INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 8, 1135–1149 (1988)

GENERATION OF THREE-DIMENSIONAL UNSTRUCTURED GRIDS BY THE ADVANCING-FRONT METHOD

RAINALD LÖHNER

Berkeley Research Associates, Springfield, VA 22150, U.S.A.
and
Laboratory for Computational Physics and Fluid Dynamics, Naval Research Laboratory, Washington, DC 20375, U.S.A.

AND

PARESH PARIKH

Vigyan Research Associates, Hampton, VA 23666, U.S.A.

SUMMARY

The generation of three-dimensional unstructured grids using the advancing-front technique is described. This technique has been shown to be effective for the generation of unstructured grids in two dimensions.[1, 2] However, its extension to three-dimensional regions required algorithms to define the surface and suitable data structures that avoid excessive CPU-time overheads for the search operations involved. After obtaining an initial triangulation of the surfaces, tetrahedra are generated by successively deleting faces from the generation front. Details of the grid generation algorithm are given, together with examples and timings.

KEY WORDS Three-dimensional unstructured grids Advancing-front technique Grid generation

1. INTRODUCTION

In recent years a wide variety of algorithms has been devised for the generation of unstructured grids around complex geometrical shapes. Among the different techniques are Watson's algorithm for Voronoi tesselations,[3–8] the modified octree method[9] and different advancing-front techniques.[1, 2, 10–12] Baker's implementation and optimization of the Voronoi algorithm[8] has shown that fast and reliable grid generators for tetrahedral meshes can be produced. We currently believe that the version of the advancing-front technique put forward by Peraire and Morgan[1] is the best approach, because it can be used for grid regeneration with directional refinement.[2] The incorporation of directional refinement in the Voronoi algorithm appears difficult unless the reconnection of points based on the purely geometrical Delauney criterion[8] is substituted by some other criterion that incorporates directionality into the triangulation. This, however, would destroy the uniqueness of the triangulation. Directional refinement is an essential ingredient in any optimal 3D algorithm for compressible flows.

2. ALGORITHMIC STEPS OF ADVANCING-FRONT GENERATORS

The advancing-front grid generation technique[1, 2, 12] consists of the following steps:

0271–2091/88/101135–15$07.50 Received 2 November 1987

F1 Set up a background grid to define the spatial variation of the size, the stretching and the stretching direction of the elements to be generated. The background grid consists of tetrahedrons. At the nodes we define the desired element size, element stretching and stretching direction. This background grid must completely cover the computational domain.
F2 Define the boundaries (surfaces) of the domain to be gridded.
F3 Using the information stored on the background grid, set up faces on all these boundaries. This yields the initial front of triangular faces. At the same time, find the generation parameters (element size, element stretching and stretching direction) for these faces from the background grid.
F4 Select the next face to be deleted from the front. In order to avoid large elements crossing over regions of small elements, the face forming the smallest new element is selected as the next face to be deleted from the list of faces.
F5 For the face to be deleted:
  F5.1 Select a 'best point' position for the introduction of a new point IPNEW.
  F5.2 Determine whether a point exists in the already generated grid that should be used in lieu of the new point. If there is such a point, set this point to IPNEW and continue searching (go to F5.2).
  F5.3 Determine whether the element formed with the selected point IPNEW does not cross any given faces. If it does, select a new point as IPNEW and try again (go to F5.3).
F6 Add the new element, point and faces to their respective lists.
F7 Find the generation parameters for the new faces from the background grid.
F8 Delete the known faces from the list of faces.
F9 If there are any faces left in the front, go to F4.

3. BACKGROUND GRID

The background grid consists of tetrahedra and is used to define the desired spatial variation of element size, element stretching and stretching direction. The background grid typically consists of only a few (10–20) elements, so that manual interactive input with a mouse on a workstation presents no significant burden. To construct a uniform mesh, the background grid consists of only one element that covers the domain. If a rapid and significant change of element size is desired, the grid generator itself may be used first to generate a background grid. At the nodes of this generated grid we then define the desired element size, element stretching and stretching direction, and proceed to generate the final mesh. Within an adaptive refinement process, the current grid and flow solver solution are used as the background grid to generate a new, better grid for the flow problem under consideration.

4. DEFINITION OF SURFACES

The advancing-front algorithm requires an initial front in order to start the tetrahedrization of the domain. This in turn means that the surfaces defining the domain to be gridded need to be defined. Two approaches are possible here:[13]

(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 (union, intersection, etc.) 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, because the desired surface of the domain is obtained immediately, and not in a post-processing step, the software involved is less complex. We chose the second approach because of the convenience of its use in external aerodynamics problems. In order to minimize the manual effort involved in defining a surface, a hierarchical data structure was adopted. Three levels of data are allowed: points, lines and surfaces. Lines are obtained by joining points, and surfaces by joining lines.

The line types implemented are:

(1) straight line segment (defined by two points);
(2) parabolic line segment (defined by three points);
(3) cubic spline segment (defined by four or more points).

The surface types implemented are:

(1) planar segment (assumes all line segments lie in one plane);
(2) triangular isoparametric parabolic surface[14] (defined by three parabolic line segments);
(3) rectangular isoparametric serendipity surface[14] (defined by four parabolic line segments);
(4) triangular Barnhill–Gregory–Nielson patch[15] (defined by three arbitrary line segments);
(5) bilinear transfinite Coon's patch[15] (defined by four arbitrary line segments).

In this paper we focus the attention on grid generation. Therefore we only mention these line segments and surface types. A description of the mapping functions used for the surface elements may be found in the Appendix.

5. GENERATION OF THE INITIAL FRONT

The generation of the initial front or surface triangulation is carried out in two main steps:

(a) the line segments are divided into straight line segments, called sides;
b) the surface segments are triangulated, starting from the sides of the corresponding line segments.

5.1. Generation of sides

Each line segment that connects surface patches is subdivided into straight line segments, called sides. The length of each side is determined by interpolation from the background grid and can vary arbitrarily along the line. The addition of a further side along a line segment is carried out iteratively. We proceed as follows:

S1 Given the current location $\mathbf{x}_0$, determine from the background grid and the tangential vector of the line the vector $\mathbf{dx}_0$ defining the next side.

S2 Add the new side tentatively. Determine from the background grid and the tangential vector of the line the corrected vector $\mathbf{dx}_1$ at the midpoint $\mathbf{x}_1 = \mathbf{x}_0 + 0{\cdot}5\, \mathbf{dx}_0$ of the newly created side.

S3 If $dx_0 \not\approx dx_1$, set $dx_0 = 0{\cdot}5(dx_0 + dx_1)$ and go back to S2.
S4 Add the new side, determine the new location x_0 and go back to S1.

This procedure is applied in turn to all line segments connecting surface patches.

5.2. *Triangulation of surface segments*

Each surface segment is triangulated independently using a 2D version of the advancing-front grid generator. The substeps are the same as outlined above (see Section 2). The initial front consists of the sides corresponding to the lines that connect the current surface segment to other neighbouring surface segments. The triangle size and stretching are computed from the 3D background grid by interpolation. As the triangulator resides in a 2D world, we need to reproduce as faithfully as possible the 3D surface in a 2D domain. To that end, mappings between the 2D and 3D worlds are used that maintain approximately the shape and size of the 3D surface triangles in 2D and *vice versa*. The mappings used are:

(1) 3D surface segment to unit 2D triangle or square $(x, y, z \rightarrow \xi, \eta)$;
(2) stretching and shearing of unit 2D triangle or square to approximate 3D surface segment in a 2D domain $(\xi, \eta \rightarrow \xi'', \eta'')$.

This mapping process is illustrated in Figure 1. The equations needed for the individual surface segments are compiled in the Appendix. When computing the element size during the triangulation, at each stage we proceed as follows:

T1 Transform the current 2D position to the 3D surface segment via the unit triangle or square $(\xi'', \eta'' \rightarrow \xi, \eta \rightarrow x, y, z)$.
T2 Determine from the background grid the desired element size and shape.
T3 Transform back the desired element size and shape to the 2D domain.

Transformation T1 is also used to transform back the new points that have been added due to the triangulation of the surface segment.

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 2. We base our checking algorithm on the following observation: two triangular faces do not intersect if no side of either face intersects

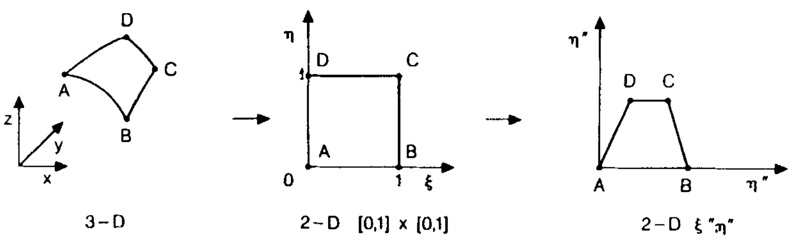


Figure 1. Mapping of surface patch to unit square with subsequent stretching and shearing

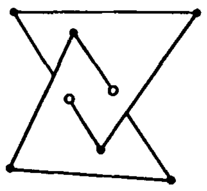


Figure 2. Crossing of two faces

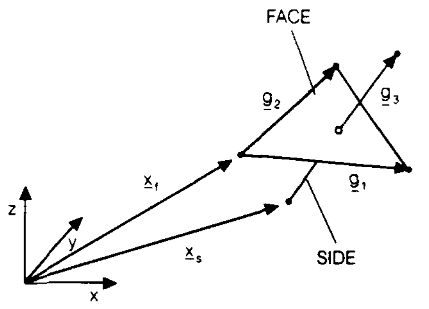


Figure 3. Face–side combination

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 3, the intersection point is found as

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

where we have used the $\mathbf{g}_1$-vectors as a covariant basis. Using the contravariant basis $\mathbf{g}^i$ defined by

$$\mathbf{g}^i\cdot\mathbf{g}_j=\delta^i_j, \tag{2}$$

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

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

Because we are only interested in a triangular surface for the $\mathbf{g}_1$, $\mathbf{g}_2$-plane, we define another quantity similar to the third shape function for a linear triangle:

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

Using the α^i, two faces can be considered as 'crossed' if they only come close together. Then, in order for the side not to cross the face, at least one of the α^i has to satisfy

$$t>\max(-\alpha^i,\alpha^i-1),\quad i=1,4, \tag{5}$$

where t is a predefined tolerance. By projecting the $\mathbf{g}_i$ onto their respective unit contravariant vectors, we can obtain the actual distance between a face and a side. The criterion given by equation (5) would then be replaced by (see Figure 4):

$$d>\frac{1}{|\mathbf{g}^i|}\max(-\alpha^i,\alpha^i-1),\quad i=1,4. \tag{6}$$

The first form (equation (5)) produces acceptable grids. If the face and the side have points in common, then the α^i will all be either 1 or 0. As both equation (5) and equation (6) will not be satisfied, we need to make special provision for these 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

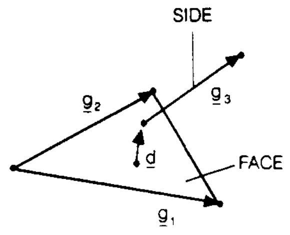


Figure 4. Distance between face and side

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 co-ordinates of each face. Faces cannot possibly cross each other if at least for one of the dimensions $i = 1, 2, 3$ they satisfy one of the following inequalities:

$$\max_{\text{face1}}(x^i_A, x^i_B, x^i_C) < \min_{\text{face2}}(x^i_A, x^i_B, x^i_C) - d, \tag{7a}$$

$$\min_{\text{face1}}(x^i_A, x^i_B, x^i_C) > \max_{\text{face2}}(x^i_A, x^i_B, x^i_C) + d, \tag{7b}$$

where A, B, C denote the corner points of each face.

(b) *Local element co-ordinates.* 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 have the same sign and lie outside the $[-t, 1+t]$ interval, then the tetrahedron cannot possibly cross the face. We therefore disregard this face.

(c) *In-depth analysis of side–face combinations.* All the faces remaining after the filtering process of steps (a) and (b) are analysed using 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 required only 25% of the total grid generation time. When operating on a vector machine, we perform loops over all the possible combinations, building the $\mathbf{g}_i$, $\mathbf{g}^i$, α^i, etc. in vector mode. Although the vector lengths are rather short, the chaining that results from the lengthy mathematical operations involved results in acceptable megaflop rates on the CRAY-XMP.

6. DATA STRUCTURES TO MINIMIZE SEARCH OVERHEADS

The operations that could potentially reduce the efficiency of the algorithm to $O(N^{1\cdot 5})$ or even $O(N^2)$ are (see Section 2).

(a) finding the next face to be deleted (step F4);
(b) finding the closest given points to a new point (step F5.2);
(c) finding the faces adjacent to a given point (step F5.3);
(d) finding for any given location the values of generation parameters from the background grid (steps F3 and F7)—this is an interpolation problem on unstructured grids.

The verb 'find' appears in all of these operations. The main task is to design the best data structures for performing the search operations (a)–(d) as efficiently as possible. The data structures used are:

(1) heap lists[16,17] to find the next face to be deleted from the front;
(2) quadtrees (2D) and octrees (3D)[9, 18] to locate points that are close to any given location;
(3) linked lists to determine which faces are adjacent to a point.

Combining these data structures, we can also derive an optimal interpolation algorithm for unstructured grids.[13] The detailed explanation of these data structures may be found in Reference 12.

7. SOME EXAMPLES AND TIMINGS

7.1. *Engine intake*

Figure 5 shows an engine intake with a centrebody. The intake cross-section varies from a square at the inlet to a circle at the outlet. The centrebody is assumed to start somewhere in the intake. Figure 5(a) shows the points and line segments defining the surfaces. Because of the symmetry, only half of the surfaces need to be defined. In Figure 5(b) the background grid is superimposed to this surface information. Figure 5(c) shows the surface triangulation obtained. In Figures 5(d) and 5(e) sections of tetrahedra along the main axis of symmetry are shown. The generated grid contains 4381 points and 22743 tetrahedra. The element volume in this case varied by more than a factor of 10^4, and the smallest element is located at the nose of the centrepiece.

7.2. *Missile launcher*

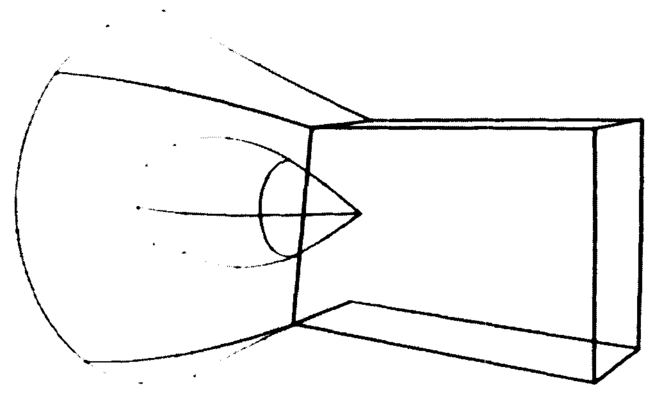
Figure 6 shows the model of a generic missile launcher. Figure 6(a) shows the points and line segments defining the surfaces. In Figure 6(b) the background grid is superimposed to this surface information. Figure 6(c) shows the surface triangulation obtained. In Figures 6(d) and 6(e) sections of tetrahedra along the main axis of symmetry are shown. The generated grid contains 3465 points and 17572 tetrahedra.

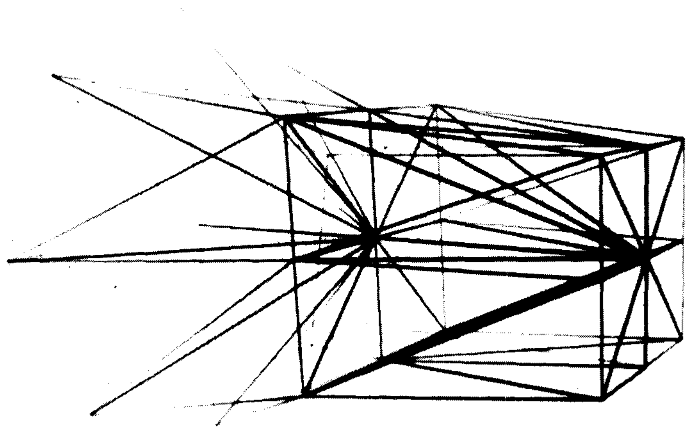
The time needed to generate a new element depends strongly on the number of faces checked for possible crossing (see above, Section 6). We find that for larger grids the number of faces checked decreases on average as the distance between 'colliding fronts' is larger. For the grids shown, the rate at which new tetrahedra were generated varied between 480 and 490 tetrahedra per second on the CRAY-XMP-24 at NRL (using one processor).

8. CONCLUSIONS

This paper describes a mesh generation procedure for three-dimensional regions. Input requirements to define objects or surfaces were minimized by adopting a hierarchical structure consisting of points, lines and surfaces. In order to reduce CPU requirements, several optimal search algorithms were adapted into the present context. Having demonstrated the validity of the approach, further enhancements can be envisioned. These are:

(1) easier interactive input of surface-defining information;
(2) enhancement of the surface patch library to achieve C^1 and C^2 continuity between surface patches;
(3) coupling of the grid generator with flow solvers;
(4) adaptive remeshing in 3D.


(a)


(b)

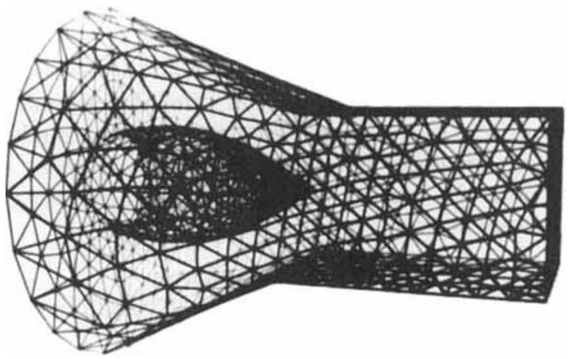
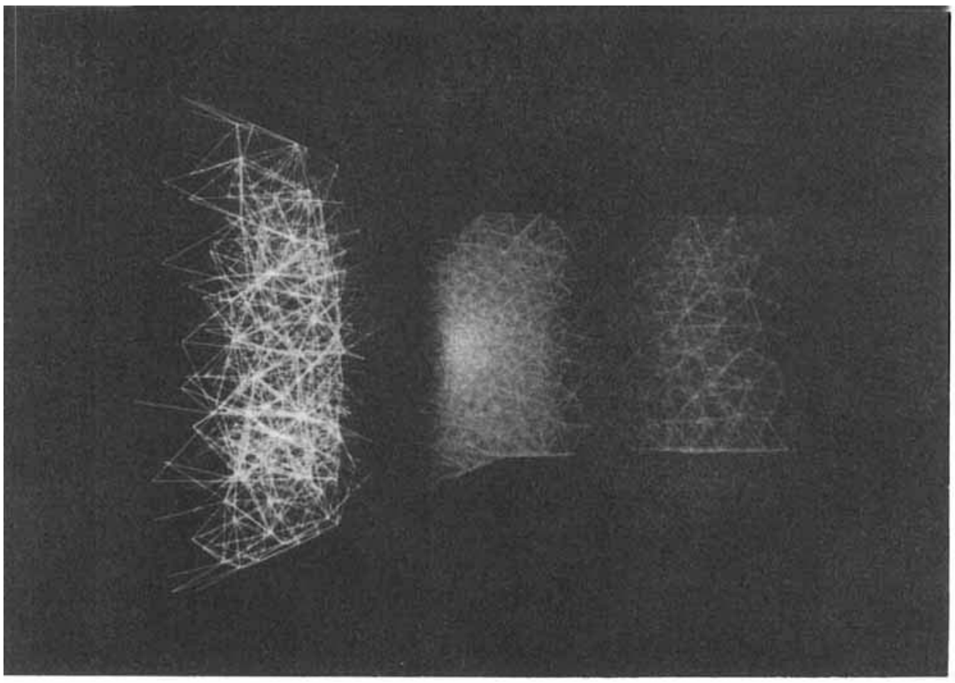

(c)

Figure 5. (a–c)


(d)

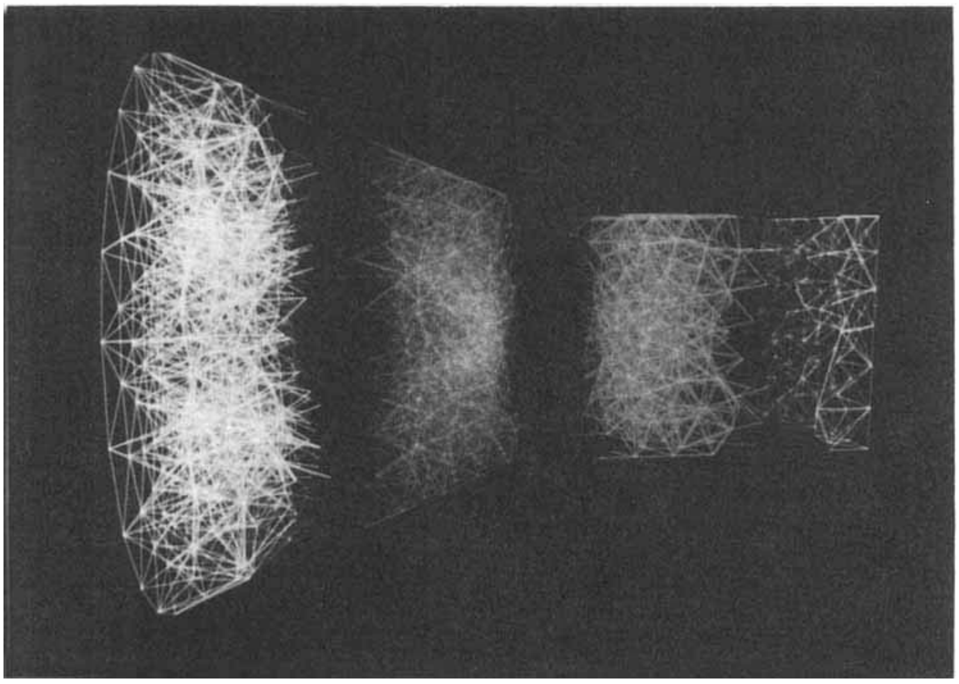
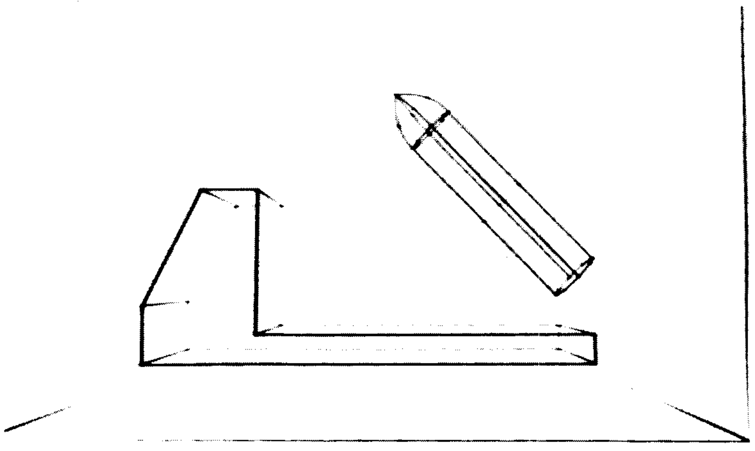
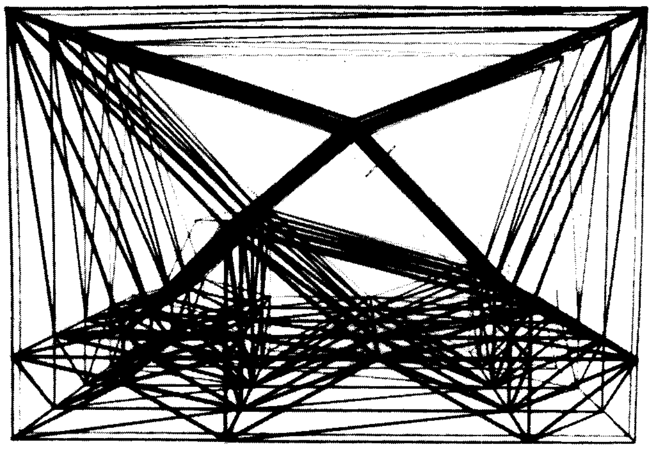

(e)

Figure 5. Engine intake: (a) Surface definition; (b) background grid; (c) surface triangulation; (d), (e) sections of tetrahedra along the main axis of symmetry


(a)


(b)

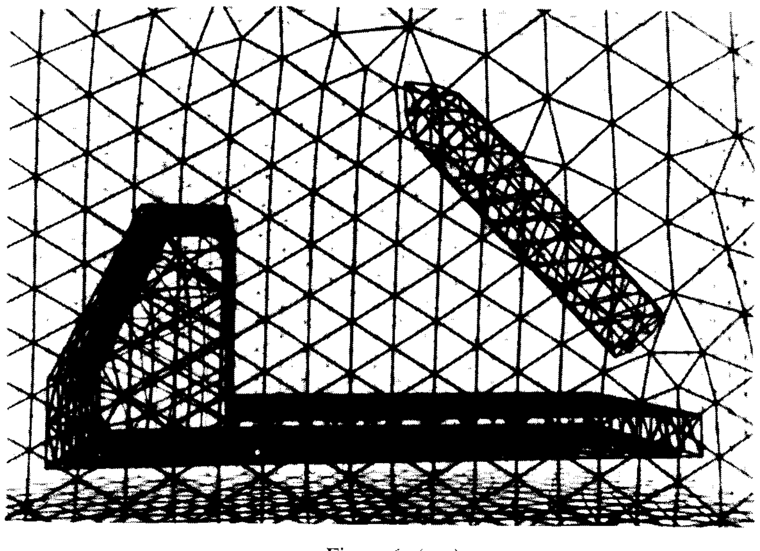
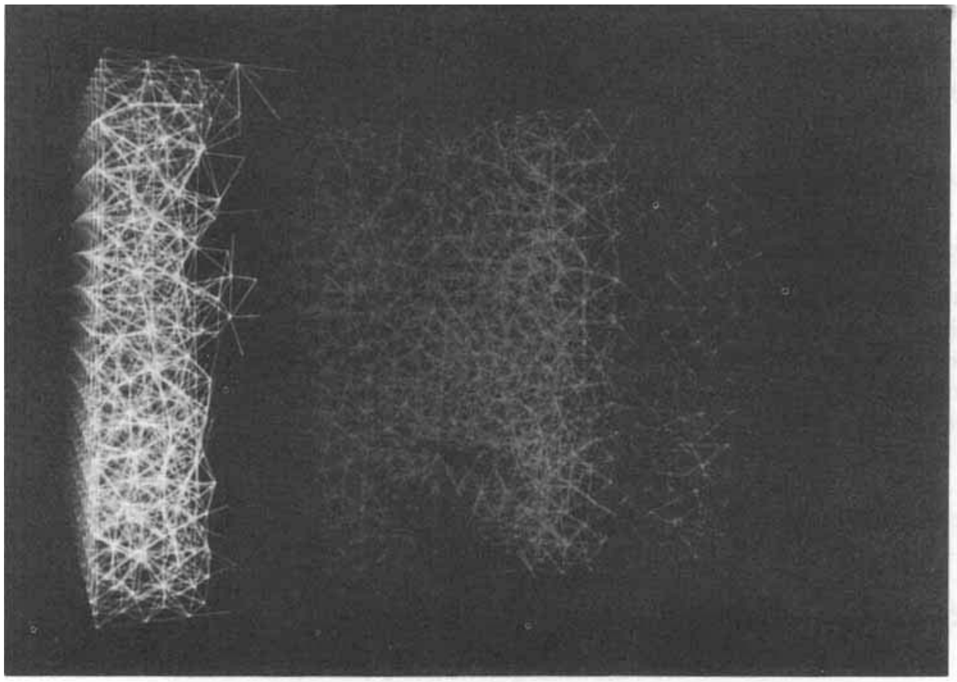
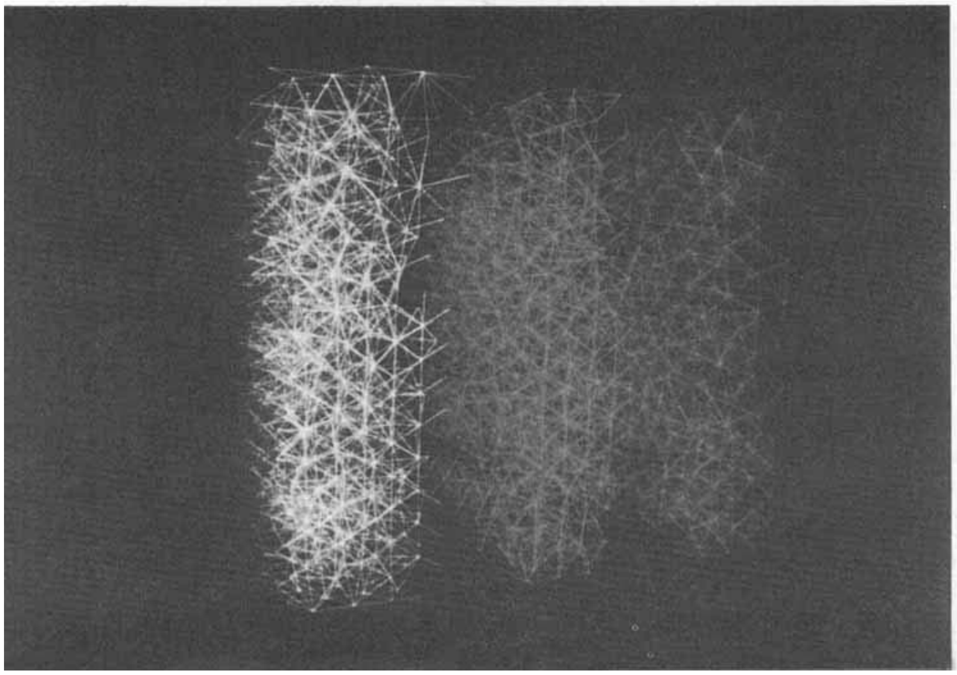

(c)

Figure 6. (a–c)



(d)



(e)

Figure 6. Generic missile launcher: (a) surface definition; (b) background grid; (c) surface triangulation; (d), (e) sections of tetrahedra along the main axis of symmetry

ACKNOWLEDGEMENTS

This work was partially funded by the Office Of Naval Research through the Naval Research Laboratory, Dr. C. P. Li of NASA Johnson Space Center and under SBIR-Grant No. NAS1-18419.

APPENDIX: MAPPING FUNCTIONS USED FOR THE SURFACE SEGMENTS

Mapping of 3D surface segment to 2D unit square ($x, y, z \to \xi, \eta$)

Planar surface segment. In the case of planar segments, stretching and shearing do not have to be applied, as we can transform directly. With the notation defined in Figure 7, we select two arbitrary vectors **a**, **b** that lie in the plane. The normal to the plane is then given by $\mathbf{n} = \mathbf{a} \times \mathbf{b}$, and a vector normal to both **a** and **n** can be computed as $\mathbf{c} = \mathbf{n} \times \mathbf{a}$. Defining the covariant basis vectors of the plane $\mathbf{x}_1$, $\mathbf{x}_2$ as

$$\mathbf{x}_1 = \mathbf{a}/|\mathbf{a}|, \qquad \mathbf{x}_2 = \mathbf{c}/|\mathbf{c}|, \tag{8}$$

any vector lying in the plane can be written as

$$\mathbf{x} = \mathbf{x}_0 + \xi''\mathbf{x}_1 + \eta''\mathbf{x}_2, \tag{9}$$

where ξ'', η'' values are given by

$$\xi'' = \mathbf{x}_1 \cdot (\mathbf{x} - \mathbf{x}_0), \qquad \eta'' = \mathbf{x}_2 \cdot (\mathbf{x} - \mathbf{x}_0). \tag{10}$$

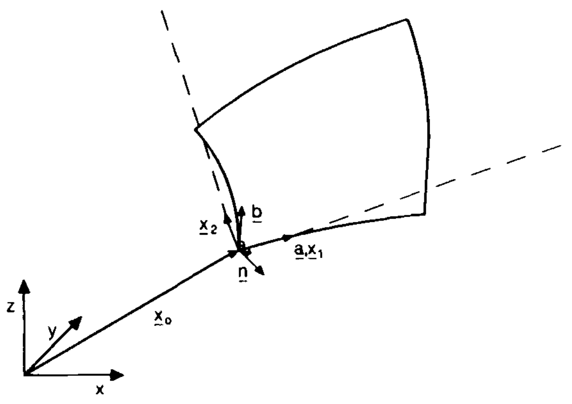


Figure 7. Covariant basis $\mathbf{x}_1$, $\mathbf{x}_2$ for planar surface patch

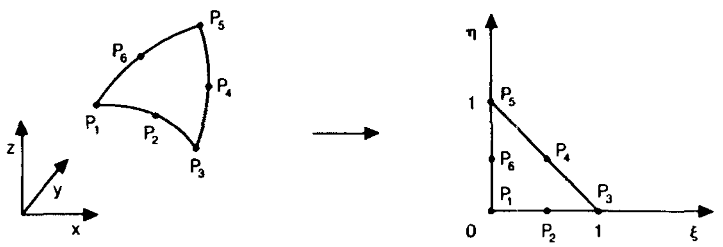


Figure 8. Triangular isoparametric parabolic surface

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 8. Any vector lying on the surface may be written as

$$\mathbf{x}=\sum_{i=1}^{6} N^i \mathbf{x}_i, \tag{11}$$

where the shape functions N^i are given by[14]

$$\begin{aligned}
N^1&=(1-\xi-\eta)(1-2\xi-2\eta),\\
N^2&=4\xi(1-\xi-\eta),\\
N^3&=\xi(2\xi-1),\\
N^4&=4\xi\eta,\\
N^5&=\eta(2\eta-1),\\
N^6&=4\eta(1-\xi-\eta).
\end{aligned} \tag{12}$$

Rectangular isoparametric serendipity surface segment. The surface segment is again most easily described in terms of the eight points defining it. The situation is shown in Figure 9. Any vector lying on the surface may be written as

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

where the shape functions N^i are given by[14]

$$\begin{aligned}
N^1&=(1-\xi)(1-\eta)(1-2\xi-2\eta),\\
N^2&=4\xi(1-\xi)(1-\eta),\\
N^3&=-\xi(1-\eta)(1-2\xi+2\eta);\\
N^4&=4\xi\eta(1-\eta),\\
N^5&=-\xi\eta(3-2\xi-2\eta),\\
N^6&=4\xi\eta(1-\xi),\\
N^7&=-\eta(1-\xi)(1+2\xi-2\eta),\\
N^8&=4\eta(1-\xi)(1-\eta).
\end{aligned} \tag{13}$$

Triangular Barnhill–Gregory–Nielson patch.[15] This surface segment is described in terms of the three line functions along its edges. The situation is shown in Figure 10. Any vector lying on the

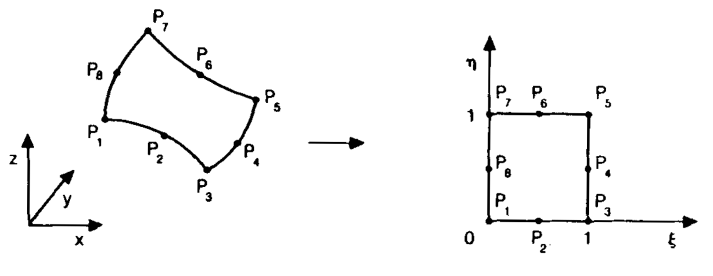


Figure 9. Rectangular isoparametric serendipity surface

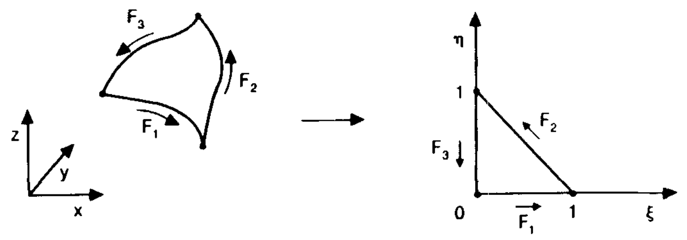


Figure 10. Triangular Barnhill–Gregory–Nielson patch

surface may be written as

$$\begin{aligned}\mathbf{x}=&\xi\,\mathbf{F}_2(\eta)+\eta\mathbf{F}_2(1-\xi)+\mathbf{F}_1(\xi)+\mathbf{F}_3(1-\eta)-\eta[\mathbf{F}_1(\xi)+\mathbf{F}_3(\xi)]\\&-\xi[\mathbf{F}_1(1-\eta)+\mathbf{F}_3(1-\eta)]-(1-\xi-\eta)\,\mathbf{F}_1(0).\end{aligned}\tag{14}$$

We observe that not all the sides are treated equally by this formula. The $\mathbf{F}_2$ 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.

Rectangular bilinear Coon's patch.[15] This surface segment is described in terms of the four line functions along its edges. The situation is shown in Figure 11. Any vector lying on the surface may be written as

$$\begin{aligned}\mathbf{x}=&(1-\eta)\mathbf{F}_1(\xi)+\xi\mathbf{F}_2(\eta)+\eta\mathbf{F}_3(\xi)+(1-\xi)\,\mathbf{F}_4(\eta)\\&-[(1-\xi)(1-\eta)\,\mathbf{F}_1(0)+\xi(1-\eta)\,\mathbf{F}_2(0)+\xi\eta\mathbf{F}_2(1)+\eta(1-\xi)\,\mathbf{F}_3(0)].\end{aligned}\tag{15}$$

Stretching and shearing 2D unit square ($\xi,\ \eta\rightarrow\xi'',\ \eta''$)

In order to obtain a reasonable triangulation of the curved surfaces, the unit square is stretched and sheared so as to approximate in 2D the 3D 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 2D surface approximation. This case is shown in Figure 1. 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

$$\xi''=x_B\xi+x_D\eta+(-x_B+x_C-x_D)\xi\eta,\qquad \eta''=y_D\eta,\tag{16}$$

or

$$\eta=\frac{\eta''}{y_D},\qquad \xi=\frac{\xi''-x_D\eta}{x_B+(-x_B+x_C-x_D)\eta}.\tag{17}$$

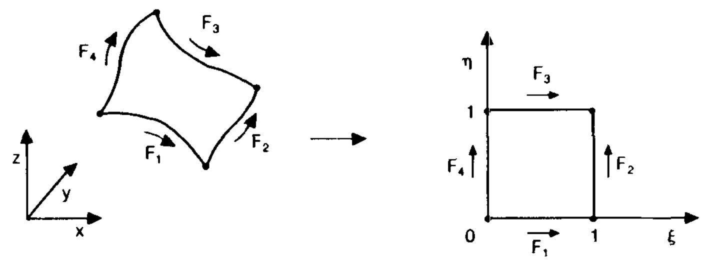


Figure 11. Bilinear transfinite Coon's patch

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 3D surface segment. For triangular surface segments point C is omitted.

REFERENCES

1. J. Peraire and K. Morgan, 'A general triangular mesh generator', *Int. j. numer. methods. eng.* (1987).
2. J. Peraire, M. Vahdati, K. Morgan and O. C. Zienkiewicz, 'Adaptive remeshing for compressible flow computations', *J. Comput. Phys.* **72,** 449–466 (1987).
3. D. F. Watson, Computing the *N*-dimensional Delaunay tesselation with application to Voronoi polytopes', *Comput. J.* **24** (2), 167–172 (1981).
4. A. Bowyer, 'Computing Dirichlet tesselations', *Comput. J.* **24** (2), 162–167 (1981).
5. S. W. Sloan and G. T. Houlsby, 'An implementation of Watson's algorithm for computing 2-dimensional Delaunay triangulations', *Adv. Eng. Software* **6**(4), 192–197 (1984).
6. A. Jameson, T. J. Baker and N. P. Weatherhill, 'Calculation of inviscid transonic flow over a complete aircraft', *AIAA-86-0103*, 1986.
7. A. Jameson and T. J. Baker, 'Improvements to the aircraft Euler method', *AIAA-87-0452*, 1987.
8. T. J. Baker, 'Three dimensional mesh generation by triangulation of arbitrary point sets', *AIAA-87-1124-CP*, 1987.
9. M. A. Yerry and M. S. Shepard, 'Automatic three-dimensional mesh generation by the modified-octree technique', *Int. j. numer. methods eng.* **20,** 1965–1990 (1984).
10. N. van Phai, 'Automatic mesh generation with tetrahedron elements', *Int. j. numer. methods. eng.* **18,** 237–289 (1982).
11. S. H. Lo, 'A new mesh generation scheme for arbitrary planar domains', *Int. j. numer. methods eng.* **21,** 1403–1426 (1985).
12. R. Löhner, 'Some useful data structures for the generation of unstructured grids', *Commun. Appl. Numer. Methods* **4,** 123–135 (1988).
13. J. Woodwark, *Computing Shape*, Butterworths, 1986.
14. O. C. Zienkiewicz and K. Morgan, *Finite Elements and Approximation*, Wiley, 1983.
15. R. E. Barnhill, 'Representation and approximation of surfaces', in J. R. Rice (ed.), *Mathematical Software III*, Academic Press, 1987, pp. 69–120.
16. D. N. Knuth, *The Art of Computer Programming, Vol. 3*, Addison-Wesley, 1973.
17. R. Sedgewick, *Algorithms*, Addison-Wesley, 1983.
18. H. Samet, 'The quadtree and related hierarchical data structures', *Comput. Surveys*, **16**(2), 187–285 (1984).