges, sides, and edges and sides. Hence, it becomes more complicated to come up with a single measure which can represent the overall shape of the element. Many different criteria have been examined. Some of these are: spherical excess, minimum spherical angle, sum of the angles between edges of the tetrahedron at the apex, minimum of the edge angles, minimum of the edge-side angles at the apex, etc. The first criterion has been extensively used; however, it does not seem to consistently give satisfactory results. While the rest of the criteria listed here are experimental, and some lack the required generality, the last one is promising and has shown good performance for the preliminary test cases.

van Phai [11] suggests that a point making the greatest spherical excess is the most favorable point for forming a tetrahedron. A spherical excess is defined as

$$\delta = A + B + C - \pi \tag{2.12}$$

where A, B, and C are the three angles of the spherical triangle ABC formed by penetration of tetrahedron 1-2-3-4 into a sphere having its center at point 4 and a radius of unit length (see Figure 18). Thus, points are ordered in the list from the largest to the smallest spherical excess. While theoretically plausible, this criterion was found inadequate to guarantee the best ordering of points all the time. Another criterion which has shown satisfactory results is referred to as the 'minimum apex angle'. We define an apex angle at the point in question as the angle between an edge of the tetrahedron and its normal projection onto the plane of the opposite face (see Figure 19), i.e.,

$$\phi_1 = \frac{\pi}{2} - cos^{-1}\left[\frac{(\overline{v}_{41} \cdot \overline{v}_n)}{|\overline{v}_{41}||\overline{v}_n|}\right] \tag{2.13}$$

where

$$\overline{v}_n = \overline{v}_{43} \times \overline{v}_{42}.$$

Since there are three apex angles, we select the smallest (most restrictive) one as our criterion, i.e.,

$$\phi_{cr} = min\,[\phi_1,\ \phi_2,\ \phi_3\ ]. \tag{2.14}$$

Points are then ordered in the list from the largest ϕ_{cr} to the smallest one. Although this criterion has shown promising results, more test cases are required to validate its accuracy. At present, the spherical excess criterion is being used in the grid generator.

2.5.6 Checking the Intersection of Faces

The most important ingredient of the advancing front generator is a reliable and fast algorithm for checking whether two faces intersect each other. We have found that even slight changes in this portion of the generator greatly influence the final mesh. As with so many other problems in computational geometry, checking whether two faces intersect each other seems trivial for the eye, but is complicated to code. The problem is shown in Figure 20. We base our checking algorithm on the following observation: two triangular faces do not intersect if no side of either face intersects the other face. The idea then is to build all possible side-face combinations between any two faces and check them in turn. If no intersection is found, then the faces do not cross. With the notation defined in Figure 21, the intersection point is found as

$$\overline{x}_f + \alpha^1 \overline{g}_1 + \alpha^2 \overline{g}_2 = \overline{x}_s + \alpha^3 \overline{g}_3 \quad , \tag{2.15}$$

where we have used the $\overline{g}_i$ vectors as a covariant basis. Using the contravariant basis $\underline{g}^i$ defined by

$$\underline{g}^i \cdot \overline{g}_j = \delta^i_j \quad , \tag{2.16}$$

where δ^i_j denotes the Kronecker-delta, we obtain the α^i as

$$\begin{aligned} \alpha^1 &= (\overline{x}_s - \overline{x}_f) \cdot \underline{g}^1 \quad , \\ \alpha^2 &= (\overline{x}_s - \overline{x}_f) \cdot \underline{g}^2 \quad , \\ \alpha^3 &= (\overline{x}_f - \overline{x}_s) \cdot \underline{g}^3 \quad . \end{aligned} \tag{2.17}$$

Since we are only interested in a triangular surface for the $\overline{g}_1, \overline{g}_2$ - plane, we define another quantity similar to the third shape function for a linear triangle as given in Section 2.5.7, i.e.,

$$\alpha^4 = 1 - \alpha^1 - \alpha^2 \quad . \tag{2.18}$$

Then, in order for a side not to cross a face, at least one of the α's has to satisfy

$$t > max(-\alpha^i, \alpha^i - 1) \quad , i = 1, 4 \quad , \tag{2.19}$$

where t is a predefined tolerance. By projecting the $\overline{g}_i$ onto their respective unit contravariant vectors, we can obtain the actual distance between a face and a side. The criterion given by Eqn.(2.19) would then be replaced by (see Figure 22):

$$d > \frac{1}{|\underline{g}^i|} max(-\alpha^i, \alpha^i - 1) \quad , i = 1,4 \quad . \tag{2.20}$$

The first form (Eqn.(2.19)) produces acceptable grids. If the face and the side have points in common, then the α's will all be either 1 or 0. If eqns. (2.19) and (2.20) are not satisfied, we need to make special provision for such cases. For each two faces, six side-face combinations are possible. Considering that on average about 40 close faces need to be checked, this way of checking the crossing of faces is very CPU-intensive. When it was first implemented, this portion of the grid generation code took more than 80% of the CPU time required. In order to reduce the work load, a three-layered approach was subsequently adopted:

(a) *Min/Max search*– The idea here is to disregard all face-face combinations where the distance between faces exceeds some prescribed minimum distance. This can be accomplished by checking the maximum and minimum value for the coordinates of each face. Faces can not possibly cross each other if at least, for one of the dimensions $i = 1, 2, 3$, they satisfy one of the following inequalities

$$max_{face1}\left(x_A^i, x_B^i, x_C^i\right) < min_{face2}\left(x_A^i, x_B^i, x_C^i\right) - d \quad , \tag{2.21a}$$

$$min_{face1}\left(x_A^i, x_B^i, x_C^i\right) > max_{face2}\left(x_A^i, x_B^i, x_C^i\right) + d \quad , \tag{2.21b}$$

where A, B, and C denote the corner points of each face, and d is a tolerance based on the local spacing parameter.

(b) *Local element coordinates*– The purpose of checking for face-crossings is to determine whether the newly formed tetrahedron breaks already given faces. The idea is to extend the previous Min/Max criterion with shape functions of the new tetrahedron. If all the points of a given face have shape-function values N^i that lie outside the interval $[-t,\ 1 + t]$, where t is a tolerance, then the tetrahedron cannot possibly cross the face.

(c) *In − depth analysis of side − face combinations*− All the remaining faces, after the filtering processes of steps (a) and (b), are thoroughly analyzed using the side-face combinations as explained above.

Each of these three filters requires about an order of magnitude more CPU-time than the preceding one. When implemented in this way, the face-crossing check requires only 25% of the total grid generation time. When operating on a vector machine, we perform loops, in vector modes, over all the possible combinations, building the $\overline{g}_i, \underline{g}^i, \alpha^i$, etc. Although the vector lengths are rather short, the chaining that results from the lengthy mathematical operations yields acceptable megaflop rates.

2.5.7 Interpolating Information from the Background Grid

After a suitable point is selected to form the next element, the heap, quad/octree, and linked lists are re-established by adding the new point, face(s), and element and removing the face(s) no longer on the new front. It should be noted that, each time, one or more new faces can be generated, depending on the local front configuration and depending on whether a new or an existing point is selected. Similarly, one or more faces may be removed from the front. For the new face or faces, local grid generation characteristics such as spacing parameters are then determined. This information is obtained by a linear interpolation of the data stored at the four nodes of the background grid element which encloses the face.

Interpolating information from one unstructured grid onto another has been an outstanding problem for quite some time. The process involves two steps: (a) finding the background grid element which surrounds the center of the face in question, and (b) interpolation from the nodes of the element. Two methods have been used, in this work, to find the enclosing background grid element. In the first method, we

(i) construct a quad/octree for the background grid nodes;

(ii) construct a node-element linked list for the background grid;

(iii) find the closest nodes in the background grid to the center-point of the face in question using the quad/octree data structure;

(iv) find the background grid elements which share the nodes found in step (iii) using the linked list structure;

(v) find, from the list of elements, the element into which the point to be interpolated falls;

(vi) enlarge the region of close elements by one layer of elements using the linked list if no enclosing element is found; and try again.

With this method, the overall procedure requires $O(NlogN)$ operations. A disadvantage of this method is that for badly deformed background grid elements, the closest points found in the background grid may not belong to the element into which the point to be interpolated falls. As a result, many elements may have to be tested by enlarging the region of close elements. In the second method, the search for the background grid element is considerably simplified by simply remembering the element which has enclosed the parent face on the front from which the present face has emanated. Thus, we

(i) store in an array, LBEFA(IFACE), the background grid element into which the mid-point of the face IFACE falls;

(ii) use LBEFA(IFACE), as the first try, for the face(s) originated from the removed parent face IFACE;

(iii) try the neighboring elements if LBEFA(IFACE) does not enclose the face in question.

The number of operations is significantly reduced by eliminating the need for finding the closest nodes in the background grid and the linked elements. A linked list is only needed to find the neighboring elements in step (iii) if LBEFA(IFACE) fails to surround the face.

To check if a point is inside a triangular or tetrahedral element, we use local element coordinates or shape functions. Considering a point P in relation to a triangular element 1-2-3, as shown in Figure 23, shape functions N^1, N^2, and N^3 are defined as

$$\begin{aligned} N^1 &= \frac{A_1}{A} \\ N^2 &= \frac{A_2}{A} \\ N^3 &= 1 - N^1 - N^2 \end{aligned} \tag{2.22}$$

where A_1 is the area of triangle P-2-3, A_2 is the area of triangle P-3-1, and A is the total area of the element. Point P is outside the element if

$$\begin{aligned} max\left(N^1,\ N^2,\ N^3\right) &> 1, \quad or \\ min\left(N^1,\ N^2,\ N^3\right) &< 0. \end{aligned} \tag{2.23}$$

Once the element surrounding the mid-point P of the face is found, information R at P is interpolated using the shape functions, i.e.,

$$R_P = \sum_{i=1}^{3} N^i R_i \tag{2.24}$$

where R_i is the information at node i of the element. For a tetrahedral element, there are four shape functions which are similarly defined in terms of volumes instead of areas, Figure 24.

2.6 GRID POST-PROCESSING

After a grid is generated, further operations may optionally be performed to refine and/or smooth the grid. These operations are briefly discussed below.

2.6.1 H-Refinement

The grid is refined by dividing each original element into a number of smaller elements. This uniform refinement is referred to as 'h-refinement'. In a two-dimensional grid made of triangular elements, the mid-points of the sides in each element are connected to divide the original triangle into four smaller elements as shown in Figure 25(a). Using the formula

$$N_e = 2N_p - N_b - 2 \tag{2.25}$$

for a two-dimensional triangular grid and the fact that the number of boundary points in the refined grid is twice as large as that in the coarse grid, one can determine the total number of grid points after h-refinement in terms of the coarse grid information as

$$(N_p)_{fine} = 4(N_p)_{coarse} - (N_b)_{coarse} - 3 \tag{2.26}$$

where N_e, N_p, and N_b are the number of elements, total number of points, and number of boundary points, respectively.

In a three-dimensional grid, each tetrahedron is divided into eight smaller tetrahedra by connecting the mid-points of the six edges as shown in Figure 25.b. The element faces are again divided into four triangles, similar to that in the 2-D case, by this partitioning. The total number of points in the fine grid becomes approximately eight times as large as that of the coarse grid, and the boundary points increase by a factor of four after each refinement.

Since one of the objectives of mesh refinement is to enrich the surface representation of bodies with curved boundaries, the positions of the newly inserted mid-points on the boundaries must be corrected to conform with the surface of the original configuration. This is automatically done by taking the line and surface definitions, described before, into consideration.

2.6.2 Mesh Smoothing

The generated grid, either before or after the refinement, can optionally be smoothed by three main operations: moving grid points around, removing 'small' elements, and removing 'distorted' elements. The mesh is smoothed by moving points in a spring system analogy. The main problem encountered here is that some distorted elements, already in the grid, may further degenerate and create negative volumes. This may also happen after a refinement when the boundary points, belonging to a distorted element, are corrected on a convex surface. Negative elements are considered here as 'small' and are removed from the grid as explained below.

The removal of 'small' elements is of particular importance for transient problems where the allowable time-step depends directly on the element size. An element is considered as 'small' if the volume and a characteristic length ratio of the element are smaller than some specified tolerances. For each of the elements considered as 'small', the smallest side is identified, and the nodes belonging to this side are collapsed into one point. The elements sharing the removed side are then taken out, and the remaining points and elements are renumbered. This is shown in Figure 26 for a two- dimensional case. The same concept is applied to three-dimensional problems.

An element is considered as 'distorted' if the ratio of the lengths of its largest and smallest side exceeds a certain tolerance. A 'distorted' element is removed similarly to that explained above. While all of the post-processing operations function properly for two-dimensional grids, the processes may still create or leave some 'small', 'distorted', or 'negative' elements in three-dimensional meshes. Better ways of smoothing 3-D grids need to be explored.

3. VGRID3D PRIMER

VGRID3D is a program for generation of 3-D unstructured tetrahedral grids given the domain definition. Grids are generated starting from the domain boundaries marching towards the interior of the computational domain using the so called advancing front technique. In this technique, the grid points are introduced while the domain of interest is filled with tetrahedra. The configuration of interest is first defined in terms of surface patches. Then, these patches are triangulated one at a time. This forms the initial front. Next, elements (tetrahedra) are generated in the 3-D field by successively removing faces (triangles) from the current front and adding new or selecting existing points (advancing the front) until the whole region is filled with tetrahedra.

Details of the theory behind the advancing front grid generator, data structures used for efficient generation of grids and other algorithmic details are given in Chapter 2. This primer mainly focuses on the preparation of the input file for the grid generator, running the VGRID3D and some explanation of the output obtained.

3.1 SETTING-UP AN INPUT FILE

VGRID3D requires an input file that defines the domain to be gridded (in the form of surface patches) and the desired spatial distribution of grid points (the background grid). The configuration of interest is defined using a hierarchical data structure made of points, lines and surface patches. Lines are obtained by joining points, and surface patches by joining lines. The free format is used to input numbers. The input file consists of the following sections.

3.1.1 Title Line

VGRID3D expects a line of text at the beginning of the input file. This line can be up to 80 characters long and is included as a user convenience to identify the configuration to be gridded. A text line is also inserted at the beginning of each new section of the input data.

3.1.2 Domain Definition

This section defines the domain to be gridded. After a text line, the number of dimensions (NDIMN), the total number of points (NPOIN), lines (NLINE) and surface patches (NSURF), defining the computational geometry including the outer boundaries, are specified. Points, lines, and surface patches are described as follows.

Points Description :

The points defining the computational domain are identified by their numbers and X/Y/Z coordinates. VGRID3D expects a right-handed coordinate system for the domain definition, usually with +X streamwise downstream,+Y along the starboard side and +Z vertical upwards. However, any right-handed system can be used. It should be noted that a point can only be defined once, i.e., no two points can have the same coordinates, even if a point is common to two or more lines. However, before beginning the grid generation process, VGRID3D checks for the user-overlooked mistakes and alerts the user if any duplicate points are found.

Line Description :

The lines connecting the points are defined next. For each line, the line number, the type of the line and the points belonging to this line are specified. The first integer is the line number while the second integer defines the line type (NTYLI). Currently, VGRID3D accepts the following three line types:

a) a straight line defined by two points $\Rightarrow$ NTYLI = 1,

b) a parabola defined by three points $\Rightarrow$ NTYLI = 2, and

c) a cubic spline defined by four or more points $\Rightarrow$ NTYLI = 3.

For each line, this information is followed by a list of points that belong to this line segment. The convention used is that a line is considered positive going from the first point in the list to the last.

Surface Description :

The configuration of interest is described using surface patches, which in turn are made up of two or more lines defined above. For each surface patch, the surface number, its type (NTYSU), the number of lines in this patch, the lines belonging to this surface, the boundary condition to be imposed on all the points generated on this surface and the parameters for the generation of a boundary layer grid are specified.

The first integer is the surface number. Since VGRID3D triangulates each surface independently, the surface numbers may be given in any order desired. The second integer defines the surface type. Currently the surface types implemented, as described in Chapter 2, are :

a) a planar surface with two or more lines, all lying in the same plane
$\Rightarrow$ NTYSU = 1,

b) a triangular isoparametric parabolic surface defined by three parabolic lines
$\Rightarrow$ NTYSU = 2,

c) a rectangular isoparametric surface defined by four parabolic lines
$\Rightarrow$ NTYSU = 3,

d) a triangular Barnhill-Gregory-Nielson patch defined by three arbitrary lines
$\Rightarrow$ NTYSU = 4, and

e) a bilinear transfinite Coon's patch defined by four arbitrary lines
$\Rightarrow$ NTYSU =5.

The surface types d) and e) are the most common in use as any combination of the line types defined above can be used in them.

The third integer is the number of lines in the surface patch. Except for the planar surface (NTYSU =1), this number is fixed as described above. Next, the lines that belong to this surface are specified. These lines are listed in such a way that when they are traversed in the order and the direction listed, the region to be triangulated always lies to the left. In doing so, if a line segment is traversed in a direction opposite to the way it was originally defined, a negative sign is placed in front of the line number in this list.

VGRID3D assigns boundary conditions to all the points generated on a surface patch. For each surface, this information is specified by three integer numbers after the list of line numbers for this patch. Currently, the numbering system used is specific to VFLOW3D family of flow solvers. If a user is not using these codes, a (3 0 0) may be specified for all the surface patches. Otherwise, the three integers denote the boundary condition type (NTYBC), the 'characteristic inflow set number' and the body number to which the current surface belongs (NBODY). The boundary conditions (NTYBC) accepted by the VFLOW3D family of flow codes are:

-1: $\Delta U = 0$, no change in flow variables (supersonic inflow);

0: $\Delta V = 0$, no change in velocities (stagnation line, no-slip for Navier-Stokes);

1: boundary condition types 2 and 3 combined;

2: tangential velocity along the surface;

3: velocity normal to the plane = 0;

4: variables chosen according to a 1-D characteristic analysis performed at the boundary in question. This analysis determines how many characteristics are coming in and going out. The final ΔU values of the variables depend on this analysis and the values given in the 'characteristic inflow set';

5: adiabatic wall boundary condition, used only for Navier-Stokes calculations;

6: 'free' boundary condition, variable computed by flow code (supersonic outflow).

The second of the three integers, specifying the boundary conditions, is non-zero only for NTYBC = 4 and denotes a set of information from which the characteristics are determined at the surface in question. Several such sets can be present in a single run. These sets consist of density, the three components of velocities and pressure. For example, an aircraft configuration with simulated flow through engines may have a set corresponding to the engine inlet plane and another for the engine exhaust. In addition, if the entire configuration is to be analyzed for a subsonic or transonic flow regime, a separate set may also be specified for the farfield outflow characteristic boundary condition. These sets are numbered sequentially, starting with 1, and are given as the second integer in the list. For non-characteristic boundary conditions, this number is specified as 0. It is noted that the grid generator needs only the set number, actual values of the variables are required only for the flow solution process (see Chapter 5).

The third integer is used to identify the body to which the surface in question belongs and is meaningful only for the cases that allow boundary movement. For such a case, all surfaces comprising a moving body (component) are grouped, and the same body number is assigned to all of them. The default value is 0 (fixed surface).

VGRID3D can, optionally, generate grids in the boundary layer region near solid surfaces. Two requirements are imposed on the grids in such a region: 1) the presence of high shear requires a much smaller grid spacing in a direction normal to the wall compared to that along the wall, and 2) the points in the direction perpendicular to the surface need be aligned for some kind of turbulence modeling. Grids satisfying both of these conditions are difficult, if not impossible, to generate using the current method. For this reason, VGRID3D uses an approach that generates semi-structured grids in the boundary layer regions. Accordingly, the surface under consideration is first triangulated, and then prisms are constructed by duplicating the triangles in a direction normal to the surface in the computational domain. Each of these prisms are then subdivided into three tetrahedra, resulting in a semi-structured tetrahedral mesh. The number and location of these duplicated triangles are based upon some parameters given for the desired normal point distribution and the boundary layer thickness.

The parameters for generating boundary layer grids are specified by two integers and a real number for each surface. These are given following the three integer parameters for the boundary condition. The first of the boundary layer parameters is an integer, NPTBL, specifying the number of points desired in the boundary layer, while the second integer, NTYBL, determines the kind of point distribution across the boundary layer. Currently, there are three options available for point distribution:

a) Geometric $\Rightarrow$ NTYBL = 1 (most commonly used),

b) Power law $\Rightarrow$ NTYBL = 2, and

c) Exponential $\Rightarrow$ NTYBL = 3.

The last of the boundary layer parameters is a real number specifying the desired increment factor between points across the boundary layer. If a surface is not a wetted surface (no boundary layer grid desired), then the three boundary layer parameters are specified as (0, 0, 0). It is noted here that the method described above for the generation of the boundary layer grids may not be the most satisfactory technique, and better methods, in terms of robustness and grid quality, may need to be explored.

3.1.3 Special Lines

This sub-section gives a list of lines along which special boundary conditions need to be specified. These boundary conditions are different from those of the surfaces to which the special lines belong. For example, a line in the plane of symmetry where flow tangency boundary condition need to be imposed or a line at the trailing edge of a wing with a 'free' boundary condition is handled differently. Accordingly, NLITA represents the number of lines with tangential boundary condition which is followed by a list of line numbers in LLITA. In a similar fashion, NLIFR and LLIFR are the total number of lines and the line numbers, respectively, for which a 'free' boundary condition is specified. If there are no special lines, both NLITA and NLIFR are given as 0. Obviously, for such a case, NLITA and NLIFR are not to be given.

3.1.4 Background Grid

This section defines the background grid which is used for determining the spatial distribution of the 3-D grid points. After a text line, indicating the beginning of the section, the total number of tetrahedra and the number of points in the background grid are specified as NELEB and NPOIB, respectively. This is followed by the connectivity array (the interdependency matrix) INTMAB made of NELEB number of lines, each consisting of five integer numbers: the background grid element number and the four node numbers associated with this element. The convention used is that the fourth node is on the same side as the normal vector to the base triangle formed by the first three nodes. The normal vector is found using the right-hand rule. Before beginning the grid generation process, VGRID3D checks the background grid tetrahedra for any inconsistency resulting in negative volumes and corrects the connectivity by exchanging the second and the third nodes, thereby changing the direction of the normal.

Next, the coordinates of the points in the background grid are given. This is given by an integer specifying the point number and its X/Y/Z coordinates. It should be noted that the coordinate system used for the background grid must be consistent with that for the domain definition.

The next sub-section specifies the parameters which control the spatial grid point distribution (UNKNB) at each node of the background grid. For any location in space, VGRID3D identifies the background grid element in which this location falls and linearly interpolates the required values from the four nodes of this element. These control parameters are given as two stretching ratios S_1 and S_2 associated with two orthogonal directions $\overline{a}_1$ and $\overline{a}_2$, respectively, and the spacing parameter δ. The generated element will then have typical dimensions δS_1 in the direction parallel to $\overline{a}_1$, δS_2 in the direction parallel to $\overline{a}_2$ and δ perpendicular to both $\overline{a}_1$ and $\overline{a}_2$ (Figure 27). The two orthogonal directions, $\overline{a}_1$ and $\overline{a}_2$, are given by their direction cosines with respect to the X/Y/Z coordinate directions. For each background grid element, the element number is followed by thirteen real numbers. The first nine real numbers are the direction cosines for the two orthogonal directions, $\overline{a}_1$ and $\overline{a}_2$, and a vector normal to them ($\overline{a}_3$). The tenth number is the spacing parameter(δ), denoting the element size desired, followed by the stretching ratios S_1 and S_2 and the boundary layer thickness. For no stretching case, the vectors $\overline{a}_1$, $\overline{a}_2$ and $\overline{a}_3$ are aligned with the X/Y/Z coordinate directions, respectively, resulting in UNKNB(1 thru 9) to be (1.0,0.0,0.0),(0.0,1.0,0.0) and (0.0,0.0,1.0). Similarly, stretching ratios are specified as (1.0,1.0) for no stretching. If no boundary layer is desired, the last value for each UNKNB is to be given as 0.0.

3.1.5 Mirroring and Refinement:

If the configuration to be gridded has a plane of symmetry, then only half of the configuration needs to be gridded and the final 3-D grid can be mirrored. Mirroring across a plane is accomplished by specifying a position vector ($\overline{x}_0$) to a point in the plane, and the direction cosines of a vector normal to the plane at this point ($\overline{x}_n$). For example, mirroring of a 3-D grid across the X-Y plane is accomplished by specifying NMIRR to be 1, indicating that a mirroring is desired, $\overline{x}_0$ to be (0.,0.,0.) and $\overline{x}_n$ as (0.,0.,1.). An NMIRR of zero means no mirroring.

The last section in the input file contains information regarding h-refinement. After the 3-D grid is generated, a finer grid may be obtained by dividing each tetrahedron into eight smaller ones, as explained in Chapter 2. The number of times the h-refinement is to be done is specified by NHREF. For no h-refinement, NHREF is set to 0.

It is, sometimes, easier to generate a coarse grid first and then do one or more h-refinements in order to get the final fine grid. The next parameter specified, on the same line as NHREF, is a factor (FACT$\geq$1.0) by which all the grid spacing parameters (δ) are multiplied before the grid generation process begins, thus generating a coarse grid. A value of 2.0 for FACT produces a grid which is eight times coarser than that generated with a FACT of 1.0.

This concludes the input file preparation for the grid generator. A sample input file for generation of a 3-D grid around an ONERA M6 wing is given in the Appendix.

3.2 RUNNING VGRID3D

VGRID3D is written in FORTRAN 77 and, as such, will run on any computer with an appropriate compiler. The complete FORTRAN source code is divided into five modules (—.f files) according to their functionality. These modules are:

vgrid3d.f$\Rightarrow$ contains the main driver routine for the declaration of all dimensioned arrays and the routines for the input/output,

gen3dsubs.f $\Rightarrow$ contains some general purpose routines for the generation of surface as well as field grids and routines for mirroring, interpolation, transformation, etc.

boulay3subs.f $\Rightarrow$ contains routines for the generation of boundary layer grids,

splitsubs.f $\Rightarrow$ contains routines for the grid post-processing. This includes h-refinement and smoothing and routines for the removal of small and distorted elements,

apmath.f $\Rightarrow$ contains routines for general purpose mathematical functions.

For running the program, the input data is to be provided in a file named '**fort.50**'. The run-time messages are written to the standard output file for the machine being used. The final 3-D grid is written to a file named '**fort.8**' which is created during the run.

3.3 OUTPUT FROM VGRID3D

The output file (fort.8) contains the 3-D tetrahedral mesh for the configuration of interest. The first line contains an integer number showing the number of title-lines, followed by that many text lines (up to 80 characters long) describing the configuration. The title of the run is followed by several integer numbers as explained below:

NDIMN is the number of dimensions;

NTYPE is the equation set the flow solver will solve. This parameter is specific to

VFLOW3D family of codes and may be ignored by others. Currently, VFLOW3D family is capable of solving three sets of equations:

NTYPE = 1, transport equation;

NTYPE = 2, Burger's equation; and

NTYPE = 3, Euler equations;

NELEM is the total number of elements (tetrahedra) generated;

NPOIN is the total number of points; and

NBOUN is the total number of boundary points for the configuration.

In the list of points (see below), the NBOUN boundary points are given first followed by the field points.

The next section describes the connectivity matrix (INTMAT) that lists nodal points belonging to each element (tetrahedron). For each element, the element number and its four nodes are given followed by twelve zeros. This array of zeros would be replaced, during the adaptive refinement process, with information such as father-son element pairs, the number and type of refinement/derefinement, etc.

The coordinates of all points are listed next after a text line. For each point, the point number and its X/Y/Z coordinates are given. As noted earlier, VGRID3D puts NBOUN boundary points at the top of the list.

The final section contains information regarding boundary conditions to be applied by the flow solver. The information in this section is specific to the VFLOW3D family of codes and may be redundant for others. The boundary condition information is given by six integers [BCONI (1 thru 6)] and three real numbers [BCONR(1 thru 3)]. The integers are:

BCONI(1) = the boundary point number,

BCONI(2) = boundary condition type (NTYBC) assigned to this point, BCONI(3) = a number which is non-zero only for characteristic boundary condition (NTYBC = 4). If BCONI(2)=4, then this number refers to the 'characteristic inflow set number', as specified in section 3.1.

BCONI(4) = boundary condition specifying movement (degrees of freedom) of the boundary point during adaptive refinement and h-refinement processes.

$= 0 \Rightarrow$ 'fixed' point (e.g. end points of a line),

$= 1 \Rightarrow$ free to move along a line, and

$= 2 \Rightarrow$ free to move on a surface.

BCONI(5) = line/surface patch number to which the point belongs. A boundary point is free to move only along the line/surface from which it originated.

$= 0 \Rightarrow$ if BCONI(4) = 0,

= line number $\Rightarrow$ if BCONI(4) = 1, and

= surface number $\Rightarrow$ if BCONI(4) = 2.

BCONI(6) = body number to which this point belongs.

The next three real numbers, for each point, correspond to the negative of the components of a unit normal vector at the boundary point. VGRID3D calculates the normal to a point facing away from the computational domain, hence the negative values. For points belonging to the lines which are common to two or more surfaces (e.g., wing-fuselage junction and the leading edge point on the tip of a wing), the normal is calculated as an average of values for all the surfaces to which the point belongs.

VGRID3D specifies an additional set of three real numbers for each boundary point. These numbers represent the X/Y/Z coordinates for the boundary point. These values are specified here just for an instant visual 'check' as to the location of a particular point and is included as a user convenience.

4. POST-PROCESSING

A rapid and interactive graphics program to manipulate and plot computational results for unstructured grids is an essential tool for efficiently analyzing the enormous data generated by today's supercomputers. For unstructured grids, this is even more important since the lack of directionality makes viewing the 3-D data impossible without a graphics program. During this study such a program, called VPLOT3D has been developed. VPLOT3D is a post-processing graphics program for vizualization of fluid dynamic data on unstructured tetrahedral meshes. It is interactive, menu-driven and user-friendly and has been written for the Silicon Graphics, Inc. IRIS 4D series workstations. Pop-up menus are used to guide the user through the display options with a click of a button.

VPLOT3D has the ability to display primitive flow variables: density, the velocity components and energy as well as derived quantities like Mach number, temperature, absolute velocity and entropy. These quantities can be displayed on the boundary surfaces, user defined arbitrarily oriented planes and iso-surfaces.

4.1 DETAILED FEATURES OF VPLOT3D:

VPLOT3D has a variety of options for displaying field or surface data. A detailed list of these options is given here. Mathematical formulation for some of these options may be found in [22].

Global Options:

- Boundary condition check:
  This option allows the user to plot boundary points colored according to the boundary condition type, thus enabling a visual check of the boundary conditions before and during the flow solution process. At present, the boundary condition types used are for VFLOW3D family of finite-element flow codes, however any user specified boundary conditions can be implemented easily. The currently available boundary conditions type include characteristic, solid wall, free, symmetry

and fixed boundary conditions. Details about the boundary conditions specific to VFLOW3D can be found in Chapter 3.

- Surfaces in 3-D space: Vizualization of data on
  - entire boundary surface or part of it
  - an arbitrary user defined plane
  - an iso-surface; a surface with a constant value of certain quantity
- 3-D field
- Grid element quality check

<u>Local Options for Surfaces:</u>

- Filter the domain to be visualized
  - within or out of a numerically or graphically specified spatial box
  - according to boundary conditions
- Surface discretization
  - wireframe with and without hidden line removal and depth cueing
  - simple and Gouraud shaded pixel plot
- Contours of scalar quantities
  - contours with and without hidden line removal with user specified range and increment
  - Gouraud shaded pixel plot with and without backface removal
  - opaque object or three levels of transparency to choose from
- Velocity vector plots
  - control of the size of vector length and arrow-head
  - colored according to a local quantity or single color
- Oilflow plots
  - graphical and numerical control of release locations
  - color and line type option

<u>Local Options for 3-D Fields:</u>

- Filter spatial domain
  - numerical and graphical input of a box with options for displaying the mesh within or out of the box
- Field discretization
  - wireframe with and without hidden line removal
  - wireframe with and without depth cueing
- Velocity vector plots
  - control of the size of vector length and arrow-head
  - colored according to a local quantity or single color
- Particle path plots
  - graphical and numerical control of release locations
  - color and line type options
  - particle paths from all points within a box
  - horizontal or verticle rake
  - plot of the particle paths in either forward or reverse direction
  - lines or ribbons option to show particle paths

Grid Element Quality Check:

- find and plot 'small' elements.
- find and plot 'distorted' elements based on sides or normal ratio criteria (see Chapter 2 for definition of 'small' and 'distorted' elements)
- plot of user specified elements

Object Display Options:

- Movement of objects on the screen
  - mouse controlled translations, rotations and zooming
- Queueing of object display
  - number and order of objects to be displayed
  - return to displaying of object from anywhere in the program

- Screendump
  - in raster metafile format
  - in Post Script raster image format
  - options for selecting plot size: IRIS 3030, IRIS 4D or user defined
- Delete objects
- Mirroring
  - mirror across a user defined plane or across a coordinate axis

Special Features

The program utilizes full graphics capabilities of the IRIS architecture in efficiently interpolating and displaying the data in a variety of forms. Some special features of the VPLOT3D are described below.

1. Handling of Objects: All the commands needed to perform a plotting operation are stored in 'Display List Objects'. This form of storage is very useful if the user desires to quickly display several objects at the same time. Since the program has a capability of plotting multiple data files, this option can also be used for a side-by-side comparison of data.
2. Transparency: For the older workstation models with no alpha-planes, a simple masking technique is implemented to get the effect of transparency. A stipple pattern is selected which permits a drawing action to affect only a selected set of pixels so that the image 'behind' shows through.
3. Z-Buffering: Z-buffering is a simple technique of hidden line and surface removal on a pixel-by-pixel basis. The screen transformed z-location of each pixel is compared with the currently displayed z-location to determined if the pixel need to be overdrawn.
4. Backface Culling: This is a technique to display only the viewable side of an object drawn on the screen.

5. X-Y Plot Capability: The program has the capability to plot the cross-sections of objects cut by a specified plane and/or variation of a plotted quantity along this cross-section. The user specifies a line by defining two points and values of the pixel colors along this line are then looked up in a colormap and plotted against the distance from the first point.

VPLOT3D-Some Special Features

VPLOT3D:

* is written using a dynamic memory allocation feature which automatically assigns memory for arrays as needed.
* reads input in either free-formatted or IRIS binary
* has the option of reading multiple data files for ease of comparison
* produces a log file containing all the commands of the current session so that the user may interrupt a session and restart later without typing all the commands again.

5. FLOW SOLUTION

In this chapter, a technique for the solution of three-dimensional, compressible fluid flow governed by Euler equations is described. The technique is based on a two-step, explicit, second order, Taylor-Galerkin finite element method (FEM). In order to handle moving bodies, the Euler equations are cast into an Arbitrary Lagrangian-Eulerian (ALE) frame of reference. The resulting procedure is implemented in a computer code called VFLOW3D. A variety of flow cases solved using this method have shown satisfactory results.

Solution methods based upon high resolution schemes [23-28] give sharper definition of flow discontinuities and are supposedly more robust. For multidimensional flows, these methods are generally implemented by using operator splitting and applying one-dimensional concepts in each coördinate direction separately. A finite element practitioner, however, finds difficulty in operating in this manner as the use of unstructured grids with no inherent directionality makes the approach complicated. A high resolution method which can be used directly on unstructured grids is Zalesak's [29] multidimensional generalization of the 1-D flux-corrected transport (FCT) ideas of Boris and Book [30-32]. This method employs a high-order scheme together with a low-order scheme and attempts to combine these in such a way that the high-order solution is used in smooth regions of the flow whereas the low-order solution is favored near discontinuities. The low-order scheme should be selected such that it produces monotonic results for the problem to be solved. Erlebacher [33] and Parrott and Christie [34] showed how FCT ideas could be interpreted in the finite element context for a single governing equation and implemented the ideas on triangular meshes. In the present work, the technique has been extended to deal with the solution of a system of equations and the incorporation of the consistent mass which yields high temporal accuracy.

5.1 THE EQUATIONS OF COMPRESSIBLE FLOW

The governing equations of inviscid compressible flow can be written in the conservation form

$$\frac{\partial U}{\partial t} + \frac{\partial F_j^a}{\partial x_j} = S \tag{5.1}$$

where the summation convention has been employed and

$$U = \begin{Bmatrix} \rho \\ \rho u_i \\ \rho e \end{Bmatrix} \quad , \quad F_j^a = \begin{Bmatrix} \rho(u_j - w_j) \\ \rho u_i(u_j - w_j) + p\delta_{ij} \\ \rho(u_j - w_j)e + u_j p \end{Bmatrix} \quad , \quad S = -\frac{\partial w_j}{\partial x_j} \begin{Bmatrix} \rho \\ \rho u_i \\ \rho e \end{Bmatrix} . \tag{5.2}$$

Here, ρ, p, and e denote the density, pressure, and specific total energy of the fluid, respectively, and u_i and w_i are the components of the fluid and grid velocities in the direction x_i of a Cartesian coordinate system. The equation set is completed by the addition of the equation of state

$$p = (\gamma - 1)\rho[e - \frac{1}{2}u_j u_j] \tag{5.3}$$

which is valid for a perfect gas, where γ is the ratio of the specific heats. In the case of no grid movement (w_i=0), the usual conservation-law form of the Euler equations are recovered.

5.2 FLOW SOLUTION METHOD

As stated earlier, high resolution, monotonicity preserving schemes must be developed in order to be able to simulate the strong nonlinear discontinuities present in the flows under consideration. Although the pertinent literature abounds with high resolution schemes [23-28], only Zalesak's generalization of the 1-D FCT schemes of Boris and Book can be considered a truly multidimensional high resolution scheme. The use of unstructured grids requires such truly multidimensional schemes, as the lack of lines or planes in the mesh makes the use of operator splitting difficult. In the present method, the multidimensional FCT concept is extended to the finite element methods for solving systems of partial differential equations with high temporal accuracy.

5.2.1 Flux-Corrected Transport (FCT)

For flows described by Eqn.(5.1), discontinuities in the variables may arise (e.g. shocks or contact discontinuities). Any numerical scheme of order higher than one will produce overshoots or ripples at such discontinuities (the so-called 'Godunov theorem' [35]). Very often, particularly for mildly nonlinear systems, these overshoots can be tolerated. However, for high-speed compressible flows, overshoots will eventually lead to numerical instability, and therefore must be suppressed.

The idea behind FCT is to combine a high-order scheme with a low-order scheme in such a way that, in regions where the variables under consideration vary smoothly, (so that a Taylor expansion makes sense) the high-order scheme is employed, whereas, in those regions where the variables vary abruptly, the schemes are combined in a conservative manner in an attempt to ensure a monotonic solution.

The temporal discretization of Eqn.(5.1) yields

$$U^{n+1} = U^n + \Delta U, \tag{5.4}$$

where ΔU is the increment of the unknowns obtained for a given scheme at time $t = t^n$. The objective is to obtain a ΔU of as high an order as possible without introducing overshoots. To this end, Eqn.(5.4) is re-written as

$$U^{n+1} = U^n + \Delta U^l + (\Delta U^h - \Delta U^l), \tag{5.5}$$

or

$$U^{n+1} = U^l + (\Delta U^h - \Delta U^l). \tag{5.6}$$

Here ΔU^h and ΔU^l denote the increments obtained by some high- and low-order scheme, respectively, whereas U^l is the monotone, ripple-free solution at time $t = t^{n+1}$ of the low-order scheme. The idea behind FCT is to limit the second term on the right-hand side of Eqn.(5.6) in such a way that no new over/undershoots are created, i.e.,

$$U^{n+1} = U^l + lim(\Delta U^h - \Delta U^l). \tag{5.7}$$

It is at this point that a further constraint, given by the conservation law (5.1) itself must be taken into account, i.e., strict conservation on the discrete level should be maintained. The simplest way to guarantee this for node-centered schemes (which are only considered here) is by constructing schemes for which the sum of the contributions of each individual element (cell) to its surrounding nodes vanishes ('all that comes in goes out'). This means that the limiting process [Eqn.(5.7)] will have to be carried out in the elements (cells).

5.2.2 Algorithmic Implementation

Following Zalesak's exposition [29], but replacing the term 'flux' by 'element contribution to a node', we can now define FCT in a quantitative way through the following six algorithmic steps.

(i) Compute LEC: the 'low-order element contribution' from some low-order scheme guaranteed to give monotonic results for the problem at hand;

(ii) compute HEC: the 'high-order element contribution', given by some high-order scheme;

(iii) define AEC: the 'antidiffusive element contributions' :

$$AEC = HEC - LEC;$$

(iv) compute the updated low-order solution :

$$U^l = U^n + \sum_{el} LEC = U^n + \Delta U^l; \tag{5.8}$$

(v) limit or 'correct' the AEC so that U^{n+1}, as computed in step (vi) below, is free of extrema not also found in U^l or U^n :

$$AEC^c = Cel * AEC, \quad 0 \leq Cel \leq 1; \tag{5.9}$$

(vi) apply the limited AEC :

$$U^{n+1} = U^l + \sum_{el} AEC^c. \tag{5.10}$$

5.2.3 The Limiting Procedure

Obviously, the whole approach depends critically on the important step (v) above. The following quantities are defined to describe the limiting procedure.

(a) $P_I^{\pm}$: the sum of all positive (negative) antidiffusive element contributions to node I

$$P_I^{\pm} = \sum_{el} \begin{Bmatrix} max \\ min \end{Bmatrix} (0, AEC_{el})$$

(b) $Q_I^{\pm}$: the maximum (minimum) increment (decrement) that node I is allowed to achieve in step (vi) above

$$Q_I^{\pm} = U_I^{\substack{max \\ min}} - U^l$$

where $U_I^{\substack{max \\ min}}$ (defined below) represents the maximum (minimum) value the unknown U at node I is allowed to achieve in step 6 above.

(c) $R^{\pm}$:

$$R^{\pm} = \begin{cases} min(1, Q^{\pm}/P^{\pm}) & if \;\; P^{+} > 0, \; P^{-} < 0 \\ 0 & if \;\; P^{\pm} = 0. \end{cases}$$

Now for each element,

$$Cel = min(element\ nodes) \begin{cases} R^{+} & if \;\; AEC > 0, \\ R^{-} & if \;\; AEC < 0. \end{cases} \tag{5.11}$$

Finally, $U_I^{\substack{max \\ min}}$ is obtained in three steps :

(i) maximum (minimum) nodal U of U^n and U^l :

$$U_I^* = \begin{Bmatrix} max \\ min \end{Bmatrix} (U_I^l, U_I^n) \quad ,$$

(ii) maximum (minimum) nodal value of element :

$$U_{el}^* = \begin{Bmatrix} max \\ min \end{Bmatrix} (U_A^*, U_B^*, ..., U_C^*) \quad ,$$

where $A, B, ..., C$ represent the nodes of element el.

(iii) maximum (minimum) U of all elements surrounding node I :

$$U_I^{\substack{max \\ min}} = \begin{Bmatrix} max \\ min \end{Bmatrix} (U_1^*, U_2^*, ..., U_m^*)$$

where $1, 2, ..., m$ represent the elements surrounding node I.

This completes the description of the limiting procedure. Up to this point a completely general description has been given, so that Eqns.(5.6)-(5.13) may be applied to any FEM-FCT scheme. In what follows, we restrict the exposition to the finite element schemes employed in the present work, describing the high and low-order schemes used.

5.2.4 The High-Order Scheme: Consistent-Mass Taylor Galerkin

As the high-order scheme, a two-step form [36-38] of the one-step Taylor-Galerkin schemes described in References [39] and [40] is employed. These schemes belong to the Lax-Wendroff class, and could be substituted by any other high-order scheme which appears more convenient, including implicit schemes. Given the system of equations (5.1), the solution is advanced from t^n to $t^{n+1} = t^n + \Delta t$ as follows.

a) First step:

$$U^{n+\frac{1}{2}} = U^n + \frac{\Delta t}{2}\left[S|^n - \frac{\partial F_j^a}{\partial x_j}\Bigg|^n \right] \tag{5.12}$$

b) Second step :

$$\Delta U^n = U^{n+1} - U^n = \Delta t \left[S|^{n+\frac{1}{2}} - \frac{\partial F_j^a}{\partial x_j}\Bigg|^{n+\frac{1}{2}} \right]. \tag{5.13}$$

The spatial discretization of (5.12) and (5.13) is performed via the classic Galerkin weighted residual method [36-38], using linear elements, i.e., 3-noded triangles in 2-D and 4-noded tetrahedra in 3-D. For (5.13), the following system of equations is obtained.

$$M_C \cdot \Delta U^n = R^n \tag{5.14}$$

where M_C denotes the consistent mass matrix, ΔU the vector of nodal increments and R the vector of added element contributions to the nodes. As M_C possesses an excellent condition number, Eqn.(5.14) is never solved directly, but iteratively, requiring typically three passes [40]. We recast the converged solution of Eqn.(5.14) into the following form, which will be of use later on.

$$M_L \cdot \Delta U^h = R + (M_L - M_C) \cdot \Delta U^h; \tag{5.15}$$

here M_L denotes the diagonal, lumped mass-matrix (see [40]).

5.2.5 The Low-Order Scheme: Lumped-Mass Taylor Galerkin Plus Diffusion

The requirement placed on the low-order scheme in any FCT-method is monotonicity. The low-order scheme must not produce any artificial, or numerical, 'ripples' or 'wiggles'. It is clear that the better the low-order scheme, the easier the resulting task of limiting will be. Therefore an obvious candidate for the low-order scheme is Godunov's method [35]. However, this scheme would be relatively expensive, and its extension to unstructured grids remains unclear.

In the context of FEM-FCT [41,42], the 'mass-diffusion' is added to the lumped-mass Taylor-Galerkin scheme. This simplest and least expensive form of diffusion is obtained by subtracting the lumped mass-matrix from the consistent mass-matrix for linear elements, i.e.,

$$DIFF = c_d \cdot (M_C - M_L) \cdot U^n. \tag{5.16}$$

The element matrix thus obtained for 2-D triangles is of the form

$$c_d \cdot (M_C - M_L)_{el} = -\frac{c_d \cdot Vol_{el}}{12} \left\{ \begin{array}{rrr} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{array} \right\}. \tag{5.17}$$

It should be noticed that this diffusion cannot be simply added to the high-order scheme in order to obtain monotonic results, as a multipoint-coupling of the right-hand side occurs due to the consistent mass-matrix employed in the high-order scheme . The imposition of monotonicity can nevertheless be achieved by using a lumped mass-matrix instead. As the terms originating from the discretization of the fluxes F^i in Eqn.(5.1) are the same as in Eqn.(5.13), the low-order scheme is given by

$$M_L \cdot \Delta U^l = R + DIFF. \tag{5.18}$$

5.2.6 Resulting Antidiffusive Element Contributions

Subtracting Eqn.(5.18) from Eqn.(5.15) yields

$$M_L \cdot (\Delta U^h - \Delta U^l) = R + (M_L - M_C) \cdot \Delta U^h - R - DIFF, \tag{5.19}$$

or using Eqn.(5.16),

$$\Delta U^h - \Delta U^l = M_L^{-1} \cdot (M_L - M_C) \cdot (c_d \cdot U^n + \Delta U^h). \tag{5.20}$$

Note that all terms arising from the discretization of the fluxes F^i in Eqns. (5.1), (5.13), and (5.18) have now disappeared. This is of particular importance for cases where the antidiffusive element contributions must be recomputed (small core memory machines) or real gas effects are taken into account (table look-up times are considerable).

5.2.7 Limiting for Systems of Equations

Results available in the literature [30-32 and 41,42] have shown that high quality results can be obtained for a single PDE with FCT. However, when trying to extend the limiting process to systems of PDEs, no immediately obvious or natural limiting procedure becomes apparent. Obviously, for 1-D problems, one could advect each simple wave system separately and then assemble the solution at the new time step. However, such a splitting is not possible for multidimensional problems as the acoustic waves are circular. Codes using FEM-FCT for production runs [43,44] have so far limited each equation separately, invoking operator-splitting arguments. This approach does not always give very good results, as may be seen from Sod's comparison of schemes for the Riemann problem [45], and has been a point of continuing criticism by those who prefer to use the more costly Riemann-solver-based, essentially one-dimensional TVD schemes [23-28]. It would, therefore, appear attractive to introduce a 'system character' for the limiter by combining the limiters for all equations of the system.

Many variations are possible and can be implemented, giving different performance for different problems. The following is a list of some of the possibilities with comments on those for which empirical experiences are available.

(a) Independent treatment of each equation as in operator-split FCT - this is the least diffusive method, which tends to produce an excessive amount of ripples in the non-conserved quantities (and ultimately in the conserved quantities also).

(b) Use of the same limiter (C_{el}) for all equations - this produces much better results, seemingly because the phase errors for all equations are 'synchronized'. This was also observed by Harten and Zwaas [46] and Zhmakin and Fursenko [47] for a class of schemes very similar to FCT.

(c) Use of a certain variable as 'indicator variable' (e.g. density, pressure, entropy).

(d) Use of the minimum of the limiters obtained for the density and the energy $C_{el} = min[C_{el}(density), C_{el}(energy)]$ - this produces acceptable results, although some undershoots for very strong shocks are present.

(e) Use of the minimum of the limiters obtained for the density and the pressure $C_{el} = min[C_{el}(density), C_{el}(pressure)]$ - this also produces acceptable results, particularly for steady-state problems.

5.2.8 Artificial Viscosities

While the FEM-FCT algorithm avoids most of the overshoots encountered at discontinuities for linear schemes, some residual noise is still left in the solution. Therefore, additional smoothing is implemented in the flow solver. Several smoothing artificial viscosities have been examined. Some of these are:

Lapidus [48]:

$$D = c_l h^2 \frac{\partial}{\partial l}(\mathbf{v} \cdot \mathbf{l}) \tag{5.21}$$

where

$$\mathbf{l} = \frac{\nabla|\mathbf{v}|}{|\nabla|\mathbf{v}||}. \tag{5.22}$$

Here $\mathbf{v}$ denotes the velocity vector of the fluid, h is a characteristic element length, and c_l is a constant. The advantages of this artificial viscosity are: invariance against coordinate rotation, and economy due to uni-directionality. On the other hand, by itself, it is under-diffusive for steady-state problems. We employ it mainly as post-smoother in transient simulations to avoid terracing of expansion waves.

<u>Jameson [49]:</u>

In this case, the diffusion operator is of the form:

$$D = c_j Cou \frac{\partial}{\partial x} h^2 f(p) \frac{\partial}{\partial x}. \tag{5.23}$$

No direct influence of the time step Δt is present in this operator. The time step is only indirectly present through the Courant-number Cou. The pressure switch is chosen as

$$f(p) = \frac{1}{\overline{p}} |\frac{\partial}{\partial x} h^2 \frac{\partial}{\partial x} p|. \tag{5.24}$$

The summation of partial derivatives is approximated by the difference of consistent and lumped mass-matrices:

$$\int \frac{\partial}{\partial x} h^2 \frac{\partial}{\partial x} p \; d\Omega = (M_C - M_L)p. \tag{5.25}$$

This avoids the difficulties of choosing the appropriate h^2, and makes the operator invariant against coordinate rotation. At the same time, it reduces CPU-costs considerably. This form of artificial viscosity appears as the most appropriate for the transonic regime. The results shown here were obtained by using this form of artificial viscosity.

Morgan et al. [50]:

The diffusion operator is the same as before. However, the pressure switch is chosen as:

$$f(p) = \frac{|\frac{\partial}{\partial x} h^2 \frac{\partial}{\partial x} p|}{|h \frac{\partial}{\partial x} p|}. \tag{5.26}$$

This switch is again approximated by

$$f(p) = \frac{|(M_L - M_C)p|}{\sum_{el} |(M_L - M_C)p|}. \tag{5.27}$$

Our experience shows that this form of artificial viscosity is too diffusive for the transonic regime. On the other hand, we use it regularly for supersonic problems. While the diffusion operator is employed with the conserved variables for the continuity and momentum equations, for the energy equation the total enthalpy is employed.

5.3 SAMPLE RESULTS

In this section sample results on an ONERA M6 wing are shown to establish the accuracy of the flow solver. This geometry has been extensively used in the past to validate newly developed codes. The ONERA M6 wing has symmetrical airfoil sections and a leading edge sweep of 30^o.

The 3-D tetrahedral grid was generated using VGRID3D. The computational domain is bounded by a rectangular box with boundaries at $-6.5 \leq X \leq 11.0$, $0.0 \leq Y \leq 2.5$ and $-6.5 \leq Z \leq 6.5$. The co-ordinate system, for the grid generation purpose, has its origin at the root leading-edge of the wing with +X streamwise downstream, +Y in the spanwise starboard direction and +Z upwards. The generated grid has 208,908 tetrahedral elements and 39,222 points. Of these, 8496 points represent the boundary surfaces (wing as well as the outer computational boundaries) while the rest are the field points.

The computations were performed using VFLOW3D at transonic condition: M_∞=0.84 and α=3.06^o. The results presented here are after 5000 iterations when residues were down by five orders of magnitude. The local time-stepping option was used to accelerate the convergence to steady-state and Jameson type artificial viscosity, described in the previous section, was used. The flow solver required about 15.9 million words of memory on CRAY-2 with about 86 microseconds/point/iteration.

Figure 28(a) shows the surface triangulation on the upper surface of the wing. The flexibility of clustering grid points at desired locations is evident as small elements are placed near the leading edge where a sharp peak in pressure is expected. Figure 28(b) shows pressure contours on the upper surface of the wing. The contour interval is $\Delta(p/p_\infty)$=0.2. As expected, the λ-shock structure can be seen. Figure 29 shows (a) the grid and (b) the pressure contours in the plane of symmetry. Once again the variation in the element size is evident.

In Figure 30, the pressure coefficient C_p, is compared with the experimental results from Ref. [51] at three spanwise stations of 44, 65 and 90%. For all the three stations, the solution misses the suction peak which may be due to the grid resolution in the leading edge area not being enough. Also the fact that inviscid Euler solution is compared with experimental results, which were obtained at a Reynolds number of 11.7 million, accounts for some differences. However, the overall solution is reasonable.

6. CONCLUSIONS

Due to their premise to easily generate 3-D grids over complex configurations, unstructured grids have received a wide attention in recent years. As a result of NASA's interest in the application of this new technology to solve fluid dynamic problems, a program was begun about three years ago (in the form of an SBIR phase I study to ViGYAN), to explore the feasibility of generating complex, 3-D grids. Once this feasibility was established beyond doubt, the second phase of SBIR was begun, this time with a challenge to develop programs not only for the grid generation but also for the flow solution and the post-processing applications.

Over the last two years, a complete set of programs has been developed that at the least show that CFD analysis can be performed over complex configurations using the unstructured grid approach. As in any exploratory study, there are certain areas that need further research. These areas are outlined below.

The grid generator needs to be made robust and some work is needed that provides an estimate of grid quality and ways of implementing them into the grid generator. By far the largest amount of time is required in setting up an appropriate background grid. This part of the program needs to be automated further. The flow solver needs to be validated for a variety of flow conditions. This may need 'tuning' the artificial viscosity routines. The next step then would be to add viscous terms to the flow solver and implementation of solution adaptive and moving grids. Of the three programs developed, the post-processing program is in the most advanced stage and is capable of displaying scalar and vector quantities on planes, surfaces or in the 3-D field. Implements like volume rendering and ability to calculate and plot stream-lines will enhance its utility even further.

In summary, the development of the programs under this contract has not only established unstructured grid methodology as an alternative to structured grids in CFD analysis but also given a timely impetus towards further research.

REFERENCES

[1] Löhner, R. and Parikh,P., "Generation of Three-Dimensional Unstructured Grids By the Advancing Front Method," *Internat. J. Numer. Methods Fluids*, 8, 1135-1149 (1988).

[2] Gumbert, C., Löhner, R., Parikh P. and Pirzadeh S., "A Package for Unstructured Grid Generation and Finite Element Flow Solvers," AIAA-89-2175 (1989).

[3] Parikh, P., Löhner, R., Gumbert C. and Pirzadeh S., "Numerical Solutions on a PATHFINDER and other Configurations Using Unstructured Grids and a Finite Element Solver," AIAA-89-0362 (1989).

[4] Watson, D.F., "Computing the N-Dimensional Delaunay Tesselation with Application to Voronoi Polytopes," *The Comput. J.*, 24, 2, 167-172 (1981).

[5] Bowyer, A., "Computing Dirichlet Tesselations," *Computer Journal* 24, 2, 162-167 (1981).

[6] Sloan, S.W. and Houlsby, G.T., "An Implementation of Watson's Algorithm for Computing 2-Dimensional Delaunay Triangulations," *Adv. Eng. Software*, 6, 4, 192-197 (1984).

[7] Tanemura, M., Ogawa, T. and Ogita, N., "A New Algorithm for Three- Dimensional Voronoi Tesselation," *J.Comput.Phys.*, 51, 191-207 (1983).

[8] Jameson, A., Baker, T.J. and Weatherhill, N.P., "Calculation of Inviscid Transonic Flow over a Complete Aircraft," AIAA-86-0103 (1986).

[9] Baker, T.J., "Three-Dimensional Mesh Generation by Triangulation of Arbitrary Point Sets," AIAA-87-1124-CP (1987).

[10] Yerry, M.A. and Shepard, M.S., "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique," *Internat.J.Numer.Methods.Engrg.*, 20, 1965-1990 (1984).

[11] van Phai, N., "Automatic Mesh generation with Tetrahedron Elements," *Internat.J.Numer.Methods.Engrg.*, 18, 237-289 (1982).

[12] Lo, S.H., "A New Mesh Generation Scheme for Arbitrary Planar Domains," *Internat.J.Numer.Methods.Engrg.*, 21, 1403-1426 (1985).

[13] Peraire, J., Vahdati, M., Morgan, K. and Zienkiewicz, O.C., " Adaptive Remeshing for Compressible Flow Computations," *J.Comput.Phys.*, 72, 449-466 (1987).

[14] Woodwark, J., *Computing Shape*, Butterworths (1986).

[15] Zienkiewicz, O.C. and Morgan, K., *Finite Elements and Approximation*, J. Wiley & Sons (1983).

[16] Barnhill, R.E., "Representation and Approximation of Surfaces," pp. 69-120 in *Mathematical Software III* (J.R. Rice, ed.), Academic Press (1977).

[17] Knuth, D.N., *The Art of Computer Programming* , Vol.3, Addison-Wesley (1973).

[18] Sedgewick, R., *Algorithms*, Addison-Wesley (1983).

[19] Broder, A.J., "An Algorithm for Incremental Nearest Neighbor Search in k-Dimensional Space," Rep. MP-86W00026, The MITRE Corp. (1986).

[20] Greengard, L. and Rokhlin, V., "A Fast Algorithm for Particle Simulations," *J.Comput. Phys.*, 73, 325-348 (1987).

[21] Samet, H., "The Quadtree and Related Hierarchical Data Structures," *Comput. Surveys*, 16, 2, 187-285 (1984).

[22] Löhner, R., Parikh, P. and Gumbert, C., "Some Algorithmic Problems of Plotting Codes for Unstructured Grids," AIAA-89-1981 (1981).

[23] Woodward, P. and Colella, P., "The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks," *J.Comput.Phys.*, 54, 115-173 (1984).

[24] van Leer, B., "Towards the Ultimate Conservative Scheme. II. Monotonicity and Conservation Combined in a Second Order Scheme," *J.Comput.Phys.*, 14, 361-370 (1974).

[25] Roe, P.L., "Approximate Riemann Solvers, Parameter Vectors and Difference Schemes," *J.Comput.Phys.*, 43, 357-372 (1981).

[26] Osher, S. and Solomon, F., "Upwind Difference Schemes for Hyperbolic Systems of Conservation Laws," *Math.Comp.*, 38, 339-374 (1982).

[27] Harten, A., "High Resolution Schemes for Hyperbolic Conservation Laws," *J.Comput.Phys.*, 49, 357-393 (1983).

[28] Sweby. P.K., "High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws," *SIAM J.Numer.Anal.*, 21, 995-1011 (1984).

[29] Zalesak, S.T., "Fully Multidimensional Flux-Corrected Transport Algorithm for Fluids," *J.Comput.Phys.*, 31, 335-362 (1979).

[30] Boris, J.B. and Book, D.L., "Flux-Corrected Transport. I. SHASTA, a Transport Algorithm that Works," *J.Comput.Phys.*, 11, 38 (1973).

[31] Book, D.L., Boris, J.P and Hain, K., "Flux-Corrected Transport. II. Generalizations of the Method," *J.Comput.Phys.*, 18, 248 (1975).

[32] Boris, J.P. and Book, D.L., "Flux-Corrected Transport. III. Minimal-Error FCT Algorithms," *J.Comput.Phys.*, 20, 397-431 (1976).

[33] Erlebacher, G., *Solution Adaptive Triangular Meshes with Application to the Simulation of Plasma Equilibrium*, Ph.D. Thesis, Columbia University , (1984).

[34] Parrott, A.K. and Christie, M.A., "FCT Applied to the 2-D Finite Element Solution of Tracer Transport by Single Phase Flow in a Porous Medium," *Proceedings of the ICFD-Conf. on Numerical Methods in Fluid Dynamics*, Reading, Academic Press (1986).

[35] Godunov, S.K., *Mat. Sb.*, 47, 271-306 (1959).

[36] Löhner, R., Morgan, K. and Zienkiewicz, O.C., "An Adaptive Finite Element Procedure for High Speed Flows," *Comput.Methods.Appl.Mech.Engrg.*, 51, 441-465 (1985).

[37] Löhner, R., Morgan, K., Peraire, J. and Zienkiewicz, O.C., "Finite Element Methods for High Speed Flows," AIAA-85-1531-CP (1985).

[38] Löhner, R., Morgan, K., Peraire, J., Zienkiewicz, O.C. and Kong, L., "Finite Element Methods for Compressible Flow," pp. 28-53 in *Numerical Methods for Fluid Dynamics* (K.W. Morton and M.J. Baines, eds.), Oxford University Press, (1986).

[39] Donea, J., "A Taylor Galerkin Method for Convective Transport Problems," *Internat.J.Numer.Methods.Engrg.*, 20, 101-119 (1984).

[40] Löhner, R., Morgan, K. and Zienkiewicz, O.C., "The Solution of Nonlinear Systems of Hyperbolic Equations by the Finite Element Method," *Internat.J.Numer.Methods.Fluids* 4, 1043-1063 (1984).

[41] Löhner, R., Morgan, K., Vahdati, K, Boris, J.P. and Book, D.L., "FEM-FCT: Combining Unstructured Grids with High Resolution," *Comm. Appl. Numer. Methods.* 4, 717-730 (1988).

[42] Morgan, K., Löhner, R., Jones, J.R., Peraire, J. and Vahdati, M., "Finite Element FCT for the Euler and Navier-Stokes Equations," *Proc. 6th Int. Symp. Finite Element Methods in Flow Problems*, INRIA (1986).

[43] Fry, M.A. and Book, D.L., "Adaptation of Flux-Corrected Transport Codes for Modelling Dusty Flows," *Proc. 14th Int. Symp. on Shock Tubes and Waves* (R.D. Archer and B.E. Milton, eds.), New South Wales University Press (1983).

[44] Fyfe, D.E., Gardner, J.H., Picone, M. and Fry, M.A., "Fast Three-Dimensional

Flux-Corrected Transport Code for Highly Resolved Compressible Flow Calculations," *Springer Lecture Notes in Physics*, 218, 230-234, Springer Verlag (1985).

[45] Sod, G., *J.Comput.Phys.*, 27, 1-31 (1978).

[46] Harten, A. and Zwas, G., "Self-Adjusting Hybrid Schemes for Shock Computations," *J.Comput.Phys.*, 6, 568-583 (1972).

[47] Zhmakin, A.I. and Fursenko, A.A., "A Class of Monotonic Shock-Capturing Difference Schemes," NRL Memo. Rep., 4567, (1981).

[48] Löhner, R., Morgan, K. and Peraire, J., "A Simple Extension to Multidimensional Problems of the Artificial Viscosity due to Lapidus," Comm. Appl. Numer. Methods., 1, 141-147 (1985).

[49] Jameson, A. and Baker, T.J., "Improvements to the Aircraft Euler Method," AIAA-87-0452 (1987).

[50] Morgan, K. and Peraire, J., "Finite Element Methods for Compressible Flows," von Karman Institute for Fluid Dynamics, Lecture Series 1987-04 (1987).

[51] Schmitt, V. and Charpin, F., "Pressure Distributions on the ONERA M6-Wing at Transonic Mach Numbers," AGARD Advisory Report 138 (1979).

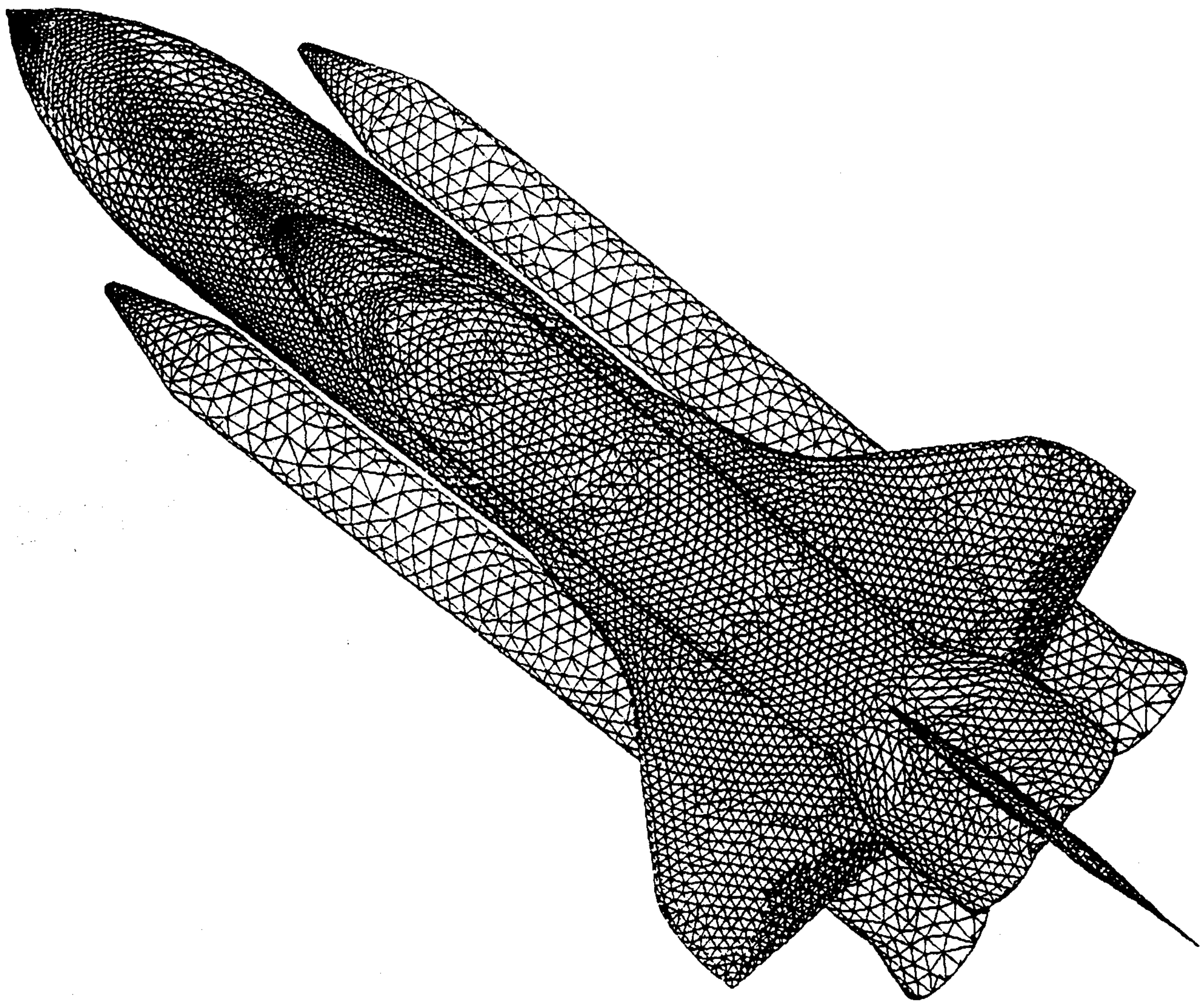


Figure 1. Surface Triangulation on a Space Transportation System Configuration

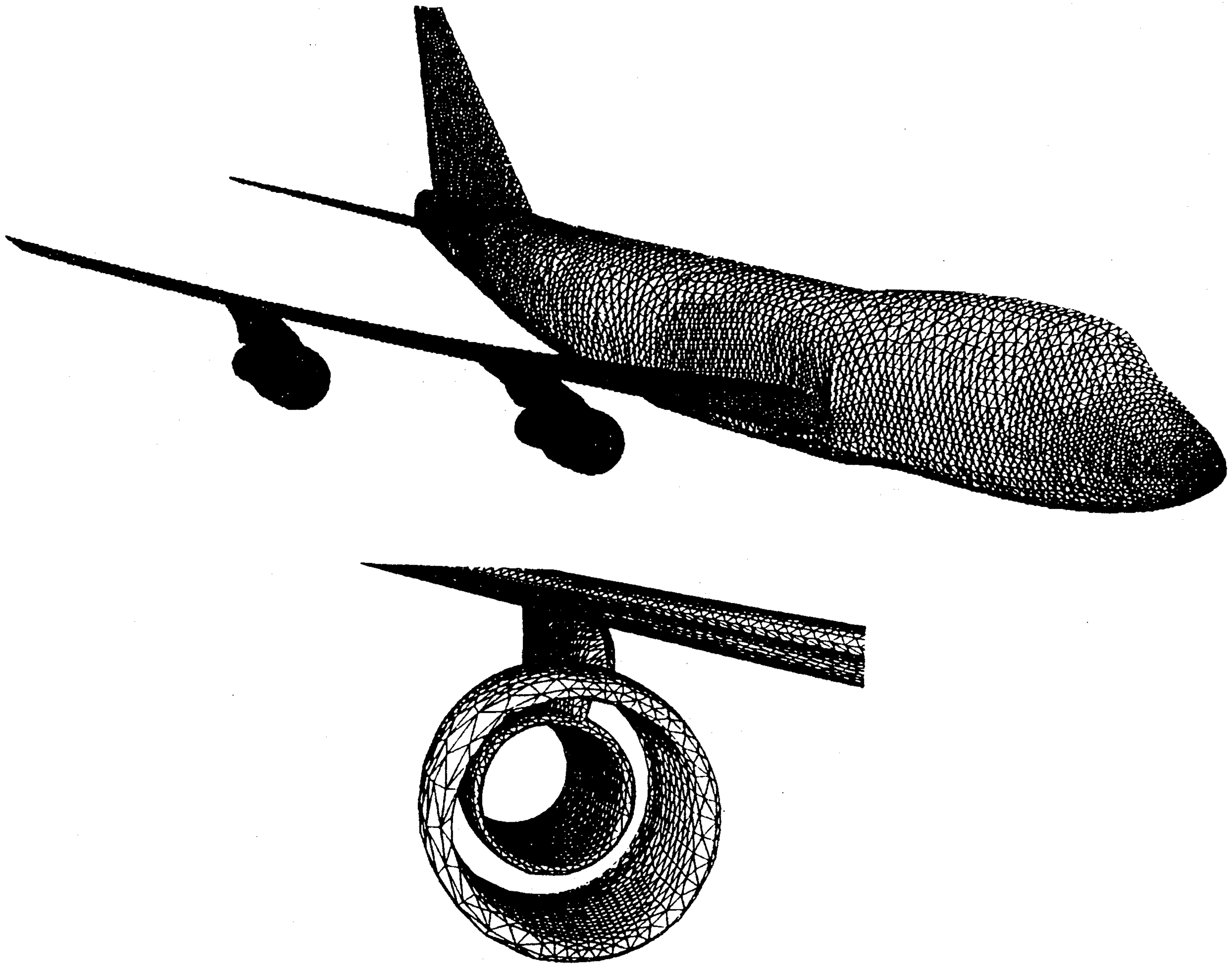


Figure 2. Surface Triangulation on a Boeing 747 configuration With Details on the Engine

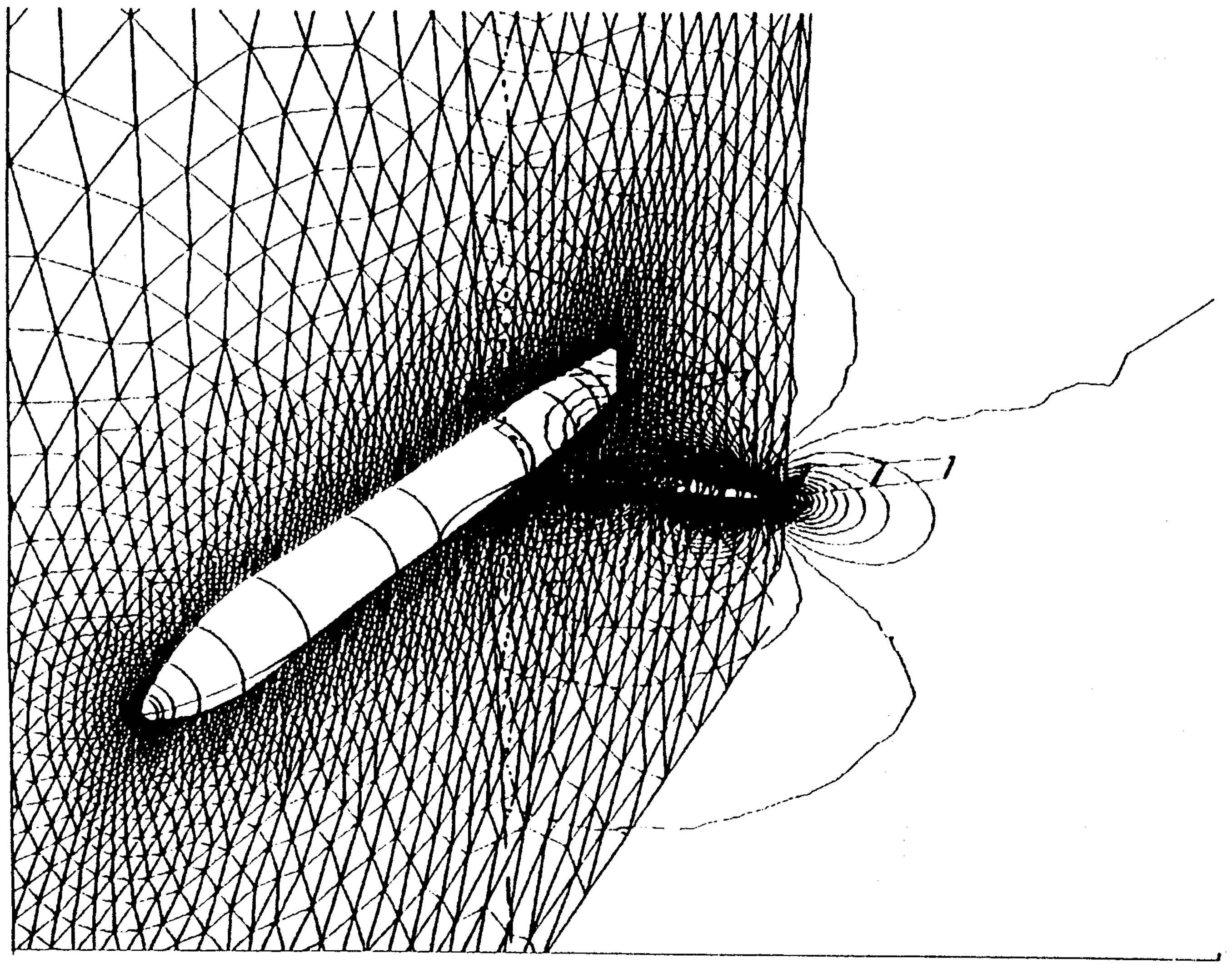


Figure 3. A Composite Picture Generated Using VPLOT3D

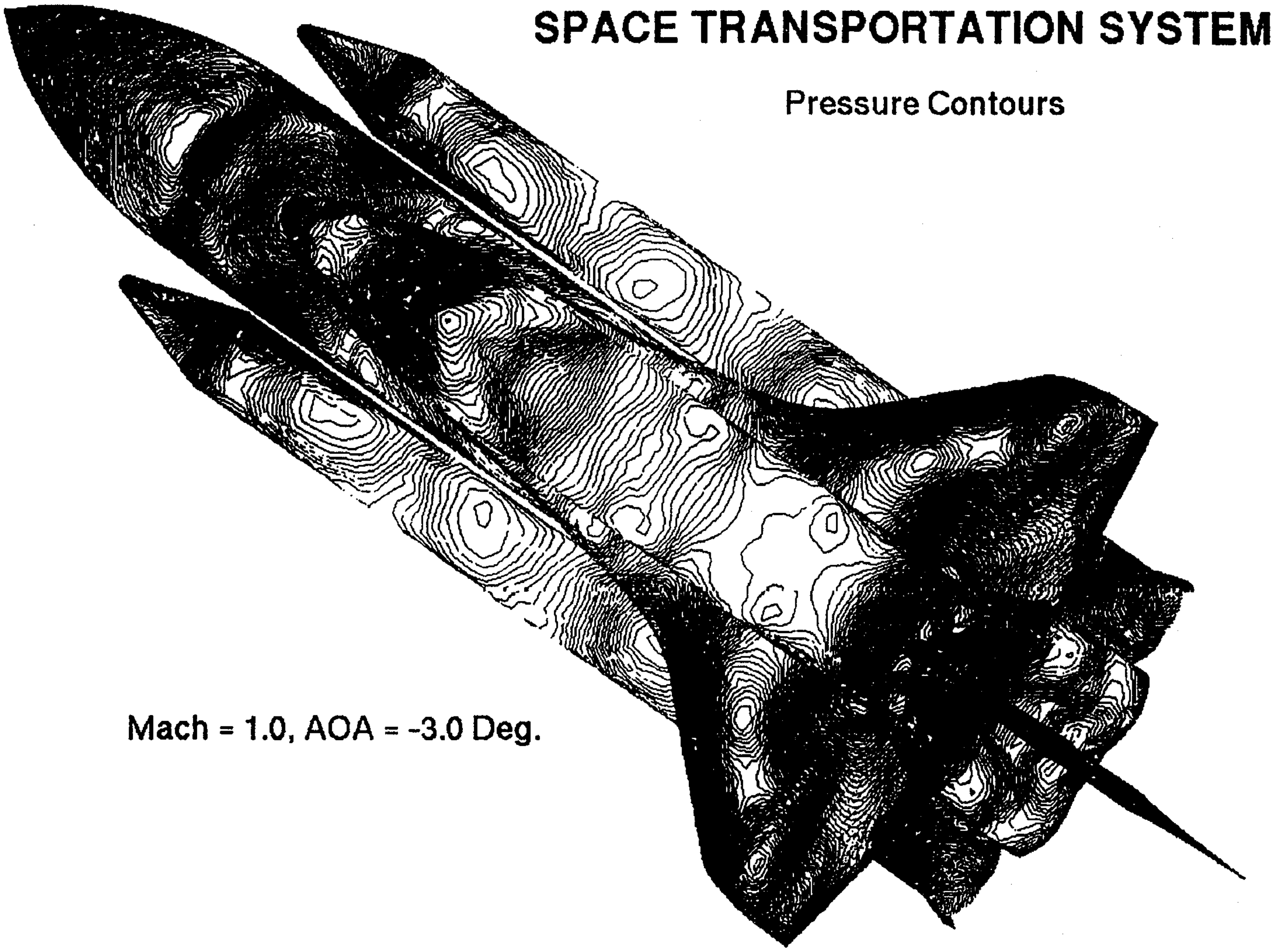


Figure 4. Surface Pressure Contours on an STS Configuration

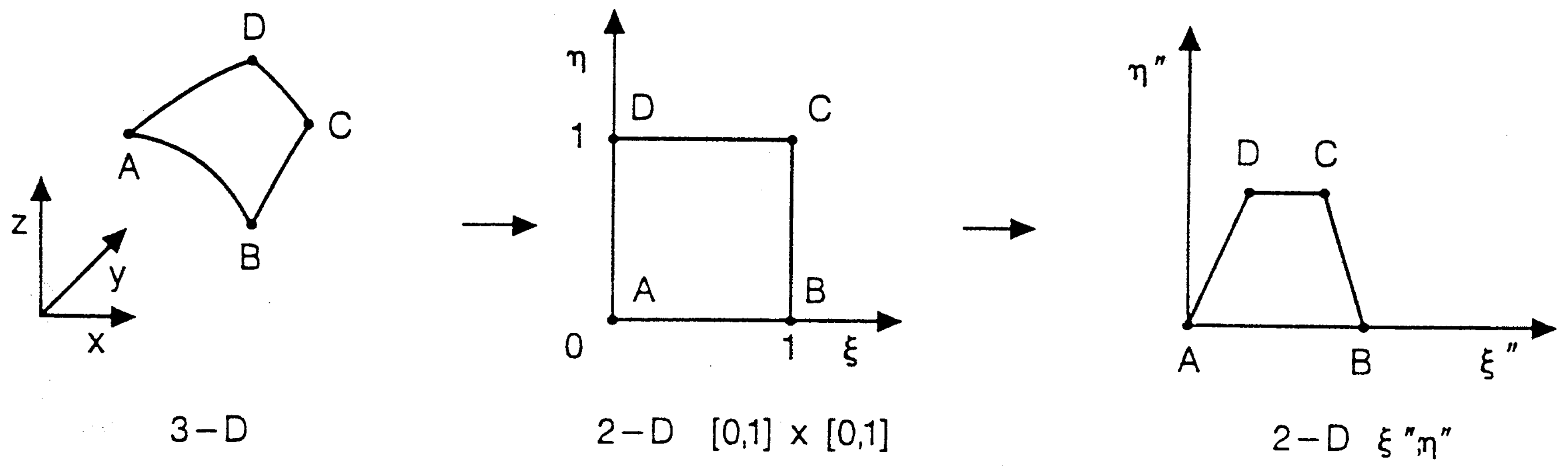


Figure 5. Mapping of a Surface Patch to a Unit Square With Stretching and Shearing

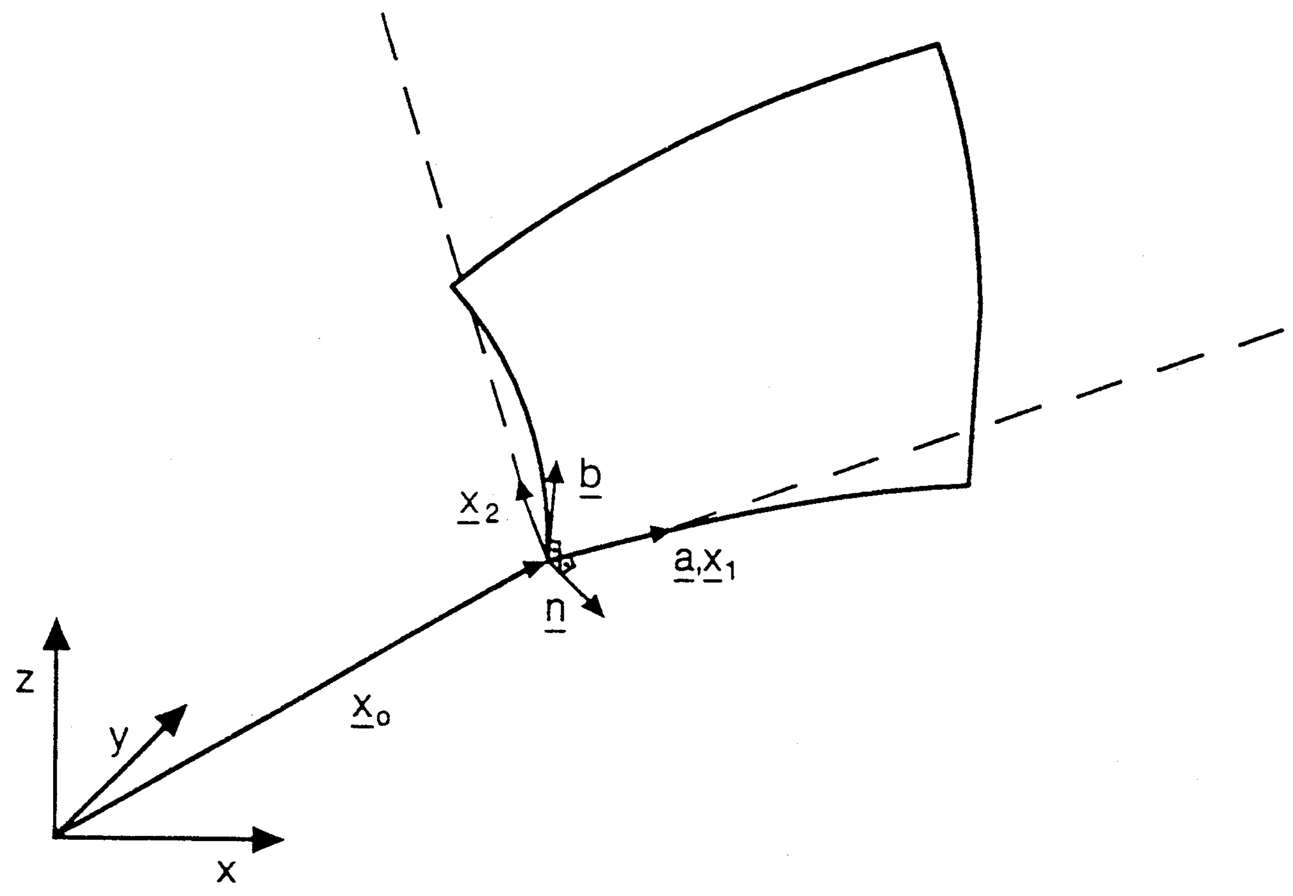


Figure 6. Covariant Bases $\underline{X}_1$ and $\underline{X}_2$ for a Planar Surface Patch

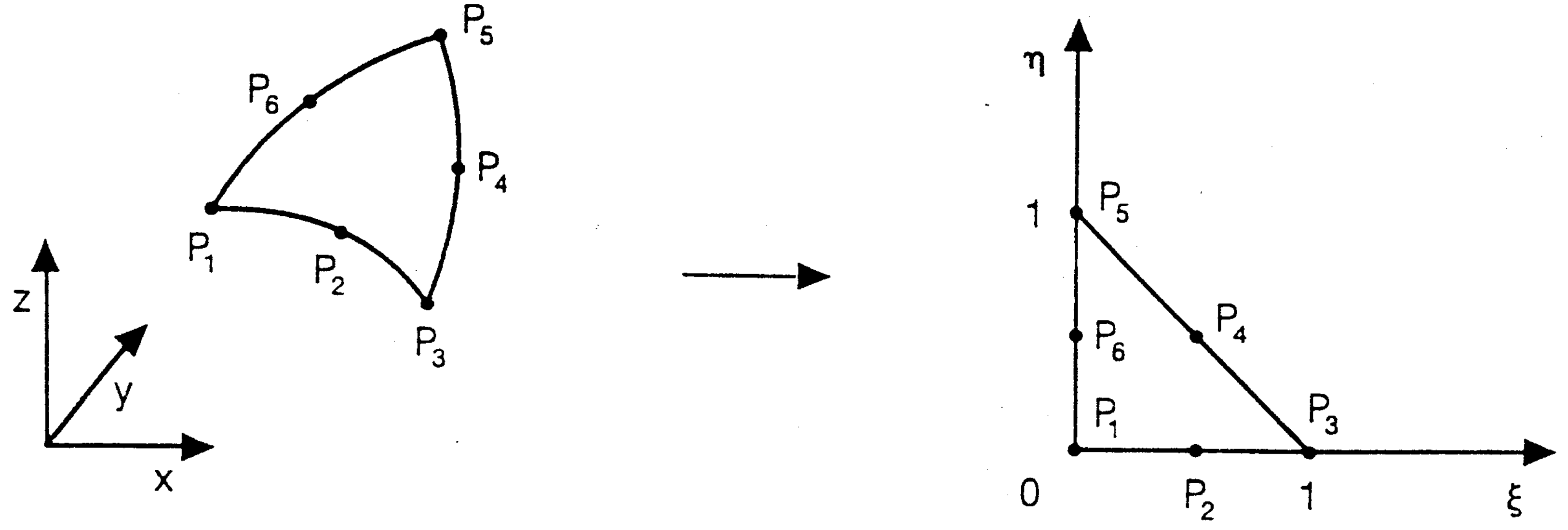


Figure 7. Triangular Isoparametric Parabolic Surface

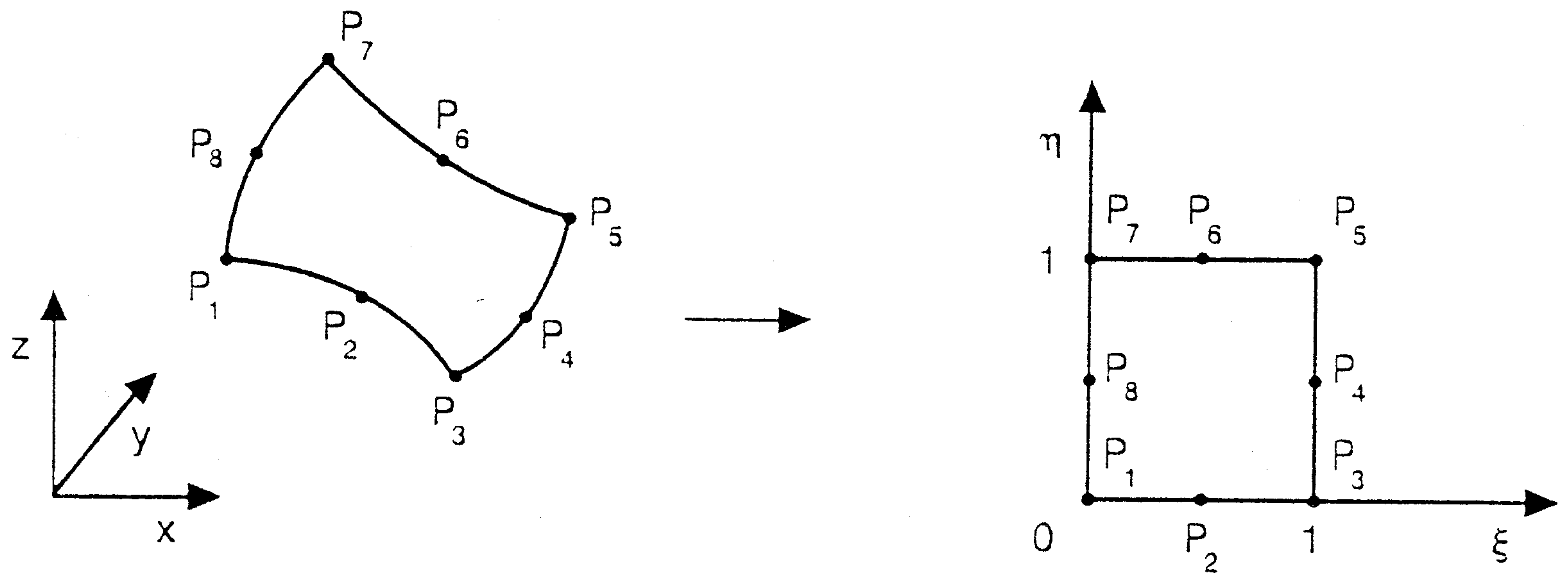


Figure 8. Rectangular Isoparametric Serendipity Surface

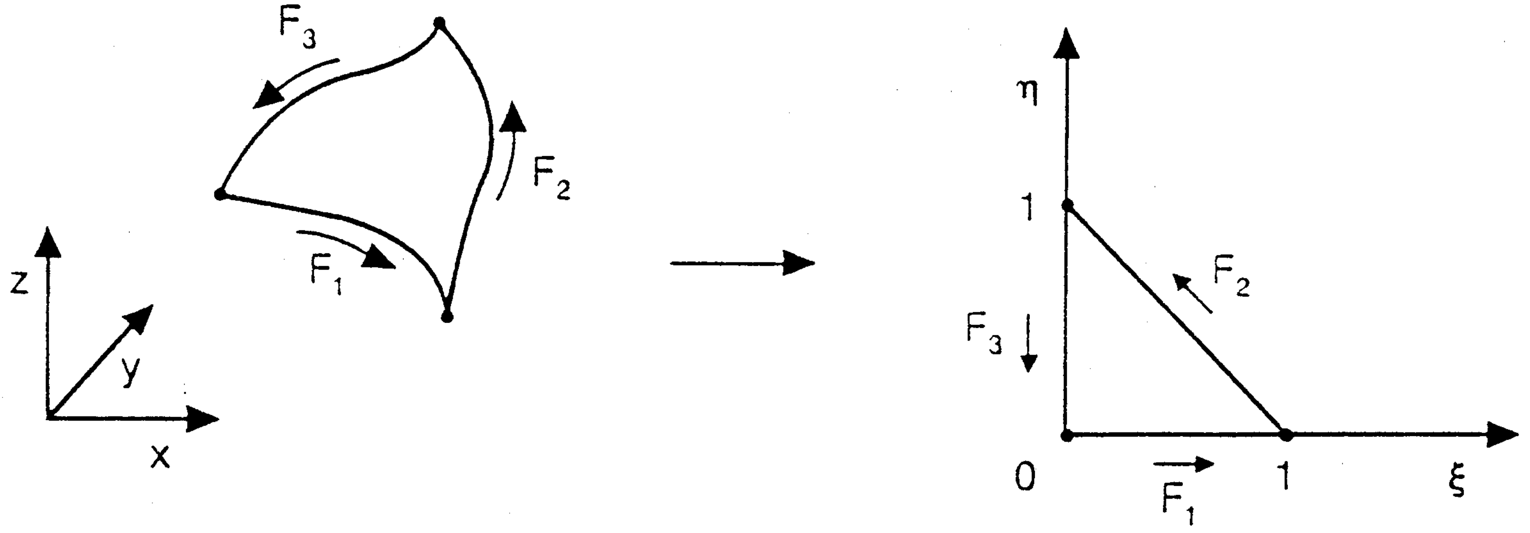


Figure 9. Triangular Barnhill-Gregory-Nielson Patch

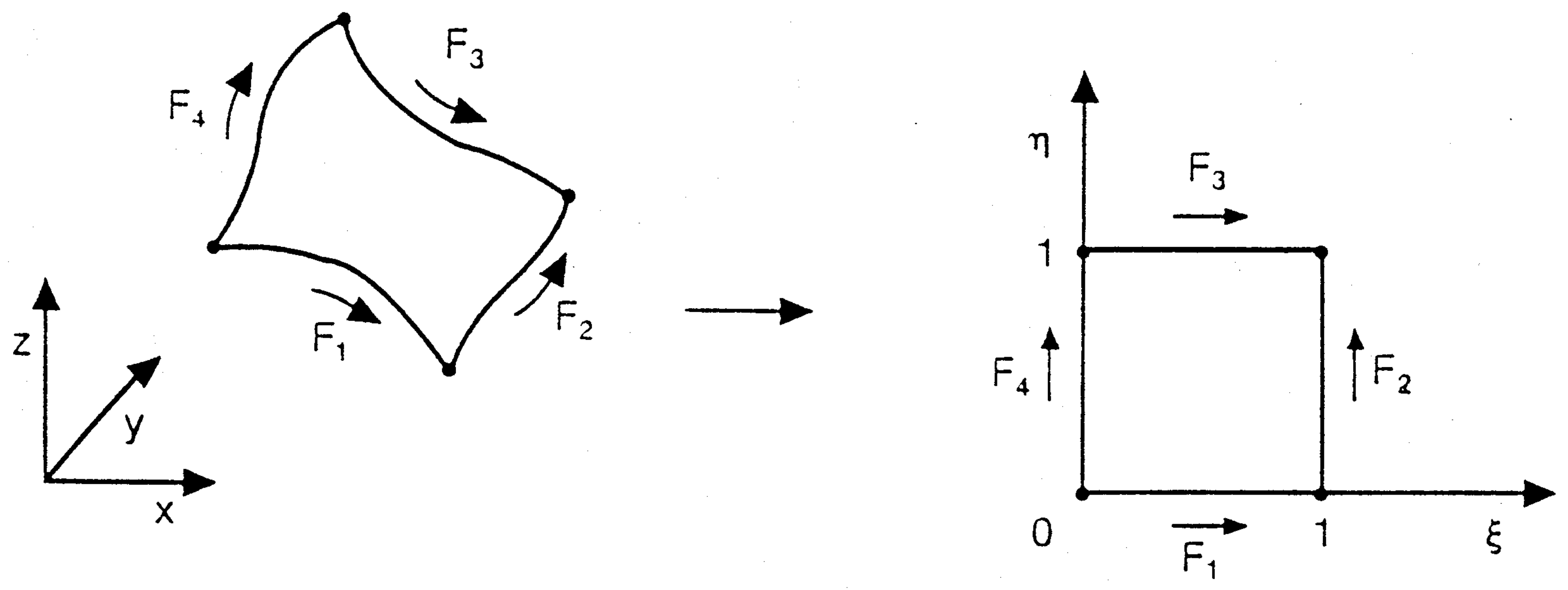


Figure10. Rectangular Bilinear Transfinite Coon's Patch

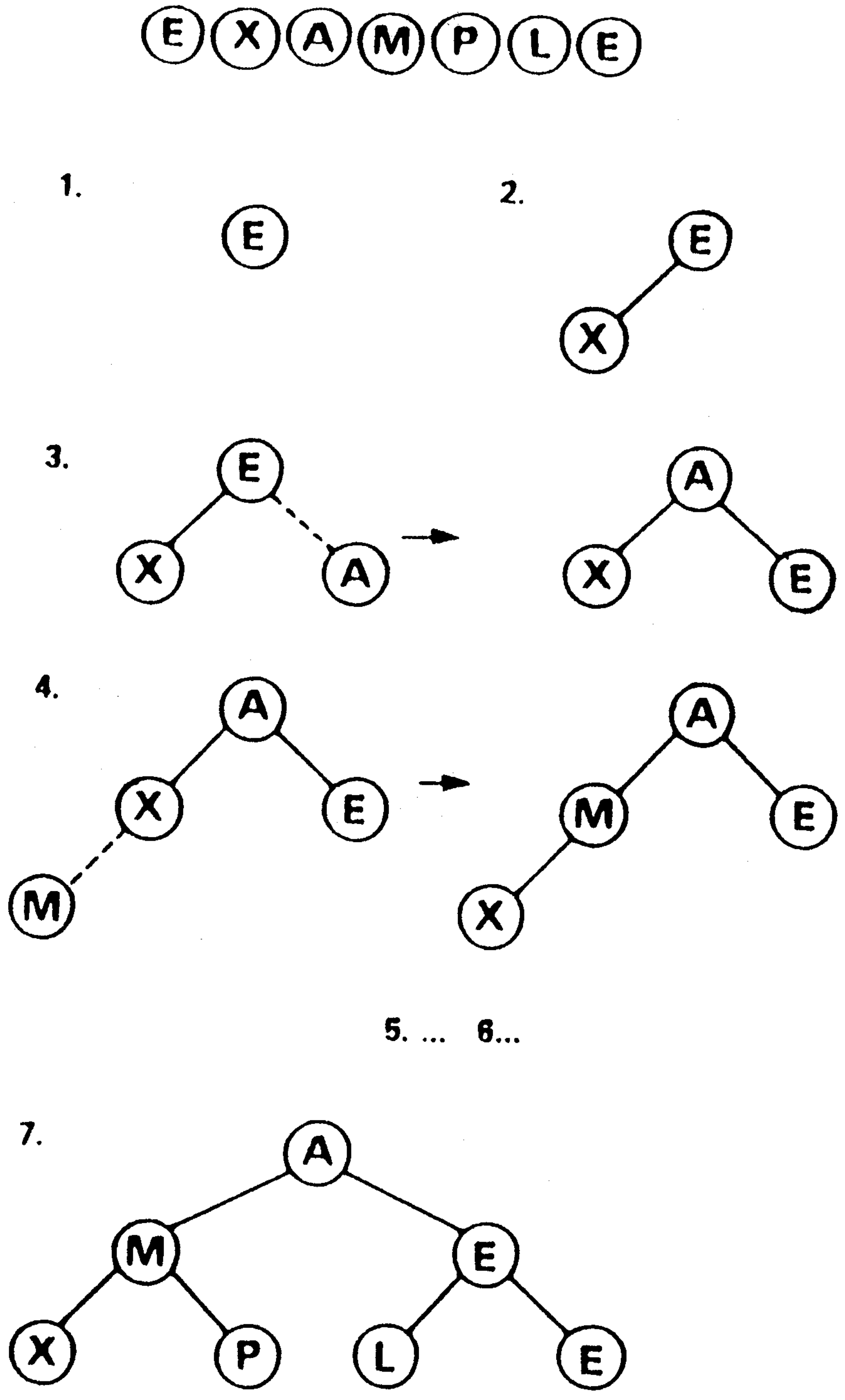
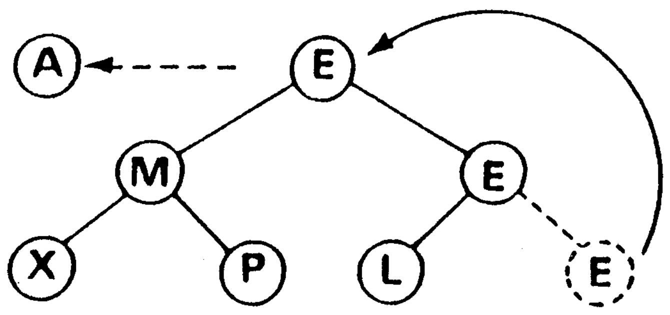
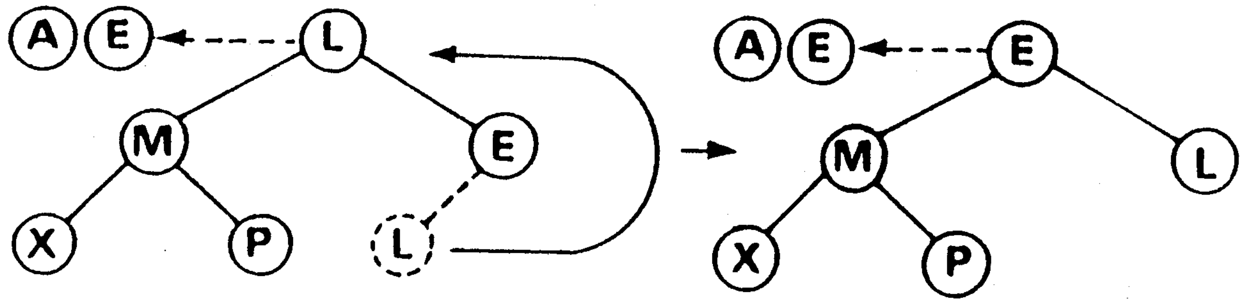


Figure 11 (a). Insertions of New Items into a Heap List

1.



2.



3. ...., 6.

7.

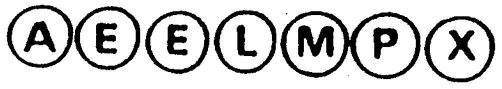


Figure 11 (b). Succesive Deletion of the Smallest Item from a Heap List

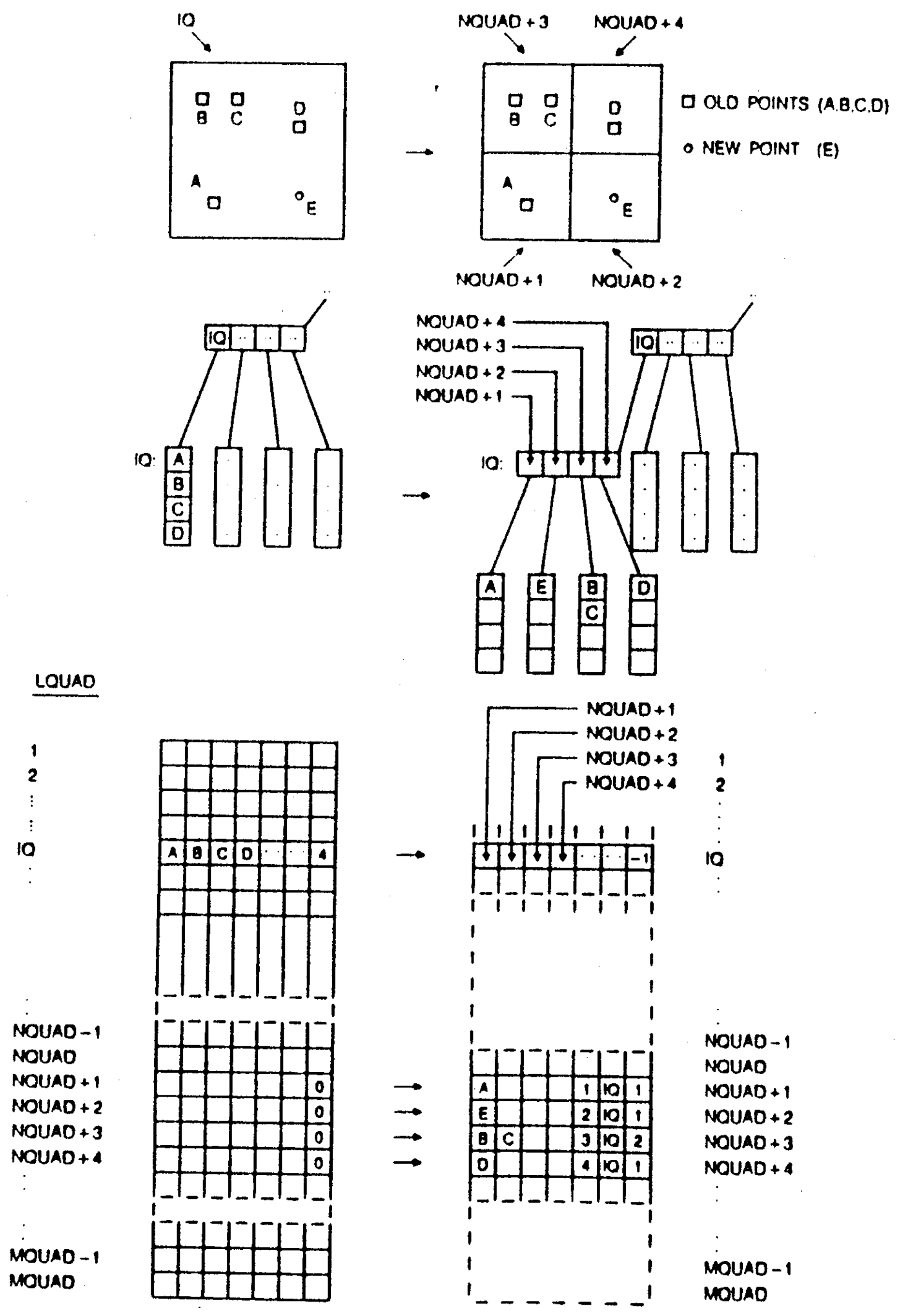
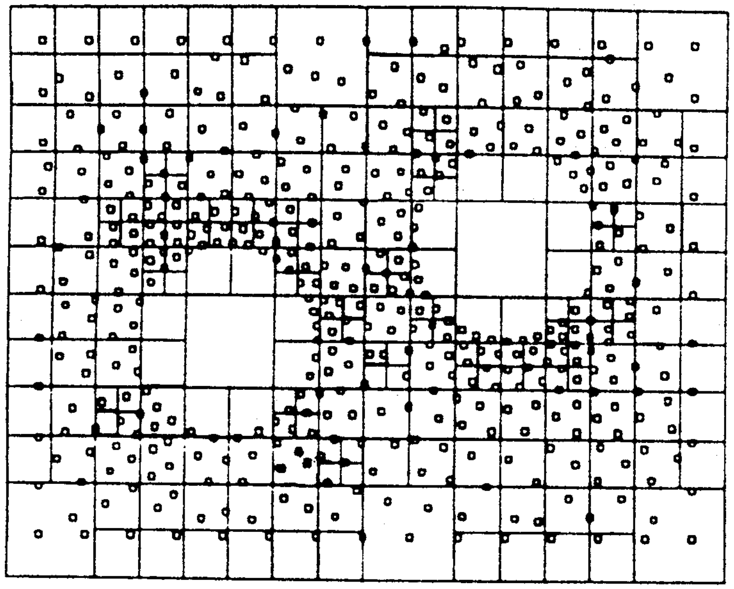


Figure 12. A Quadtree Structure


(a)

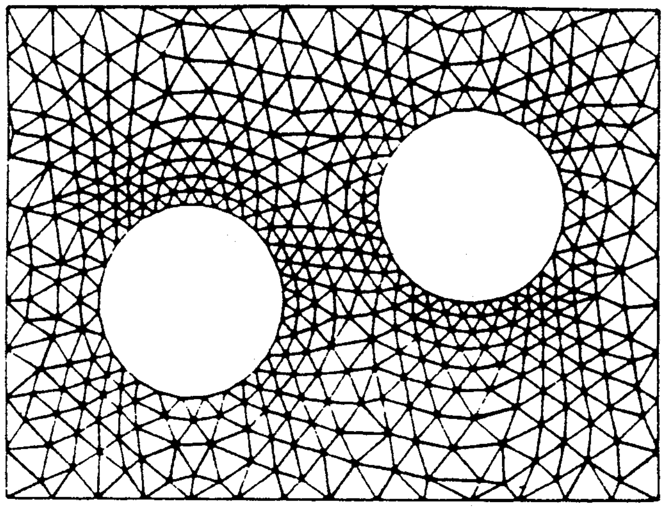

(b)

Figure 13. A Quadtree Example for an Unstructured Triangular Mesh Around a Simple Geometry (a) Quadtree (b) mesh

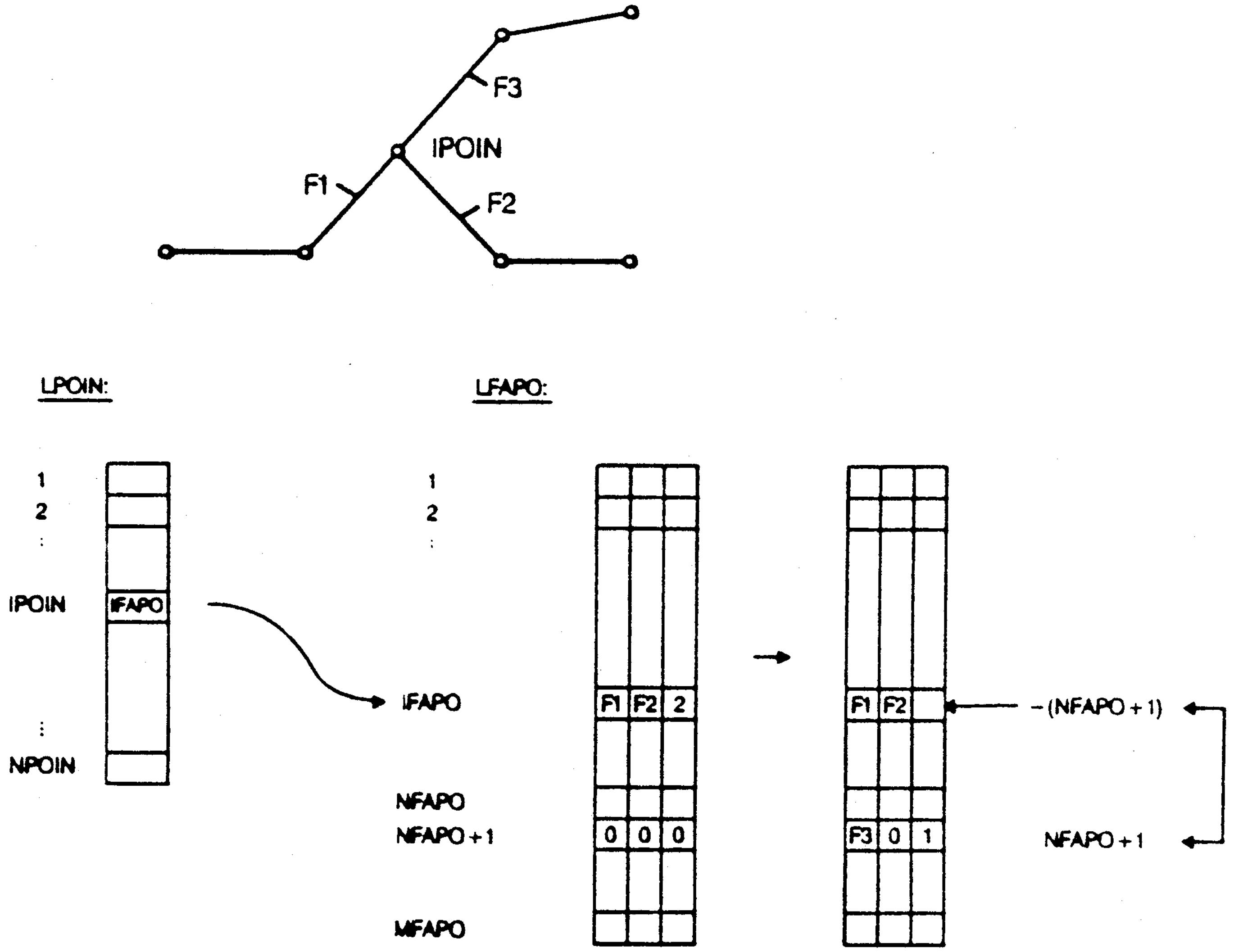


Figure 14. A Linked List Structure

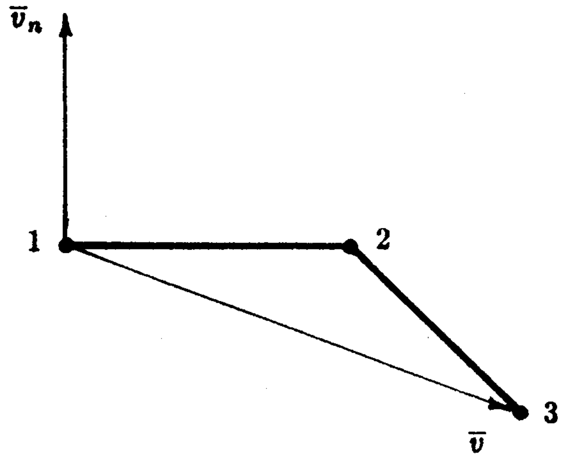


Figure 15. Point 3 in the 'Wrong' Side of the Face to be Removed (Face 1-2)

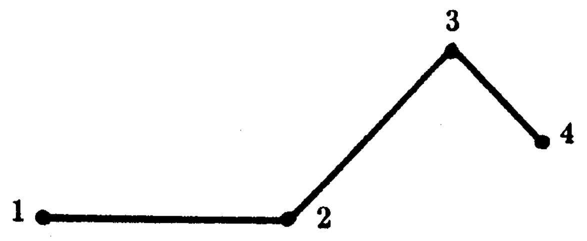


Figure 16. Point 4 in the 'Wrong' Side of the Adjacent Face (2-3) to the Face to be Removed (Face 1-2)

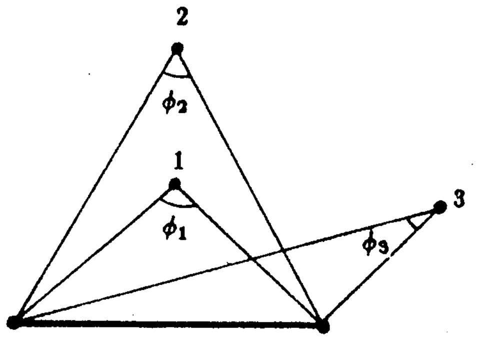


Figure 17. Apex Angles Corresponding to points 1, 2, and 3 (2-D)

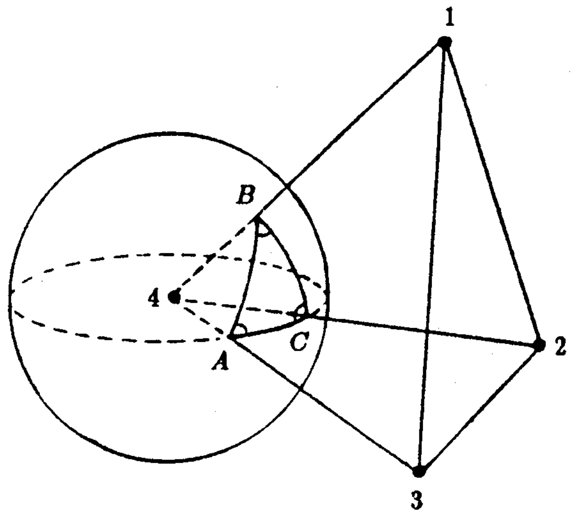


Figure 18. Spherical Angles A, B, and C

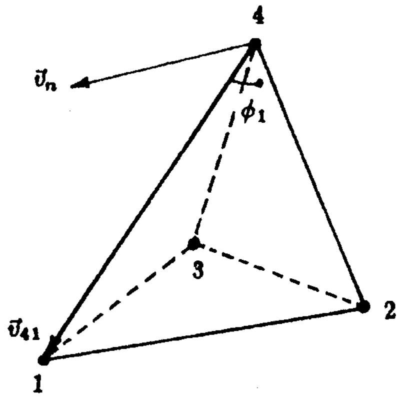


Figure 19. An Apex Angle Corresponding to Point 4 (3-D)

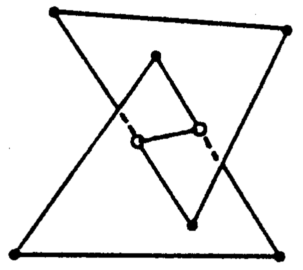


Figure 20. Crossing of Two Faces

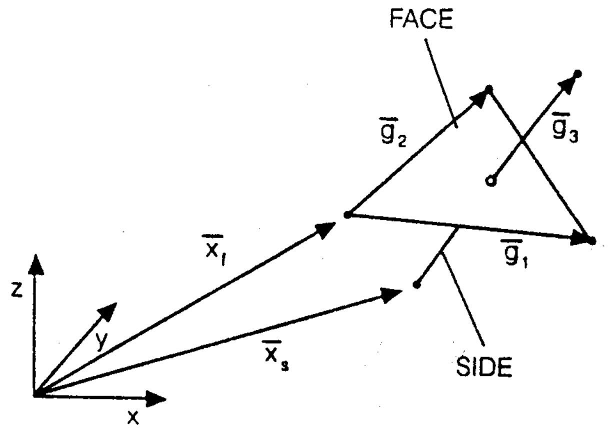


Figure 21. Face-Side Combination

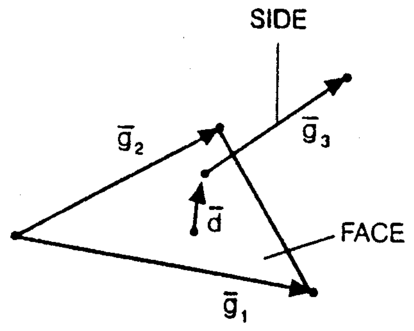


Figure 22. Distance Between Face and Side

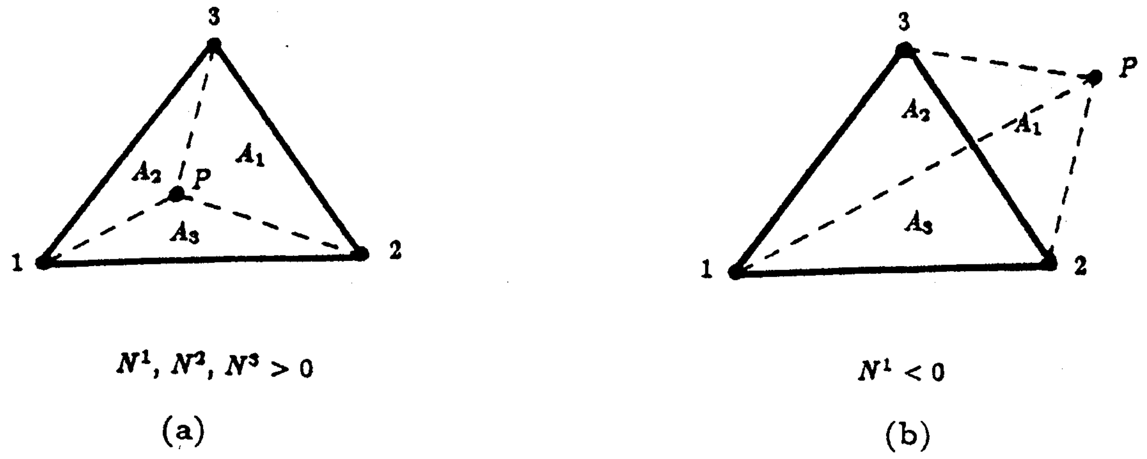
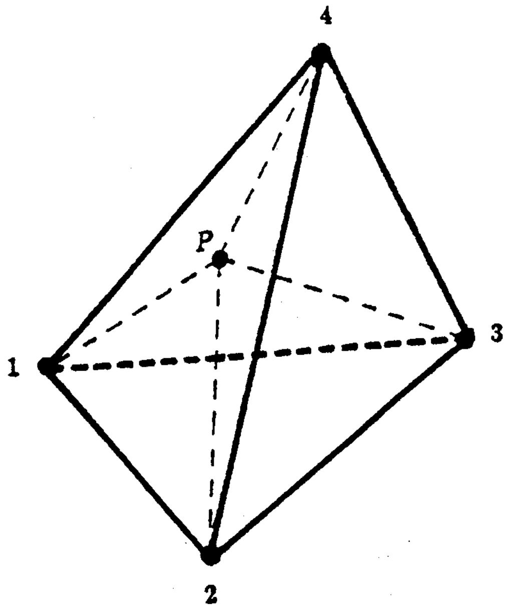


Figure 23. Point P (a) Inside and (b) Outside of Triangle 1-2-3



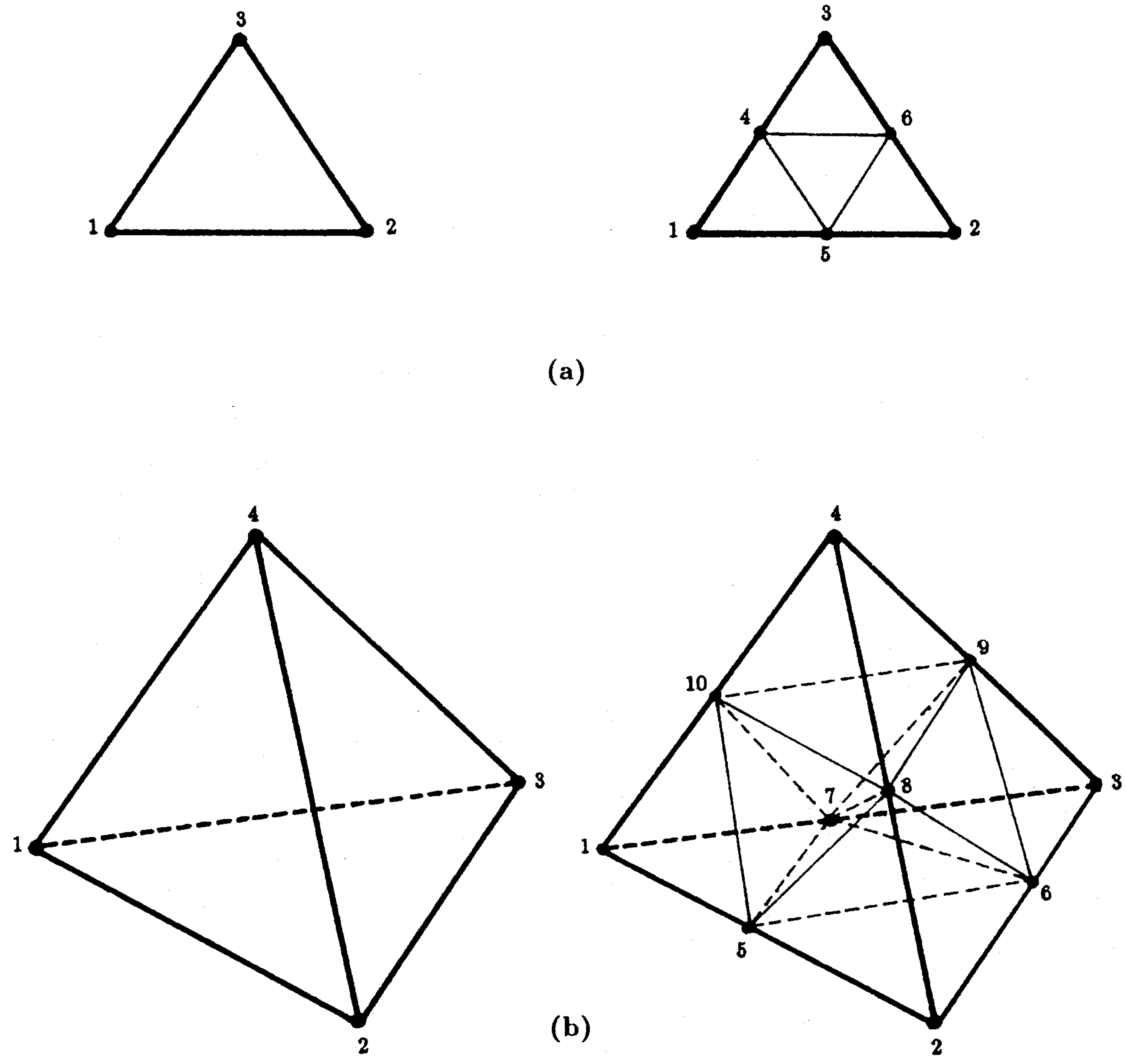
$$N^1 = \frac{V_{4-3-2-p}}{V} > 0$$

$$N^2 = \frac{V_{1-3-4-p}}{V} > 0$$

$$N^3 = \frac{V_{4-2-1-p}}{V} > 0$$

$$N^4 = \frac{V_{1-2-3-p}}{V}$$
$$= 1 - N^1 - N^2 - N^3 > 0$$

Figure 24. Point P Inside Tetrahedron 1-2-3-4



NEW ELEMENTS :

corner elements

10 – 8 – 9 – 4

10 – 7 – 5 – 1

5 – 6 – 8 – 2

6 – 7 – 9 – 3

center elements

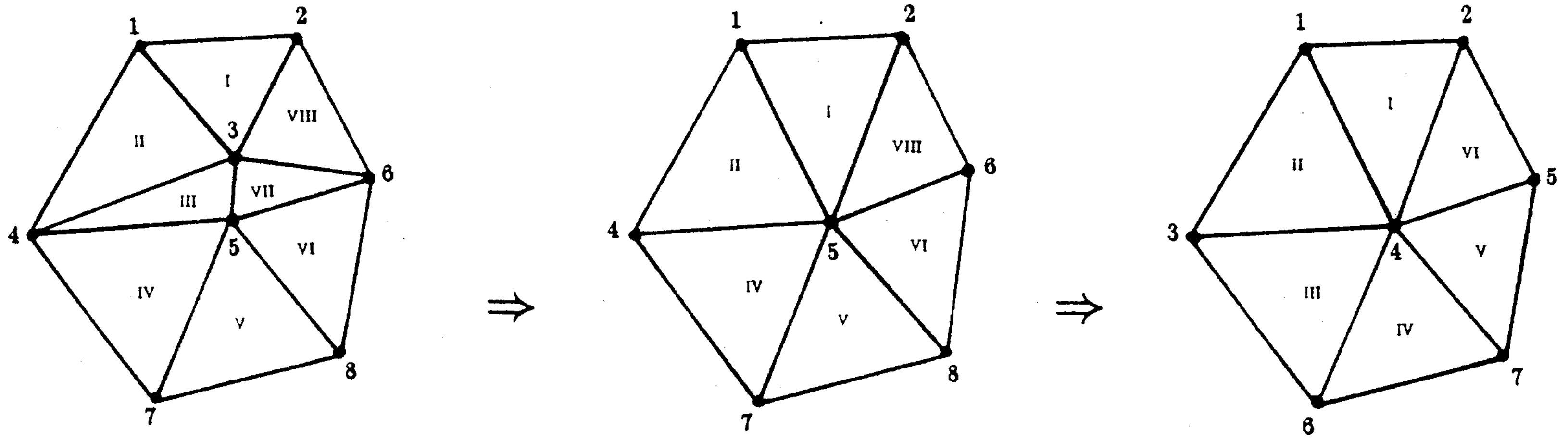
10 – 8 – 5 – 7

5 – 6 – 7 – 8

6 – 8 – 9 – 7

10 – 7 – 9 – 8

Figure 25. H-refinement: (a) Division of a Triangle into Four (b) Division of a Tetrahedron into Eight



to remove elemnet VII,
collapse side 3 – 5 *into point* 5,
(*a*)

remove elements
III *and* VII,
(*b*)

and renumber points
and elements
(*c*)

Figure 26. Removal of a Small Element

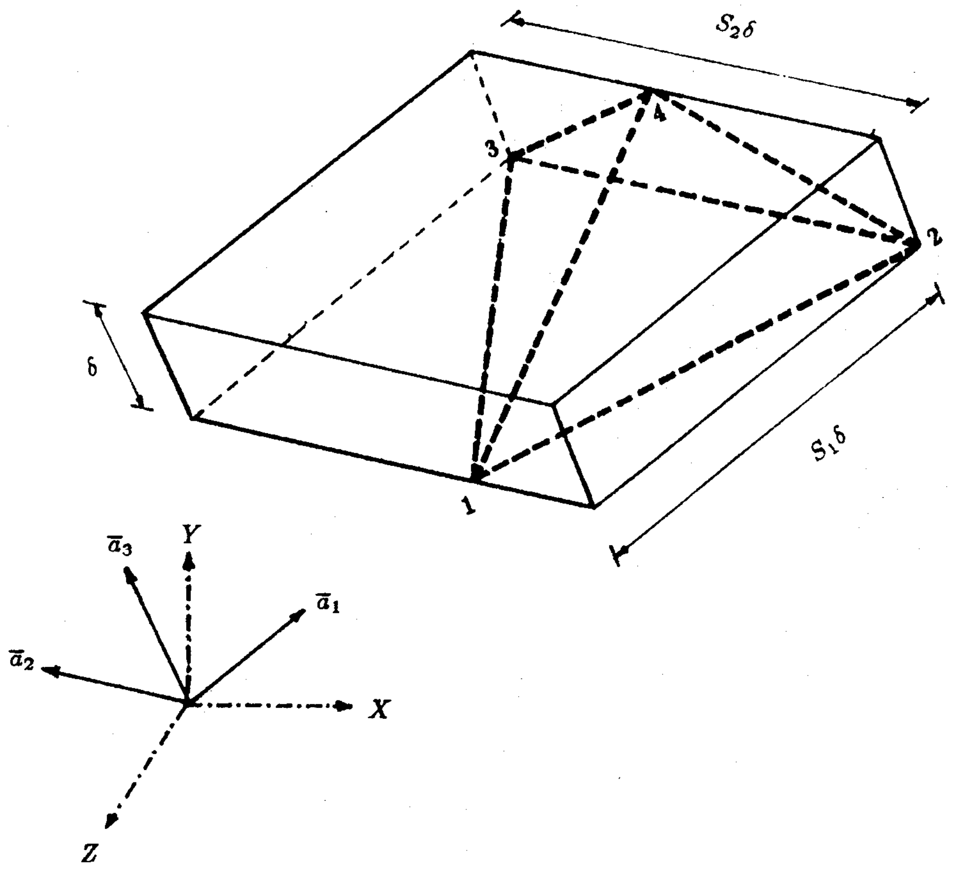
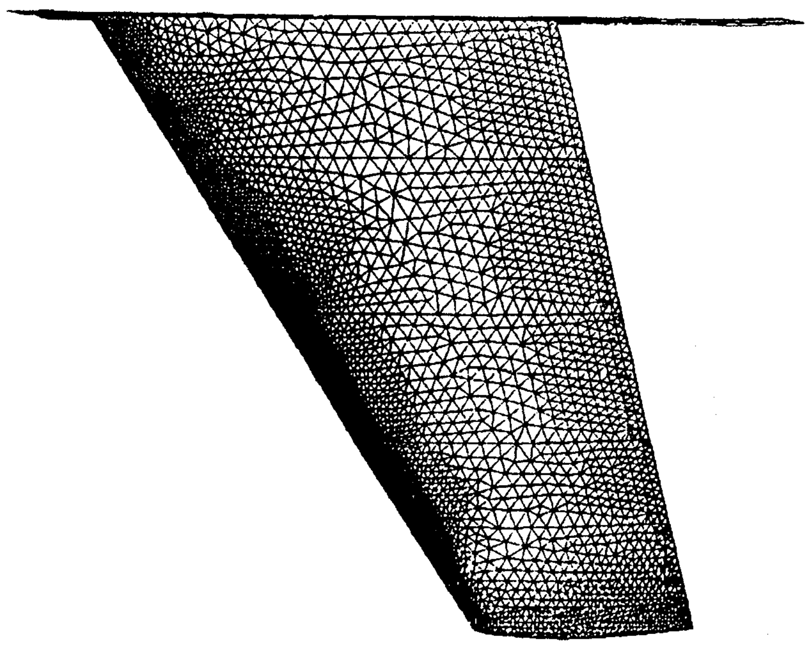
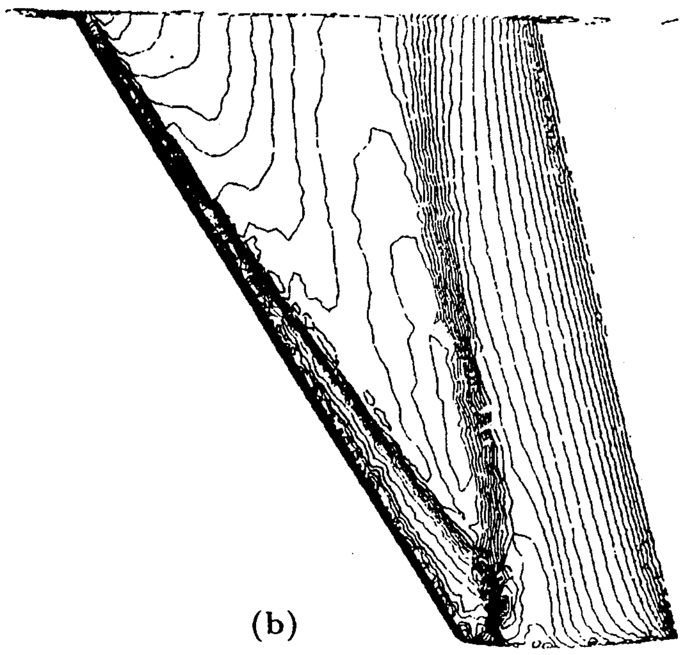


Figure 27. Grid Characteristics Parameters



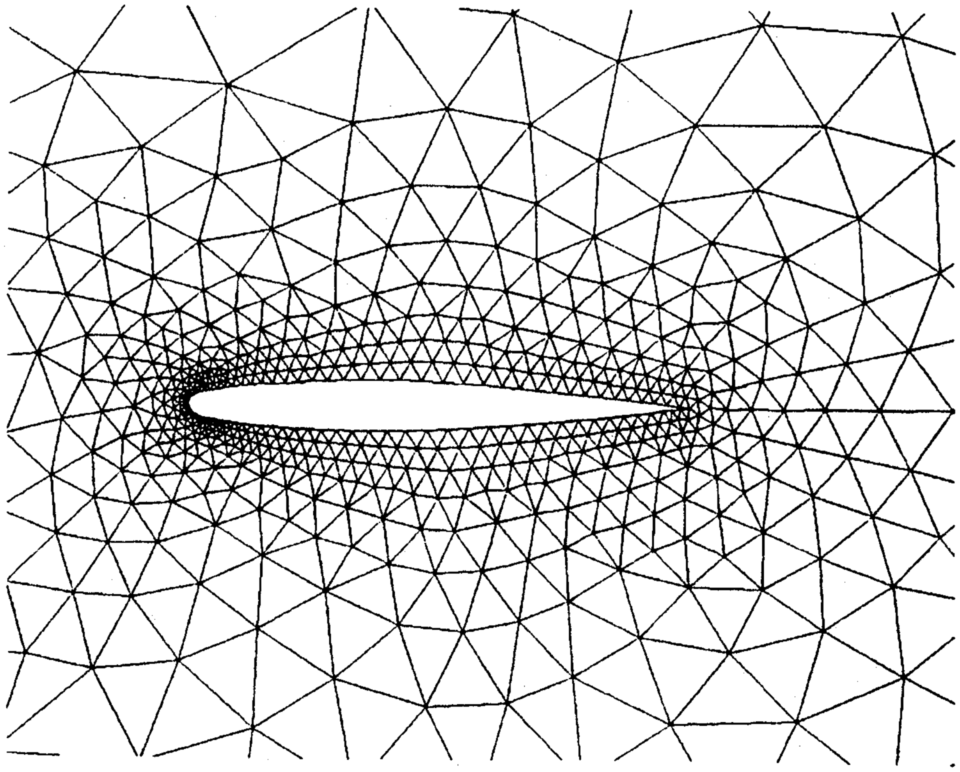
(a)



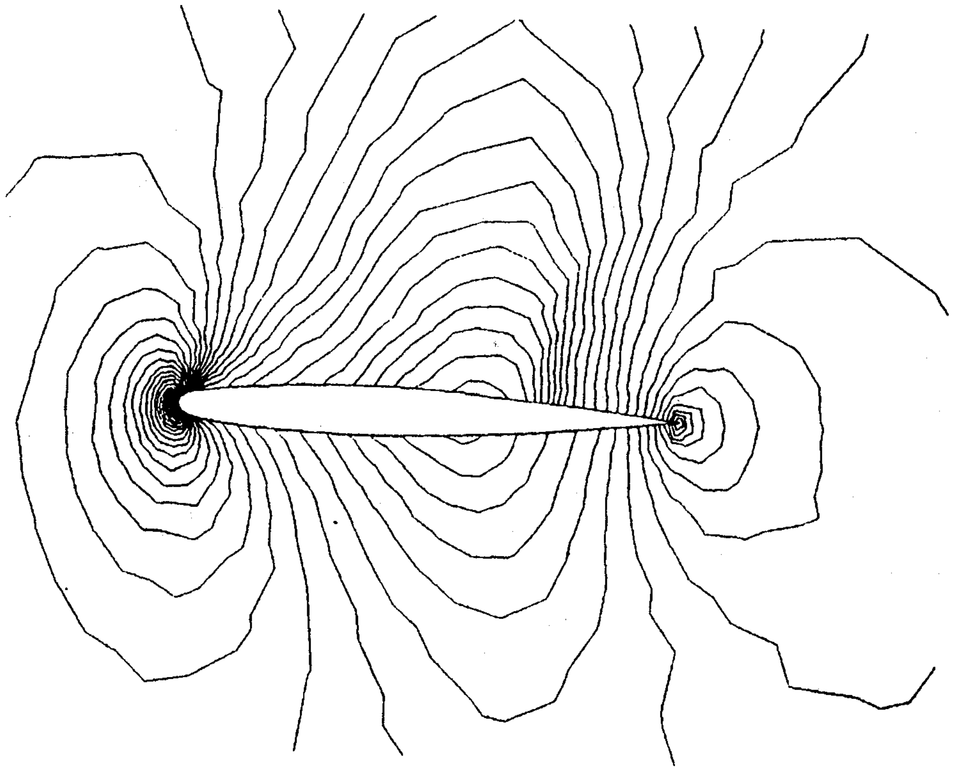
(b)

Figure 28. ONERA M6-Wing, M=0.84, α=3.06^o, Upper Surface (a) Triangulation (b) Pressure Contours

C-2



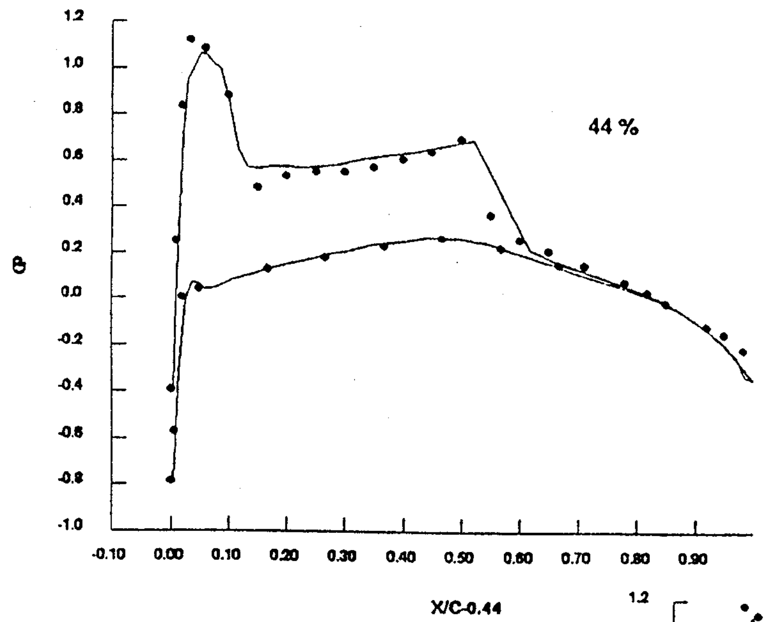
(a)

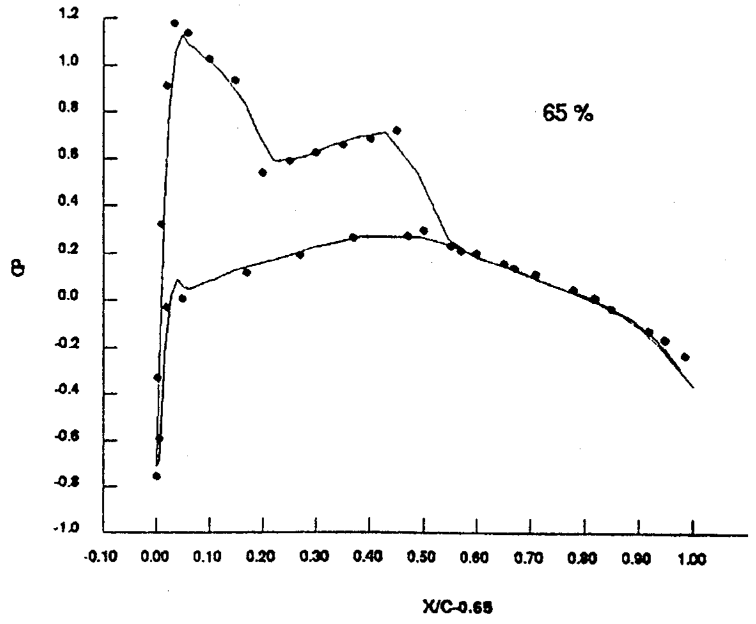


(b)

Figure 29. ONERA M6 Wing, M=0.84, α=3.06°, Symmetry Plane

(a) Triangulation (b) Pressure Contours





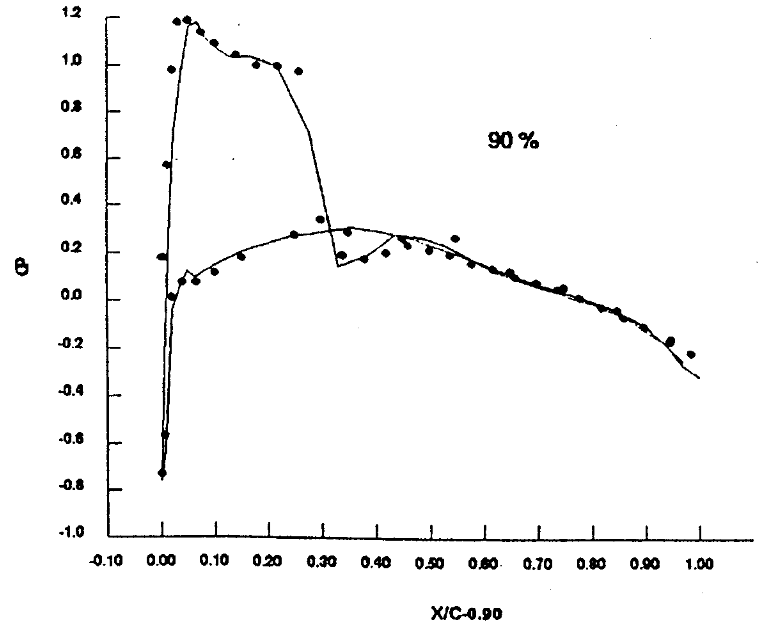


Figure 30. Comparison of Surface Pressure Coefficients with Experimental Data

APPENDIX

A sample input file for the grid generator is given here for an ONERA M6 wing. The wing configuration comprises of 36 surface patches made of 69 lines and 675 points. For brevity only partial listing of the file is shown.

The initial section specifying the title etc., is followed by the coordinates of the points. Next, information regarding lines is given. Lines 1,2,and 3 are cubic splines while the last three lines shown, are straight lines. The information regarding the surfaces follows next. It may be noticed that boundary layer grid is specified on surface number 1 with 15 points within the boundary layer, distributed in a geometric progression (NTYBL = 1) and the distance between two consecutive points is specified to be increasing at a ratio of 1.2. Please also notice that on surfaces 32,33,34 and 36, characteristic boundary condition is specified with 'characteristic inflow set' numberd as 1. As mentioned in Chapter 3, the actual values of the variables to be used for "characteristic inflow analysis" need only be specified during execution of the flow solver. Surface number 35 is the inflow plane for the computational domain, hence a boundary condition of (-1,0,0) is specified corresponding to no change in unknowns. The two lines, 50 and 57, are the lines forming the wing upper and the lower surface in the plane of symmetry. So these lines are specified to have tangential boundary condition. Similarly, the six lines listed under LLITA are the lines forming the trailing edge of the wing and hence, are specified to have 'free' boundary condition.

The background grid information is self explanatory, except that on points 1 and 2 of the background grid, a boundary layer thickness of 0.1 is specified. The desired mirroring of the 3-D grid is specified by NMIRR = 1, while the values of X0VEC and XNVEC provide for a mirroring across a X-Y plane located at the origin. The run is specified to generate a coarse grid, FACT = 2.0, and do one h-refinement.

SAMPLE INPUT FILE FOR VGRID3D

```
ONERA M6 WING
NDIMN      NPOIN      NLINE      NSURF
    3        675         69         36
COORDINATES OF THE POINTS
   1  0.57499999E+00  0.00000000E+00  0.10000000E+01
   2  0.57520121E+00  0.00000000E+00  0.10003500E+01
   3  0.57663625E+00  0.00000000E+00  0.10021501E+01
   4  0.57911402E+00  0.00000000E+00  0.10043200E+01
   5  0.58245730E+00  0.00000000E+00  0.10064300E+01
   6  0.58663100E+00  0.00000000E+00  0.10082800E+01
   .
   .
   .
   .
   .
 667  0.67369998E+00 -0.47999999E-03  0.00000000E+00
 668  0.11000000E+02  0.65000000E+01  0.00000000E+00
 669 -0.65000000E+01  0.65000000E+01  0.00000000E+00
 670 -0.65000000E+01 -0.65000000E+01  0.00000000E+00
 671  0.11000000E+02 -0.65000000E+01  0.00000000E+00
 672  0.11000000E+02  0.65000000E+01  0.25000000E+01
 673 -0.65000000E+01  0.65000000E+01  0.25000000E+01
 674 -0.65000000E+01 -0.65000000E+01  0.25000000E+01
 675  0.11000000E+02 -0.65000000E+01  0.25000000E+01
LINES: NUMBER  NTYLI     #POINTS    LIST
         1       3          6    1    2    3    4    5    6
         2       3          7    6    7    8    9   10   11   12
         3       3          6   12   13   14   15   16    1
         .
         .
         .
         .
         .
         .
        67      1          2   673  669
        68      1          2   674  670
        69      1          2   675  671
SURFACES: NUMBER NTYSU  #LINES    LIST                B.COND.     B.LAYER
            1      4      3         3   1   2          0  0  0    15  1  1.2
            2      4      3         4   5  -1          3  0  0     0  0  0.
            3      5      4         8  -2   6     7    3  0  0     0  1  0.
            .
            .
            .
            .
           32      1      4        62  67 -58 -66    4  1  0     0  0  0.
           33      1      4       -62 -67 -64 -63    4  1  0     0  0  0.
           34      1      4        64  69 -60 -68    4  1  0     0  0  0.
           35      1      4        63  68 -59 -67   -1  0  0     0  0  0.
           36      1      4        65  66 -61 -69    4  1  0     0  0  0.
 NLITA
 2
 LLITA
 50  57
 NLIFR
 7
 LLITA
 49  46  43  40  37  34  25
 BACKGROUND GRID DATA
 NELEB   NPOINB
   612      154
INTMAB
             1           1           3           2           23
             2           1           2           4           23
             3           2           5           4           24
             .
```

```
                .
                .
                .
                .
               608         152         153         131         144
               609         144         153         131         143
               610         153         154         132         143
               611         143         154         132         141
               612         147         132         154         141
 COORDINATES OF THE BACKGROUND GRID NODES
   1  0.00000E+00  0.00000E+00  0.00000E+00
   2  0.50127E-01  0.00000E+00  0.00000E+00
   3  0.24132E-01 -0.30325E-01  0.00000E+00
   .
   .
   .
   .
 152  0.11000E+02  0.65000E+01  0.25000E+01
 153  0.67758E+00  0.65000E+01  0.25000E+01
 154 -0.65000E+01  0.65000E+01  0.25000E+01
 UNKNOWNS AT THE NODES - UNKNB
  1 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 4.000E-03 1.0 1.0 0.1
  2 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 4.000E-03 1.0 1.0 0.1
  3 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 8.000E-03 1.0 1.0 0.
  4 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 8.000E-03 1.0 1.0 0.
  5 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 1.000E-02 1.0 1.0 0.
  .
  .
  .
  .
151 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 2.500E+00 1.0 1.0 0.
152 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 2.600E+00 1.0 1.0 0.
153 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 2.500E+00 1.0 1.0 0.
154 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 2.600E+00 1.0 1.0 0.
 MIRROR -NMIRR
 1
 X0VEC
 0.  0.  0.
 XNVEC
 0.  0.  1.
 NHREF   FACT
 1       2.00
```

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CR-182090	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A Package for 3-D Unstructured Grid Generation Finite-Element Flow Solution and Flow Field Visualization		5. Report Date September 1990
		6. Performing Organization Code
7. Author(s) Paresh Parikh, Shahyar Pirzadeh and Rainald Löhner		8. Performing Organization Report No.
		10. Work Unit No. 324-02-00
9. Performing Organization Name and Address ViGYAN, Inc. 30 Research Drive Hampton, VA 23666-1325		11. Contract or Grant No. NAS1-18670
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Clyde Gumbert
Final Report - SBIR Phase II

16. Abstract

A set of computer programs for 3-D unstructured grid generation, fluid flow calculations and flow field visualization has been developed. The grid generation program, called VGRID3D, generates grids over complex configurations using the advancing front method. In this method, the point and element generation is accomplished simultaneously, VPLOT3D is an interactive, menu-driven pre- and post-processor graphics program for interpolation and display of unstructured grid data. The flow solver, VFLOW3D, is an Euler equation solver based on an explicit, two-step, Taylor-Galerkin algorithm which uses the Flux Corrected Transport (FCT) concept for a wriggle-free solution. Using these programs, increasingly complex 3-D configurations of interest to aerospace community have been gridded including a complete Space Transportation System comprising of the space-shuttle orbitor, the solid-rocket boosters, and the external tank. Flow solutions have been obtained on various configurations in subsonic, transonic, and supersonic flow regimes.

17. Key Words (Suggested by Author(s)) Unstructured Grids Finite Element Solver Computer Graphics		18. Distribution Statement Unclassified - Unlimited Subject Category 02	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 104	22. Price A06

NASA FORM 1626 OCT 86