N89-20941

GENERATION OF UNSTRUCTURED GRIDS AND EULER SOLUTIONS FOR COMPLEX GEOMETRIES

Rainald Löhner
Berkeley Research Associates, Springfield, VA
and
Laboratory for Computational Physics and Fluid Dynamics
Naval Research Laboratory, Washington, D.C.

Paresh Parikh
Vigyan Research Associates, Hampton, VA

Manuel D. Salas
Theoretical Aerodynamics Branch
NASA Langley Research Center, Hampton, VA

ABSTRACT

We describe algorithms for the generation and adaptation of unstructured grids in two and three dimensions, as well as Euler solvers for unstructured grids. The main purpose of the paper is to demonstrate how unstructured grids may be employed advantageously for the economic simulation of both geometrically as well as physically complex flowfields.

1. INTRODUCTION

Over the past years the development of flow solvers for unstructured grids has progressed rapidly. While the main emphasis at the beginning was on Euler solvers [1-5], the current emphasis has shifted onto those fields which may be considered 'peripheral', yet very important for a complete simulation capability. These are: efficient implementation on vector and parallel machines [6], adaptive refinement [4,7-14], mesh generation [5,11,15-21] and flow visualization. In what follows, we describe briefly the capabilities we developed in this area. The main emphasis of the present paper in on applications. Therefore, rather than dwelling in depth on each subject, we will show examples to illustrate the relevant points.

2. THE GRID GENERATOR: GENERALIZED ADVANCING FRONT

Several algorithms for the generation of unstructured grids have been proposed in the literature. We may group them into two classes: a) those that introduce gridpoints before constructing the triangulation, and b) those that introduce gridpoints while constructing the triangulation. Members of the first family of schemes include the Voronoi/Delauney triangulation schemes [5,7,15,22-24], as well as the advancing-front algorithms [16,20]. The generalized advancing-front scheme [11,12,17-19] belongs to the second family. It is advocated here because it does not require a separate library of modules to introduce points

before triangulating, and also allows to regenerate adaptively the domain in a very simple manner [11]. The main steps involved when generating a grid using the advancing-front technique [11,12,17-19] are as follows:

F.1 Define the boundaries (surfaces) of the domain to be gridded.

F.2 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, stretching and stretching direction. This background grid must completely cover the domain to be gridded.

F.3 Using the information stored on the background grid, set up faces on all these boundaries. This yields the initial front. At the same time, find the generation parameters (element size, stretching and stretching direction) for the new faces from the background grid.

F.4 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.

F.5 For the face to be deleted:

F.5.1 Select a 'best point' position for the introduction of a new point `IPNEW`.

F.5.2 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 F.5.2).

F.5.3 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 F.5.3).

F.6 Add the new element, point, and faces to their respective lists.

F.7 Find the generation parameters for the new faces from the background grid.

F.8 Delete the known faces from the list of faces.

F.9 If there are any faces left in the front, go to F.4.

There are several interesting algorithmic aspects which should be mentioned:

a) Extensive use is made of optimal data structures to perform the search operations involved. In particular, we use heap-lists to find the next face to be deleted (step F.4), quad-trees to find the closest given points to a new point (step F.5.2), and linked lists to find the faces adjacent to a given point (step F.5.3). We combine quad-trees and linked lists to find for any given location the values of generation parameters from the background grid (steps F.3 and F.7). The algorithmic complexity of the overall algorithm should be of $O(N \log N)$. In practice, we find it to be closer to $O(N)$, as we continuously delete domain points from the lists, and the subroutine-calls require some overhead.

b) The checking of face-intersection is a non-trivial problem in 3-D. It also requires a large amount of CPU-time due to the algorithmic complexity involved. In order to

make this process faster, a layered face-checking approach was implemented [18]. This resulted in a significant reduction of CPU time.

More details about the implementation and the data structures used can be found in [17,18].

Examples include

a) Multi-Element Airfoil Configuration (2-D)

Figure 1 shows a multi-element airfoil case. Figure 1a gives the boundary information, Figure 1b the background grid used and Figure 1c the final generated grid.

b) Missile Launcher (3-D)

Figure 2 shows a generic missile launcher model. Only half the model is required for computational purposes. The surface definition was given by 39 points, 44 lines and 19 surface segments. The background grid had 48 points and 132 elements. Figure 2a shows the complete surface triangulation. The generated grid contains 14,508 points and 75,894 tetrahedrons. Figure 2b shows the grid along the axis of symmetry, and one can observe finer grid zonings close to the missile and the launcher.

3. THE FLOW SOLVER: FINITE-ELEMENT FLUX-CORRECTED TRANSPORT (FEM-FCT)

The design of flow solvers for compressible flows has reached a high degree of sophistication, as witnessed by the many publications dealing with this subject [25-27]. At present, the central question no longer is:'will it work ?', but rather:'how well will it work for the class of problems to be simulated ?'. Therefore, we define the design criteria we applied when constructing our Euler solver:

a) The solver should be able to operate on unstructured grids.

b) The solver should be able to operate on moving grids, so that body or interface movement can be simulated.

c) The solver should be able to simulate transient compressible flow problems with strong shocks.

d) The phase-accuracy of the solver should be better than second order, so that vortex propagation and linear discontinuities are not spread too fast in time.

e) The solver should be fast, as transient simulations tend to require longer run-times than steady-state simulations.

Item a) implies that we can only discretize operators, and not stencils. Item b) implies that we must formulate the equations of motion for the fluid in an Arbitrary Lagrangian-Eulerian (ALE) [28] frame of reference. Item c) implies that the solver must have Total Variation Diminishing (TVD) [27,29] properties, and be of first-order accuracy near discontinuities. Item d) implies that for smooth flow regions, the flow solver must reverse to a third- or fourth-order accurate scheme in phase. Finally, item e) implies that if possible we ought not to use Riemann-solvers, as these require more CPU-time per update than ordinary schemes.

We start by writing the governing equations of compressible flow in the following ALE-form

$$\frac{\partial U}{\partial t} + \frac{\partial F_j^a}{\partial x_j} = S, \tag{1}$$

where the summation convention has been employed and

$$U = \left\{ \begin{array}{c} \rho \\ \rho u_i \\ \rho e \end{array} \right\} \quad , \quad F_j^a = \left\{ \begin{array}{c} \rho(u_j - w_j) \\ \rho u_i(u_j - w_j) + p\delta_{ij} \\ \rho(u_j - w_j)e + u_j p \end{array} \right\} \quad , \quad S = -\frac{\partial w_j}{\partial x_j} \left\{ \begin{array}{c} \rho \\ \rho u_i \\ \rho e \end{array} \right\}. \tag{2}$$

Here ρ, p, and e denote the density, pressure and specific total energy of the fluid respectively, and u_i, 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 state equation

$$p = (\gamma - 1)\rho[e - \frac{1}{2}u_j u_j] \quad , \tag{3}$$

which is valid for a perfect gas, where γ is the ratio of the specific heats. We now focus on the TVD aspects of the flow solver. As is well known, for the compressible flows described by Eqn.(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'). The idea behind Flux-Corrected Transport (FCT), and more generally all TVD schemes, 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. Note that even if the original partial differential equation is linear, the resulting scheme will be nonlinear. The temporal discretization of Eqn.(1) yields

$$U^{n+1} = U^n + \Delta U \quad , \tag{4}$$

where ΔU is the increment of the unknowns obtained for a given scheme at time $t = t^n$. Our aim is to obtain a ΔU of as high an order as possible without introducing overshoots. To this end, we re-write Eqn.(4) as:

$$U^{n+1} = U^n + \Delta U^l + (\Delta U^h - \Delta U^l) \quad , \tag{5}$$

or

$$U^{n+1} = U^l + (\Delta U^h - \Delta U^l) \quad . \tag{6}$$

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.(6):

$$U^{n+1} = U^l + lim(\Delta U^h - \Delta U^l) \ , \tag{7}$$

in such a way that no new over/undershoots are created. More details on how the limiting is performed may be found in [30,31]. As the high-order scheme, we employ a two-step form [4,32] of the one-step Taylor-Galerkin schemes described in [1,3]. 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. They have been chosen here, because they appear to offer the best accuracy per cost performance of all the schemes tried. Given the system of equations (1), we advance the solution from t^n to $t^{n+1} = t^n + \Delta t$ as follows:

a) First step:

$$U^{n+\frac{1}{2}} = U^n + \frac{\Delta t}{2} \cdot \left[S|^n - \left. \frac{\partial F_j^a}{\partial x_j} \right|^n \right] \tag{8}$$

b) Second step :

$$\Delta U^n = U^{n+1} - U^n = \Delta t \left[S|^{n+\frac{1}{2}} - \left. \frac{\partial F_j^a}{\partial x_j} \right|^{n+\frac{1}{2}} \right] \tag{9}$$

The spatial discretization of Eqns.(8,9) is performed via the classic Galerkin weighted residual method [4,32], using linear elements, i.e. 3-noded triangles in 2-D and 4-noded tetrahedra in 3-D. For Eqn.(9) the following system of equations is obtained:

$$M_C \cdot \Delta U^n = R^n, \tag{10}$$

where M_C denotes the consistent mass matrix [1,3], 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.(10) is never solved directly, but iteratively, requiring typically three passes [1]. We can then recast the converged solution of Eqn.(10) into the following form:

$$M_L \cdot \Delta U^h = R + (M_L - M_C) \cdot \Delta U^h. \tag{11}$$

Here M_L denotes the diagonal, lumped mass-matrix (see [1]). We remark that this high-order scheme is better than second-order in phase-space, as the consistent mass adds additional information beyond nearest neighbors when iterating. Numerical experiments (see below) show that this scheme, when applied on a structured mesh, is comparable to a fourth-order accurate scheme. As the low-order scheme, we use the same discretization for

the fluxes and source-terms, but revert to a lumped mass and an added numerical diffusion term on the right:

$$M_L \cdot \Delta U^l = R + DIFF \quad . \tag{12}$$

NUMERICAL EXAMPLES

a) Passive Advection of a Square Wave

This is the same example as was used by Boris and Book [33] to demonstrate the accuracy and monotonicity of their FCT-schemes. As the equation being solved (the transport equation) is linear, both amplification- and phase-errors can be identified easily. The wave extends over 20 gridpoints, and is convected with a Courant-number of C=0.2 for 800 steps. In Figure 3 we compare the solutions obtained when using, in the high-order scheme, a lumped mass-matrix and a consistent mass-matrix. Observe that the consistent mass-matrix gives better phase-accuracy. After 800 steps the initial discontinuity is spread over only 5 gridpoints.

b) Passive Advection in 2-D

This is the same example that Zalesak [34] used to test his FCT-algorithms. Again, as the equation solved is linear, both amplification and phase-errors can be identified easily. The problem statement may be found in [34]. The mesh used for this case, as well as the results obtained are depicted in Figure 4.

c) Sod-Problem (1-D)

This classic example, taken from [35], solves the Riemann-problem for the compressible Euler equations in 1-D. The same grid as in [35] is employed, and the solutions are shown at times t=7.36 and t=14.75. Figure 5 shows the results obtained for the described FEM-FCT Euler solver.

d) Shock-Reflection at a Wall

This problem has also been used extensively to assess the accuracy of schemes used for the solution of steady state problems [29]. As in all the following steady-state examples, local timestepping was used. The problem statement, as well as the pressure distributions obtained for the original Taylor-Galerkin scheme and FEM-FCT are shown in Figure 6. Observe that the shocks are captured so sharply that the underlying grid structure becomes visible in the contour-plots. The steady state solution for this problem took 300 iterations, the residuals dropping 4 orders of magnitude. Figure 6d depicts the variation of the density along the line y=0.5, and, as one can see, no over/undershoots are present.

e) Flow Past an Airfoil in Transonic Flow

This example shows that acceptable solutions can be achieved with the present algorithm for the transonic flow regime. The case at hand is a NACA-0012 airfoil, and the Mach number at infinity and angle of attack were set to $M_\infty = 0.85$ and $\alpha = 0.0$. The grid-point distribution was taken from Jameson's FLO52-code [36], and corresponds to a 96 by 16 mesh. The c_p -distributions on the airfoil-surface obtained for FEM-FCT and

the original two-step Taylor-Galerkin scheme presented in [4,32] are given in Figure 7. Although the solution achieved by FEM-FCT is better than that of the ordinary Taylor-Galerkin scheme, for steady-state aerodynamic applications, where shocks are only locally important, the additional cost of the high-resolution schemes does not make them attractive for production runs. Adaptive refinement [4,7,11,32] is a much more effective way of obtaining sharp shocks for steady flows.

f) Supersonic Flow Past an Object in a Cavity (3-D)

This example shows a 3-D solution for a fairly complex geometry. The problem statement, as well as the grid employed and the solution obtained, may be seen in Figure 8. The grid consisted of 5,426 points and 27,462 tetrahedra. The free-stream Mach-number was set to $M_\infty = 1.5$. The flow impinges on the back wall of the cavity, creating a recirculation zone in the cavity. This is visible in the particle paths plotted in Figure 8c.

h) Transonic Flow Past Pathfinder in a Wind-Tunnel (3-D)

This example shows a 3-D solution for the Pathfinder wing-body-tail configuration in a windtunnel. The problem statement, as well as the grid employed and the solution obtained may be found in Figures 9a-c. The grid consisted of 10,280 points and 55,865 tetrahedra. The free-stream Mach-number was set to $M_\infty = 0.82$, and convergence was achieved in 2000 steps.

4. ADAPTIVE REFINEMENT

A very attractive feature of unstructured grids is the ease with which adaptive refinement can be incorporated into them. The addition of further degrees of freedom does not destroy any previous structure. Thus, the flow solver requires no further modification when operating on an adapted grid. For many practical problems, the regions that need to be refined are extremely small as compared to the overall domain. Therefore, the savings in storage and CPU requirements typically range between 10-100 as compared to an overall fine mesh. We find that for the majority of the daily production-type runs, adaptive refinement makes the difference between being or not being able to run the problems to an acceptable accuracy in a reasonable time. Without it, we would be forced to use much coarser grids, with lower accuracy, for the same expense.

Any adaptive refinement scheme is composed of three main ingredients. These are
1) an optimal-mesh criterion,
2) an error indicator, and
3) a method to refine and derefine the mesh.
They give answers to the questions
1) how should the optimal mesh be ?,
2) where is refinement (derefinement) required ?, and
3) how should the refinement (derefinement) be accomplished ?
Many variants of each of these subtopics have been explored and shown to be useful for a certain class of problems [4,7,8,9-14,37]. Therefore, as was the case with the flow solvers, we need to define our design criteria before proceeding further. Again, we seek a method

that is efficient and reliable for transient compressible flow problems. This leads us to the following design criteria for the error indicator:

a) The error indicator should be fast.

b) The error indicator should be dimensionless, so that several 'key variables' can be monitored at the same time.

c) The error indicator should be bounded, so that no further user intervention becomes necessary as the solution evolves.

d) The error indicator should not only mark the regions with strong shocks to be refined, but also weak shocks, contact discontinuities and other 'weak features' in the flow.

For the refinement method, the design criteria are as follows:

e) The method should be conservative, i.e. a mesh change should not result in the production or loss of mass, momentum or energy.

f) The method should not produce elements that are too small, as this would reduce too severely the allowable timestep of the explicit flow solvers employed.

g) The method should be fast. In particular, it should lend itself to some degree of parallelism.

h) The method should not involve major storage overhead.

An error indicator that meets the design criteria a)-d) was proposed in [37]. In general terms, it is of the form

$$error = \frac{h^2 \; |second\ derivatives|}{h \; |first\ derivatives| + \epsilon \; |mean\ value|}$$

By dividing the second derivatives by the absolute value of the first derivatives the error indicator becomes bounded, dimensionless, and the 'eating up' effect of strong shocks is avoided. The terms following ϵ are added as a 'noise' filter in order not to refine 'wiggles' or 'ripples' which may appear due to loss of monotonicity. The value for ϵ thus depends on the algorithm chosen to solve the PDEs describing the physical process at hand. The multidimensional form of this error indicator is given by

$$E^I = \sqrt{\frac{\sum_{k,l} (\int_\Omega N^I_{,k} N^J_{,l} d\Omega \cdot U_J)^2}{\sum_{k,l} (\int_\Omega |N^I_{,k}| \left[|N^J_{,l} U_J| + \epsilon \left(|N^J_{,l}| |U_J| \right) \right] d\Omega)^2}}, \tag{13}$$

where N^I denotes the shape-function of node I.

Two different methods for grid refinement have been explored. These are 1) local h-refinement, and 2) adaptive remeshing. Table 1 compares the relative merits of both approaches. We may summarize our experience with both approaches as follows:

- For steady-state or mildly unsteady problems, adaptive remeshing represents the best adaptive refinement method currently available. Particularly if large stretching ratios can be realized, it easily outperforms all other methods.

- For strongly unsteady problems, where a new grid is required every 5-10 timesteps, local h-refinement seems to be preferable. Several reasons can be given for this choice. Firstly, h-refinement is more robust than remeshing. The amount of things that can go wrong seems to be much less than when remeshing. Secondly, h-refinement is very well suited to vector- and parallel processors. Thirdly, conservation presents no problem for h-refinement.
- Table 1 also quotes CPU times measured on the CRAY-XMP-24 at NRL. While these times are approximate and could be improved, they indicate a trend. The described grid generator is a scalar procedure. Moreover, the introduction of a new point or element requires substantially more operations than the equivalent operation done with h-refinement. A partial solution to this problem is to generate first a coarser, yet stretched grid, and then to apply global h-refinement [8]. As vectorization of the global h-refinement is straightforward, the savings as compared to just remeshing are substantial (a factor of 4 in 2-D, a factor of 8 in 3-D).
- In our experiments adaptive remeshing maintained the conservation sums very well. The change during one mesh change was less than 0.01%. We attribute this very good performance to the fact that the elements close to shocks were uniform in size.
- The combination of both approaches should be pursued further. In this way, the advantages of each approach can be employed to its fullest extent.

	H-Refinement	Remeshing
- Interpolation/Conservation	easy	not easy
- Minimum h-size	easy	not easy
- Directional Refinement	not easy	easy
- Body/Interface Movement	not easy	easy
- Parallelizable	easy	not easy
- Timings ($\mu sec/pt./grid$)	120	1800

Table 1: H-Refinement vs. Remeshing

NUMERICAL EXAMPLES

a) Shock Impinging on Two Obstacles

We consider a strong shock ($M_s = 10.0$) coming from the left that collides with the two bodies shown in Figure 10a. Four levels of refinement were activated. Had the grid been refined uniformly, this would correspond to 152,320 points. For the run shown the highest number of gridpoints required was about 17,000, and during most of the computation

considerably less gridpoints were used. As the physics become more complicated, more grid points are required, and correspondingly larger portions of the domain are refined. The solutions obtained at different times are depicted in Figures 10b-c. We show the grid, the density contours, and a blow-up of the velocities at times T=4.0, T=6.0 and T=8.0. For the density contours about 100 levels were chosen, so that any 'wiggle' in the plot corresponds to approximately 1% deviation from a smooth solution. The adaptive refinement procedure is capable of producing very accurate calculations with sharp, detailed resolution of flow structures.

b) Shock-Box Interaction

In this case we consider a weak shock ($M_s = 1.4$) that interacts with a box some distance from the ground. A maximum of four layers was activated, but we also imposed a minimum h-size limit for the refinement. Thus, in the finest regions of the base mesh only three levels of refinement were used. The solutions obtained at different times are depicted in Figures 11a-c. Again, we typically show 60-80 contour levels for the relevant quantities. Besides the obvious shock reflections, one can also see the vortex shedding process that begins at the sharp corner of the box. More details and further studies of this class of problems may be found in [38].

c) Object Falling into Supersonic Free Stream

The problem statement is as follows: an object is placed in a cavity surrounded by a free stream at $M_\infty = 1.5$. After the steady-state solution is reached (time T=0.0), a body motion is prescribed, and the resulting flowfield disturbance is computed. Adaptive remeshing was performed every 100 timesteps initially, while at later times the grid was modified every 50 timesteps. No subsequent h-refinement was used in this case, as the mesh adaptation process is not done very often. The maximum stretching ratio specified was $S = 5.0$. Figures 12a,b show two different stages during the computation at times T=20 and T=175. Initially, the velocity flows counterclockwise around the object. At later times, this motion is reversed in the cavity. One can also see how the location and strength of the shocks changes due to the motion of the object. Notice how the directionality of the flow features is reflected in the mesh.

5. CONCLUSIONS

We have described several algorithms for the generation and adaptation of unstructured grids in two and three dimensions, as well as Euler solvers that operate on these grids. The main purpose of the paper was to demonstrate how unstructured grids may be employed advantageously for the economic simulation of both geometrically as well as physically complex flowfields. Numerous examples taken from daily production runs were shown, demonstrating the capabilities developed. Future developments will center on the following areas:

- extension of Euler-solvers to the Navier-Stokes equations,
- incorporation of chemistry in the current non-reacting flow models,
- extension of local adaptive h-refinement and adaptive remeshing to 3-D, and
- adaptive gridding for time-dependent, viscous flows.

6. ACKNOWLEDGEMENTS

This work was partially funded by the Office of Naval Reseach through the Naval Research Laboratory, the Applied and Computational Mathematics Program of DARPA, the NASA Johnson Space Center and SBIR-Grant-No. NAS1-18419.

We would also like to acknowledge the support of C. Gumbert (TAB, NASA LRC), who provided a number of very useful graphics interfaces for the IRIS.

7. REFERENCES

[1] J. Donea - A Taylor Galerkin Method for Convective Transport Problems; *Int. J. Num. Meth. Eng.* 20, 101-119 (1984).

[2] T.J.R. Hughes, M. Mallet and A. Mizukami - A New Finite Element Formulation for Computational Fluid Dynamics: II. Beyond SUPG; *Comp. Meth. Appl. Mech. Eng.* 54, 3, 341-355 (1986).

[3] R. Löhner, K. Morgan and O.C. Zienkiewicz - The Solution of Nonlinear Systems of Hyperbolic Equations by the Finite Element Method; *Int. J. Num. Meth. Fluids* 4, 1043-1063 (1984).

[4] R. Löhner, K. Morgan and O.C. Zienkiewicz - An Adaptive Finite Element Procedure for High Speed Flows; *Comp. Meth. Appl. Mech. Eng.* 51, 441-465 (1985).

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

[6] R. Löhner, K. Morgan and O.C. Zienkiewicz - Effective Programming of Finite Element Methods for CFD on Supercomputers; pp.117-125 in *The Efficient Use of Vector Computers with Emphasis on CFD* (W. Schönauer and W. Gentzsch eds.), Vieweg Notes on Numerical Fluid Mechanics, Vol 9, Vieweg Verlag (1985).

[7] D.G. Holmes, S.H. Lamson and S.D. Connell - Quasi-3D Solutions for Transonic, Inviscid Flows by Adaptive Triangulation; Proc. ASME Gas Turbine Conf., Amsterdam, June 1988.

[8] R. Löhner - Adaptive Remeshing for Transient Problems; accepted for publication in *Comp. Meth. Appl. Mech. Eng.* (1988).

[9] K. Morgan, J. Peraire, R.R. Thareja and J.R. Steward - An Adaptive Finite Element Method for the Euler and Navier-Stokes Equations; AIAA-87-1172-CP (1987).

[10] J.T. Oden, P. Devloo and T. Strouboulis - Adaptive Finite Element Methods for the Analysis of Inviscid Compressible Flow: I. Fast Refinement/Unrefinement and Moving Mesh Methods for Unstructured Meshes; *Comp. Meth. Appl. Mech. Eng.* 59, 327-362 (1986).

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

[12] J. Peraire, K. Morgan, J. Peiro and O.C. Zienkiewicz - An Adaptive Finite Element Method for High Speed Flows; AIAA-87-0558 (1987).

[13] M.M. Pervaiz and J.R. Baron - Spatio-Temporal Adaptation Algorithm for Two-Dimensional Reacting Flows; AIAA-88-0510 (1988).

[14] R.A. Shapiro and E.M. Murman - Adaptive Finite Element Methods for the Euler Equations; AIAA-88-0034 (1988).

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

[16] S.H. Lo - A New Mesh Generation Scheme for Arbitrary Planar Domains; *Int. J. Num. Meth. Eng.* 21, 1403-1426 (1985).

[17] R. Löhner - Some Useful Data Structures for the Generation of Unstructured Grids; *Comm. Appl. Num. Meth.* 4, 123-135 (1988).

[18] R. Löhner and P. Parikh - Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method; AIAA-88-0515 (1988).

[19] J. Peraire, J. Peiro, L. Formaggia, K. Morgan and O.C. Zienkiewicz - Finite Element Euler Computations in Three Dimensions; AIAA-88-0032 (1988).

[20] N. van Phai - Automatic Mesh Generation with Tetrahedron Elements; *Int. J. Num. Meth. Eng.* 18, 237-289 (1982).

[21] M.A. Yerry and M.S. Shepard - Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique; *Int. J. Num. Meth. Eng.* 20, 1965-1990 (1984).

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

[23] 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).

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

[25] A. Jameson - The Evolution of Computational Methods in Aerodynamics; *ASME J. Appl. Mech.* 50, 1052-1069 (1983).

[26] R. Löhner - Finite Elements in CFD: What Lies Ahead; *Int. J. Num. Meth. Eng.* 24, 1741-1756 (1987).

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

[28] J. Donea - An Arbitrary Lagrangian-Eulerian Finite Element Method for Transient Dynamic Fluid-Structure Interactions; *Comp. Meth. Appl. Mech. Eng.* 33, 689-723 (1982).

[29] H.C. Yee, R.F. Warming and A. Harten - Implicit Total Variation Diminishing (TVD) Schemes for Steady-State Calculations, *J. Comp. Phys.* 57, 327-360 (1985).

[30] R. Löhner, K. Morgan, M. Vahdati, J.P. Boris and D.L. Book - FEM-FCT: Combining Unstructured Grids with High Resolution; accepted for publication in *Comm. Appl. Num. Meth.* (1988).

[31] R. Löhner, K. Morgan, J. Peraire and M. Vahdati - Finite Element Flux-Corrected Transport (FEM-FCT) for the Euler and Navier-Stokes Equations; ICASE Rep. 87-4, *Int. J. Num. Meth. Fluids* 7, 1093-1109 (1987).

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

[33] J.P. Boris and D.L. Book - Flux-corrected Transport. I. SHASTA, a Transport Algorithm That Works; *J. Comp. Phys.* 11, 38 (1973).

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

[35] G. Sod - A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws; *J. Comp. Phys.* 27, 1-31 (1978).

[36] A. Jameson, W. Schmidt and E. Turkel - Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes; AIAA-81-1259 (1981).

[37] R. Löhner - An Adaptive Finite Element Scheme for Transient Problems in CFD; *Comp. Meth. Appl. Mech. Eng.* 61, 323-338 (1987).

[38] J.D. Baum and R. Löhner - Numerical Simulation of Shock-Elevated Box Using an Adaptive Finite Element Shock Capturing Scheme, AIAA-89-0653.

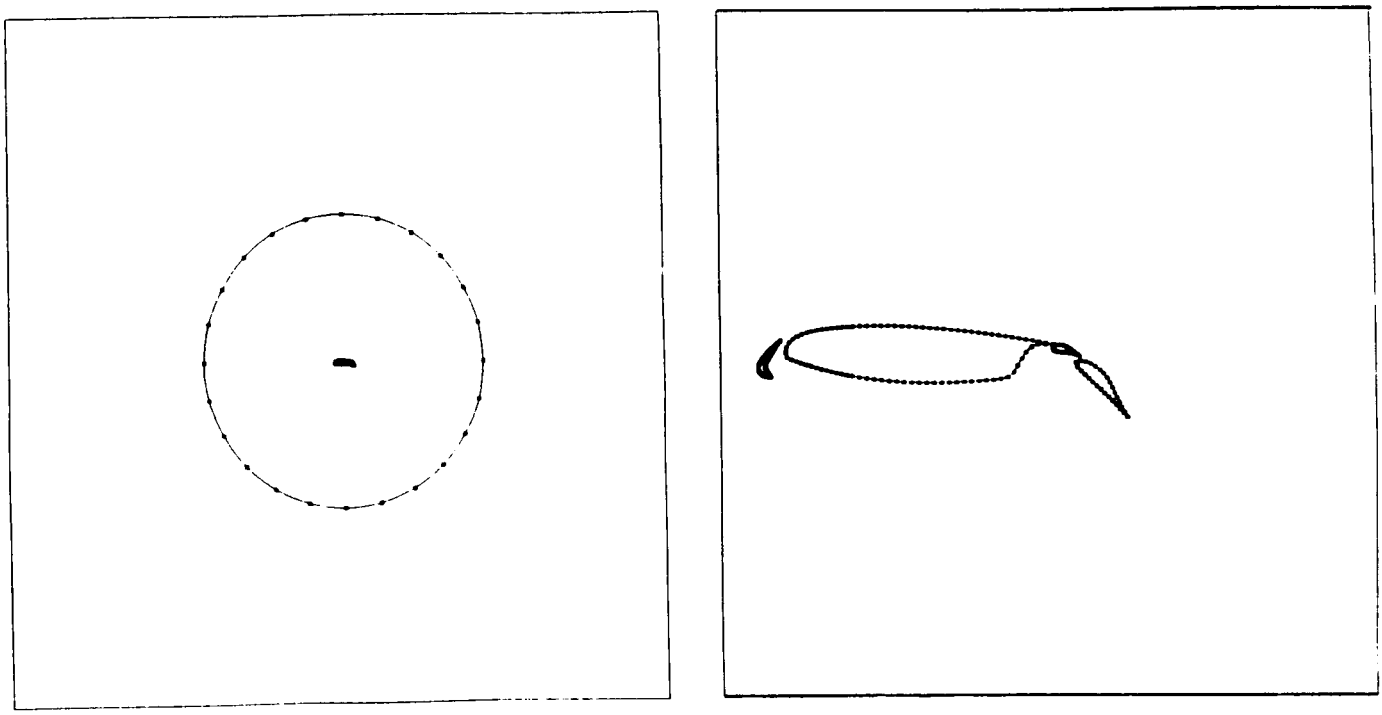


Figure 1(a). Multielement airfoil configuration: boundary information.

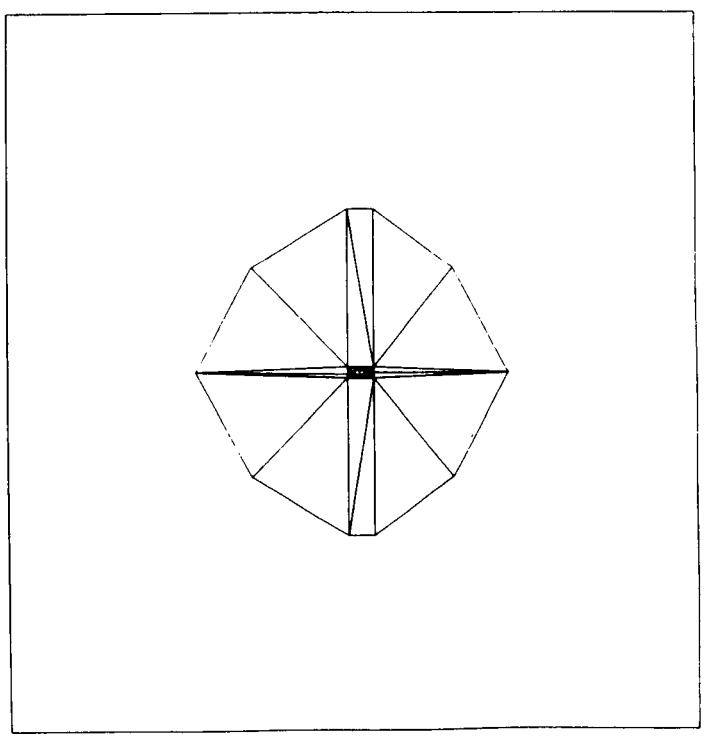


Figure 1(b). Multielement airfoil configuration: background grid.

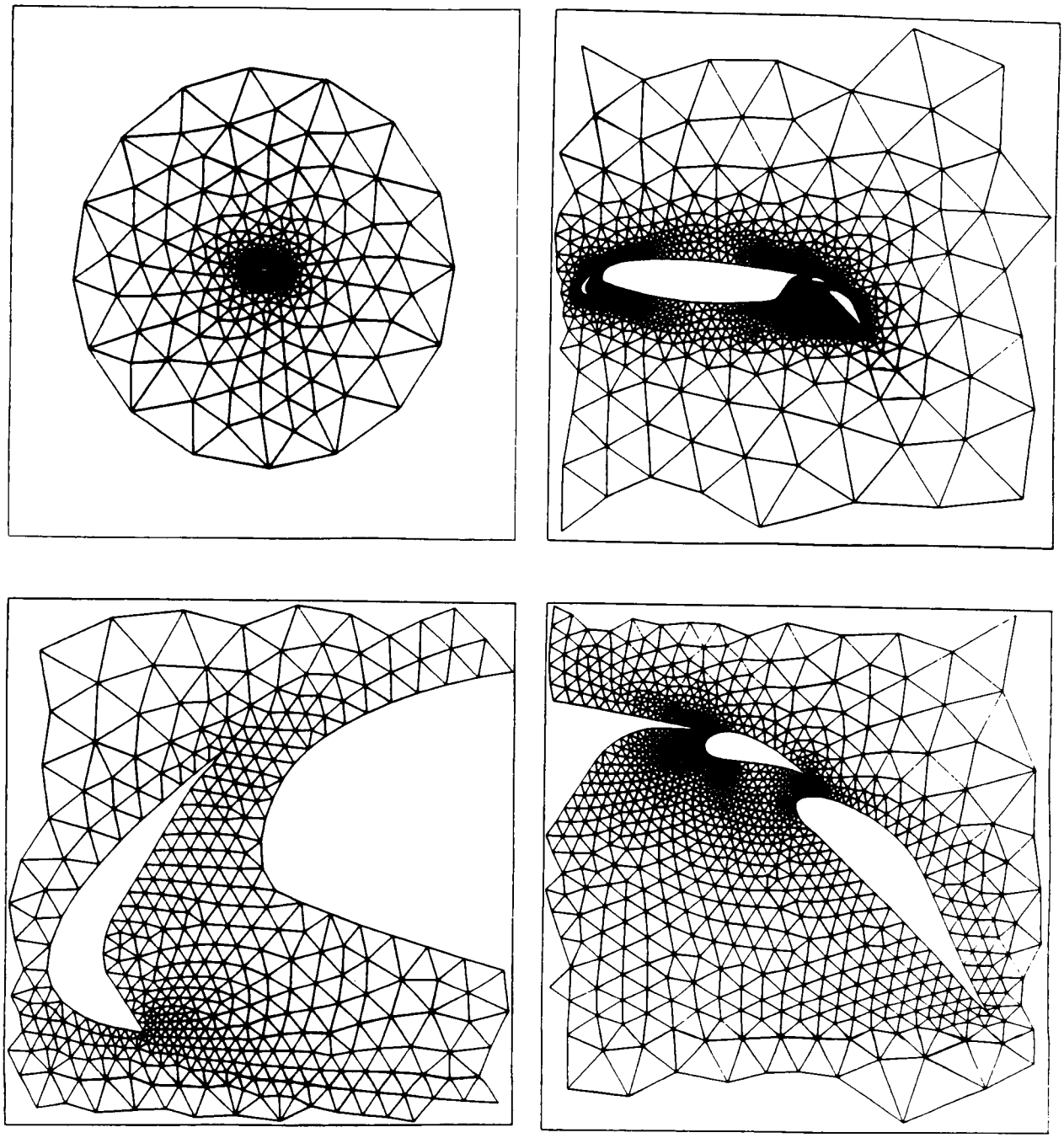


Figure 1(c). Multielement airfoil configuration: generated grid.

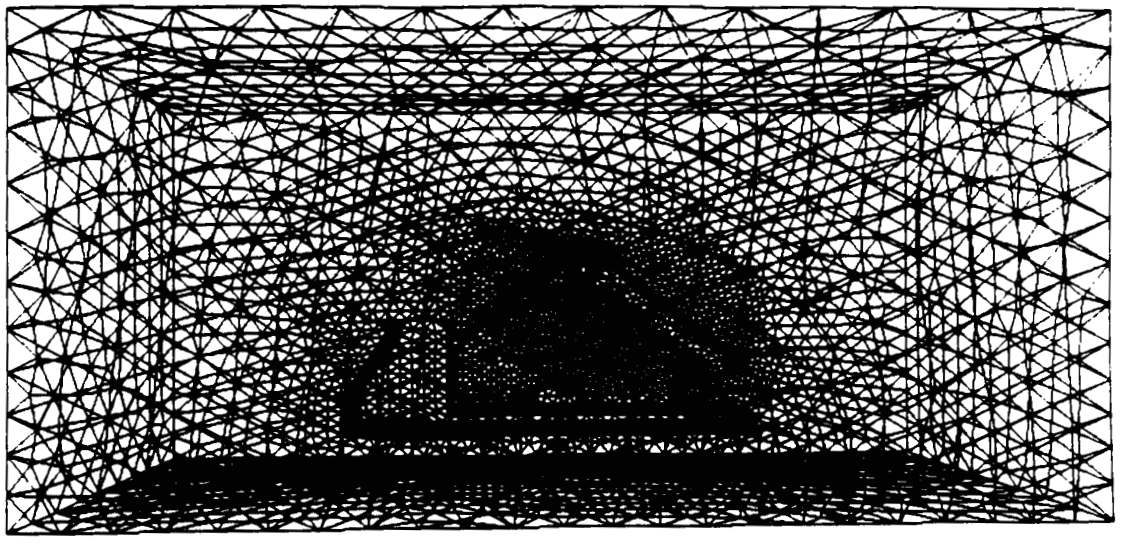


Figure 2(a). Missile launcher: surface triangulation.

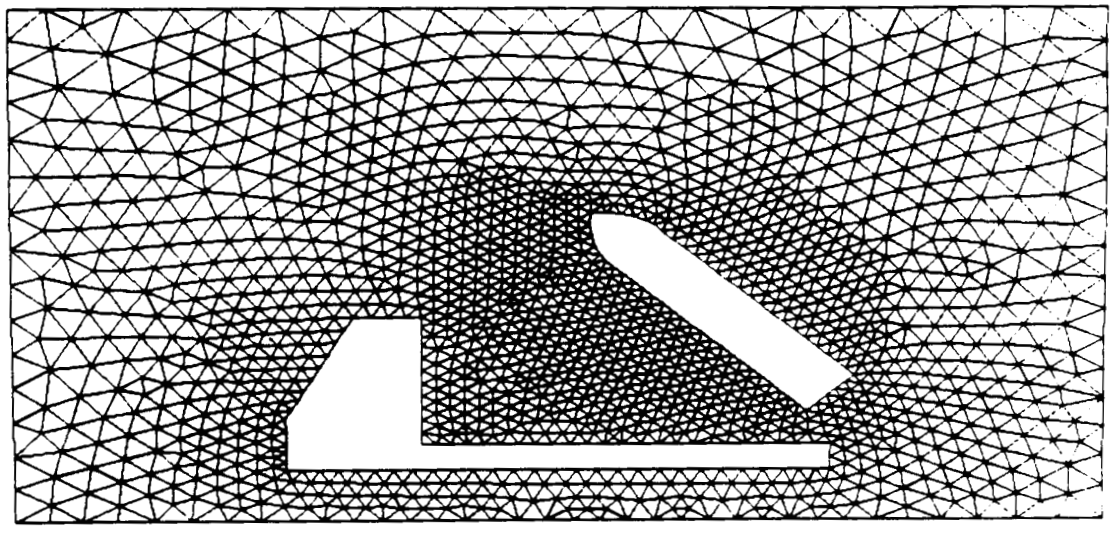
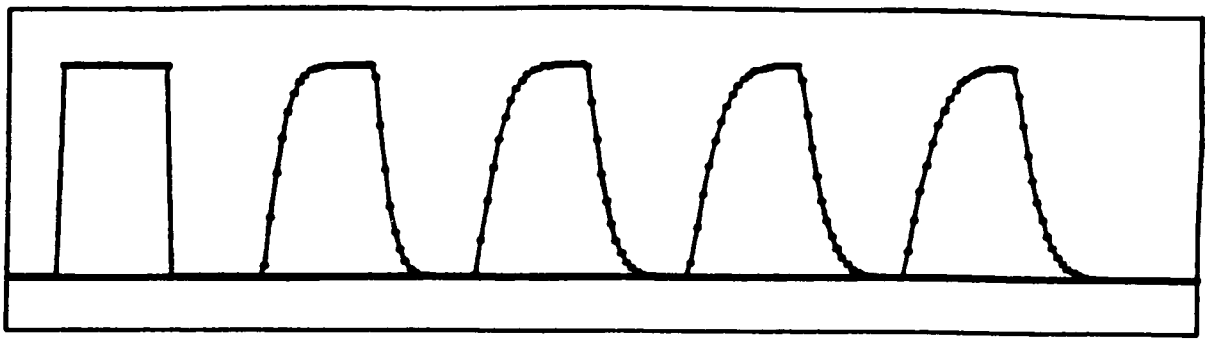
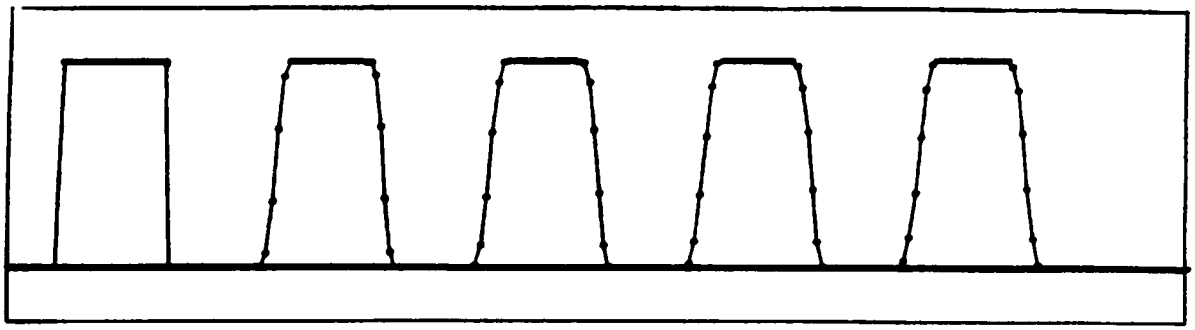


Figure 2(b). Missile launcher: grid in plane of symmetry.

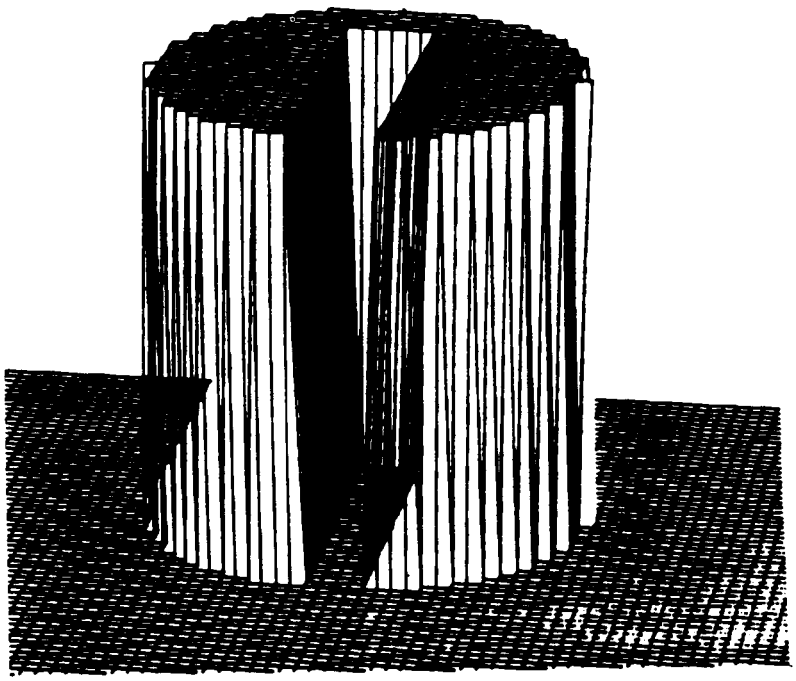


(a) High-order scheme: lumped-mass Taylor-Galerkin (niter=1).

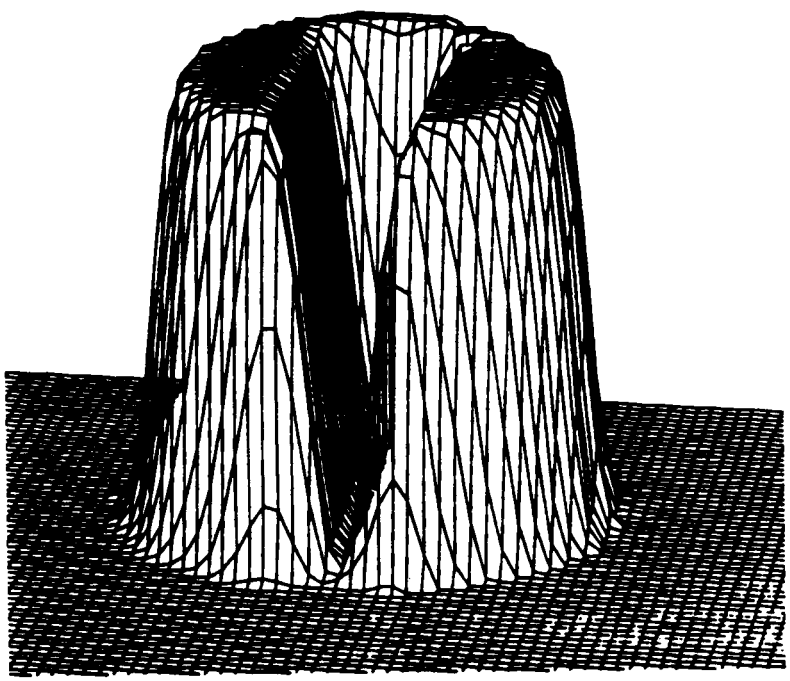


(b) High-order scheme: consistent-mass Taylor-Galerkin (niter=3).

Figure 3. Passive advection of a square wave (1-D); C=0.2, plot every 200 steps.



(a) Solution at time t=0.0



(b) Solution after 628 iterations $\left(\frac{1}{4}\text{ revolution}\right)$. The perspective view has been rotated with the cylinder.

Figure 4. Passive advection in 2-D: Zalesak's example.

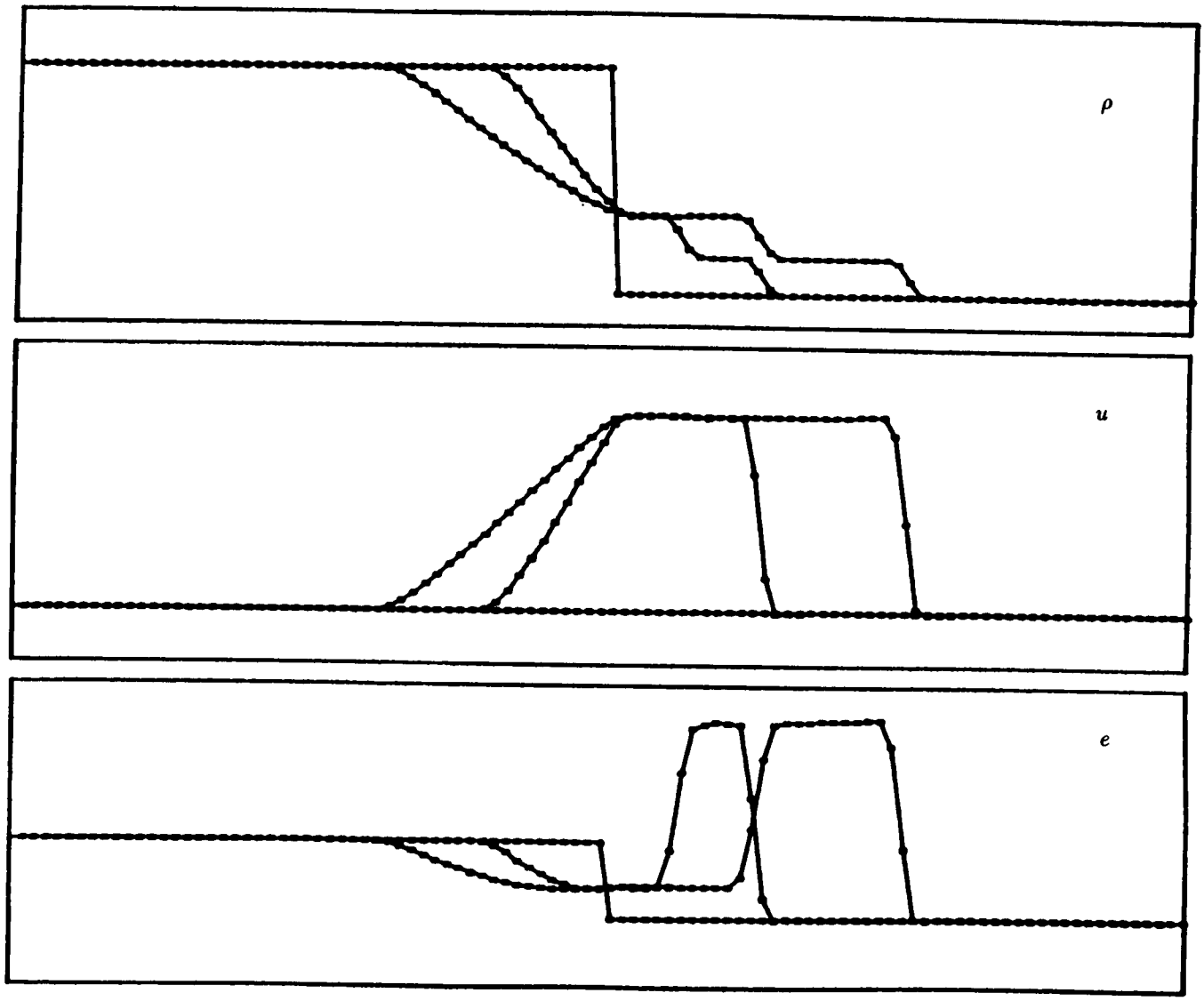
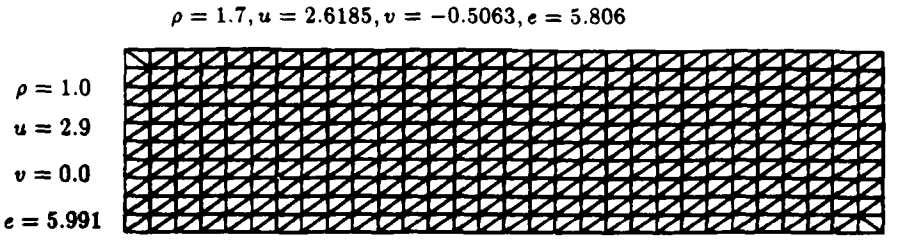
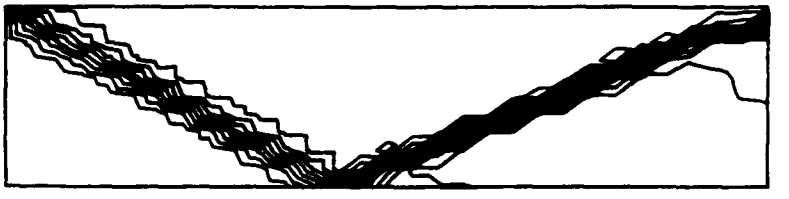


Figure 5. Sod-problem (1-D). Solutions at times t=7.37 and t=14.75.

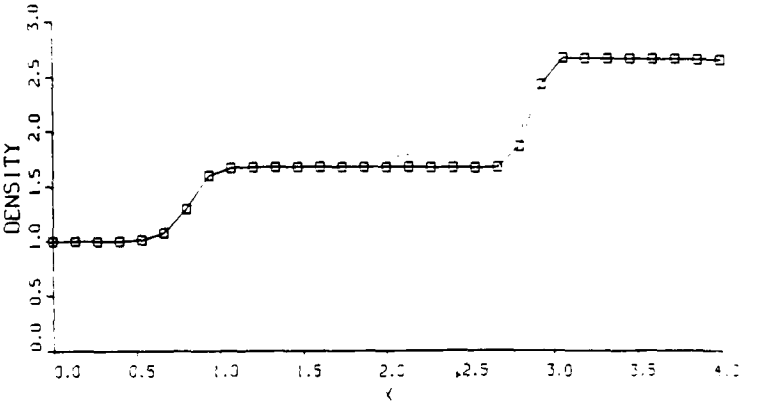


(a) Grid.

(b) Pressure distribution obtained for Taylor-Galerkin scheme [10,12] (C.I.=0.1).



(c) Pressure distribution obtained for FEM-FCT (C.I.=0.1).



(d) Density distribution along line y=0.5.

Figure 6. Shock reflexion at a wall (steady state).

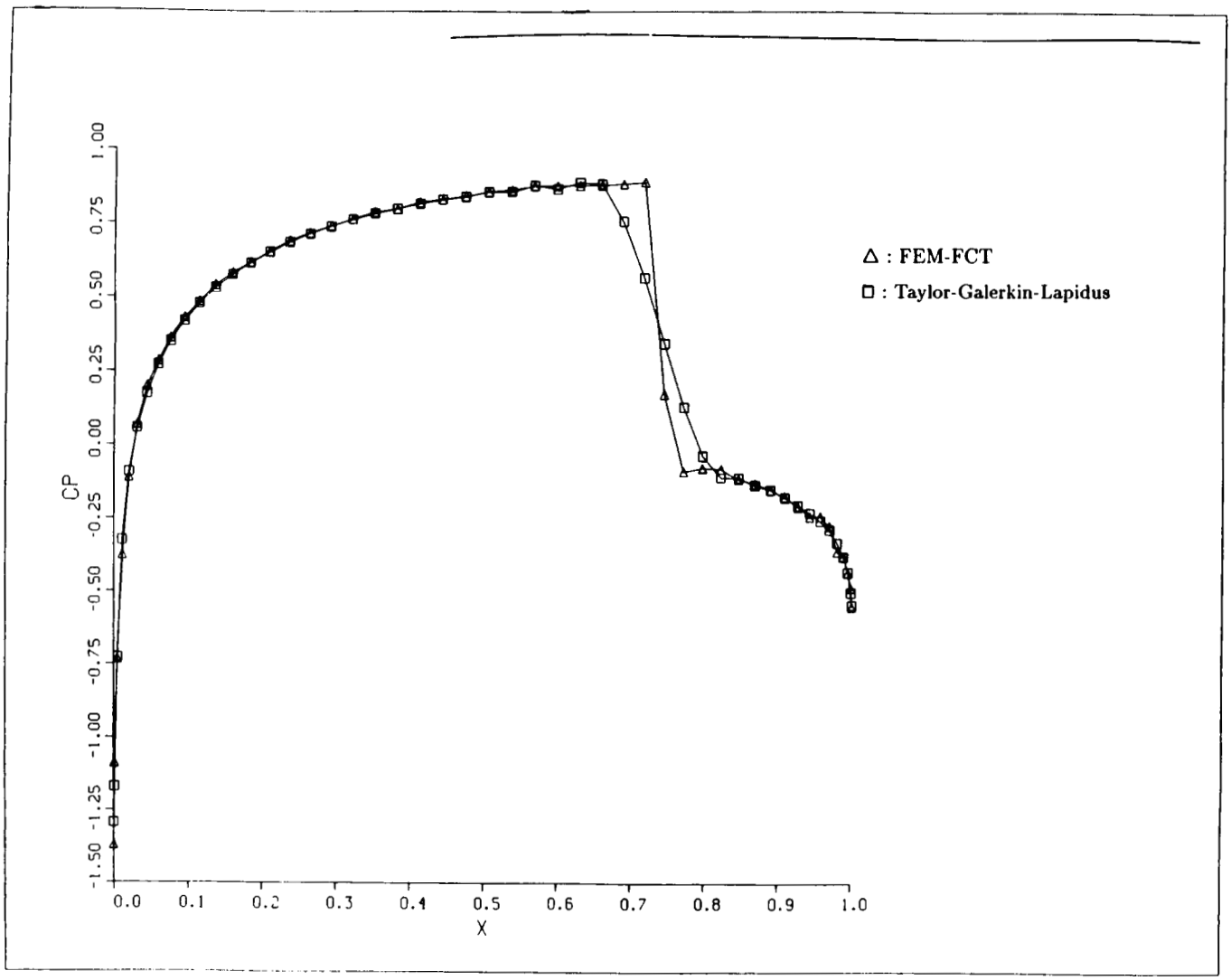
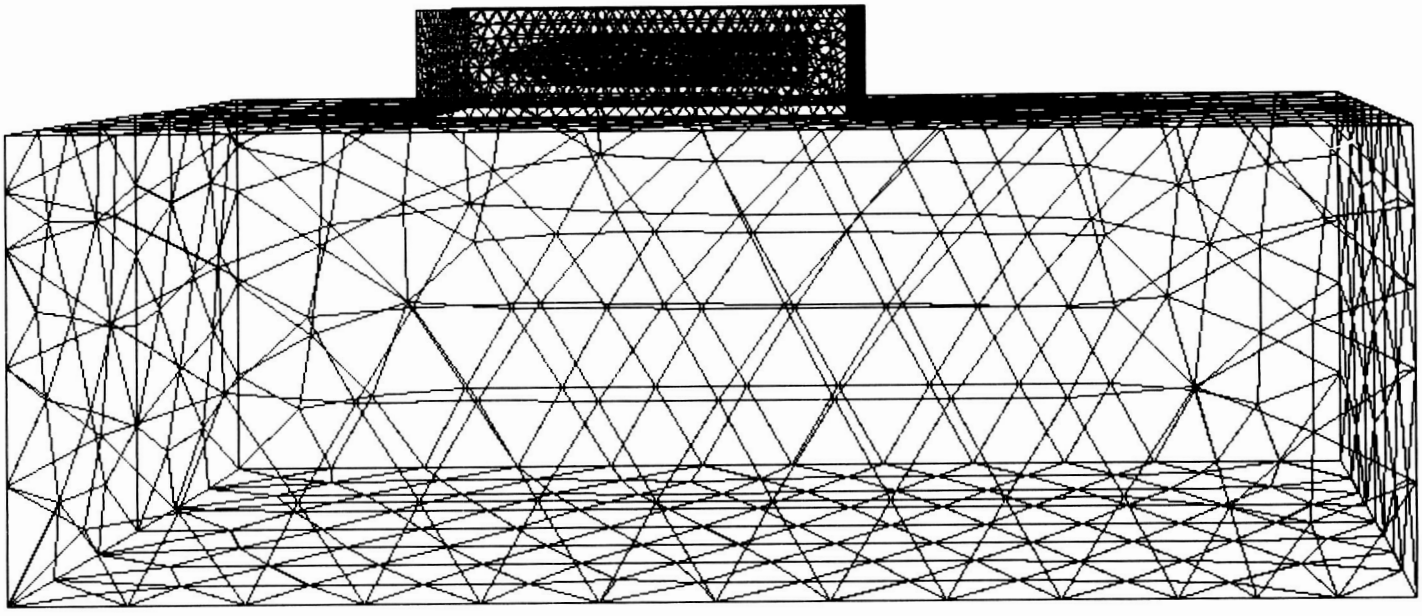
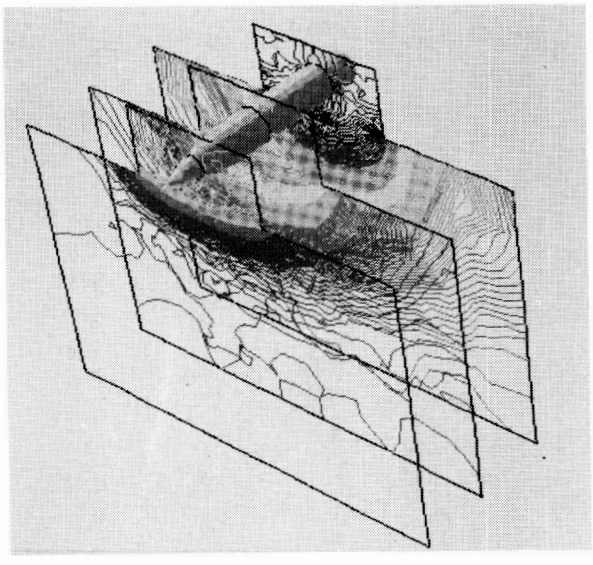


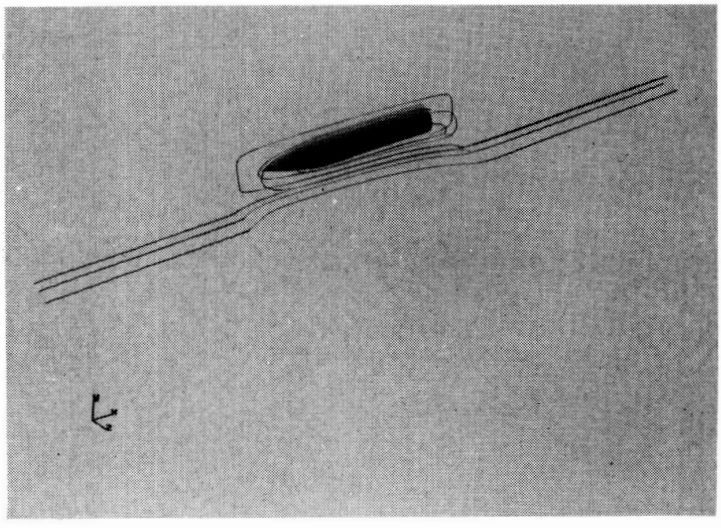
Figure 7. Comparison of Taylor-Galerkin-Lapidus and FEM-FCT for NACA-0012.



(a) Surface triangulation.

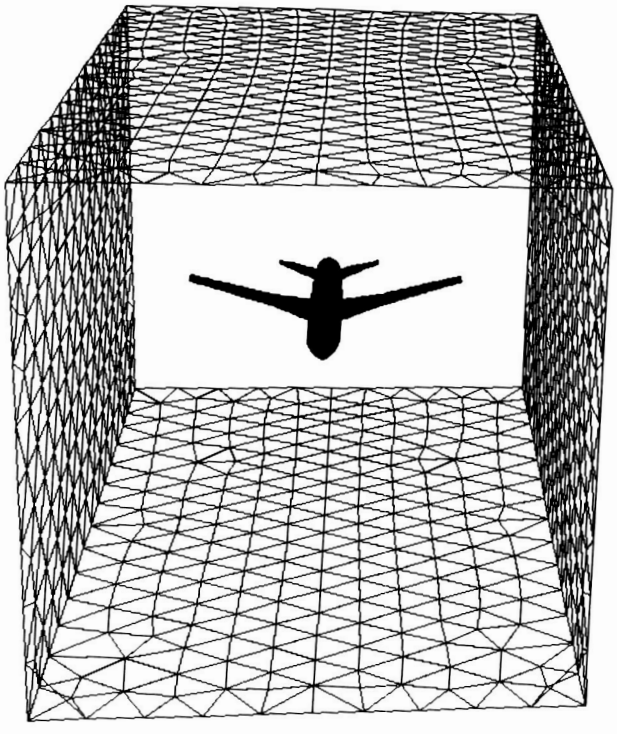


(b) Pressure contours in selected planes.

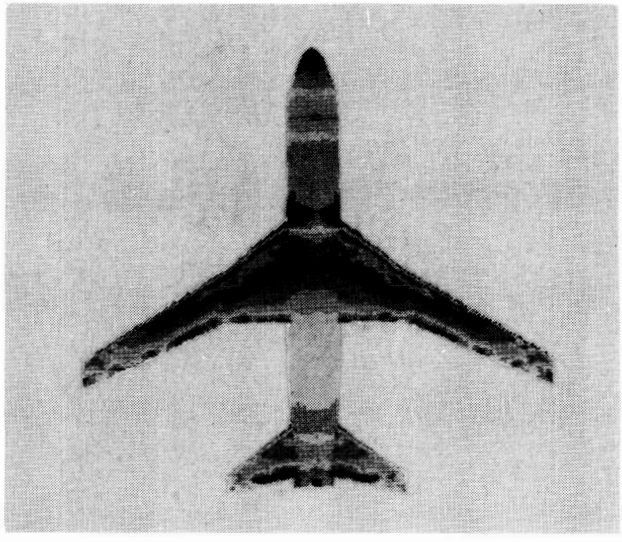


(c) Particle paths.

Figure 8. Supersonic flow past an object in cavity.



(a) Surface triangulation.



(b) Pressure contours.

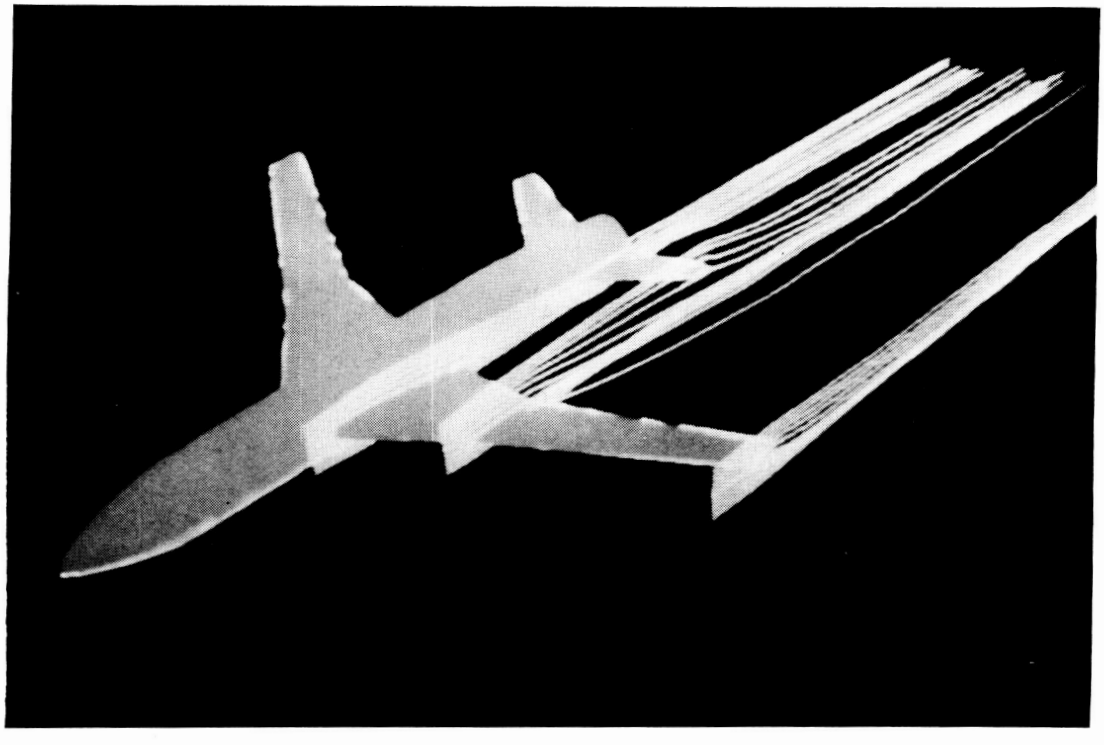
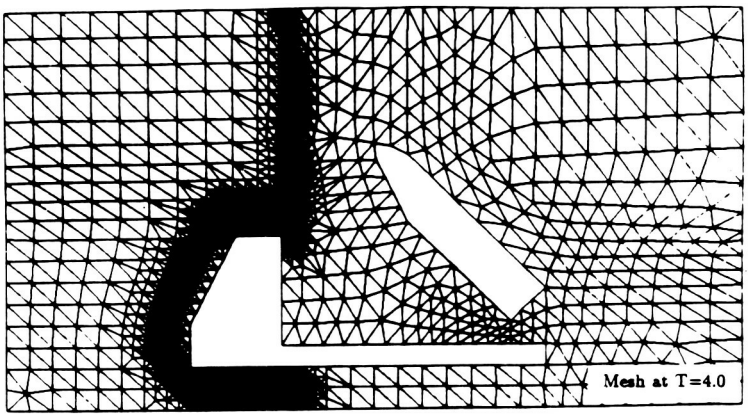
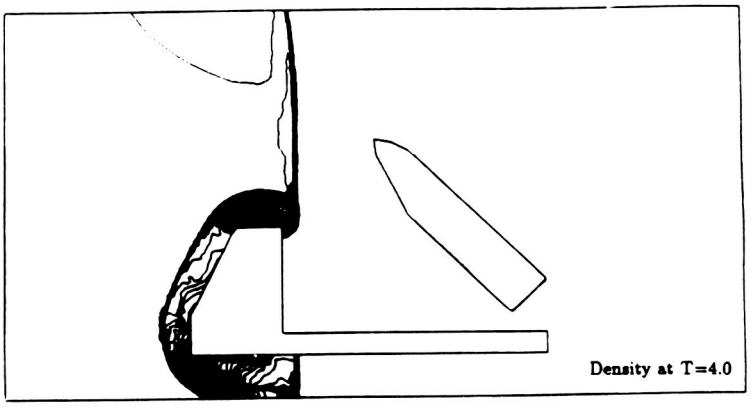


Figure 9. Transonic flow past Pathfinder in tunnel.





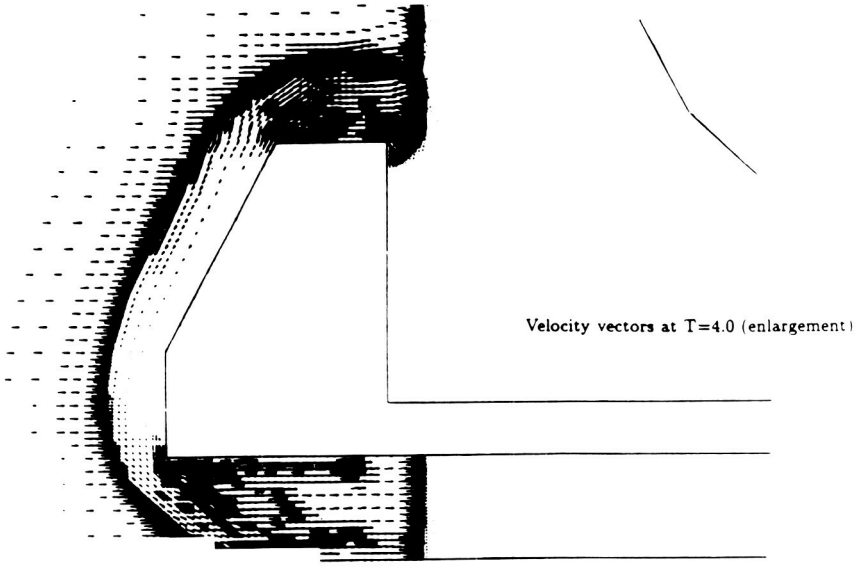
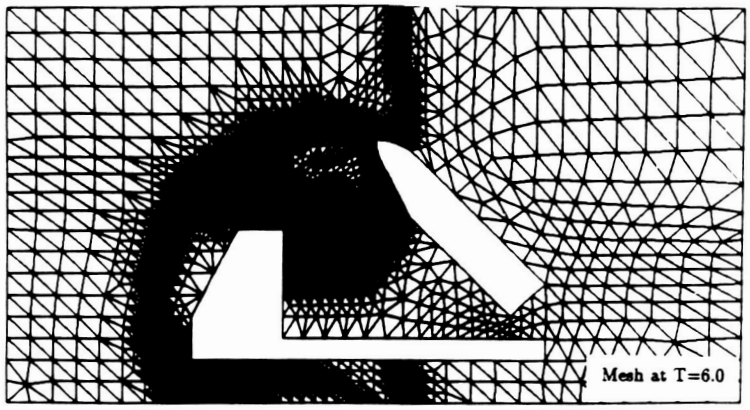
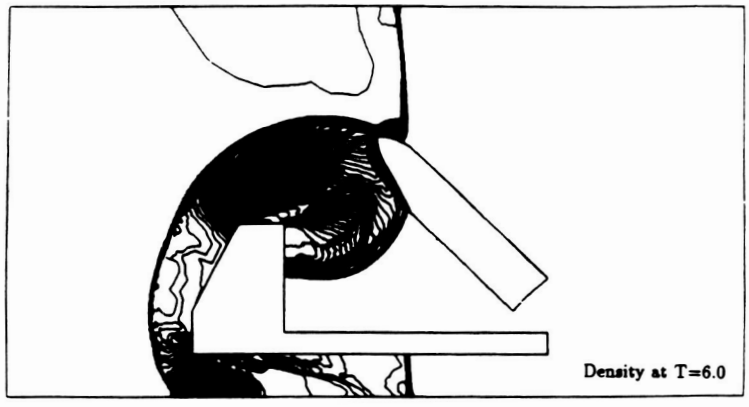


Figure 10(a). Shock impinging on two obstacles.





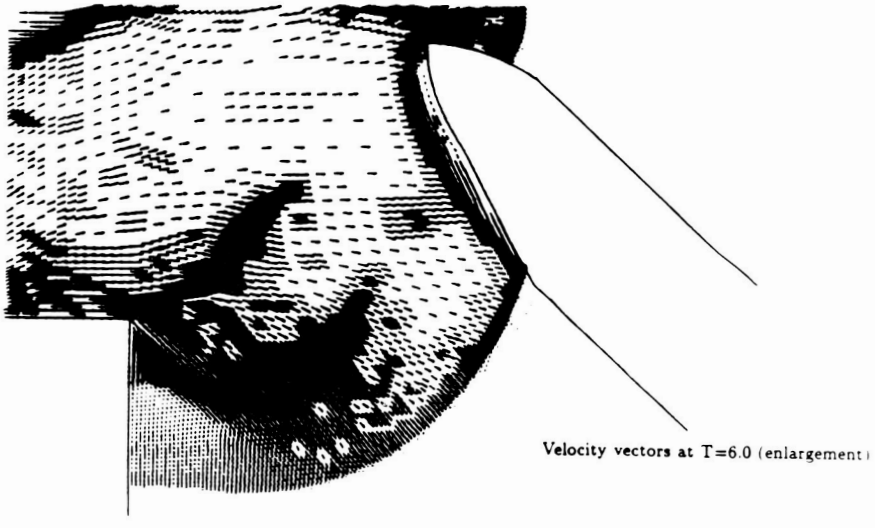
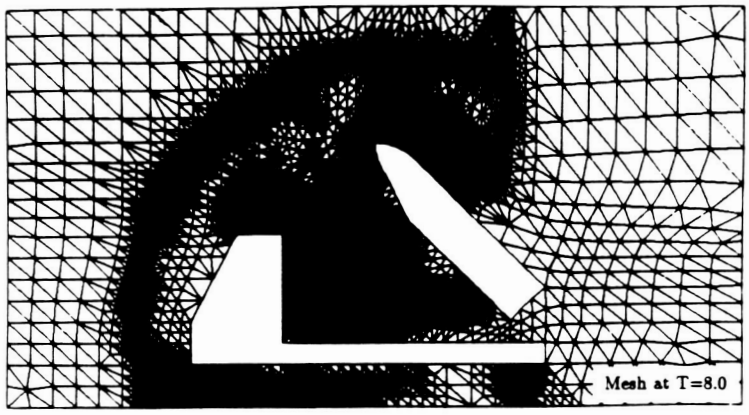


Figure 10(b). Shock impinging on two obstacles.



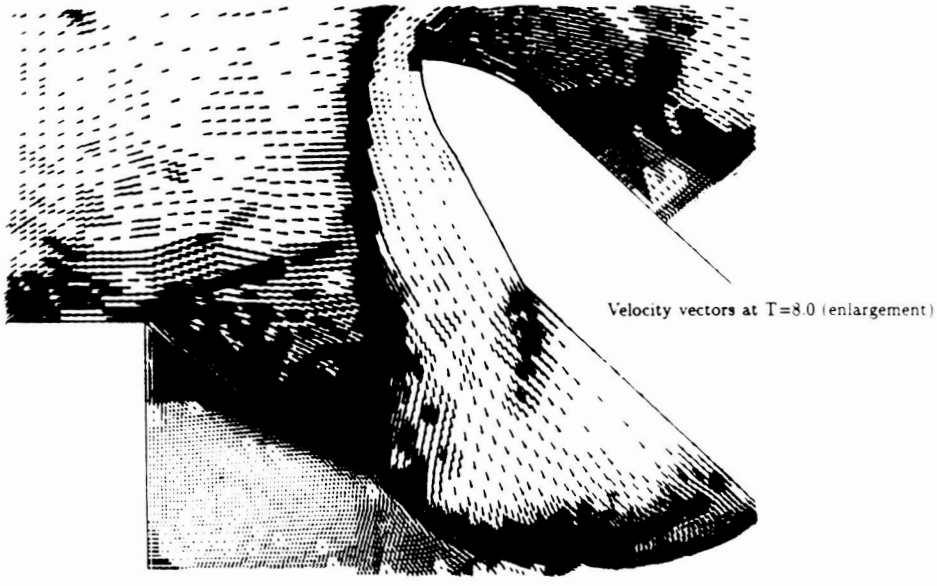
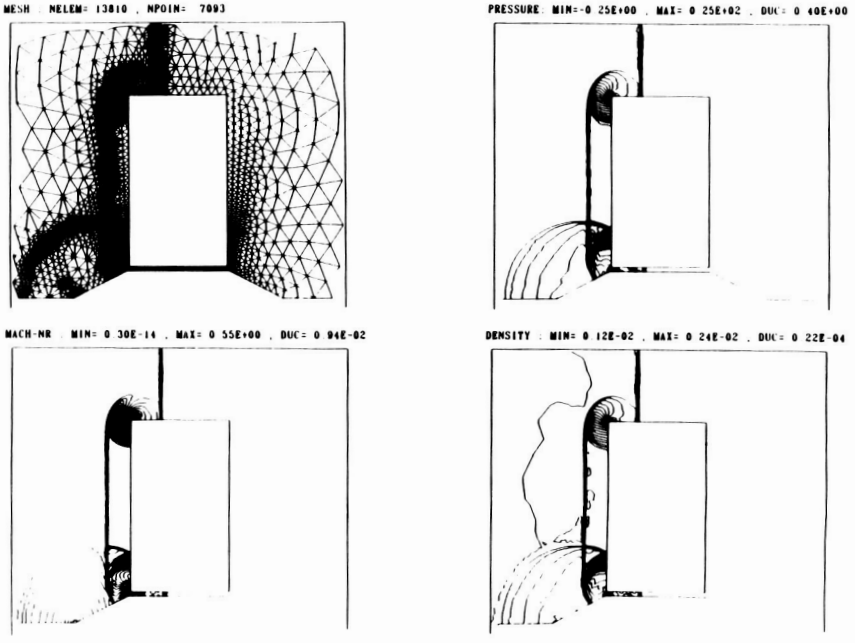


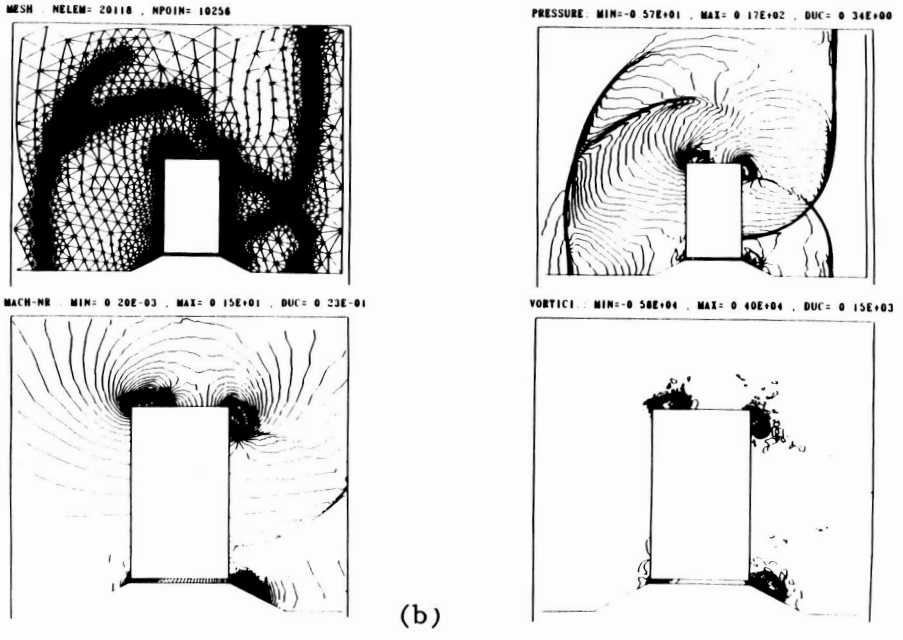
Figure 10(c). Shock impinging on two obstacles.

SHOCK INTERACTION WITH AN ELEVATED RAIL CAR
t=6.0 ms



(a)

SHOCK INTERACTION WITH AN ELEVATED RAIL CAR
t=22.0 ms



(b)

Figure 11. Shock-box interaction.

SHOCK INTERACTION WITH AN ELEVATED RAIL CAR
t=38.0 ms

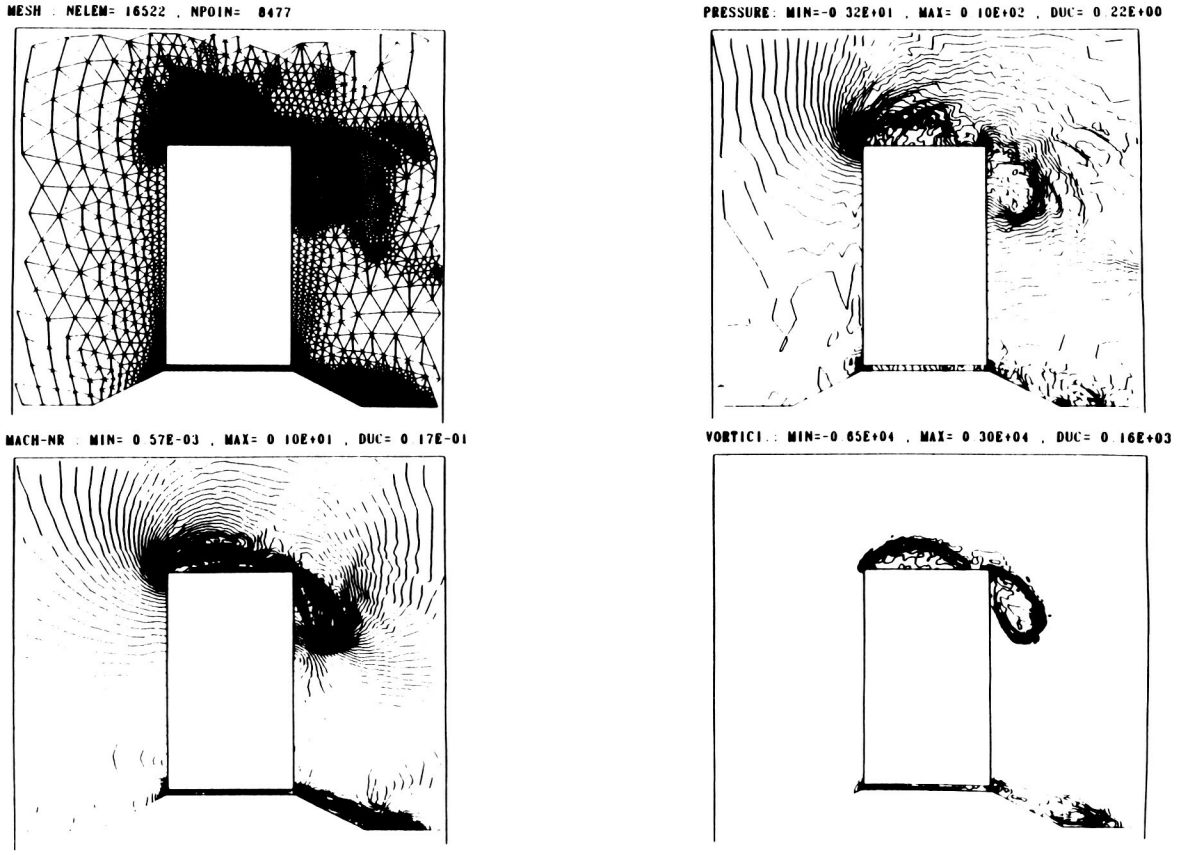
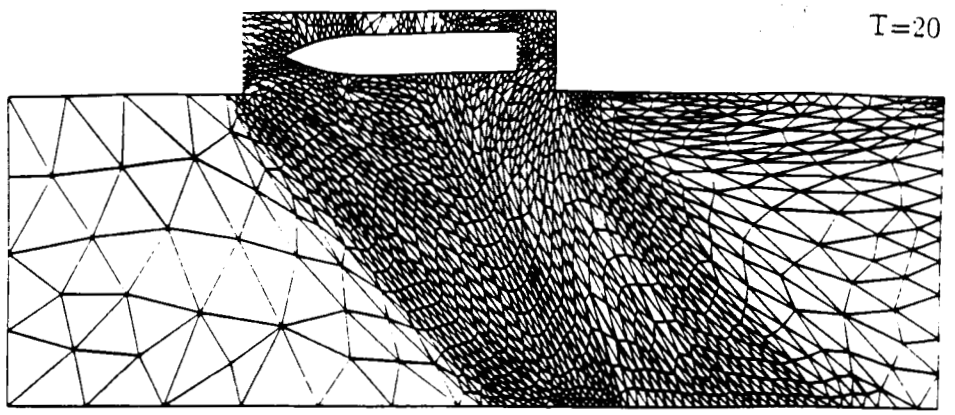
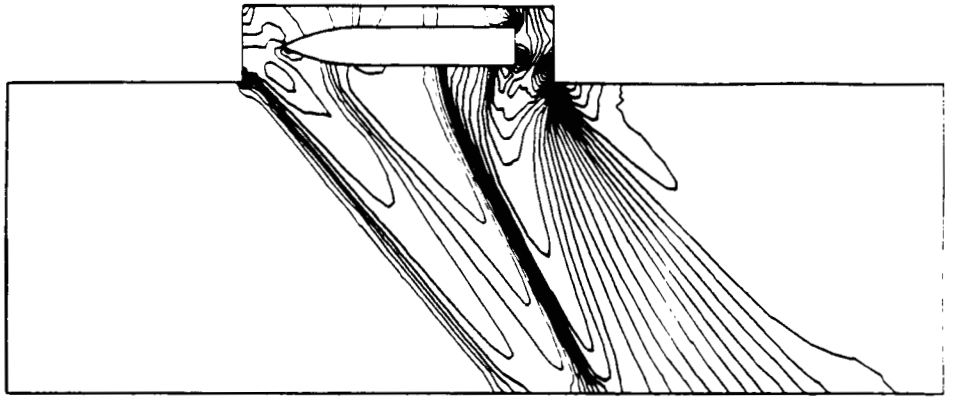


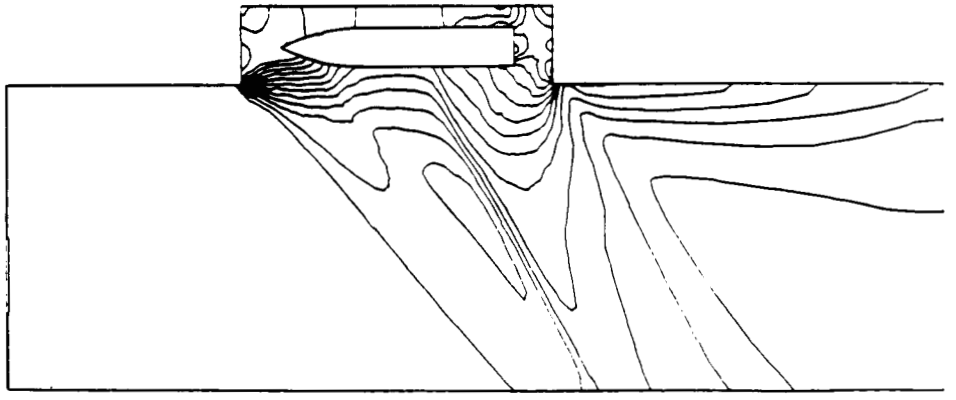
Figure 11(c) Shock-box interaction (concluded).



Mesh: NELEM=2264, NPOIN=1251

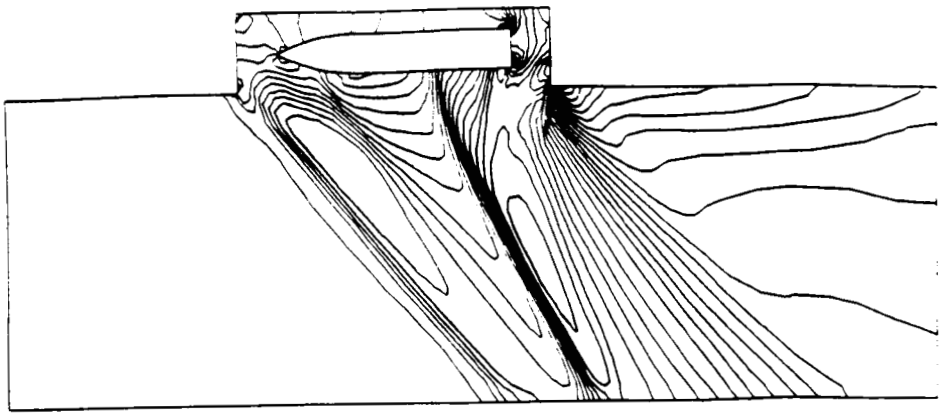


Pressure: Min=0.60, Max=2.30, Duc=0.05

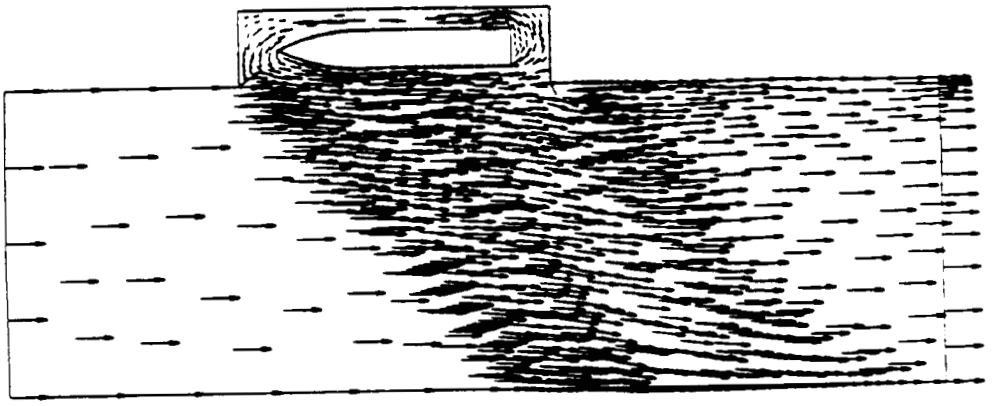


Mach-Nr.: Min=0.00, Max=1.60, Duc=0.10

Figure 12(a). Object falling into Ma = 1.5 free stream, T = 20.

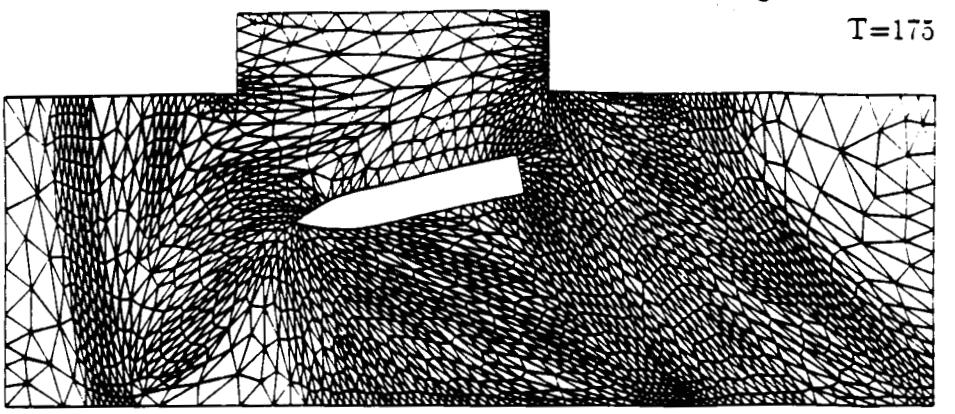


Density: Min=0.81, Max=2.20, Duc=0.05

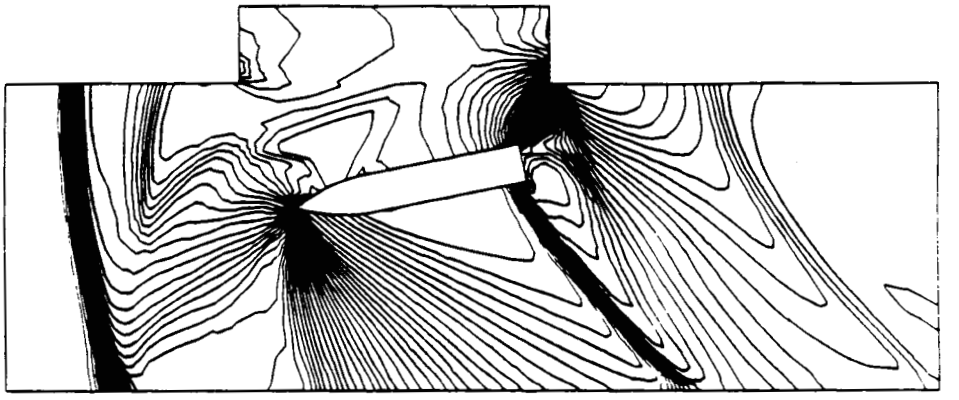


Velocity Vectors

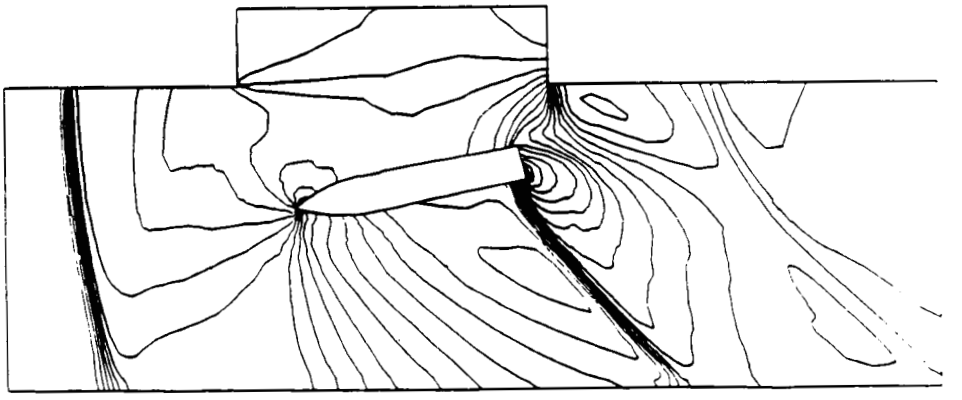
Figure 12(a). Concluded.



Mesh: NELEM=3047, NPOIN=1648

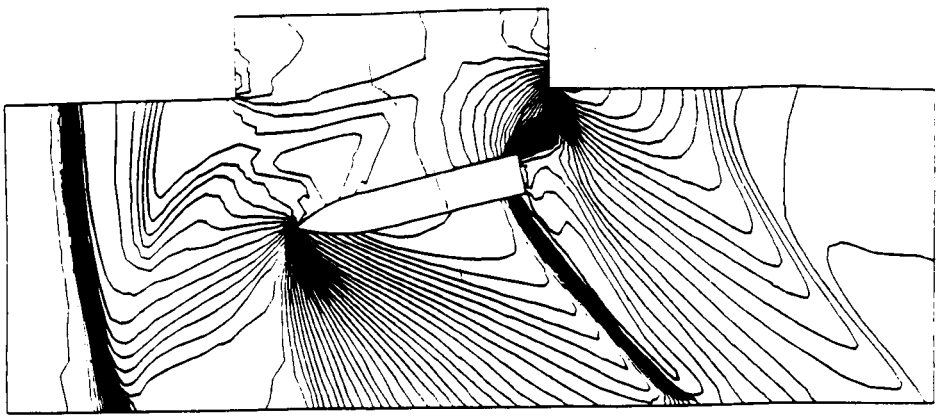


Pressure: Min=0.40, Max=3.30, Duc=0.05

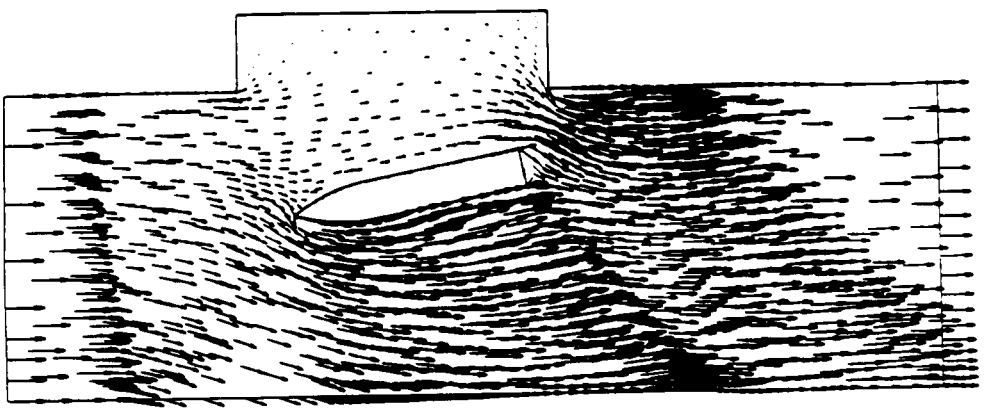


Mach-Nr.: Min=0.00, Max=1.90, Duc=0.10

Figure 12(b). Object falling into Ma = 1.5 free stream, t = 175.



Density: Min=0.56, Max=3.10, Duc=0.05



Velocity Vectors

Figure 12(b). Concluded.