REPORT DOCUMENTATION PAGE

plates: All DTIC reproductions will be in black and white"

AD-A250 232

(2)

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPO	9/15/90 - 9/14/91
	3/3/92	Final	

4. TITLE AND SUBTITLE

Development and Application of New Algorithms for The Simulation of Compressible Flows with Moving Bodies in Three Dimensions

5. FUNDING NUMBERS

61102F
2307/AS
AFOSR-89-0540

6. AUTHOR(S)

Rainald Lohner
Jean Cabello

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The George Washington University
School of Engineering and Applied Science
Washington, D.C. 20052

8. PERFORMING ORGANIZATION REPORT NUMBER

AFOSR-TR- 92 0369

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR/NA
Building 410
Bolling AFB, DC 20332-6448

DTIC
ELECTE
MAY 19 1992
S C D

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFOSR-
89-0540

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

A new CFD capability for compressible flows with moving bodies was developed. The salient features of this capability are:
a) Fast and reliable 3-D unstructured grid generation;
b) Flow solvers for moving frames of reference;
c) Adaptive mesh regeneration during transient runs;
d) On-line display of results;
e) Post-processing and movie-making capability.

92-12949

14. SUBJECT TERMS

UNSTRUCTURED GRID, 3-D, ADAPTIVE MESH

15. NUMBER OF PAGES

16 Plus 5 Append.

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	**UNCLASSIFIED**	**UNCLASSIFIED**	U

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std. 239-18
298-102

02 5 TA 075

Final Report for Contract AFOSR-89-0540

Contract Monitor: Dr. Leonidas Sakell

Prepared by:

Rainald Löhner and Jean Cabello
CMEE/SEAS, The George Washington University
Washington, D.C. 20052

Accomplishments

The work carried out under this contract may be subdivided according to the following topics:
1) Grid Generation and Pre-Processing;
2) Flow Solvers;
3) Grid Adaptation;
4) Rigid Body Motion;
5) Visualization and Animation;
6) Demonstration Calculation and Results.

The description of the main accomplishments of the present work are listed according to these topics in the following.

1. GRID GENERATION AND PRE-PROCESSING

All the unstructured grids required for the simulation of compressible flows with moving bodies were generated using the advancing front technique [1]. At the beginning of the present contract, we had a working, but slow and unreliable unstructured grid generator. The following improvements to this original capability were implemented as part of this grant.

1.1 Reliable 3-D Grid Unstructured Generator: We improved the reliability of the 3-D grid generator to a point where it could be used for the many regenerations that typically take place during a transient flow simulation with moving bodies. This significant increase in reliability was achieved by:

a) not allowing any bad elements during the generation process; and
b) enlarging and remeshing those regions where new elements could not be introduced.

Thus, we first attempt to complete the mesh, skipping those faces that do not give rise to good elements. If pockets of unmeshed regions remain, we enlarge them somewhat,

and regrid them. This 'sweep and retry' technique has proven extremely robust and reliable. In fact, the 3-D unstructured grid generator has not failed a single time since this capability was implemented.

1.2 Incorporation of Stretching in 3-D Grid Generator: The ability to generate stretched elements has the potential payoff of order-of-magnitude reductions in the required number of elements and CPU time. Stretching was taken into account during all phases of the grid generation process: when generating sides along lines, triangles on the surfaces, and tetrahedra in the domain. The most difficult of these was the generation of triangles on surfaces.

1.3 Fast 3-D Grid Generator: A typical CFD run with moving bodies requires many regriddings. Thus, it may cost as much to advance the flow solver as to regrid. Therefore, the grid generator has to be made as fast as possible. The following techniques were used to improve the performance of the advancing front grid generator:

a) Filtering: Typically, the number of close points and faces is far too conservative, i.e. large. As an example, consider the search for close points: there may be up to eight points inside an octant, but of these only one may be close to the face to be taken out. The idea is to filter out these 'distant' faces and points in order to avoid extra work afterwards. While the search operations are difficult to vectorize, these filtering operations lend themselves to vectorization in a straightforward way, leading to a considerable overall reduction in CPU requirements.

b) Automatic Reduction of Unused Points: As the front advances into the domain and more and more tetrahedra are generated, the number of tree-levels increases. This automatically implies an increase in CPU-time, as more steps are required to reach the lower levels of the trees. In order to reduce this CPU-increase as much as possible, all trees are automatically restructured. All points which are completely surrounded by tetrahedra are eliminated from the trees. This procedure has proven to be extremely effective. It reduces the asymptotic complexity of the grid generator to less than $O(N \log N)$. In fact, in most practical cases one observes a linear $O(N)$ asymptotic complexity, as CPU is traded between subroutine call overheads and less close faces on average for large problems.

c) Global H-refinement: While the basic advancing front algorithm is a scalar algorithm, h-refinement can be completely vectorized. Therefore, the grid generation process can be made considerably faster by first generating a coarser, but stretched mesh, and then refining globally this first mesh with classic h-refinement [2]. Typical speed-ups achieved by using this approach are 1:6 to 1:7.

1.4 Local Remeshing: Practical simulations revealed that the appearance of badly distorted elements occurred at a frequency that was much higher than expected from the element size prescribed. Given the relatively high cost of global remeshing, we explored the idea of local remeshing in the vicinity of the elements that became too distorted. Thus, wherever the elements become too distorted, we open up 'holes' in the mesh. We

then recompute the error indicators, and remesh adaptively the 'holes' using the advancing front method. Typically, only a very small number of elements (< 10) becomes so distorted that a remeshing is required. Thus, local remeshing is a very economical tool that has allowed us to reduce CPU-requirements by more than 60% for typical runs.

1.5 Mesh Smoothing and Optimization: After the advancing grid generator has filled the region to be meshed with elements, it is advisable to smooth the mesh in order to improve the element quality further. This not only avoids bad results due to deformed elements, but also allows larger timesteps to be taken, significantly reducing CPU requirements. Two different ways of optimizing or smoothing unstructured grids were explored.

1.5.1 Spring System Analogy with Local Regeneration: The spring system analogy has been widely used for smoothing unstructured grids. Each side or edge in the mesh is supposed to represent a spring. Thus, the force acting on each point is given by:

$$\mathbf{f}_i = c \sum_{j=1}^{ns_i} (\mathbf{x}_j - \mathbf{x}_i) \quad , \tag{1}$$

where c denotes the spring constant, $\mathbf{x}_i$ the coordinates of the point, and the sum extends over all surrounding points. The time-advancement for the coordinates is accomplished as follows:

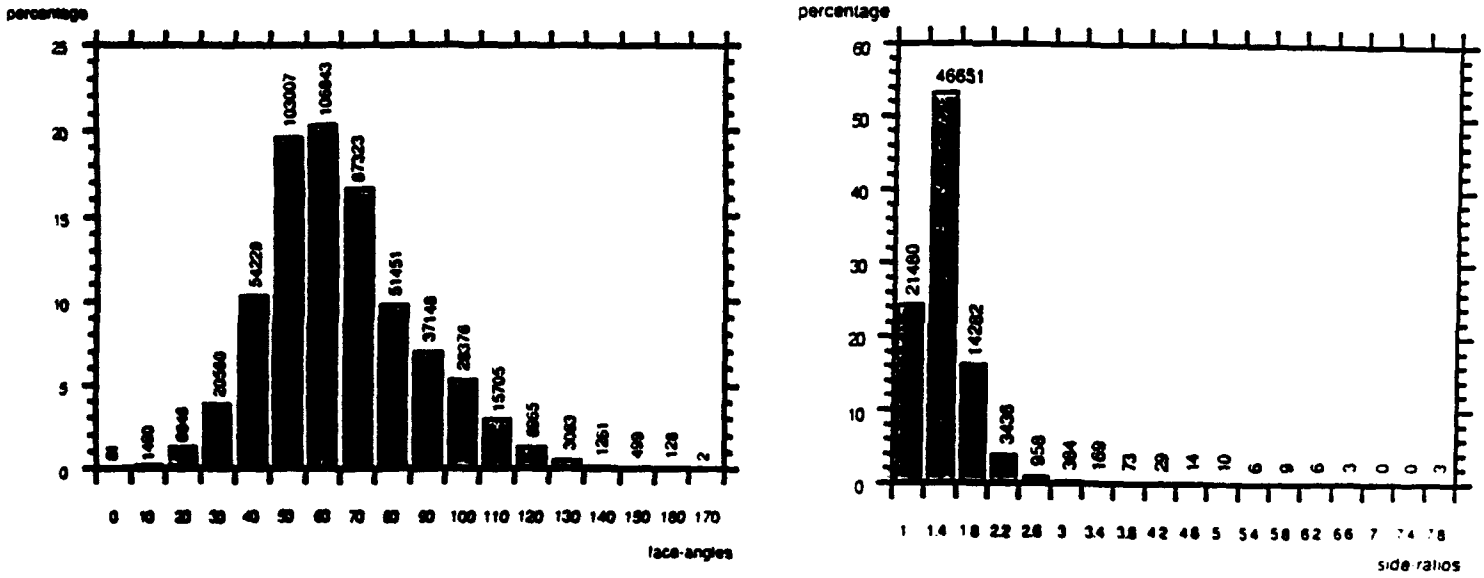
$$\Delta \mathbf{x}_i = \Delta t \frac{1}{ns_i} \mathbf{f}_i \quad . \tag{2}$$

At the boundary of the domain, $\Delta \mathbf{x} = 0$. Usually, 3-4 passes over the mesh yield an acceptable mesh. Unfortunately, this type of smoothing can produce negative elements. Typically, these are very few compared to the overall number of elements ($< 1\%$). This makes local remeshing an attractive option. Therefore, we treat these negative elements as distorted, remove them, and regrid these 'holes' again. To our knowledge, no other grid generator presently in operation uses smoothing on a routine basis. This has enabled us to produce the best unstructured grids presently available, as the statistics clearly show. Figure 1 is an example of such a statistic.

1.5.2 Smoothing via Optimization: The idea here is to state a functional that describes the optimal mesh. Then, the present mesh is optimized using a Pollak-Ribiere conjugate gradient algorithm. This technique has been used extensively, and very successfully, for the optimization of structured grids. However, unlike most finite difference techniques, it is completely general, and thus can be extended to unstructured grids. Its main advantage is that it will never give rise to elements with negative Jacobians. This technique has been described in detail in Refs.[3-5]. Starting from a mesh that was already smoothed using the spring system analogy, one observes always an increase in quality. In some cases, this improvement is dramatic, leading to increases in the

MESH QUALITY STATISTICS

Before Mesh Improvement



After Mesh Improvement

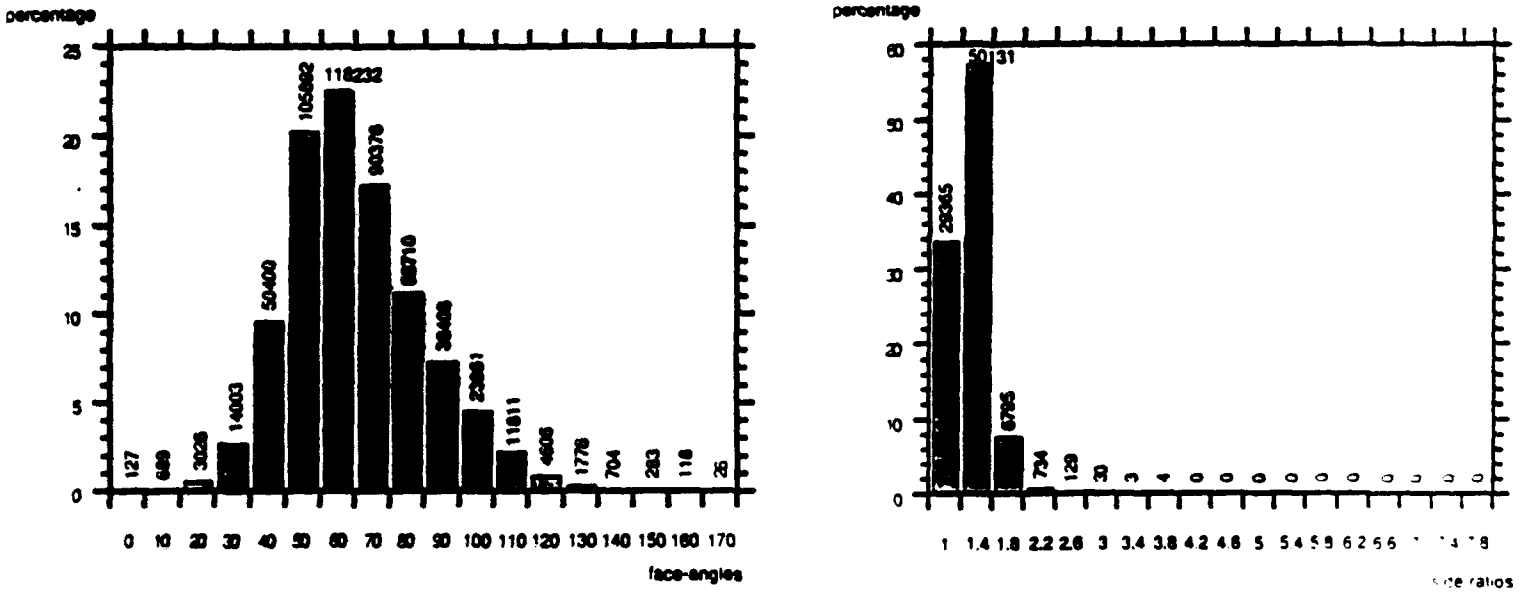


Fig 1 . Mesh Quality Statistic. This Figure Demonstrates the Significant Mesh-Quality Improvement Obtained using the Advanced 3-D Smoothing Algorithm Recently Developed

allowable timestep of an order of magnitude [5]. We have included more information on this optimization technique in Appendix 1-4.

1.6 Mesh Post-Processor: Practical runs show that even after smoothing or otherwise improving the mesh a few badly distorted elements may still remain. Remember that in 3-D, where even grids for inviscid flow calculations exceed 1 million tetrahedra, every small possibility becomes a reality. These badly deformed elements that remain at the end are removed with a special post-processor [5]. The elements in question are deleted by collapsing one of their sides and removing one of the points without creating any elements of worse shape.

1.7 Diagnostics: An extensive suite of diagnostics subroutines that check for input errors was added to the grid generator. The most common types of errors (double point definition, unclosed surfaces, surface orientation, surface crossings, incomplete boundary condition definition, etc.) are checked internally by the grid generator at run-time. Our experience has been that these diagnostics not only significantly improves the reliability of the CFD runs, but also reduces problem set-up times considerably by pointing the user to potential problems.

1.8 PREGEN/PREBACK: During the course of the present effort it became clear that the error-free problem setup was the most real-time intensive factor of a typical CFD run with moving bodies. Therefore, a considerable amount of effort was spent in reducing the burden of specifying the problem to be solved. Two tools were developed to this end:

1.8.1 PREGEN: PREGEN consists of a suite of subroutines that allows the user to produce grid generator compatible, error-free input in a faster way. PREGEN not only allows the user to exercise basic CAD-CAM operations (shrinking, translations, rotations, surface lofting, etc.), but also eases the merging of several parts of the surface into one cohesive, well-defined input-file. This allows the merger of files produced by different users and/or different surface generators. PREGEN has a whole series of built-in diagnostics to avoid such undesirable features as doubly defined points, isolated points or lines, badly defined lines or surfaces, etc. PREGEN is highly interactive, allowing the user to monitor every step during problem set-up in a graphical way.

1.8.2 PREBACK: PREBACK generates 3-D background grids by lofting or rotating a 2-D triangulation. In addition, it allows interactive operations to move background grid-points around (translation, rotation, shrinking, etc.), as well as to modify the grid-generation parameters in space (size, shape). By being able to see both the surface-defining CAD-CAM data and the background grid, the user can quickly generate a background grid that is suited for the desired distribution of element size, stretching and stretching directions in 3-D space.

2. FLOW SOLVERS

All the flow solvers used for the present effort are based on finite elements and operate on unstructured grids. At the beginning of the period of performance, the flow solvers were based on FEM-FCT techniques [6], and worked on element loops. This type of flow code has been the workhorse for many of the runs done. In the last half year we have developed some new flow solvers that were aimed specifically at reducing the indirect addressing (i/a) cost and the memory requirements. These newer flow codes are based on an edge data-structure, and use new renumbering techniques to reduce both the cache-misses as well as to increase the performance on vector machines. The main accomplishments in this area are detailed in the following.

2.1 ALE Capability: In order to handle the moving frames of reference associated with the moving finite elements, the partial differential equations need to be modified. This is most easily accomplished by the Arbitrary Lagrangian-Eulerian (ALE) formulation. The derivation of the equations may be found in [7]. Here, we just state the final form of the equations of motion. Given the velocity field $\mathbf{w}$ for the elements

$$\mathbf{w} = (w^x, w^y, w^z) \quad , \tag{3}$$

the Euler equations that describe an inviscid, compressible fluid may be written as

$$\begin{Bmatrix} \rho \\ \rho u^x \\ \rho u^y \\ \rho u^z \\ \rho e \end{Bmatrix}_{,t} + \begin{Bmatrix} (u^i - w^i)\rho \\ (u^i - w^i)\rho u^x + p \\ (u^i - w^i)\rho u^y \\ (u^i - w^i)\rho u^z \\ (u^i - w^i)\rho e + u^i p \end{Bmatrix}_{,i} = -\nabla \cdot \mathbf{w} \begin{Bmatrix} \rho \\ \rho u^x \\ \rho u^y \\ \rho u^z \\ \rho e \end{Bmatrix} \quad . \tag{4}$$

Observe that in the case of no element movement ($\mathbf{w} = \mathbf{0}$), we recover the usual Eulerian conservation-law form of the Euler equations. If, however, the elements move with the particle velocity ($\mathbf{w} = \mathbf{v}$), we recover the Lagrangian form of the equations of motion. From the numerical point of view, Eqn.(4) implies that all that is required when going from an Eulerian frame to an ALE-frame is a modified evaluation of the fluxes on the left-hand side, and the additional evaluation of source-terms on the right-hand side. These ALE equations were implemented in all the flow solvers used for the present effort.

2.2 Edge-Based Solvers: A significant reduction in indirect addressing (i/a) costs can be realized by going from an element to an edge-based data structure for the flow solver. We developed a new series of edge-based flow solvers. When developing these new codes, we incorporated all the coding lessons that we learned over the years. The latest code, FEFLO93, runs at a sustained rate of 115 MFlops on the CRAY-YMP, and has significantly less Flops per update than the previous code (FEFLO54). At the same

time, i/a costs were reduced by a factor of 3.5. We plan to continue the development of this new capability in the future, and expect to reduce i/a costs by another factor of 3 shortly.

2.3 Renumbering Strategies: This development was carried out specifically for machines with small cache-memory, like the CONVEX or the INTEL hypercubes. The idea is to still be able to operate in vector-mode, but to avoid large jumps in data accessed. Of the many techniques explored, the following two were the mosts successful:

2.3.1 Element Renumbering According to Smallest Point: The elements are renumbered according to their smallest point. This technique by itself reduced CPU times for typical runs by 50% on the IBM RISC-6000/530 for large grids.

2.3.2 Small Group Element Colouring: In order to avoid memory conflicts inside vector-loops, elements or edges have to be grouped together, or 'coloured'. For a CRAY, the aim is to achieve as small a number of groups with as large a vector as possible. For a CONVEX, such a modus operandi will lead to a large number of cache-misses. The idea is to reduce the vector lengths to 64 or 128, but to operate on local data as much as possible. This is accomplished by first renumbering according to smallest point, and then by colouring in small groups, always starting at the beginning.

2.4 Linelets: Any implicit flow solver (and we will need these for Navier-Stokes simulations) requires the solution of a large system of linear equations of the form

$$\mathbf{K}\mathbf{u} = \mathbf{r} \ , \tag{5}$$

where $\mathbf{K}$ is the matrix arising from implicit timestepping, $\mathbf{u}$ the desired vector of unknowns, and $\mathbf{r}$ the right hand side vector. The fastest way to solve such large systems of equations is through the use of unstructured multigrid solvers. They require good smoothers, as well as efficient intergrid transfer operators. During the present year, we developed a class of iterative solvers that lie between the complexity of multiple grids and the excessive memory requirements of direct solvers. Codes based on structured grids have explored for a long time the useful properties of line-relaxation. Line-relaxation offers an economical way to circumvent directional stiffness by joining together neighbors of neighbors along the line. Thus, it offers a practical way to derive good preconditioners and smoothers. The concept of lines translates to snakes in the context of an unstructured grid. The corresponding relaxation scheme to solve Eqn.(5) becomes

$$\mathbf{K}_1\mathbf{K}_2\Delta\mathbf{u} = \mathbf{r} - \mathbf{K}\mathbf{u} \ . \tag{6}$$

Here $\mathbf{K}_1, \mathbf{K}_2$ denote the entries of $\mathbf{K}$ for the active point-point combinations that define the snake. Observe that this formulation is not dimensionally consistent. Therefore, a better formulation is to use

$$\mathbf{K} = \mathbf{D}^{\frac{1}{2}}(\mathbf{I}+\mathbf{E})\mathbf{D}^{\frac{1}{2}} \approx \mathbf{D}^{\frac{1}{2}}(\mathbf{I}+\mathbf{E}_1)(\mathbf{I}+\mathbf{E}_2)\mathbf{D}^{\frac{1}{2}} \ , \tag{7}$$

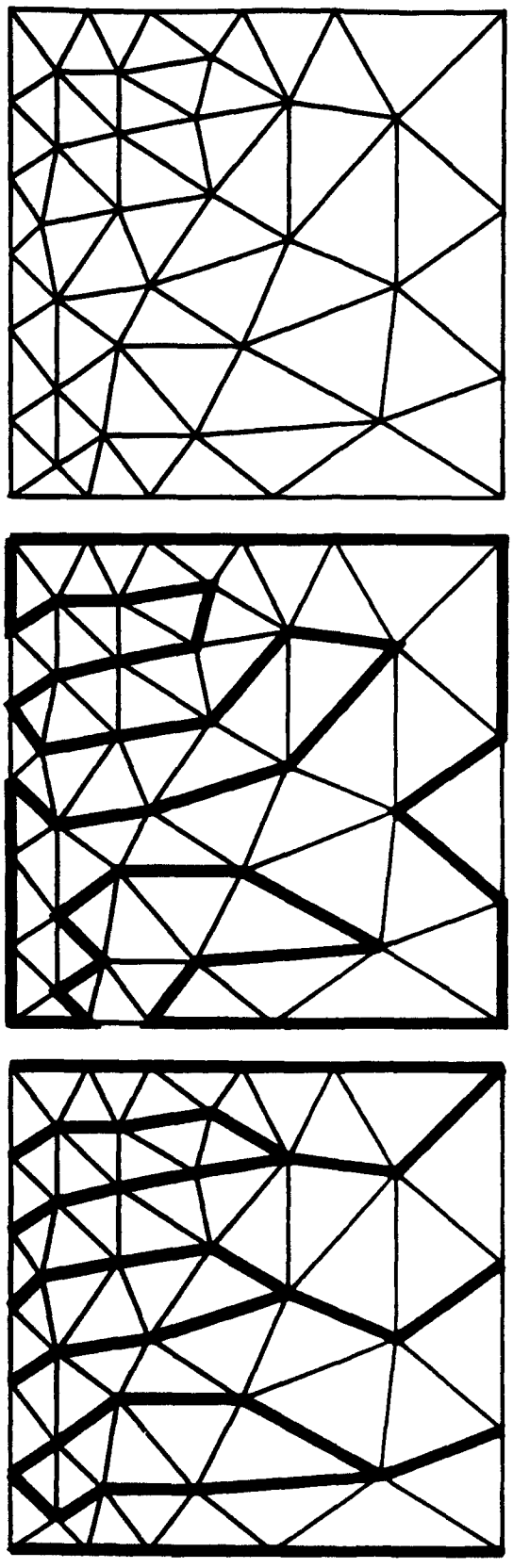


Figure 2: Snake and Linelets for a 2-D Discretization
Preferred Direction: (1,0)

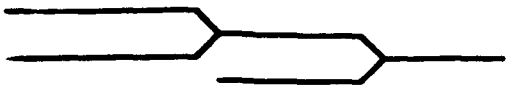
where $\mathbf{D}, \mathbf{E}$ denote the diagonal and off-diagonal entries of $\mathbf{K}$. Thus, the relaxation scheme becomes

$$\mathbf{D}^{\frac{1}{2}}(\mathbf{I}+\mathbf{E}_1)(\mathbf{I}+\mathbf{E}_2)\mathbf{D}^{\frac{1}{2}}\Delta\mathbf{u} = \mathbf{r} - \mathbf{K}\mathbf{u}\,. \tag{8}$$

In order to achieve higher rates of convergence, we use Eqn.(8) within a preconditioned Conjugate gradient algorithm. As an unstructured grid usually does not possess an equal number of gridpoints along a certain direction, the resulting snakes may often exhibit folding (see Figure 2b). This implies that the information flow from the domain to the boundary may be slowed down considerably. This is not important for smoothers, but crucial for the preconditioners required in one-grid solvers. In order to obtain a steady flow of information towards the boundaries in all directions, we reconnect the snake in the direction it intended to continue wherever it folds. This gives rise to a more complex structure, which we call linelet (see Figure 2c). Whereas the storage requirements of snakes are fixed ($3N$, where N is the number of unknowns), the storage requirements of linelets depend on the structure of the mesh and the renumbering chosen. Using reverse Cuthill-McKee ordering, as well as octrees, one observes $O(5N - 10N)$ storage requirements. This is deemed acceptable, as it takes much more than $6N$ operations to build a new right-hand side. As expected, the convergence rate of the preconditioned Conjugate gradient algorithm increases considerably when going from snakes to linelets. This completely new concept went through several stages of development. Starting with snakes, we soon realized the problems with folding, and had to develop ways to construct the linelets in an efficient way. Because the size of the system of equations to be solved depends on the numbering of nodes, near-optimal renumbering schemes had to be tested. We went through four generations of renumbering strategies. Compared to our current scheme, a simple-minded renumbering would yield equation systems that require 4-8 times more storage and CPU for the same performance. So far, we have used this linelet-based solver for scalar elliptic problems. In the coming year, we will extend it to the fully coupled compressible Navier-Stokes case.

2.4.1 Vectorization of Linelet-Matrix Solution: The first linelet-matrix solvers we implemented were based on the general Crout LU decomposition, which can be found in many Finite Element textbooks. Performance tracing on the CRAY-YMP showed that this portion of the code was running at about 2.5Mflops, dismal for a machine that is rated at 250Mflops per processor. This very low speed was the result of short vector lengths in the mainly tridiagonal structure of the linelet-matrix. Vectorization of both the LU factorization, as well as the forward and backward reduction during solution can be accomplished if one processes in parallel as many linelets as possible. The entries in the linelet-matrix are categorized into diagonal (points with prescribed unknowns), tridiagonal or non-tridiagonal. At each stage, all active non-tridiagonal inhibitors are processed first. Thereafter, all available tridiagonal entries are processed in vector-mode. Towards the end of the factorization or reduction process, the available number of tridiagonal locations may diminish to a point where scalar processing of the remaining entries is faster than extremely short vector-loops. For this reason, a scalar

Linelet



Matrix Solution

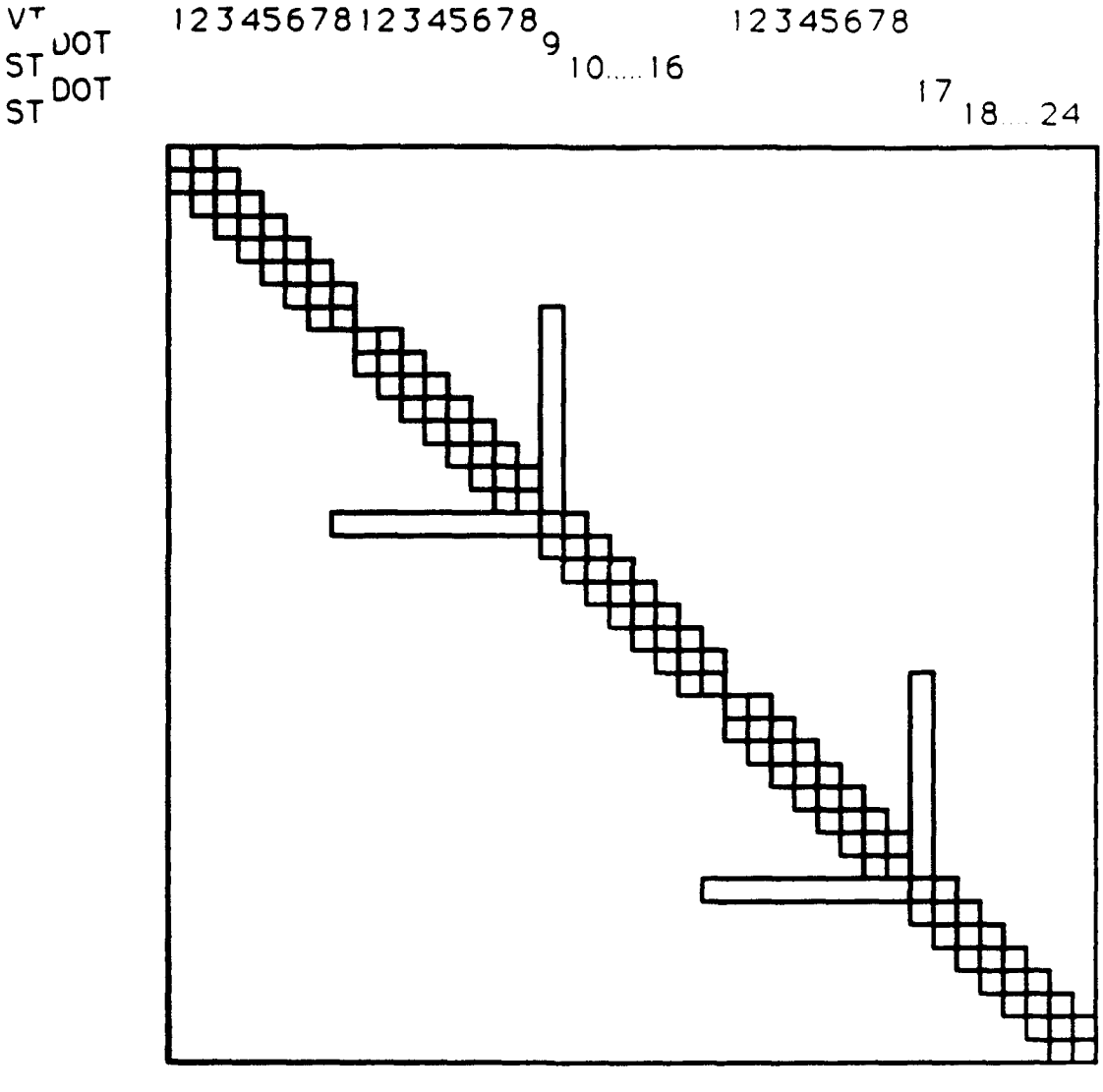


Figure 3: Efficient Solution of Linelet Matrix Systems

tridiagonal category is added to the complete solver. The complete factorization or reduction process, illustrated in Figure 3, is then given by:

1. Identify 'Origins' (V)
2. Solve For Non-Tridiag. 'Inhibitors' (SV,DOT)
3. Solve For Available Tridiagonals (V)
4. Solve For Remaining Linelets (FS)
5. If Not Finished: GOTO 2.

The vectorized version of the linelet-solver presently runs at 15Mflops. This represents a speed-up of 7 as compared to the scalar version, but is still deemed unacceptable. Further research is being devoted to this subject at the present time. Of course, for the solution of blocks of unknowns (as required for the compressible Navier-Stokes equations), as compared to a single variable, the achievable Mflop-rate is expected to be significantly higher.

2.5 Coding of Body Force Reduction Subroutines: In order to couple the flowfields with the rigid body motion one has to extract the forces and moments exerted by the fluid on the bodies present. These data reduction subroutines were coded and tested.

2.6 Turbulence Model: The algebraic Baldwin-Lomax turbulence model [8] was implemented in the 2-D flow solver. This is the simplest way to add the effects of turbulence. The model is straightforward to implement as the only difference in the flow equations is the increase of viscosity. The model represents the inner and outer parts of a boundary layer as follows:

Inner part:

$$\mu_t = \rho|\omega|\left[ky(1 - exp(-y/A))\right]^2$$

Outer part:

$$\mu_t = \alpha C_{cp}\rho Y_{max} F_{max}\gamma \quad ,$$

with

$$F(y) = y|\omega|(1 - exp(-y/A)) \quad .$$

The exponential factor in the inner part is the Van Driest damping factor which matches the damping of the wall. In the second equation, F_{max} is computed along a normal to the wall. This is particularly easy when dealing with structured grids, but requires some effort for unstructured grids. The required data structures were discussed by Rostand [9]. One complete evaluation of the turbulent viscosity requires the following transfer of information:
a) Vorticity from elements or points in the mesh to the appropriate normals to the walls. Along each normal, the vorticity $|\omega|$ is required to evaluate the turbulent viscosity. This

transfer of vorticity is accomplished with a linked list of intersections of normals to the wall with elements of the mesh. This list is constructed by starting from the surface and moving along the normal. All that is required to do so is a list of elements surrounding elements.

b) Transfer of the turbulent viscosity from the normals to the walls to the elements or points of the mesh. In the present case, we transfer to points. This not only reduces the transfers required (there are less points than elements in a mesh), but also introduces a beneficial smoothing effect at element level. For each point in the mesh close to wetted surfaces, we find the closest normals surrounding it, and then the two closest points along each of the normals. Thus, a point in the mesh assembles the turbulent viscosity from four points along the normals of a wetted surface. The search for the closest points is done using quad-trees [10]. In the case of the mixing of several boundary layers, μ_t is computed from the contribution of each wall weighted by its distance to the point.

3. ADAPTIVE REFINEMENT

Any adaptive refinement scheme is composed of an error indicator and a refinement strategy. Adaptive remeshing was chosen for the refinement strategy. This seems natural, as it allows to couple adaptation and body motion in a straightforward way.

3.1 Development of Suitable Error Indicators for Adaptive Remeshing: We extended the highly successful modified interpolation theory error indicator [2,11] to 3-D. This error indicator can be generalized to multidimensional situations by defining the following tensors:

$$(D^0)^I_{kl} = h^2 c_n \int_\Omega |N^I_{,k}||N^J_{,l}||U_J| d\Omega \quad , \tag{9}$$

$$(D^1)^I_{kl} = h^2 \int_\Omega |N^I_{,k}||N^J_{,l} U_J| d\Omega \quad , \quad (D^2)^I_{kl} = h^2 \left| \int_\Omega N^I_{,k} N^J_{,l} d\Omega \; U_J \right| \quad , \tag{10}$$

which yield an error matrix $\mathbf{E}$ of the form:

$$\mathbf{E} = \begin{Bmatrix} E_{xx} & E_{yx} & E_{zx} \\ E_{xy} & E_{yy} & E_{zy} \\ E_{xz} & E_{yz} & E_{zz} \end{Bmatrix} = \mathbf{X} \cdot \begin{Bmatrix} E_{11} & 0 & 0 \\ 0 & E_{22} & 0 \\ 0 & 0 & E_{33} \end{Bmatrix} \cdot \mathbf{X}^{-1} \tag{11}$$

The principal eigenvalues of this matrix are then used to obtain reduction parameters $\xi_{j'j'}$ in the three associated eigenvector directions. Note that due to the symmetry of $\mathbf{E}$ this is an orthogonal system of eigenvectors that defines a local coordinate system. The required eigenvalue/eigenvector solvers for 3X3 matrices were vectorized and incorporated into the code.

3.2 Smoothing of Element Size, Stretching and Stretching Directions: In the context of transient problems it is very important to generate grids which do not exhibit minimum element sizes that are much smaller than the prescribed minimum element size.

Practical calculations indicated that the grids produced by simply taking the described error indicator and the resulting distribution of element sizes, stretchings and stretching directions did not meet this requirement. In other words, they were too irregular. Far superior grids were obtained by smoothing and limiting the initial distributions obtained for element sizes, stretchings and stretching directions.

4. RIGID BODY MOTION

Rigid body motion algorithms are essential for moving body simulations. On the other hand, just as CFD is more than a flow solver, so rigid body motion is more than a rigid body timestepping scheme. We found that in order to make rigid body motion user-friendly, i.e. accessible to a larger user group, significant graphical interface developments were required.

4.1 Development of a Rigourous Rigid Body Motion Algorithm: If one simply integrates the motion of rigid bodies without enforcing the rigidness explicitly, the shape may change over the course of the calculation. By transforming at every timestep to the body reference system, this is avoided. At the same time, the moments of inertia tensor only needs to be inverted once, and is much easier to construct.

4.2 PREMOV: In order to avoid costly mistakes by wrong user-input, we developed a graphical pre-processor for cases where the body motion is prescribed. This code reads the same input data as the actual flow-solver, but displays immediately the trajectories prescribed. In doing so, the user can assess before the run whether the correct trajectories were specified in the input file. Given that a user may have very specific motion-types in mind, a graphical checking tool like PREMOV is essential.

5. VISUALIZATION AND ANIMATION

Many of the breakthroughs that took place over the performance period of the present grant were only possible due to the advent of powerful 3-D graphics workstations. On the other hand, we had to harness this power by developing extensive libraries to display whatever was required. The following tools were developed as part of this effort.

5.1 ONLIDISPL: This library allows to display any desired quantity while the CFD code is running interactively on a workstation. Thus, it allows to monitor and check a run as it proceeds in real time. ONLIDISPL has proven invaluable for input data checking and debugging, adding a new dimension of user-friendliness. On the other hand, it is clear that big runs can not be done on a workstation. But the user may want to run a coarser mesh interactively on the workstation, make sure everything is working as desired, and then run the big problem on a supercomputer in batch mode.

5.2 FEPOST3D/MOVIESUBS: FEPOST3D is based on FEPLOT4D, our standard 3-D plotting capability, but performs all the CPU-intensive filtering operations on the supercomputer within any of the 3-D codes. Only the plane or surface triangulations are sent back to the SGI-IRIS-4D/80GT for plotting. The user specifies before the

run the planes and surfaces to be inspected. The typical size of plot-files is reduced from 130-160Mbytes (complete 3-D flowfield, 2Mtetra) to 2.5-4Mbytes (plane/surface). Although seemingly trivial, this capability is essential when trying to produce a movie, or running large 3-D problems via a network.

5.3 FEMOVIE3D: We completed a first movie-making capablity for 3-D runs. After obtaining the dumps from the CRAY, FEMOVIE3D will run in batch mode on the IRIS and generate a series of pixel-dumps. These pixel-dumps are then shipped to a movie-making center to obtain either VHS or 0.25in format movies. Although we consider this as only the first generation of a more sophisticated movie-making software, it still represents a significant development, as it enabled us to assess the problems and important ingredients that a good movie-making toolkit must possess.

6. DEMONSTRATION CALCULATIONS AND RESULTS

6.1 Unstructured Grid/Remeshing Transient 3-D Runs: With the developed tools we performed the first ever transient unstructured 3-D runs with adaptive remeshing. The results are summarized in [2], which is included here as Appendix 5.

7. PUBLICATIONS

All of the developments listed above were reported extensively in the literature. The main papers published are listed in chronological order:

[1] R. Löhner and J.D. Baum - Unstructured Grid Methods for Store Separation; *Proc. 8th JOCG Airframe/Stores Compatibility Symp.* , Ft. Walton Beach, Fl., October 23-25, 1990.

[2] R. Löhner - Three-Dimensional Fluid-Structure Interaction Using a Finite Element Solver and Adaptive Remeshing; *Computer Systems in Engineering* 1, 2-4, 257-272 (1990).

[3] R. Löhner and J.D. Baum - Three-Dimensional Store Separation Using a Finite Element Solver and Adaptive Remeshing; *AIAA*-91-0602 (1991).

[4] J. Cabello, R. Löhner and O.P. Jacquotte -A Variational Method For the Optimization of Directionally Stretched Elements Generated by the Advancing Front Method (AFM) - *Proc. Third Int. Conf. on Numerical Grid Generation in CFD and Related Fields*, Barcelona, Spain, June 3-6 (1991).

[5] J. Cabello, R. Löhner and O.P. Jacquotte - A Variational Method for The Optimization of Two- and Three-Dimensional Unstructured Meshes - *Proc. 1st U.S. National Congress on Computational Mechanics*, Chicago, Illinois, July 21-24 (1991).

[6] J. Cabello, R. Löhner and O-P. Jacquotte - A Variational Method for the Optimization of Two- and Three-Dimensional Unstructured Meshes; *AIAA*-92-0450 (1992).

8. CONCLUSIONS AND OUTLOOK

We have developed a CFD capability to compute 3-D compressible flows with moving bodies. This is a new capability that so far is unmatched. It represents a new major step in CFD, as it allows to compute flows of engineering interest that were previously untractable by CFD means. The developments that took place over the last two years have been decribed.

In the future, we plan to expand the developed capabilities as follows:

a) Development of Flow Solvers for high Re-number Viscous Flows: We will extend the linelet-concept to the fully coupled linear equation systems that arise for implicit Navier-Stokes solvers.

b) Turbulence Models: We will incorporate the $k-\epsilon$ model into the implicit Navier-Stokes solver.

c) Development of suitable gridding algorithms for high Re-number viscous flows: We plan to start with a semi-structured approach, whereby we construct prisms away from the triangulated surfaces. Although not general enough, this will serve as a starting point for later developments, and a way to gather experience for simulations of high Re-number flows.

d) Development of suitable error indicators for high Re-number viscous flows: The idea here is to develop error indicators that sense were to refine boundary layers and shear layers, and that work even for highly stretched grids.

e) Improvements in movie-making capabilities: The main aim of this improvement is to be able to run the movie on the workstation before going to the movie-making center. At the same time, we must be able to compress the information to an extent that a 3min movie can be stored effortlessly on a small disk. This will be accomplished through image compression algorithms.

f) Further test runs: we plan to look for available experimental and/or numerical data in the literature in order to set up runs to test the algorithms developed.

9. REFERENCES

[1] R. Löhner and P. Parikh - Three-Dimensional Grid Generation by the Advancing Front Method; *Int. J. Num. Meth. Fluids* 8, 1135-1149 (1988).

[2] R. Löhner - Adaptive Remeshing for Transient Problems; *Comp. Meth. Appl. Mech. Eng.* 75, 195-214 (1989).

[3] J. Cabello, R. Löhner and O.P. Jacquotte -A Variational Method For the Optimization of Directionally Stretched Elements Generated by the Advancing Front Method (AFM) - *Proc. Third Int. Conf. on Numerical Grid Generation in CFD and Related Fields*, Barcelona, Spain, June 3-6 (1991).

[4] J. Cabello, R. Löhner and O.P. Jacquotte - A Variational Method for The Optimization of Two- and Three-Dimensional Unstructured Meshes - *Proc. 1st U.S. National Congress on Computational Mechanics*, Chicago, Illinois, July 21-24 (1991).

[5] J. Cabello, R. Löhner and O-P. Jacquotte - A Variational Method for the Optimization of Two- and Three-Dimensional Unstructured Meshes; *AIAA*-92-0450 (1992).

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

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

[8] B.S. Baldwin and H. Lomax - Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows AIAA 78-257 (1978).

[9] P. Rostand - Algebraic Turbulence Models for the Computation of 2-D High Speed Flows Using Unstructured Grids; *ICASE Rep.* 88-63 (1988).

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

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

APPENDIX 1-4: GRID OPTIMIZATION

1 - Introduction

Unstructured meshes, especially triangles in two dimensions and tetrahedra in three dimensions, are recognized as the more suitable to fit any complex arbitrary domain shape. Various algorithms [1-7,15] have been developed to fill any arbitrary domain while preserving element quality to some extent. Unfortunately, for most techniques, the required versatility of the algorithms used to generate unstructured grids leads to some loss of mesh quality. Since finite element solvers are sensitive to the quality of the grid employed and all generation techniques to date may generate bad elements, means to improve or smooth an initial mesh must be found.

Most of the smoothing techniques available in the literature are based on the spring system analogy [8,9]. The mesh is viewed as a network of nodes connected by springs of constant stiffness. The smoothing process consists in moving the nodes until the spring system is in equilibrium. Even though this method is easy to implement and presents the advantage of requiring low CPU-time and memory, some drawbacks have been reported. For example, for non-convex domains, the spring analogy smoothing may lead to negative jacobians in the vicinity of a concave boundary.

2 - Optimization of two-dimensional unstretched meshes

In the present work, we started using a variational method [10-14] to optimize two-dimensional unstretched triangular meshes. The approach presented is based on the principles of continuum mechanics (appendix 1). The theoretical framework is inspired by the three-dimensional non linear elasticity theory. Starting from basics geometrical axioms, lying on mechanical properties, we introduce a functional defined as a measure of the deformation between a reference cell and a current cell. This functional depends upon the principal invariants of the deformation matrix (metric tensor) derived from the transformation between the two cells. The mesh optimization leads to a minimization problem of an energy-like hyperelastic material, quasi incompressible. A convexity property is imposed to the functional; then the minimization problem is well-posed. The minimization will be performed by a conjugate gradient method (appendix 4).

3 - Optimization of two-dimensional stretched meshes (art. Barcelona)

Dimensional features appear quite often in multi-dimensional fluid flows (e.g. shocks, contact discontinuities, etc ...). It has been shown by previous work [7] that allowing directionally stretched elements is an efficient way to increase the solution accuracy without drastically increasing the number of elements.

The advancing front method (AFM) proposed by Peraire et al. [7], Löhner [15] , Löhner and Parikh [6] is able to generate stretched elements in the direction of the flow features. Unfortunately, the unstructured grid generated with the advancing

front technique may lack the desired distribution of the grid points in some regions. As the numerical stability and accuracy of flow solutions may be affected if the grid points are not distributed smoothly, the initial mesh generated by the AFM must be smoothed.

The variational method was extended to the optimization of two-dimensional stretched triangular meshes generated by the AFM. Being able to cope with the optimization of two-dimensional unstretched meshes, we needed to devise a way to transform the stretched mesh into an unstretched mesh. This was done by the introduction of a transformed space. A transformation, depending on the mesh parameters used to construct a stretched element, was performed to obtain an element which should look like an equilateral element. This transformation is just a rotation of the element, in order to align the stretching direction with the x-axis and a shrinking of the element (along the x-axis) of a factor equal to the inverse of the stretching factor. Asuming that the element generated in the physical space has exactly the characteristics given by the mesh parameters, then the resulting element after transformation would be an equilateral triangle. The optimization is performed on the transformed space. Finally, the optimized mesh is transformed back (using the inverse transformation) from the transformed into the physical space.

4 - Optimization of three-dimensional stretched meshes

In three dimensions, a transformation between the physical and transformed space was devised as an extension of the two-dimensional transformation. In the transformed space, all tetrahedra are assumed to be regulars (all faces being equilateral triangles). The variational method is applied in the transformed space. First, a measure of the deformation between a transformed and reference element is computed. The total deformation is obtained by summation of the elementary contributions. Second, the interior mesh points are repositioned in order to minimize the total deformation. Finally, the elements are transformed back to the physical space.

5 - Optimization of three-dimensional unstretched meshes (art. AIAA-92-0450)

For the special case of three-dimensional unstretched tetrahedral meshes there is no need to go back and forth between the physical space and the transformed space. The method is directly applied in the physical space. A special version of the code has been written for the optimization of unstretched meshes. This version requires less memory and less cpu-time than the general one.

6 - Post-processing

Using a moving node method we are tied to the topology given by the initial mesh. Some routines have been written which change the topology of the mesh.

These routines are used to either remove cells with negative jacobians or cells which are marked as distorted according to a quality criterion. These routines have been isolated and can be used as a "black box" during the post-processing step of any initial or optimized mesh. During the conjugate gradient iterations we may get folded elements with negative jacobians. These elements will be unfolded as the conjugate gradient iterations are carried along and no cell with negative jacobian will remain when the minimum is reached. Nevertheless, these negative jacobians will usually appear in some pathological regions of the mesh. We suspect that these elements originate where the advancing front closes small gaps in order to fill the whole domain. Therefore, it is preferable to take out these elements, changing locally the topology of the mesh, rather than carrying the conjugate gradient iterations to convergence.

7 - Minimization algorithm

Within the domain, the optimal location of the physical nodes is computed by an iterative conjugate gradient algorithm (appendix 4). The multidimensional minimization problem is reduced to a succession of one dimensional problems in a descent direction. The one dimensional search for a minimum leads to find the root of a three (in two dimensions) or five (in three dimensions) degree polynomial of one variable. The coefficients of this polynomial depend on the differential of σ. The formulas for these differentials are given in appendices 2 and 3.

8 - Conclusion

The scope of a variational method, widely used for the optimization of structured grids, has been extended in order to perform the optimization of two- and three-dimensional stretched or unstretched unstructured meshes. This moving node method is not tied to the advancing front method or to any particular generation technique and can be initialized by any arbitrary grid, even a grid with overlapped cells..

In the paper presented in Barcelona, the variational method was able to give acceptable results in a two-dimensional case where the spring analogy failed. The robustness and efficiency of the method was tested for a mesh in which the location of the points was altered. The method was able to unravel this "chaotic" mesh and to give a smooth one. For stretched meshes the method was able to exert more control on the grid according to the parameters specified by the user in the background mesh.

In the paper presented at the AIAA meeting, for the optimization of unstretched three-dimensional tetrahedral meshes, a more appropriate formulation than the general one, have been presented in order to save cpu-time and memory space. For all the examples presented the quality of initial meshes, already smoothed by the spring analogy, have been improved with an increase of about eight to nine

per cent around the optimal angle (angles between fifty and eighty degrees) along with a decrease of angles leading to flat elements (i.e. angles lower than thirty and greater than one hundred and fifty degrees). Also, for all examples, an increase of a factor two approximately is obtained for the minimum radii ratio of the inscribed over the circumscribed sphere. This is an important result, given that the minimum radii ratio drives the allowable time step of finite element solvers.

References

[1] J. F. Thompson - Numerical Grid Generation - *North-Holland* (1980)

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

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

[4] T. J. Baker - Developments and Trends in Three-Dimensional Mesh Generation - *Appl. Num. Math.* Vol. 5, 275-304 (1989)

[5] T. J. Baker - Three Dimensional Mesh Generation by Triangulation of Arbitrary Point Sets - *AIAA-87-1124*

[6] R. Löhner and P. Parikh - Three-Dimensional Grid Generation by the Advancing Front Method - *Int. J. Num. Meth. in Fluids* , Vol 8, 1135-1149 (1988).

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

[8] J. C. Cavendish - Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method- *Int. J. Num. Meth. Eng.* , 48, 679-696 (1974).

[9] R. Carcaillet, S. R. Kennon and G. S. Dulikravich - Optimization of Three-Dimensional Computational Grids - *AIAA Paper 85-4087*, Colorado Springs, CO.

[10] O.- P. Jacquotte - A Mechanical Model for a New Generation Method in C.F.D. - *Comp. Meth. Applied Mech. Eng.*, Vol. 26, 323-338; also *ONERA T.P. N^0 1988-48.*

[11] O.- P. Jacquotte and J. Cabello - A Variational Method for the Optimization and Adaptation of Grids in C.F.D. - 2^{nd} *Int. Conf. on Num. Grid Generation in C.F.D., Miami Beach, Florida, USA* , December 5-8, 1988, Pineridge Press; also *ONERA T.P. N^0 1988-99.*

[12] O.- P. Jacquotte and J. Cabello - A New Variational Method for the Generation of Two- and Three-Dimensional Adapted Grids in C.F.D. - *7 th Int. Conf. in*

Finite Elements Meth. in Flow Problems, Huntsville, Alabama, USA , April 3-7, 1989; **also** *ONERA T.P. N^0 1988-99.*

[13] O.- P. Jacquotte and J. Cabello - Three-Dimensional Grid Generation Method Based on a Variational Principle - *La Recherche Aérospatiale, English version, N^0* 1990-4.

[14] J. Cabello - Méthode Variationnelle
d'Optimisation et d'Adaptation de Maillages Structurés Tridimensionnels - *Doctoral Dissertation Université Paris Sud (ORSAY)*, February, 1990.

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

[16] J. Cabello, R. Löhner and O.P. Jacquotte - A Variational Method For the Optimization of Directionally Stretched Elements Generated by the Advancing Front Method (AFM) - *The Third Int. Conf. on Numerical Grid Generation in C.F.D and Related Fields*, Barcelona, June 3-6 1991.

[17] J. Cabello, R. Löhner and O.P. Jacquotte - A Variational Method For the Optimization of Two- and Tree-Dimensional Unstructured Meshes *Abstract in The first U.S. National Congress on Computational Mechanics* , Chicago, Illinois, July 21-24, 1991.

APPENDIX 1

Construction of the functional

Theory

We review the mechanical and mathematical properties of the functional used to measure the deformation. A more detailed explanation of the foundations of the method can be found elsewhere [10,14].

1.1. - Mechanical Assumptions

Let us consider a transformation $\mathbf{x}(\xi)$ between a reference element and a physical element. First, we restrict the dependence of the functional up to the first order derivative of the transformation :

$$\sigma = \sigma(\mathbf{x}, \mathbf{F}) \quad \text{with} \quad \mathbf{F} = \nabla \mathbf{x} \tag{1}$$

As the functional should not change if we apply a translation to the physical element, we further restrict σ to rely on the transformation $\mathbf{x}$ solely through its gradient $\mathbf{F}$.

$$\sigma(\mathbf{x}, \mathbf{F}) = \sigma(\mathbf{F}) \tag{2}$$

Since any rotation of the physical element must not change the measure we require the following identity to hold for σ :

$$\sigma(\mathbf{F}) = \sigma(\mathbf{Q} \cdot \mathbf{F}) \qquad \text{for any orthogonal matrix } \mathbf{Q} \tag{3}$$

Furthermore, since the measure must also be invariant for any reference rotation we impose :

$$\sigma(\mathbf{F}) = \sigma(\mathbf{F} \cdot \mathbf{Q}) \qquad \text{for any orthogonal matrix } \mathbf{Q} \tag{4}$$

To sum up, we require the functional's invariance to rigid motions of both reference and physical element. It means that a functional following the four axioms and properties aforementioned is independent of the orthonormal basis in which the gradient $\mathbf{F}$ is computed. This property can be written as,

$$\sigma = \sigma(\mathbf{Q} \cdot \mathbf{F} \cdot \mathbf{Q}^t) \qquad \text{for any orthogonal matrix} \mathbf{Q} \tag{5}$$

The axioms and properties (1) to (4) are well known in three-dimensional elasticity theory [10]. They characterize the energy of an hyperelastic (1), isotropic (4), homogeneous (2) material satisfying the axiom of frame indifference (3).

From these properties, it can be shown that the functional σ depends only on the invariants of a matrix $\mathbf{C}$, called the right Cauchy-Green tensor of the transformation $\mathbf{x}$, and defined as

$$\boxed{\mathbf{C} = \mathbf{F}^t \cdot \mathbf{F} = \nabla \mathbf{x}^t \cdot \nabla \mathbf{x}} \tag{6}$$

At any point $\underline{\xi} = (\xi, \eta, \zeta)$ in the reference space, the invariants, noted I_1, I_2 and I_3 are given by the formulae :

$$\begin{aligned} I_1 &= \text{tr}\,(\underline{\mathbf{C}}) \\ &= ||\mathbf{F}||_m^2 \\ &= ||\mathbf{x}_\xi||_v^2 + ||\mathbf{x}_\eta||_v^2 + ||\mathbf{x}_\zeta||_v^2 \end{aligned} \tag{7}$$

$$\begin{aligned} I_2 &= \text{tr}(\text{Cof}(\underline{\mathbf{C}})) \\ &= ||\text{Cof}\,\mathbf{F}||_m^2 \\ &= ||\mathbf{x}_\xi \wedge \mathbf{x}_\eta||_v^2 + ||\mathbf{x}_\xi \wedge \mathbf{x}_\zeta||_v^2 + ||\mathbf{x}_\eta \wedge \mathbf{x}_\zeta||_v^2 \end{aligned} \tag{8}$$

$$\begin{aligned} I_3 &= \det\,(\underline{\mathbf{C}}) \\ &= (\det \mathbf{F})^2 \\ &= J^2 \end{aligned} \tag{9}$$

with,

$$\begin{aligned} J &= \det \mathbf{F} \\ &= \det(\mathbf{x}_\xi, \mathbf{x}_\eta, \mathbf{x}_\zeta) \quad \text{(jacobian)} \end{aligned} \tag{10}$$

and the symbols $||.||_m$, $||.||_v$, tr (.), det (.) and Cof(.) denoting, respectively, the matrix norm, the vector norm, the linear operator trace and the two non linear operators determinant and cofactor.

$$\sigma = \sigma(I_1, I_2, I_3) \tag{11}$$

In order to control the element orientation and prevent negative jacobians, the invariant I_3 (insensitive to the orientation) is replaced by the jacobian. From now on, we are seeking for a functional in the general form,

$$\boxed{\sigma = \sigma(I_1, I_2,\ J\ )} \tag{12}$$

Vowing to establish a well-posed minimization problem, we force the functional to be locally convex in the neighborhood of a rigid transformation.

1.2. - Mathematical Assumptions

Rigid transformations - i.e. translations, rotations - verify the following equivalent properties :

$$\mathbf{x}(\xi) \text{is a rigid transformation} \tag{13}$$

$$\mathbf{F} = \nabla \mathbf{x} \text{ is a direct orthogonal matrix} \tag{14}$$

$$\mathbf{F} = \text{Cof}\,\mathbf{F} \quad \text{and} \quad \mathrm{J} = +1 \tag{15}$$

$$(\mathrm{I}_1, \mathrm{I}_2, \mathrm{J}) = (3, 3, 1) \tag{16}$$

In the sequel, rigid transformations will be characterized by the subscript index 0.

Owing to the shape preserving property of a rigid transformation we impose a normalization condition :

$$\sigma|_0 = 0 \tag{17}$$

Then, an equilibrium condition is settled for rigid transformations,

$$\mathrm{D}\sigma|_0 = 0 \quad \text{(the null tensor)} \tag{18}$$

Finally, in an attempt to obtain a unique minimum, the functional is assumed to be convex in the neighborhood of a rigid transformation :

$$\mathrm{D}^2\sigma|_0 > 0 \quad \text{(positive-definite tensor)} \tag{19}$$

In these expressions D and D^2 denote respectively the first and second-order differential of the functional.

1.3. - Functional Chosen

We are now in position to move on to the practical choice of the functional. Several candidates verifying both mechanical and mathematical requirements can be found in the literature [11]. In order to devise a functional more amenable to numerical computations we choose a polynomial form, discussed in details in [5], that can be expressed in three dimensions as :

$$\boxed{\sigma_{3d} = C_1(\mathrm{I}_1 - \mathrm{I}_3 - 2) + C_2(\mathrm{I}_2 - 2\mathrm{I}_3 - 1) + C_3(\mathrm{J} - 1)^2} \tag{20}$$

with the convexity condition (19) leading to the inequality

$$3\,C_3 > 4(\,C_1 + C_2) > 0 \tag{21}$$

In two dimensions, the expression reduces to

$$\boxed{\sigma_{2d} = C\,(I_1 - I_3 - 1) + K\,(J - 1)^2} \tag{22}$$

with, the inequality

$$K > C > 0 \tag{23}$$

APPENDIX 2

Two-Dimensional Optimization

Numerical aspects

The numerical approximation of the transformation and its derivatives are presented.

introduction

We want to compute the deformation between a current element in the physical space (x, y) and a reference element in a reference space ($\xi,\ \eta$). The current element is a triangle given by its vertices coordinates $\left((\mathbf{x}_i);\ i = 1,3\right)$ while the reference element is a triangle given by its vertices coordinates $\left((\underline{\xi}_i);\ i = 1,3\right)$. A linear transformation $\mathbf{x}$ is assumed between the reference triangle and the current triangle.

2.1. - Linear transformation $\mathbf{x}$:

For the vector transformation $\mathbf{x}$ we assume that each scalar transformation, i.e. x and y, is a linear function of $\underline{\xi} = (\ \xi,\ \eta)$ i.e. :

$$\begin{aligned} &\mathbf{x}(\underline{\xi}) = (x(\ \xi,\ \eta),\ y(\ \xi,\ \eta))^t \\ &\mathbf{x} = \mathbf{a}\xi + \mathbf{b}\eta + \mathbf{c} \qquad \mathbf{a} = (a_x,\ a_y)^t,\ \mathbf{b} = (b_x,\ b_y)^t,\ \mathbf{c} = (c_x,\ c_y)^t \end{aligned} \tag{1}$$

Following the line of the classical finite element method the above vector transformation can also be defined by :

$$\mathbf{x} = \sum_{i=1}^{i=3} \mathbf{x}_i\ N_i \tag{2}$$

Knowing the vertices coordinates in the physical space the vector transformation is completely defined as soon as the element shape functions $N_i,\ i = 1,3$ are known.

2.2. - Element Shape Functions :

From expression (2), the linear shape functions are defined such that N_i has the value unity at node i and is zero for the remaining nodes :

$$N_i(\xi_j,\ \eta_j) = \delta_{ij} \qquad \forall i,j;\quad 1 \leq i,j \leq 3 \tag{3}$$

$$N_i(\ \xi,\ \eta) = a_i\xi + b_i\eta + c_i \qquad \forall i;\quad 1 \leq i \leq 3 \tag{4}$$

with δ_{ij} referring to the kronecker symbol and a_i, b_i and c_i being scalar values.

2.3. - Computation of the Shape Functions :

From equations (3) and (4) we deduce that the unknowns (a_i, b_i, c_i) are solutions of the system :

$$\begin{bmatrix} \xi_1 & \eta_1 & 1 \\ \xi_2 & \eta_2 & 1 \\ \xi_3 & \eta_3 & 1 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix} = \begin{bmatrix} \delta_{i1} \\ \delta_{i2} \\ \delta_{i3} \end{bmatrix} \tag{5}$$

That we can rewrite,

$$\mathbf{A} \cdot \mathbf{u} = \mathbf{r} \tag{6}$$

with,

$$\mathbf{A} = \begin{bmatrix} \xi_1 & \eta_1 & 1 \\ \xi_2 & \eta_2 & 1 \\ \xi_3 & \eta_3 & 1 \end{bmatrix}; \qquad \mathbf{u} = \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix}; \quad \mathbf{r} = \begin{bmatrix} \delta_{i1} \\ \delta_{i2} \\ \delta_{i3} \end{bmatrix}$$

The system of equations defined by (6) has a unique solution if the determinant of the matrix **A** does not vanish. Introducing the circular permutation (i,j,k) between indices 1,2,3 we obtain :

$$\begin{aligned} a_i &= (\eta_j - \eta_k) \,/\, \text{deter} \\ b_i &= (\xi_k - \xi_j) \,/\, \text{deter} \\ c_i &= (\xi_j \eta_k - \xi_k \eta_j) \,/\, \text{deter} \end{aligned} \tag{7}$$

with

$$\begin{aligned} \text{deter} &= \det \mathbf{A} \\ &= 2(\text{ area of the reference element }) \end{aligned} \tag{8}$$

Remark 2 :

- The area of the reference triangle is defined by

$$\text{area} = \frac{1}{2} \left| (\xi_2 - \xi_1) \wedge (\xi_3 - \xi_1) \right|$$

where the symbol $\wedge$ represents the vector product.

From expression (4) and (7) the element shape functions are completely defined. Afterwards, the vector transformation is defined by expression (2).

Remark 3 :

- One convenient and practical way to verify that the basis functions are correctly defined consists in noticing that the sum of the three basis functions is a linear function taking the value unity on three points. Consequently, the sum $N_1 + N_2 + N_3$ is the constant function unity and we have :

$$N_1 + N_2 + N_3 = 1 \tag{9}$$

We will need to compute the derivatives of the vector function $\mathbf{x}$. According to expression (2) the derivatives are :

$$\begin{aligned} \mathbf{x}_\xi &= \mathbf{x}_1 \frac{\partial N_1}{\partial \xi} + \mathbf{x}_2 \frac{\partial N_2}{\partial \xi} + \mathbf{x}_3 \frac{\partial N_3}{\partial \xi} \\ \mathbf{x}_\eta &= \mathbf{x}_1 \frac{\partial N_1}{\partial \eta} + \mathbf{x}_2 \frac{\partial N_2}{\partial \eta} + \mathbf{x}_3 \frac{\partial N_3}{\partial \eta} \end{aligned} \tag{10}$$

2.4. - Computation of the derivatives

The elements shape functions being linear, their derivatives are constants. These constants are given by derivation of formula (4) leading to :

$$\frac{\partial N_i}{\partial \xi} = a_i \quad ; \qquad \frac{\partial N_i}{\eta} = b_i \tag{11}$$

Remark 4 :

- Using expression (9) we deduce that the sum of all the derivatives vanishes (i.e.) :

$$\frac{\partial N_1}{\partial \xi} + \frac{\partial N_2}{\partial \xi} + \frac{\partial N_3}{\partial \xi} = \frac{\partial N_1}{\partial \eta} + \frac{\partial N_2}{\partial \eta} + \frac{\partial N_3}{\partial \eta} = 0 \tag{12}$$

  The equalities (12) offers a good way to verify numerically the computation of the element shape function derivatives .

At this point, we defined all the elements required to compute the transformation and its derivatives. The forthcoming section is devoted to the computation of the invariants for the right Cauchy-Green tensor and their derivatives which will be used for the minimization of the functional. The computation of the functional and its derivatives will be used during the minimization performed by the conjugate gradient.

2.5. - Computation of the invariants

Let us remind that we have chosen to approximate the vector transformation by a linear function :

$$\mathbf{x} = \mathbf{x}_1 N_1 + \mathbf{x}_2 N_2 + \mathbf{x}_3 N_3$$

with

$$\underline{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

the derivatives are constants given by the expressions :

$$\begin{cases} \mathbf{x}_\xi(\ \underline{x}\ ) = \mathbf{x}_1 a_1 + \mathbf{x}_2 a_2 + \mathbf{x}_3 a_3 \\ \mathbf{x}_\eta(\ \underline{x}\ ) = \mathbf{x}_1 b_1 + \mathbf{x}_2 b_2 + \mathbf{x}_3 b_3 \end{cases} \tag{13}$$

Remark 5 :

- From expression (13), the derivatives are linear function of the vertices coordinates $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$

The vector function gradient, $\mathbf{F}$, is a (2×2) matrix defined by

$$\begin{aligned} \mathbf{F} &= \nabla \mathbf{x} \\ &= [\mathbf{x}_\xi\ ,\ \mathbf{x}_\eta] \end{aligned} \tag{14}$$

The Cauchy-Green matrix is :

$$\mathbf{C} = \mathbf{F}^t\ \mathbf{F} \tag{15}$$

Finally, the two invariants of the Cauchy-Green tensor are :

$$\begin{aligned} \mathrm{I}_1 &= \mathrm{tr}\ [\mathbf{C}] \\ \mathrm{I}_3 &= \det\ (\mathbf{C}) = \mathrm{J}^2 \end{aligned} \tag{16}$$

with

$$\mathrm{J} = \det\ (\mathbf{F})\ , \tag{17}$$

the jacobian of the transformation

2.5 - Computation of the invariants

2.5.1. - Computation of the jacobian

The jacobian is computed using the expression :

$$\mathrm{J} = \det\ (\mathbf{x}_\xi,\ \mathbf{x}_\eta) \tag{18}$$

Remark 6 :

1 - The jacobian is independent of ξ and η. It only depends on the vertices coordinates of the current element (physical space).

2 - The jacobian is an homogeneous polynomial of degree 2 i.e.

$$J = \sum_{\substack{1 \le i \le 3 \\ 1 \le j \le 3}} a_{i,j}\, x_i\, y_j \qquad \text{with} \quad a_{i,j} \in \Re \tag{19}$$

where x_i, y_i represents the cartesian components of vertex $\mathbf{x}_i$.

3 - The jacobian represents the ratio of the current element surface over the reference element surface.

2.5.2. - Computation of the first invariant I_1

By definition, the first invariant is :

$$I_1 = \text{tr } [C] \tag{23}$$

thus, we deduce from expressions (14) and (15),

$$I_1 = ||\mathbf{x}_\xi||^2 + ||\mathbf{x}_\eta||^2 \tag{24}$$

which allows us to compute I_1 through expression (13) of the vector function derivatives.

Remark 7 :

1 - The first invariant is independent of ξ and η, depending only on the vertices coordinates.

2 - I_1 is a second order homogeneous polynomial of the coordinates (x_1, y_1, x_2, y_2, x_3, y_3).

2.6. - Differentiation of the invariants

So far, we have seen how the two invariants are computed. From expressions (18) and (21), the computation of the invariants J and I_1 is related to the computation of the vector functions $\mathbf{x}_\xi$ and $\mathbf{x}_\eta$. Now, we are going to give the formulas for the differentiation of the invariants and their derivatives.

2.6.1. - Notations :

In the following we note

$$\mathbf{x} = (x_1,\ y_1,\ x_2,\ y_2,\ x_3,\ y_3) = (\mathbf{x}_1{}^t,\ \mathbf{x}_2{}^t,\ \mathbf{x}_3{}^t) \tag{22}$$

and

$$\underline{h} = (h_1^x,\ h_1^y,\ h_2^x,\ h_2^y,\ h_3^x,\ h_3^y) = (\ \underline{h}_1{}^t,\ \underline{h}_2{}^t,\ \underline{h}_3{}^t) \qquad (23)$$

with,

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{and} \quad \underline{h}_i = \begin{bmatrix} h_i^x \\ h_i^y \end{bmatrix}$$

representing respectively the vertices of the triangle and a direction vector at the vertices (This direction vector may be the descent direction at the vertex).

Remark 8 :

- The $\Re^6$-vector $\mathbf{x}$ is the variable of our problem ($\mathbf{x}$ represents the three vertices coordinates). According to expression (2), the vector transformation $\mathbf{x}$ is completely defined when the three vertices coordinates $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ are defined. Thus, there is a one-to-one correspondence between the vector transformation $\mathbf{x}$ and the $\Re^6$ vector $\mathbf{x}$.

2.6.2. - Differentiation of the transformation derivatives

From expression (13) we get for the vector function derivatives :

$$\mathbf{x}_\xi = \begin{bmatrix} x_\xi(x_1,x_2,x_3) \\ y_\xi(y_1,y_2,y_3) \end{bmatrix} \qquad \mathbf{x}_\eta = \begin{bmatrix} x_\eta(x_1,x_2,x_3) \\ y_\eta(y_1,y_2,y_3) \end{bmatrix}$$

The above expressions show the derivatives dependence on the variable $\mathbf{x}$. Using the standard differentiation rules, the vector function differential is :

$$\mathrm{d}\ \mathbf{x}_\xi(\mathbf{x}) \cdot \underline{h} = \begin{bmatrix} \mathrm{d}\ x_\xi(x_1,x_2,x_3) \cdot (h_1^x, h_2^x, h_3^x) \\ \mathrm{d}\ y_\xi(y_1,y_2,y_3) \cdot (h_1^y, h_2^y, h_3^y) \end{bmatrix} \qquad (24)$$

and,

$$\mathrm{d}\ \mathbf{x}_\eta(\mathbf{x}) \cdot \underline{h} = \begin{bmatrix} \mathrm{d}\ x_\eta(x_1,x_2,x_3) \cdot (h_1^x, h_2^x, h_3^x) \\ \mathrm{d}\ y_\eta(y_1,y_2,y_3) \cdot (h_1^y, h_2^y, h_3^y) \end{bmatrix} \qquad (25)$$

with,

$$\mathrm{d}\ x_\xi(x_1,x_2,x_3) \cdot (h_1^x, h_2^x, h_3^x) = \frac{\partial x_\xi}{\partial x_1} h_1^x + \frac{\partial x_\xi}{\partial x_2} h_2^x + \frac{\partial x_\xi}{\partial x_3} h_3^x \qquad (26)$$

and similar formulas holding when interchanging x by y and the subscript ξ by η.

2.6.2. - Differentiation of the jacobian

From definition (18) of the jacobian and using the differentiation rules we obtain :

$$\mathrm{d}\ J(\mathbf{x}) \cdot (\ \underline{h}\ ) = \det(\ \mathrm{d}\ \mathbf{x}_\xi(\mathbf{x})(\ \underline{h}\ ), \mathbf{x}_\eta) + \det(\mathbf{x}_\xi,\ \mathrm{d}\ \mathbf{x}_\eta(\mathbf{x})(\ \underline{h}\ )) \qquad (27)$$

Remark 9

1 - As the vector functions $\mathbf{x}_\xi$ and $\mathbf{x}_\eta$ are linear functions of the variable $\mathbf{x}$ we have the following relation :

$$\begin{aligned} d\,\mathbf{x}_\xi(\mathbf{x}) \cdot (\,\underline{h}\,) &= \mathbf{x}_\xi(\,\underline{h}\,) \\ &= \underline{h}_\xi \end{aligned} \tag{28}$$

where $\underline{h}_\xi$ is the vector function defined by equation (13) when replacing $\mathbf{x}$ by $\underline{h}$. Then, we can rewrite (27) in the form :

$$d\ J(\mathbf{x}) \cdot (\,\underline{h}\,) = \det(\,\underline{h}_\xi, \mathbf{x}_\eta) + \det(\mathbf{x}_\xi,\ \underline{h}_\eta) \tag{29}$$

2 - Using the differentiation rule we get the higher order differential of the jacobian.

2.6.4. - Differentiation of I_1

Using the same notations and applying the differentiation rules to formula (21) we deduce :

$$d\,I_1(\mathbf{x})(\,\underline{h}\,) = 2(\,\underline{h}_\xi \cdot \mathbf{x}_\xi + \underline{h}_\eta \cdot \mathbf{x}_\eta) \tag{30}$$

2.2.3. - Computation of the functional

$$\sigma = C_1(I_1 - I_3 - 1) + C_2(\,J\, - 1)^2 \tag{31}$$

- I_1 is a polynomial of degree 2.
- J is a polynomial of degree 2.
- I_3 is a polynomial of degree 4.

From all the previous sections we are able to compute numerically σ and its derivatives.

APPENDIX 3

Three-Dimensional Optimization

Numerical aspects

The numerical approximation of the transformation and its derivatives are presented in three dimensions.

introduction

By the same token, following the same line of thinking, the numerical approximation presented in two dimensions can be extended to three dimensions. We want to compute the deformation between a current element in the physical space (x, y, z) and a reference element in a reference space (ξ, η, ζ). The current element is a tetrahedron given by its four vertices coordinates $((\underline{x}_i);\ i = 1,4)$ while the reference element is a tetrahedron given by its vertices coordinates $((\xi_i);\ i = 1,3)$. A linear transformation **x** is assumed between the reference triangle and the current triangle.

3.1. - Element Shape Functions :

The basis functions N_i are linear functions defined by the relations :

$$N_i(\xi_j, \eta_j, \zeta_j) = \delta_{ij} \qquad \forall i, j; \quad 1 \leq i, j \leq 4 \tag{1}$$

$$N_i(\xi, \eta, \zeta) = a_i\xi + b_i\eta + c_i\zeta + d_i \tag{2}$$

with $\xi_j = (\xi_j, \eta_j, \zeta_j)$ the coordinates of the vertices of the tetrahedron in the reference space (ξ, η, ζ).

3.2. - computation of $N_i(\xi, \eta, \zeta)$

The unknowns a_i, b_i, c_i and d_i are solutions of the linear system :

$$\{\Sigma\} \begin{cases} a_i\xi_1 + b_i\eta_1 + c_i\zeta_1 + d_i &= \delta_{i1} \\ a_i\xi_2 + b_i\eta_2 + c_i\zeta_2 + d_i &= \delta_{i2} \\ a_i\xi_3 + b_i\eta_3 + c_i\zeta_3 + d_i &= \delta_{i3} \\ a_i\xi_4 + b_i\eta_4 + c_i\zeta_4 + d_i &= \delta_{i4} \end{cases}$$

The determinant of this system is given by :

$$\text{deter} = \begin{vmatrix} \xi_1 & \eta_1 & \zeta_1 & \delta_{i1} \\ \xi_2 & \eta_2 & \zeta_2 & \delta_{i2} \\ \xi_3 & \eta_3 & \zeta_3 & \delta_{i3} \\ \xi_4 & \eta_4 & \zeta_4 & \delta_{i4} \end{vmatrix}$$

If deter $\neq 0$ then the system Σ has a unique solution.

Remark 1 :

- The summation of the basis functions is a linear polynomial of degree 1, equal to 1 at four points. Therefore, this polynomial is the constant polynomial equal to 1 and we have :

$$N_1 + N_2 + N_3 + N_4 = 1$$

3.3. - Computation of the derivatives

All derivatives are constant and given by the expressions :

$$\frac{\partial N_1}{\partial \xi} = a_1 \quad ; \frac{\partial N_1}{\partial \eta} = b_1 \quad ; \frac{\partial N_1}{\partial \zeta} = c_1$$

$$\frac{\partial N_2}{\partial \xi} = a_2 \quad ; \frac{\partial N_2}{\partial \eta} = b_2 \quad ; \frac{\partial N_2}{\partial \zeta} = c_2$$

$$\frac{\partial N_3}{\partial \xi} = a_3 \quad ; \frac{\partial N_3}{\partial \eta} = b_3 \quad ; \frac{\partial N_3}{\partial \zeta} = c_3$$

$$\frac{\partial N_4}{\partial \xi} = a_4 \quad ; \frac{\partial N_4}{\partial \eta} = b_4 \quad ; \frac{\partial N_4}{\partial \zeta} = c_4$$

Remark

- The derivatives of the sum of the basis functions is null (i.e.)

$$\sum_{j=1}^{j=4} \frac{\partial N_j}{\partial \xi^i} = 0 \qquad \text{for } i = 1,2,3 \qquad \text{with } (\xi^1, \xi^2, \xi^3) = (\xi, \eta, \zeta)$$

3.4. - Computation of the invariants

The transformatiom **x** may be rewritten as :

$$\mathbf{x} = \sum_{i=1}^{i=4} \underline{x}_i N_i \tag{3}$$

with,

$$\underline{x}_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \qquad ; i = 1,2,3,4$$

The derivatives are constants given by the expression :

$$\mathbf{x}_{\xi^j} = \sum_{i=1}^{i=4} \underline{\mathbf{x}}_{,N_{i,\xi^j}} \qquad \text{for } i = 1,2,3 \qquad \text{with } (\xi^1,\xi^2,\xi^3) = (\xi,\eta,\zeta) \tag{4}$$

and,

$$N_{i,\xi^j} = \frac{\partial N_i}{\partial \xi^j}$$

representing the derivative of the shape function N_i with respect to the variable ξ^j. We obtain,

$$\begin{aligned} \mathbf{x}_\xi(\underline{\mathbf{x}}) &= a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + a_3\mathbf{x}_3 + a_4\mathbf{x}_4 \\ \mathbf{x}_\eta(\underline{\mathbf{x}}) &= b_1\mathbf{x}_1 + b_2\mathbf{x}_2 + b_3\mathbf{x}_3 + b_4\mathbf{x}_4 \\ \mathbf{x}_\zeta(\underline{\mathbf{x}}) &= c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + c_3\mathbf{x}_3 + c_4\mathbf{x}_4 \end{aligned} \tag{5}$$

The vector function gradient, $\mathbf{F}$, is a (3×3) matrix defined by

$$\begin{aligned} \mathbf{F} &= \nabla\mathbf{x} \\ &= [\mathbf{x}_\xi\ ,\ \mathbf{x}_\eta,\ \mathbf{x}_\zeta] \end{aligned} \tag{6}$$

The Cauchy-Green matrix is :

$$\mathbf{C} = \mathbf{F}^t\ \mathbf{F} \tag{7}$$

Finally, the three invariants of the Cauchy-Green tensor are :

$$\begin{aligned} \mathbf{I}_1 &= \text{tr}\ (\underline{\mathbf{C}}) \\ &= ||\mathbf{F}||^2_{\mathbf{m}} \\ &= ||\mathbf{x}_\xi||^2_{\mathbf{v}} + ||\mathbf{x}_\eta||^2_{\mathbf{v}} + ||\mathbf{x}_\zeta||^2_{\mathbf{v}} \end{aligned} \tag{8}$$

$$\begin{aligned} \mathbf{I}_2 &= \text{tr}(\text{Cof}(\underline{\mathbf{C}})) \\ &= ||\text{Cof}\ \mathbf{F}||^2_{\mathbf{m}} \\ &= ||\mathbf{x}_\xi\ \wedge\ \mathbf{x}_\eta||^2_{\mathbf{v}} + ||\mathbf{x}_\xi\ \wedge\ \mathbf{x}_\zeta||^2_{\mathbf{v}} + ||\mathbf{x}_\eta\ \wedge\ \mathbf{x}_\zeta||^2_{\mathbf{v}} \end{aligned} \tag{9}$$

$$\begin{aligned} \mathbf{I}_3 &= \det\ (\underline{\mathbf{C}}) \\ &= (\det\mathbf{F})^2 \\ &= \mathbf{J}^{\,2} \end{aligned} \tag{10}$$

with,

$$\begin{aligned} J &= \det \mathbf{F} \\ &= \det(\mathbf{x}_\xi, \mathbf{x}_\eta, \mathbf{x}_\zeta) \quad \text{(jacobian)} \end{aligned} \tag{11}$$

and the symbols $||.||_m$, $||.||_v$, tr (.), det (.) and Cof(.) denoting, respectively, the matrix norm, the vector norm, the linear operator trace and the two non linear operators determinant and cofactor.

3.5. - Differentiation of the invariants

The computations of the invariants are carried out using formulas (3) of the basis functions derivatives. The next step is to find the formulas for the differential of these invariant which will be used to compute the differential of the functional.

3.5.1 - Notations :

In the following we note,

$$\underline{x} = (x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4, z_1, z_2, z_3, z_4)$$

and

$$\underline{h} = (h_1^x, h_2^x, h_3^x, h_4^x, h_1^y, h_2^y, h_3^y, h_4^y, h_1^z, h_2^z, h_3^z, h_4^z)$$

with,

$$\underline{x}_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \qquad \underline{h}_i = \begin{bmatrix} h_i^x \\ h_i^y \\ h_i^z \end{bmatrix}$$

representing respectively the vertices of the tetrahedron and a direction vector at these vertices.

3.5.2 - Differentiation of the transformation derivatives :

The derivative of the transformation $\mathbf{x}$ is :

$$\mathbf{x}_\xi = \begin{bmatrix} x_\xi(x_1, x_2, x_3, x_4) \\ y_\xi(y_1, y_2, y_3, y_4) \\ z_\xi(z_1, z_2, z_3, z_4) \end{bmatrix}.$$

For the differential of the vector function $\mathbf{x}_\xi$ at point $\mathbf{x}$ applied to vector $\underline{h}$ we have :

$$
\mathrm{d}\,\mathbf{x}_\xi(\mathbf{x})\cdot\,\underline{\mathbf{h}} \;=\; \left[\begin{array}{lcl}
\mathrm{d}\,x_\xi(\mathbf{x})\cdot\,\underline{\mathbf{h}} & = & \dfrac{\partial x_\xi}{\partial x_1}h_1^x+\dfrac{\partial x_\xi}{\partial x_2}h_2^x+\dfrac{\partial x_\xi}{\partial x_3}h_3^x+\dfrac{\partial x_\xi}{\partial x_4}h_4^x \\
\mathrm{d}\,y_\xi(\mathbf{x})\cdot\,\underline{\mathbf{h}} & = & \dfrac{\partial y_\xi}{\partial y_1}h_1^y+\dfrac{\partial y_\xi}{\partial y_2}h_2^y+\dfrac{\partial y_\xi}{\partial y_3}h_3^y+\dfrac{\partial y_\xi}{\partial y_4}h_4^y \\
\mathrm{d}\,z_\xi(\mathbf{x})\cdot\,\underline{\mathbf{h}} & = & \dfrac{\partial z_\xi}{\partial z_1}h_1^z+\dfrac{\partial z_\xi}{\partial z_2}h_2^z+\dfrac{\partial z_\xi}{\partial z_3}h_3^z+\dfrac{\partial z_\xi}{\partial z_4}h_4^z
\end{array}\right.
$$

and the same relations holding for η and ζ.

3.5.3 - Differentiation of the jacobian

The jacobian being related to the derivatives by the expression :

$$
\mathrm{J}\,(\mathbf{x}) = (\mathbf{x}_\xi, \mathbf{x}_\eta, \mathbf{x}_\zeta)
$$

with $\mathbf{x}_\xi, \mathbf{x}_\eta, \mathbf{x}_\zeta$, linear functions of the variable $\mathbf{x}$, the differentiation gives :

$$
\begin{aligned}
\mathrm{d}\;\mathrm{J}\,(\,\underline{\mathbf{x}}\,)\cdot(\,\underline{\mathbf{h}}\,) = {} & \mathrm{d}\,\mathbf{x}_\xi(\,\underline{\mathbf{x}}\,)(\,\underline{\mathbf{h}}\,)\cdot(\mathbf{x}_\eta \wedge \mathbf{x}_\zeta)+ \\
& \mathrm{d}\,\mathbf{x}_\eta(\,\underline{\mathbf{x}}\,)(\,\underline{\mathbf{h}}\,)\cdot(\mathbf{x}_\zeta \wedge \mathbf{x}_\xi)+ \\
& \mathrm{d}\,\mathbf{x}_\zeta(\,\underline{\mathbf{x}}\,)(\,\underline{\mathbf{h}}\,)\cdot(\mathbf{x}_\xi \wedge \mathbf{x}_\eta) \qquad (12)
\end{aligned}
$$

The scalar obtained above is the summation of the scalar product between the differential of the derivative function and the vector product of the two remaining derivatives. The differentiation of the derivatives simplifies to :

$$
\begin{aligned}
\mathrm{d}\,\mathbf{x}_\xi(\,\underline{\mathbf{x}}\,)\cdot(\,\underline{\mathbf{h}}\,) &= \mathbf{x}_\xi(\,\underline{\mathbf{h}}\,) \\
&= \underline{\mathbf{h}}_{\,\xi} \qquad (13)
\end{aligned}
$$

where $\underline{\mathbf{h}}_{\,\xi}$ is the vector defined by relation (3) while replacing variable $\underline{\mathbf{x}}$ by $\underline{\mathbf{h}}$. The differential of the jacobian rewrites :

$$
\mathrm{d}\;\mathrm{J}\,(\,\underline{\mathbf{x}}\,)\cdot(\,\underline{\mathbf{h}}\,) = \underline{\mathbf{h}}_{\,\xi}\cdot(\mathbf{x}_\eta \wedge \mathbf{x}_\zeta) + \underline{\mathbf{h}}_{\,\eta}\cdot(\mathbf{x}_\zeta \wedge \mathbf{x}_\xi) + \underline{\mathbf{h}}_{\,\zeta}\cdot(\mathbf{x}_\xi \wedge \mathbf{x}_\eta) \qquad (14)
$$

We can also deduce the higher orders differentiation of the functional J and with the same notations and considerations introduced previously we obtain :

$$
\begin{aligned}
\mathrm{d}^2\mathrm{J}(\,\underline{\mathbf{x}}\,)\cdot(\,\underline{\mathbf{h}}\,,\,\underline{\mathbf{h}}\,) = 2[\; & (\mathbf{x}_\xi,\ \underline{\mathbf{h}}_{\,\eta},\ \underline{\mathbf{h}}_{\,\zeta})+ \\
& (\,\underline{\mathbf{h}}_{\,\xi}, \mathbf{x}_\eta,\ \underline{\mathbf{h}}_{\,\zeta})+ \\
& (\,\underline{\mathbf{h}}_{\,\xi},\ \underline{\mathbf{h}}_{\,\eta}, \mathbf{x}_\zeta)\;\;] \qquad (15)
\end{aligned}
$$

$$d^3 J(\underline{x})\cdot(\underline{h},\underline{h},\underline{h}) = 6(\underline{h}_\xi, \underline{h}_\eta, \underline{h}_\zeta) \tag{16}$$

3.5.4. - Differentiation of I_1

$$I_1 = x_\xi^2 + x_\eta^2 + x_\zeta^2 \tag{17}$$

$$d\,I_1(\underline{x})(\underline{h}) = 2\ (\underline{h}_\xi . x_\xi + \underline{h}_\eta . x_\eta + \underline{h}_\zeta . x_\zeta) \tag{18}$$

$$d^2 I_1(\underline{x})(\underline{h},\underline{h}) = 2\ (\underline{h}_\xi \cdot \underline{h}_\xi + \underline{h}_\eta \cdot \underline{h}_\eta + \underline{h}_\zeta \cdot \underline{h}_\zeta) \tag{19}$$

3.5.5. - Differentiation of I_2

$$I_2 = (x_\xi \wedge x_\eta)^2 + (x_\eta \wedge x_\zeta)^2 + (x_\zeta \wedge x_\xi)^2 \tag{20}$$

$$\begin{aligned} d\,I_2 = 2\ (\ & (\underline{x}_\xi \wedge \underline{x}_\eta)\cdot\{\underline{h}_\xi \wedge \underline{x}_\eta + \underline{x}_\xi \wedge \underline{h}_\eta\} \\ & + (\underline{x}_\eta \wedge \underline{x}_\zeta)\cdot\{\underline{h}_\eta \wedge \underline{x}_\zeta + \underline{x}_\eta \wedge \underline{h}_\zeta\} \\ & + (\underline{x}_\zeta \wedge \underline{x}_\xi)\cdot\{\underline{h}_\zeta \wedge \underline{x}_\xi + \underline{x}_\zeta \wedge \underline{h}_\xi\}\ ) \end{aligned} \tag{21}$$

$$\begin{aligned} d^2 I_2 = 2\ (\ & (\underline{h}_\xi \wedge \underline{x}_\eta + \underline{x}_\xi \wedge \underline{h}_\eta)^2 \\ & + (\underline{h}_\eta \wedge \underline{x}_\zeta + \underline{x}_\eta \wedge \underline{h}_\zeta)^2 \\ & + (\underline{h}_\zeta \wedge \underline{x}_\xi + \underline{x}_\zeta \wedge \underline{h}_\xi)^2 \\ +2\ & [\ (\underline{x}_\xi \wedge \underline{x}_\eta)\cdot(\underline{h}_\xi \wedge \underline{h}_\eta) \\ & (\underline{x}_\eta \wedge \underline{x}_\zeta)\cdot(\underline{h}_\eta \wedge \underline{h}_\zeta) \\ & (\underline{x}_\zeta \wedge \underline{x}_\xi)\cdot(\underline{h}_\zeta \wedge \underline{h}_\xi) \quad]\ ) \end{aligned} \tag{22}$$

$$\begin{aligned} d^3 I_2 = 12\ (\ & [(\underline{h}_\xi \wedge \underline{x}_\eta + \underline{x}_\xi \wedge \underline{h}_\eta)\cdot(\underline{h}_\xi \wedge \underline{h}_\eta) \\ & + (\underline{h}_\eta \wedge \underline{x}_\zeta + \underline{x}_\eta \wedge \underline{h}_\zeta)\cdot(\underline{h}_\eta \wedge \underline{h}_\zeta) \\ & + (\underline{h}_\zeta \wedge \underline{x}_\xi + \underline{x}_\zeta \wedge \underline{h}_\xi)\cdot(\underline{h}_\zeta \wedge \underline{h}_\xi)\ ]\ ) \end{aligned} \tag{23}$$

From expression (23) we see that the scalar function $d^3 I_2$ is linear and then for the computation of $d^4 I_2$ (constant i.e. independent of the variable $\mathbf{x}$) we use the previous expression replacing $\mathbf{x}$ by $\underline{h}$.

3.5.6. - Differentiation of the functional

$$\sigma = C_1(I_1 - I_3 - 2) + C_2(I_2 - 2I_3 - 1) + C_3(J - 1)^2 \qquad (24)$$

- I_1 is a polynomial of degree 2 (with respect to the variable $\underline{x}$)
- I_2 is a polynomial of degree 4 (with respect to the variable $\underline{x}$)
- J is a polynomial of degree 3 (with respect to the variable $\underline{x}$)

Therefore the functional σ is a polynomial of degree 6 of the vertices coordinates. Reordering and gathering the terms, σ rewrites in the general form :

$$\sigma = C_1 I_1 + C_2 I_2 + \gamma J^2 - 2C_3 J + \delta$$

with,

$$\gamma = C_3 - C_1 - 2C_2 \quad \text{and} \quad \delta = C_3 - 2C_1 - C_2$$

The differentiation of σ gives :

$$d\sigma(\underline{x}).\underline{h} = C_1\, d I_1(\underline{x})\cdot\underline{h} + C_2\, d I_2(\underline{x})\cdot\underline{h} + (2\gamma J(\underline{x}) - 2C_3)\, dJ(\underline{x})\cdot\underline{h}$$

$$d^2\sigma(\underline{x}).(\underline{h},\underline{h}) = C_1\, d I_1(\underline{h})\cdot\underline{h} + C_2\, d I_2(\underline{x})\cdot(\underline{h},\underline{h}) + 2\gamma(dJ(\underline{x})\cdot\underline{h})^2 + [2\gamma J(\underline{x}) - 2C_3]\, d J(\underline{x})\cdot(\underline{h},\underline{h})$$

$$d^3\sigma(\underline{x})\cdot(\underline{h}^3) = C_2\, d I_2(\underline{x})\cdot(\underline{h},\underline{h},\underline{h}) + 4\gamma(dJ(\underline{x})\cdot\underline{h})\cdot(d J(\underline{x})\cdot(\underline{h},\underline{h})) + [2\gamma J(\underline{x}) - 2C_3]\, d J(\underline{h})\cdot(\underline{h},\underline{h}) + 2\gamma dJ(\underline{x})\cdot\underline{h}\; d J(\underline{x})\cdot(\underline{h},\underline{h})$$

$$d^4\sigma(\underline{x}).(\underline{h}^4) = C_2\, d I_2(\underline{h})\cdot(\underline{h},\underline{h},\underline{h}) + 8\gamma(dJ(\underline{x})\cdot\underline{h})\cdot(d J(\underline{h})\cdot(\underline{h},\underline{h})) + 6\gamma(d J(\underline{x})\cdot(\underline{h},\underline{h}))^2$$

$$d^5\sigma(\underline{x}).(\underline{h}^5) = 20\gamma\, d J(\underline{x})\cdot(\underline{h},\underline{h})\, d J(\underline{h})\cdot(\underline{h},\underline{h})$$

$$d^6\sigma(\underline{x}).(\underline{h}^6) = 20\gamma\, d J(\underline{h})\cdot(\underline{h},\underline{h})^2$$

APPENDIX 4

Conjugate Gradient Algorithm

$V = \{\mathbf{x}; \mathbf{x} \in \Re^{nunk}\}$

nunk	= ndimn*npoin	(unstretched meshes)
	= ndimn*nnodes*nelem	(stretched meshes)

σ is a polynomial of degree 2*ndimn (ndimn = 2 or 3)

To find the point $\mathbf{x}$ minimizing σin the multidimensional space V a Conjugate Gradient Algorithm is used. This algorithm reduces the multidimensional minimization problem to a succession of one-dimensional minimization problems. The different steps are sketched below :

Initialization :

$\mathbf{x}_0 \in V$; coordinates of the initial mesh.

$\underline{h}_0 = -\nabla\sigma(\mathbf{x}_0)$

Assume that $\mathbf{x}_{n-1}$ and $\underline{h}_{n-1}$ are known,

1 - **Computation of** $\mathbf{x}_n$:

$\mathbf{x}_n = \mathbf{x}_{n-1} + \rho_n \underline{h}_{n-1}$

with, $\rho_n = Arg\ min_\rho \sigma(\mathbf{x}_{n-1} + \rho \underline{h}_{n-1})$

2 - **Computation of** γ_n :

$$\gamma_n = \frac{(G_n - G_{n-1}) \cdot G_n}{|G_{n-1}|^2}$$

with, $G_n = -\nabla\sigma(\mathbf{x}_n)$

3 - **Compute the descent direction** $\underline{h}$:

$\underline{h}_n = G_n + \gamma_n \underline{h}_{n-1}$

4 - **convergence test** :

if $|\underline{h}_n| > \epsilon$, $n = n + 1$, goto 1

APPENDIX 5: SAMPLE CALCULATIONS

Computing Systems in Engineering Vol. 1, Nos 2–4, pp. 257–272, 1990
Printed in Great Britain.

0956-0521/90 $3.00 + 0.00

THREE-DIMENSIONAL FLUID–STRUCTURE INTERACTION USING A FINITE ELEMENT SOLVER AND ADAPTIVE REMESHING

R. LÖHNER

Department of Civil, Mechanical and Environmental Engineering, School of Engineering and Applied Science, George Washington University, Washington, D.C. 20052, U.S.A.

(*Received* 1 *May* 1990)

Abstract—The combination of adaptive remeshing techniques, flow solvers for transient problems with moving grids, and consistent rigid body motion integrators in three dimensions is presented. The resulting scheme allows the economical simulation of fully coupled fluid–rigid body interaction problems of arbitrary geometric complexity. Several results, showing three-dimensional store separation, are given to demonstrate the capabilities developed.

1. INTRODUCTION

Practical military applications where we encounter a significant flowfield perturbation due to body motion are as follows: ordnance store separation from aircraft, interstage separation in rockets, shroud removal for interceptors, separation of MIRVs, and torpedo launch. Civilian applications where we encounter a significant perturbation of the flowfields due to body motion are as follows: reciprocating engines, turbines, propellers, ventilators and valves.

1.1. *Store separation as a model problem*

In order to focus the discussion and the relevant ideas, consider the separation of ordnance stores from airplanes flying at supersonic speeds. Figure 1 illustrates some of the relevant physical processes involved in these situations.

—*Shock/shock interactions*: at supersonic speeds, the presence of shocks in the flowfields becomes unavoidable. With several bodies interfering with each other, the shocks emanating from them interact with each other in sometimes extremely complicated ways.[1,2]

—*Shock–boundary layer interaction*: when shocks impact on a surface, the boundary layer is greatly influenced by parameters such as shock-reflection angle, shock-strength, the pressure gradients upstream and downstream of the impact zone, and body curvature. The resulting flowfield may vary abruptly with only minor changes of flight conditions.

—*Turbulent, separated flows*: given the highly complex, and not aerodynamically streamlined geometries of bomb-bays, many of the flowfields contemplated will have vast regions of separated, turbulent flow. This implies that any hope of simulating them accurately with a conventional, algebraic turbulence model has to be forfeited. The lowest order turbulence model that can yield acceptable results for flows of this type is the k, ϵ model.

—*Body motion*: a futher degree of complexity is added for the class of problems considered here due to the relation motion of the bodies present. The bodies move through an already complex, highly nonlinear flowfield, modifying it constantly.

All of these aspects, taken together, make the intuitive prediction of these flows, as well as any extrapolation from past experience, a very unreliable design approach. The nonlinear character of these flows also implies that safe deployment from a mid-cavity position does not guarantee safe deployment from a side-cavity position. The only other two alternative design procedures besides computational fluid dynamics (CFD), wind-tunnel measurements and flight testing, are either extremely expensive or impossible.

Wind-tunnel experiments for store separation in the supersonic regime are difficult because:

—Three non-dimensional numbers need to be reproduced on the scaled model at the same time: Reynolds number, Mach number and Froude number. For supersonic flows in particular, the reproduction of all three non-dimensional numbers in the wind-tunnel is practically impossible.

—The release of ordnance stores requires several seconds, a time-frame that would be too power-consuming—and thus expensive—for most large wind-tunnels.

—The release of ordnance stores into a supersonic free stream tends to accelerate these objects drastically, propelling them to high velocities very

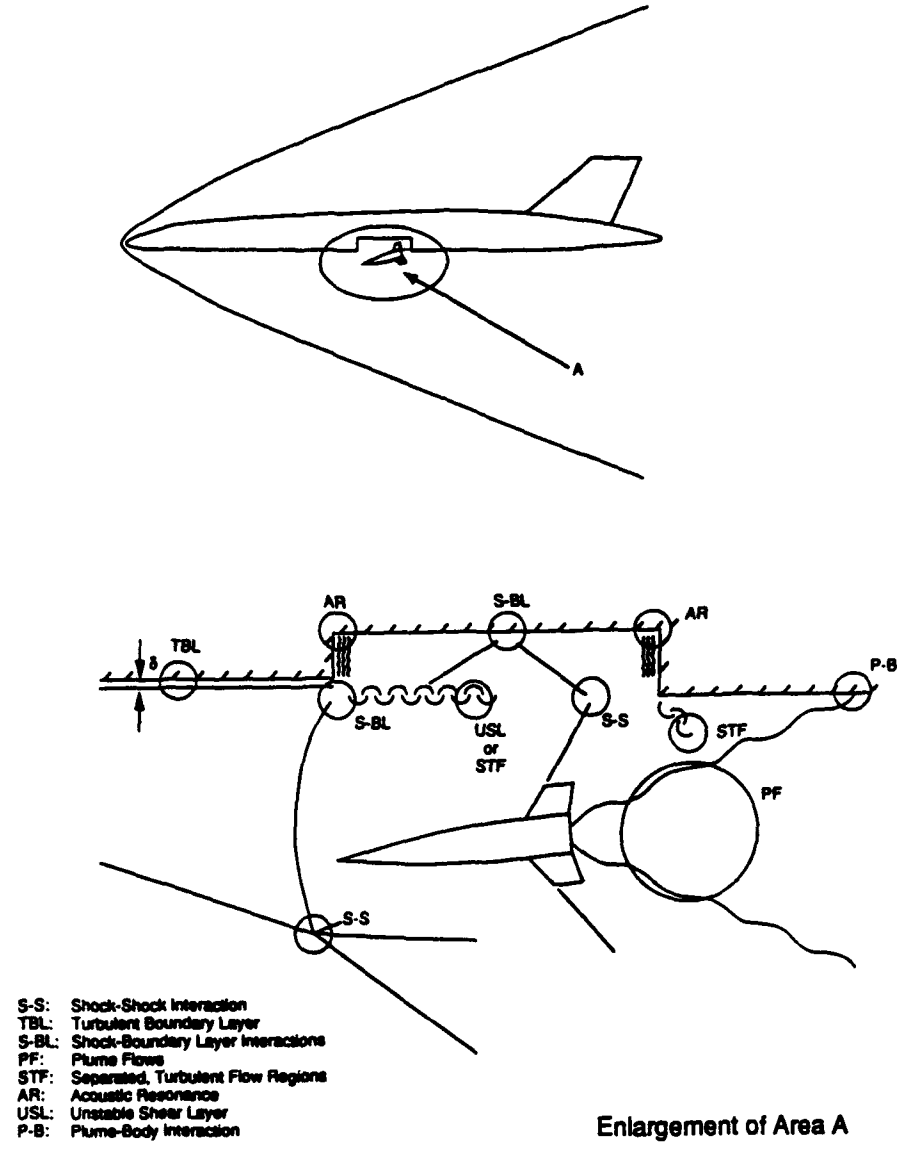


Fig. 1. Store separation: relevant physical processes.

rapidly. Thus, one can expect extensive damage from any experimental program of this sort.

Therefore, in-flight experiments appear as the only viable choice. However, this is a very primitive, and extremely expensive, design process.

—A protoype has to be built to attain certainty in the safety of the design. Production of a complete prototype on such uncertain terms—there is no guarantee that it will deploy safely—appears almost unjustifiable.
—Unsafe deployment may damage or destroy the carrier vehicle. This high risk implies additional expenses in any test program.
—The prototype has to be tested for each new release position: safe deployment from a given position at a certain speed and height does not imply safe deployment from any other position and/or any other speed and/or any other height. As one can see, this can lead to an extremely lengthy, and costly, certification procedure for each new ordnance store entering service.

The situation outlined above for store separation is not much different from that encountered in all of the other engineering applications listed above. The main difficulty in predicting all of these flowfields stems from the fact that body motion will, in most cases, lead to complex, time dependent flows. Advances in computer speed and memory over the next decade will allow the simulation of these flows on a routine basis. Thus, one can expect CFD to gradually take the lead role in the design process for these applications. The present effort represents a first step in this direction.

1.2. *General features of any computational fluid dynamics methodology for moving bodies*

The accurate simulation of three-dimensional time-dependent, compressible flows with moving bodies requires the following capabilities.

(a) *Interactive grid generation methods.* The fast, userfriendly, interactive generation of grids in three dimensions is essential to the success and widespread acceptance of any CFD tool in the user community. Without such a capability, the set-up times for new problems run in the order of months, significantly reducing the benefits which may be realized from any CFD methodology. Thus, interactive grid generation methods for unstructured grids that reduce set-up times to days if not hours are a prime requirement.

(b) *Solvers that can handle moving frames of reference.* Since at the very least the portions of the mesh close to the moving bodies will move in time, the ability to describe the equations of motion for the fluids in moving frames of reference becomes mandatory.

(c) *High-order, monotonicity-preserving schemes.* These schemes are needed to simulate time-dependent flows with strong shocks and other discontinuities that will arise in the flows of interest in this effort (supersonic and hypersonic speeds).

(d) *Proper modeling of turbulent, separated flows.* The flowfields considered will have vast regions of separated, turbulent flow. The lowest order turbulence model that can yield acceptable results for the flows considered here is the k, ϵ model.

(e) *Fast regridding capability for regions with distorted elements.* These are needed because the motion of bodies may be severe, leading to distorted elements which in turn lead to poor numerical results.

(f) *Adaptive refinement schemes.* Experience over recent years[3-7] has demonstrated that self-adaptive refinement schemes are essential in reducing the total number of degrees of freedom without deteriorating the accuracy of the solution. In three dimensions, the ability to refine locally regions of interest will determine the accuracy of the result and whether it can be obtained in a reasonable time. Given the currently available hardware, it is impossible to solve three-dimensional problems using uniformly fine grids everywhere in the computational domain.

(g) *Consistent rigid body motion integrators.* In order to fully couple the motion of rigid bodies with the aerodynamic forces exerted on them, consistent rigid body motion integrators must be developed. This task is relatively simple in two dimensions. However, in three dimensions the temporal variation of the moments of inertia tensor can lead to difficulties.

(h) *Interactive post-processing capabilities.* Understanding of the complex, time-dependent, three-dimensional flowfields requires instantaneous visualization of several key parameters such as pressure, Mach number, density, etc. An engineer that cannot visualize immediately a computed flowfield, in order to make judicious changes in the design, will never accept CFD as a design tool. Thus, a fast, interactive, workstation-based post-processing capability is required.

This list indicates that techniques from several different areas of CFD and computer science must be combined to meet the desired goal. It is therefore not surprising that very few attempts have been made to tackle the complete class of problems. Currently, the chimera grid scheme[8] seems to be the most promising approach for structured grids. In this approach, local grids for each body are overset on a major grid that covers the complete computational domain. For unstructured grids, Formaggia *et al.*[9] used local remeshing for regions of distorted elements in combination with Eulerian and arbitrary Eulerian–Lagrangian solvers in two dimensions to simulate store separation problems. In both cases, the body motion was prescribed, and no adaptive refinement techniques were employed. In 1988 the present author presented a fully coupled two-dimensional fluid–rigid body interaction algorithm.[5,6] This algorithm also employed adaptive remeshing to accurately simulate the flowfield at hand. This development represented the first attempt to combine and incorporate in a single, coherent software package all of the requirements listed above.

The present paper extends this methodology to three dimensions. While conceptually the same as the two-dimensional algorithm, the three-dimensional extension requires several important improvements: better three-dimensional grid generators, consistent three-dimensional rigid body motion integrators, interactive plotting tools, and access to a large memory supercomputer for debugging. Given the currently available computer hardware, and our lack of knowledge in turbulence modeling, it seems unreasonable to include turbulence modeling at the present stage of development. Therefore, the present discussion will center on Euler solvers, rather than Navier–Stokes solvers for compressible flows.

The rest of this paper is divided as follows. Section 2 treats the equations of motion for the flowfield in arbitrary frames of reference, as well as their solution [items (b) and (c) above]. Section 3 deals with the equations of motion for the moving bodies [item (g)]. In the present case, we restrict the description to rigid bodies. Section 4 outlines the gridding technique used [items (a) and (e)]. The gridding technique is also used to adaptively regrid the computational domain [item (f)]. Finally, Sec. 5 contains numerical examples that demonstrate the capabilities developed.

2. THE EQUATIONS OF MOTION FOR THE FLUID

In order to handle the moving frames of reference associated with the moving finite elements, the partial differential equations need to be modified. This is most easily accomplished by the Arbitrary Lagrangian–Eulerian (ALE) formulation. The derivation of the equations may be found in Ref. 10. Here, we just state the final form of the equations of motion. Given the velocity field $\mathbf{w}$ for the elements

$$\mathbf{w} = (w^x, w^y, w^z), \tag{1}$$

the Euler equations that describe an inviscid, compressible fluid may be written as

$$\begin{Bmatrix} \rho \\ \rho u^x \\ \rho u^y \\ \rho u^z \\ \rho e \end{Bmatrix}_{,t} + \begin{Bmatrix} (u^x - w^x)\rho \\ (u^x - w^x)\rho u^x + p \\ (u^x - w^x)\rho u^y \\ (u^x - w^x)\rho u^z \\ (u^x - w^x)\rho e + u^x p \end{Bmatrix}_{,x} + \begin{Bmatrix} (u^y - w^y)\rho \\ (u^y - w^y)\rho u^x \\ (u^y - w^y)\rho u^y + p \\ (u^y - w^y)\rho u^z \\ (u^y - w^y)\rho e \times u^y p \end{Bmatrix}_{,y} + \begin{Bmatrix} (u^z - w^z)\rho \\ (u^z - w^z)\rho u^x \\ (u^z - w^z)\rho u^y \\ (u^z - w^z)\rho u^z + p \\ (u^z - w^z)\rho e + u^z p \end{Bmatrix}_{,y} = -\nabla \cdot \mathbf{w} \begin{Bmatrix} \rho \\ \rho u^x \\ \rho u^y \\ \rho u^z \\ \rho e \end{Bmatrix}. \tag{2}$$

Observe that in the case of no element movement ($\mathbf{w} = \mathbf{0}$), we recover the usual Eulerian conservation-law form of the Euler equations. If, however, the elements move with the particle velocity ($\mathbf{w} = \mathbf{v}$), we recover the Lagrangian form of the equations of motion. From the numerical point of view, Eq. (2) implies that all that is required when going from an Eulerian frame to an ALE frame is a modified evaluation of the fluxes on the left-hand side, and the additional evaluation of source-terms on the right-hand side.

As the elements move, their geometric parameters (shape-function derivatives, Jacobians, etc.) need to be recomputed every timestep. If the whole mesh is assumed to be in motion, then these geometric parameters need to be recomputed globally. In order to save CPU time, only a small number of elements surrounding the bodies are actually moved. The remainder of the field is then treated in the usual Eulerian frame of reference, avoiding the need to recompute geometric parameters. This is accomplished by identifying several layers of elements surrounding the bodies, which are then moved. As the number of layers increases, the time-interval between regridding increases, but so also does the cost per timestep. Therefore, one has to strike a balance between the CPU requirements per timestep and the CPU requirements per regridding. In the present case, we found that two to five layers of elements represented a good compromise.

2.1. *Boundary conditions*

When imposing the boundary conditions for the velocities at solid walls, we need to take the velocity of the surface $\mathbf{w}$ into consideration. Denoting the predicted momentum at the surface as $\Delta\rho\mathbf{v}^*$, we can decompose it as follows:

$$\Delta\rho\mathbf{v}^* = \Delta[\rho(\mathbf{w} + \alpha\mathbf{t} + \beta\mathbf{n})], \tag{3}$$

where $\mathbf{t}$ and $\mathbf{n}$ are the tangential and normal vectors. The desired momentum at the new timestep should, however, have no normal velocity component ($\beta = 0$) and has the form

$$\Delta\rho\mathbf{v}^{n+1} = \Delta[\rho(\mathbf{w} + \alpha\mathbf{t})]. \tag{4}$$

Combining Eqs (3) and (4), we obtain for the two following cases:

(a) given $\mathbf{t}$

$$\Delta\rho\mathbf{v}^{n+1} = \Delta\rho\mathbf{w} + [(\Delta\rho\mathbf{v}^* - \Delta\rho\mathbf{w})\cdot\mathbf{t}]\cdot\mathbf{t}; \tag{5}$$

(b) given $\mathbf{n}$

$$\Delta\rho\mathbf{v}^{n+1} = \Delta\rho\mathbf{v}^* - [(\Delta\rho\mathbf{v}^* - \Delta\rho\mathbf{w})\cdot\mathbf{n}]\cdot\mathbf{n}. \tag{6}$$

2.2. *The flow solver (FEM–FCT)*

For the compressible flows described by Eqn (2), 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"). In the present case the appearance of these overshoots, which may lead to numerical instability, is avoided by combining, in a conservative manner, a high-order scheme with a low-order scheme.[11] The temporal discretization of Eq. (2) yields

$$U^{n+1} = U^n + \Delta U, \tag{7}$$

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 rewrite Eq. (7) as

$$U^{n+1} = U^n + \Delta U^l + (\Delta U^h - \Delta U^l) \tag{8}$$

or

$$U^{n+1} = U^l + (\Delta U^h - \Delta U^l). \tag{9}$$

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 Eq. (9)

$$U^{n+1} = U^l + \lim(\Delta U^h - \Delta U^l), \tag{10}$$

in such a way that no new overshoots or undershoots are created. It is at this point that a further constraint, given by the conservation law (2) itself, must be taken into account: strict conservation on the discrete level should be maintained. The simplest way to guarantee this for the node-centered schemes considered here is by constructing schemes for which the sum of the contributions of each individual element (cell) to its surrounding nodes vanishes. This means that the limiting process [Eq. (10)] will have to be carried out in the elements (cells). Further details on the limiting procedure, its algorithmic implementation, and the high- and low-order schemes employed may be found in Ref. 11.

3. THE EQUATIONS OF MOTION FOR THE RIGID BODIES

The movement of rigid bodies can be found in standard textbooks on classical mechanics (see e.g. Ref. 12). Due to its nonlinear character, rigid body motion in three dimensions is not as straightforward as it may seem. Therefore, a more detailed description of the numerical implementation used is given here. The situation under consideration is shown in Fig. 2. Given the position vector of any point of the body

$$\mathbf{r} = \mathbf{r}_c + \mathbf{r}_0 \tag{11}$$

the velocity and acceleration of this point will be

$$\dot{\mathbf{r}} = \dot{\mathbf{r}}_c + \dot{\mathbf{r}}_0 = \mathbf{v}_c + \boldsymbol{\omega} \times \mathbf{r}_0 \tag{12}$$

$$\ddot{\mathbf{r}} = \dot{\mathbf{v}}_c + \dot{\boldsymbol{\omega}} \times \mathbf{r}_0 + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}_0). \tag{13}$$

Using the vector relationships

$$\mathbf{r} \times (\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r})) = (\mathbf{r} \cdot (\boldsymbol{\omega} \times \mathbf{r}))\boldsymbol{\omega} - (\mathbf{r} \cdot \boldsymbol{\omega})(\boldsymbol{\omega} \times \mathbf{r})$$
$$= -\boldsymbol{\omega} \times (\mathbf{t} \otimes \mathbf{r}) \cdot \boldsymbol{\omega} \tag{14}$$

and the abbreviations

$$m = \int_\Omega \mathrm{d}m = \int_\Omega \rho \, \mathrm{d}\Omega \tag{15}$$

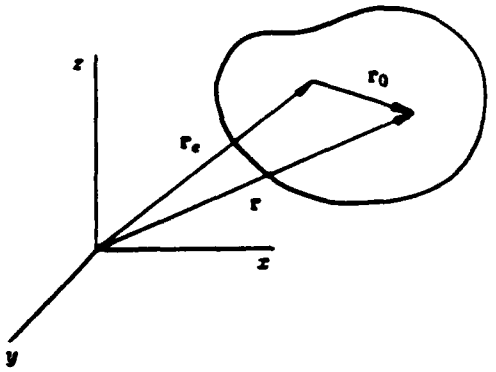


Fig. 2. Rigid body motion.

$$I_{ij} = \int_\Omega r_0^i r_0^j \rho \, \mathrm{d}\Omega \tag{16}$$

$$\Theta = tr(\mathbf{I}) \cdot \mathbf{1} - \mathbf{I}$$
$$= \begin{Bmatrix} I_{yy} + I_{zz} & -I_{xy} & -I_{xz} \\ -I_{xy}, & I_{xx} + I_{zz} & -I_{yz} \\ -I_{xx} & -I_{yz} & I_{xx} + I_{yy} \end{Bmatrix}, \tag{17}$$

we then have the following equations describing balance of forces and moments:

$$m\dot{\mathbf{v}}_c = \sum \mathbf{F} = mg - \int_\Gamma p\mathbf{n} \, \mathrm{d}\Gamma \tag{18}$$

$$\Theta\dot{\boldsymbol{\omega}} - \boldsymbol{\omega} \times (\mathbf{I} \cdot \boldsymbol{\omega}) = \sum \mathbf{r}_0 \times \mathbf{F} = -\int_\Gamma p\mathbf{r}_0 \times \mathbf{n} \, \mathrm{d}\Gamma. \tag{19}$$

Observe that, in two dimensions, the second term on the left-hand side disappears, considerably simplifying the equations. However, in three dimensions it usually does not. Another complication that arises only in three dimensions is the temporal variation of the inertial matrix Θ. As one can see from Eq. (16), the values of Θ will vary as the body rotates. This implies that during the simulation one has to follow the local frame of reference of the body.

In order to update the velocities and positions of the bodies in time, we employ an explicit time-marching scheme. This seems reasonable, as in practical calculations the timescales of the body-movement are much larger than those associated with the fluid flow. Thus, we update $\mathbf{v}_c$, $\boldsymbol{\omega}$ as follows:

$$\mathbf{v}_c^{n+1} = \mathbf{v}_c^n + \Delta t \dot{\mathbf{v}}_c^n \tag{20}$$

$$\boldsymbol{\omega}^{n+1} = \boldsymbol{\omega}^n + \Delta t \dot{\boldsymbol{\omega}}^n. \tag{21}$$

A minor difficulty now becomes apparent: the magnitude of the timestep Δt is unknown before the start of the flowfield update. In the present case, the timestep of the previous timestep was taken instead. This implies that the body movement is "lagging" behind the flowfield by at most one timestep. However, practical simulations show that the actual error is much smaller, as the magnitude of Δt does not change abruptly. For the time-interval $[t^n, t^{n+1}]$, we then have the average velocities

$$\mathbf{v}_c^{av} = 0.5 * (\mathbf{v}_c^{n+1} + \mathbf{v}_c^n) \tag{22}$$

$$\boldsymbol{\omega}^{av} = 0.5 * (\boldsymbol{\omega}^{n+1} + \boldsymbol{\omega}^n). \tag{23}$$

Combining Eqs (22) and (23) with Eq. (12), we are now in a position to compute the velocities at the surface of the bodies, $\mathbf{w}_\Gamma$.

Some of the simulations shown below required several thousand timesteps. If one simply uses the velocities obtained at the boundary from Eqs (22) and (23), the body shape becomes more and more distorted. This is a purely numerical artifact. It can

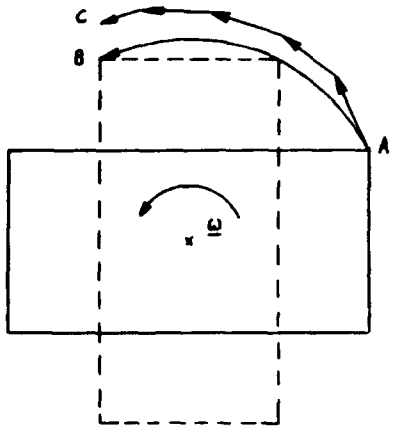


Fig. 3. Elongation of body. Correct path: AB. Computed: AC.

be explained by looking at the situation depicted in Fig. 3. The portions of the body with higher velocity tend to "elongate" the body. This implies that one ought to impose the exact rigid body motion when updating points on the surface. With reference to Fig. 4, we decompose a point lying on the body at time $t = t^n$ into three components

$$\mathbf{r}^n = \mathbf{r}_c + \mathbf{r}_\varphi + \mathbf{r}_r. \tag{24}$$

We can then define unit vectors in the directions of $\mathbf{r}_\varphi$ and $\mathbf{r}_r$

$$\mathbf{e}_\varphi = \frac{\mathbf{r}_\varphi}{|\mathbf{r}_\varphi|} \quad \mathbf{e}_r = \frac{\mathbf{r}_r}{|\mathbf{r}_r|}. \tag{25}$$

Furthermore, we define the vector $\mathbf{e}_n$ as

$$\mathbf{e}_n = \mathbf{e}_\varphi \times \mathbf{e}_r. \tag{26}$$

Then, given the incremental rotation angle $\Delta\varphi = |\omega^{av}|\Delta t$, the new position for $\mathbf{r}$ is obtained from

$$\mathbf{r}^{n+1} = \mathbf{r}_c + \Delta t \mathbf{v}_c^{av} + \mathbf{r}_\varphi + |\mathbf{r}_r|(\cos(\Delta\varphi)\mathbf{e}_r + \sin(\Delta\varphi)\mathbf{e}_n). \tag{27}$$

The complete *rigid body algorithm* then consists of the following steps.

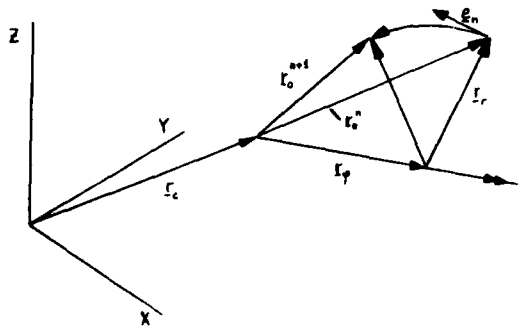


Fig. 4. Decomposition of surface vector for rigid body motion.

(B.1) Compute body forces and moments from Eqs (18) and (19).

(B.2) Transform moments to the local frame of reference of the body

$$M^i = (\mathbf{e}^i \cdot \mathbf{e}^j)M^j. \tag{28}$$

(B.3) Given the estimated timestep Δt, obtain the accelerations $\dot{\mathbf{v}}_c$, $\dot{\omega}$ from Eqs (18) and (19).

(B.4) Given the accelerations, compute average velocities $\mathbf{v}_c^{av}$, ω^{av} for time-interval $[t^n, t^{n+1}]$ from Eqs (22) and (23).

(B.5) Transform back the angular velocity ω^{av} from the local frame of reference of the body to Cartesian coordinates

$$\omega^i = (\mathbf{e}^i \cdot \mathbf{e}^j)\omega^j. \tag{29}$$

(B.6) Given the actual timestep Δt, update the positions of the points lying on the surface of the body, as well as the points defining the body geometry using Eqs (24)–(27).

(B.7) Given the actual timestep Δt, update the positions of the centers of mass and the rotational frame of reference using Eqs (24)–(27).

4. ADAPTIVE REMESHING

For typical compressible flow problems, we have small regions of rapid change in the solution embedded in large regions where the solution is smooth. In order to simulate correctly the interaction of these discontinuities or fronts, an appropriately fine mesh is required. It would, however, be extremely wasteful to have an overall fine mesh, as the regions where a fine mesh is required are small. Therefore, the use of adaptive refinement techniques becomes imperative. As the bodies in the flowfield may undergo arbitrary movement (see examples below), a fixed mesh structure will lead to badly distorted elements. This means that at least a partial regeneration of the computational domain is required. On the other hand, as the bodies move through the flowfield, the positions of relevant flow features will change. Therefore, in most of the computational domain a new mesh distribution will be required. The idea is to regenerate the whole computational domain adaptively, taking into consideration the current flowfield solution. In order to generate or regenerate a mesh we use the advancing front technique:[4-7,13-15]

(F.1) Use the current grid and solution, together with appropriate error indicators, to define the spatial variation of the size, the stretching, and the stretching direction of the elements to be generated. At the nodes of the current grid we define the desired element size, element stretching, and stretching direction. In what follows we will denote this grid as the background grid.

(F.2) Define the boundaries of the domain to be gridded. This is typically accomplished by splines in two dimensions and surface patches in three dimensions.

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

4.1. *Recent developments*

A typical simulation where bodies undergo severe motion typically requires several tens, if not hundreds, of remeshings. Therefore, the grid generator must be reliable and fast.

4.1.1. *Reliability*. We have recently increased the reliability of the grid generator to a point where it can be applied on a routine basis in a production environment. This significant increase in reliability was achieved by: (a) not allowing any bad elements during the generation process; and (b) enlarging and remeshing those regions where new elements could not be introduced. Thus, we first attempt to complete the mesh, skipping those faces that do not give rise to good elements. If pockets of unmeshed regions remain, we enlarge them somewhat, and regrid them. This technique has proven extremely robust and reliable. It has also made smoothing of meshes possible: if elements with negative or small Jacobians appear during smoothing, these elements are removed. The unmeshed regions of space are then regridded. By being able to smooth, the mesh quality was improved substantially.

4.1.2. *Speed*. The following means are used to achieve speed.

(a) *Use of optimal data structures*. The operations that could potentially reduce the efficiency of the algorithm to $O(N^{1.5})$ or even $O(N^2)$ are (see Sec. 2) as follows.

—Finding the next face to be deleted (step F.4).
—Finding the closest given point to a new point (step F.5.2).
—Finding the faces adjacent to a given point (step F.5.3).
—Finding for any giving location the values of generation parameters from the background grid (steps F.3 and F.7). 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 these search operations as efficiently as possible. The data structures used are as follows.

—Heap-lists to find the next face to be deleted from the front.
—Quad-trees (two-dimensional) and octrees (three-dimensional) to locate points that are close to any given location.
—Linked lists to determine which faces are adjacent to a point.

The detailed implementation of these data-structures may be found in Ref. 13.

(b) *Filtering*. Typically, the number of close points and faces is far too conservative, i.e. large. As an example, consider the search for close points: there may be up to eight points inside an octant, but of these only one may be close to the face to be taken out. The idea is to filter out these "distant" faces and points in order to avoid extra work afterwards. While the search operations are difficult to vectorize, these filtering operations lend themselves to vectorization in a straightforward way, leading to a considerable overall reduction in CPU requirements.

(c) *Automatic reduction of unused points*. As the front advances into the domain and more and more tetrahedra are generated, the number of tree-levels increases. This automatically implies an increase in CPU time, as more steps are required to reach the lower levels of the trees. In order to reduce this CPU increase as much as possible, all trees are automatically restructured. All points which are completely surrounded by tetrahedra are eliminated from the trees. We have found this procedure to be extremely effective. It reduces the asymptotic complexity of

the grid generator to less than $O(N \log N)$. In fact, in most practical cases one observes a linear $O(N)$ asymptotic complexity, as CPU is traded between subroutine call overheads, and less close faces on average for large problems.

(d) *Global* h-*refinement*. While the basic advancing front algorithm is a scalar algorithm, h-refinement can be completely vectorized. Therefore, the adaptive remeshing process can be made considerably faster by first generating a coarser, but stretched mesh, and then refining globally this first mesh with classic h-refinement.[6] Typical speed-ups achieved by using this approach are 1:6 to 1:7.

Currently, the advancing front algorithm constructs grids at a rate of 25,000 tetrahedra per minute on the Cray-XMP or Cray-2. With one level of h-refinement, the rate is 190,000–200,000 tetrahedra per minute. This rate is essentially independent of grid-size, but may decrease for very small grids.

4.2. *Local remeshing*

Practical simulations revealed that the appearance of badly distorted elements occurred at a frequency that was much higher than that expected from the element size prescribed. Given the relatively high cost of global remeshing, we explored the idea of local remeshing in the vicinity of the elements that became too distorted. Thus, we proceed as follows.

(L.1) Identify the badly distorted elements in the layers that move, writing them into a list LEREM(1: NEREM).
(L.2) Add to this list the elements surrounding these badly distorted elements.
(L.3) Form "holes" in the present mesh by:
  (L.3.1) Forming a new background mesh with the elements stored in the list LEREM.
  (L.3.2) Deleting the elements stored in LEREM from the current mesh.
  (L.3.3) Removing all unused points from the grid thus obtained.
(L.4) Recompute the error indicators and new element distribution for the background grid.
(L.5) Regrid the "holes" using the advancing front method.

Typically, only a very small number of elements (< 10) becomes so distorted that a remeshing is required. Thus, local remeshing is a very economical tool that has allowed us to reduce CPU requirements by more than 60% for typical runs.

4.3. *Determination of element sizes*

In order to estimate the element size, stretchings, and stretching directions, we employ the modified interpolation theory error indicator proposed in Ref. 3. In one dimension, on a uniform grid of element size h, this error indicator reduces to the following form:

$$E_i = \frac{|U_{i+1} - 2 \cdot U_{i-1}|}{|U_{i+1} - U_i| + |U_i - U_{i-1}| + c_n[|U_{i+1}| + 2 \cdot |U_i| + |U_{i-1}|]}. \quad (30)$$

Defining the following "derivative quantities":

$$D_i^0 = c_n(|U_{i+1}| + 2 \cdot |U_i| + |U_{i-1}|) \quad (31)$$

$$D_i^1 = |U_{i+1} - U_i| + |U_i - U_{i-1}| \quad (32)$$

$$D_i^2 = |U_{i+1} - 2 \cdot U_i + U_{i-1}|, \quad (33)$$

the error on the present ("old") grid is given by

$$E_i^{\text{old}} = \frac{D_i^2}{D_i^1 + D_i^0}. \quad (34)$$

This implies that a reduction of the current element size h^{old} by a fraction ξ to

$$h^{\text{new}} = \xi \cdot h^{\text{old}} \quad (35)$$

will lead to the following estimated errors:

$$E_i^{\text{new}} = \frac{D_i^2 \xi^2}{D_i^1 \xi + D_i^0}. \quad (36)$$

Thus, given the desired error value E^{new}, the reduction factor ξ becomes

$$\xi = \frac{E^{\text{new}}}{E^{\text{old}}} \frac{1}{2} \left[\frac{D_i^1 + \sqrt{\left((D_i^1)^2 + 4 D_i^0 \frac{E^{\text{old}}}{E^{\text{new}}} [D_i^1 + D_i^0] \right)}}{[D_i^1 + D_i^0]} \right]. \quad (37)$$

Notice that if the solution is smooth, implying $D^1 \ll D^0$, then the reduction factor reverts to

$$\xi = \sqrt{\left(\frac{E^{\text{new}}}{E^{\text{old}}} \right)}, \quad (38)$$

consistent with the second-order accuracy assumption of linear elements. However, close to a discontinuity, where $D^1 \gg D^0$, the reduction factor ξ is given by

$$\xi = \frac{E^{\text{new}}}{E^{\text{old}}}. \quad (39)$$

In two and three dimensions we define the corresponding matrices

$$(D^0)_{kl}^I = h^2 c_n \int_\Omega |N_k^I| \, |N_l^J| \, |U_J| \, d\Omega \quad (40)$$

$$(D^1)_{kl}^I = h^2 \int_\Omega |N_k^I| \, |N_l^J U_J| \, d\Omega \quad (41)$$

$$(D^2)_{kl}^I = h^2 \left| \int_\Omega N_k^I N_l^J \, d\Omega U_J \right|, \quad (42)$$

where N^I denotes shape-function of node I, and h is a typical element length. Given these matrices, we obtain the error-indicator matrix $\mathbf{E}$ and its modal decomposition

$$\mathbf{E} = \begin{Bmatrix} E_{xx} & E_{yx} & E_{zx} \\ E_{xy} & E_{yy} & E_{zy} \\ E_{xz} & E_{yz} & E_{zz} \end{Bmatrix} = \mathbf{X} \cdot \begin{Bmatrix} E_{11} & 0 & 0 \\ 0 & E_{22} & 0 \\ 0 & 0 & E_{33} \end{Bmatrix} \cdot \mathbf{X}^{-1}. \tag{43}$$

Each principal direction is then treated as a one-dimensional problem. Using Eq. (37), we obtain three different element sizes δ_1, δ_2, δ_3 along the principal directions. This information is then used to regenerate a better grid for the problem at hand. We remark on the following characteristics of the present error indicator.

(a) The error indicator is *non-dimensional*. Therefore, several variables may be monitored at the same time in order to accurately track all physical phenomena present. Thus, we can monitor both density (shocks, contact discontinuities) and vorticity (boundary layers) for viscous flow problems.
(b) The error indicator is *bounded*. This implies that the user does not have to change specified error tolerances from run to run. We have found that for large classes of problems the specified error tolerances could be left untouched without impediment to the adaptation process. We find this of particular value for the non-expert user environment.

Before proceeding to the overall algorithm, we summarize the steps required for one adaptive remeshing as follows.

(R.1) Obtain the error indicator matrix for the grid-points of the present grid.
(R.2) Given the error indicator matrix, obtain the element size, element stretching and stretching direction for the new grid.
(R.3) Using the old grid as the "background grid," remesh the computational domain using the advancing front technique.
(R.4) If further levels of global h-refinement are desired: refine the new grid globally.
(R.5) Interpolate the solution from the old grid to the new one.

5. THE OVERALL ALGORITHM

The overall algorithm for the advancement of the solution in time is as follows.

(0.1) Advance the solution one timestep.
  (A.1) Compute the body forces and moments from the pressure field and any exterior forces.
  (A.2) Taking into consideration the kinematic constraints for the body movements, update the velocities of the bodies at $t = t^{n+1}$: $\mathbf{v}_c^{n+1}$, ω^{n+1}. At the same time, obtain the average velocities $\mathbf{v}_c^{av}$, ω^{av} for the time-interval $[t^n, t^{n+1}]$.
  (A.3) With the average velocities $\mathbf{v}_c^{av}$, ω^{av}, obtain the velocities $\mathbf{w}_\Gamma$ on the surface of each body for the time-interval $[n, t^{n+1}]$.
  (A.4) Given the surface velocities $\mathbf{w}_\Gamma$ on the boundaries of the global domain, obtain the global velocity field $\mathbf{w}_\Omega$ for the element movement.
  (A.5) Advance the solution by one timestep using the ALE–FEM–FCT solver. This yields the actual timestep Δt^n.
  (A.6) Given the actual timestep Δt^n and the velocity field for the element movement $\mathbf{w}_\Omega$, update the coordinates of the points.
  (A.7) Update the shape-function derivatives and other geometric parameters for the elements that have been moved.
  (A.8) Update the centers of mass $\mathbf{r}_c$ for the bodies, as well as the coordinates of the points defining the body geometry.
(0.2) If the grid has become too distorted close to the moving bodies: adaptively remesh these regions.
(0.3) If the desired number of timesteps between global remeshings has elapsed: adaptively remesh the complete computational domain.
(0.4) If the desired time-interval has elapsed: stop. Otherwise, advance the solution further (go to 0.1).

6. NUMERICAL EXAMPLES

We consider two numerical examples that demonstrate the effectiveness of the algorithms developed. In both cases an idealized store release from a bay at supersonic speed ($\mathrm{Ma}_\infty = 2.0$) is simulated. Because of symmetry, only half the flowfield domain needs to be simulated. Release into a supersonic flowfield will necessitate the forceful ejection of stores. Therefore, the motion of the stores was prescribed, and the resulting forces computed. Adaptive remeshing was performed every 100 timesteps initially, while at latter times the grid was modified every 40 timesteps. The maximum stretching ratio specified was $S = 1.5$. Density and the absolute value of the velocity were chosen as indicator variables. The latter provided suitable mesh adaptation to the shear layers in the cavity. Even though the grid shows considerable variation in element size, the average grid size was of the order of 280,000 tetrahedra for the first example, and 350,000 tetrahedra for the second one. On a uniform mesh, the required number of elements would have increased by more than an order of magnitude. The required CPU time for runs of this kind is of the order of several CRAY-XMP processor hours.

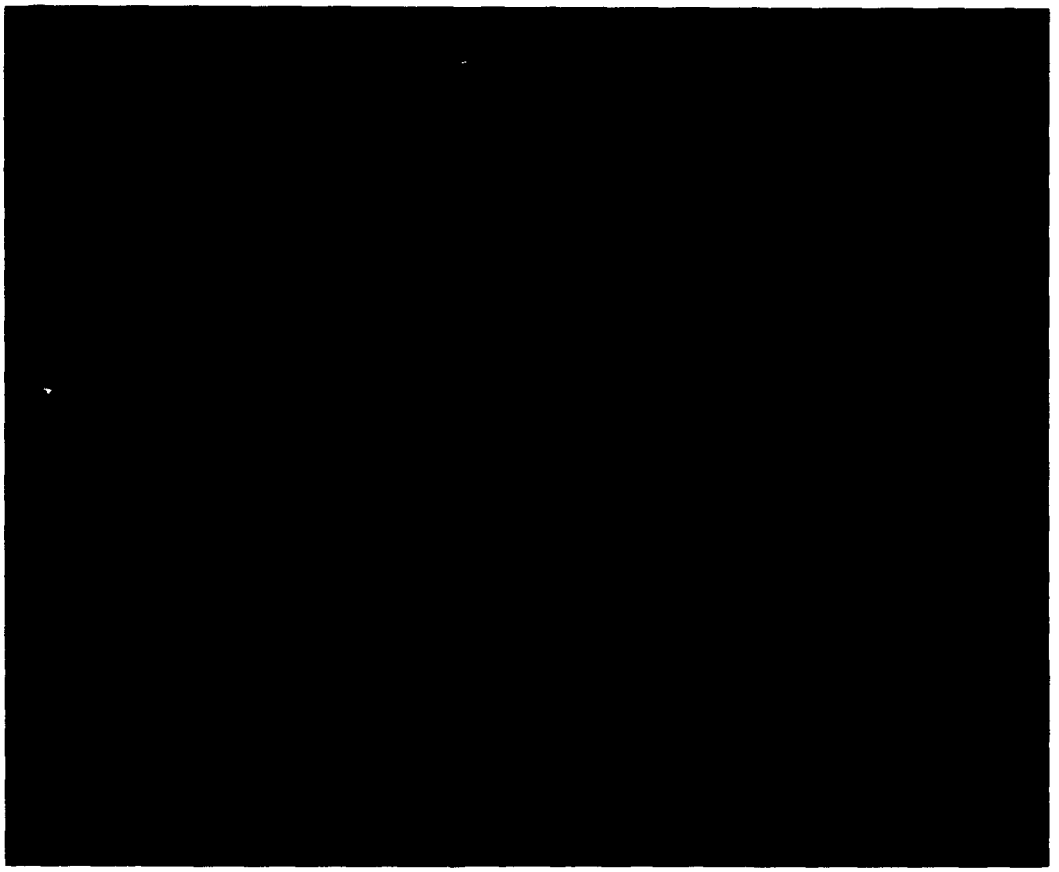


Fig. 5.1. Single store separation. Surface mesh at $T = 0.0$.

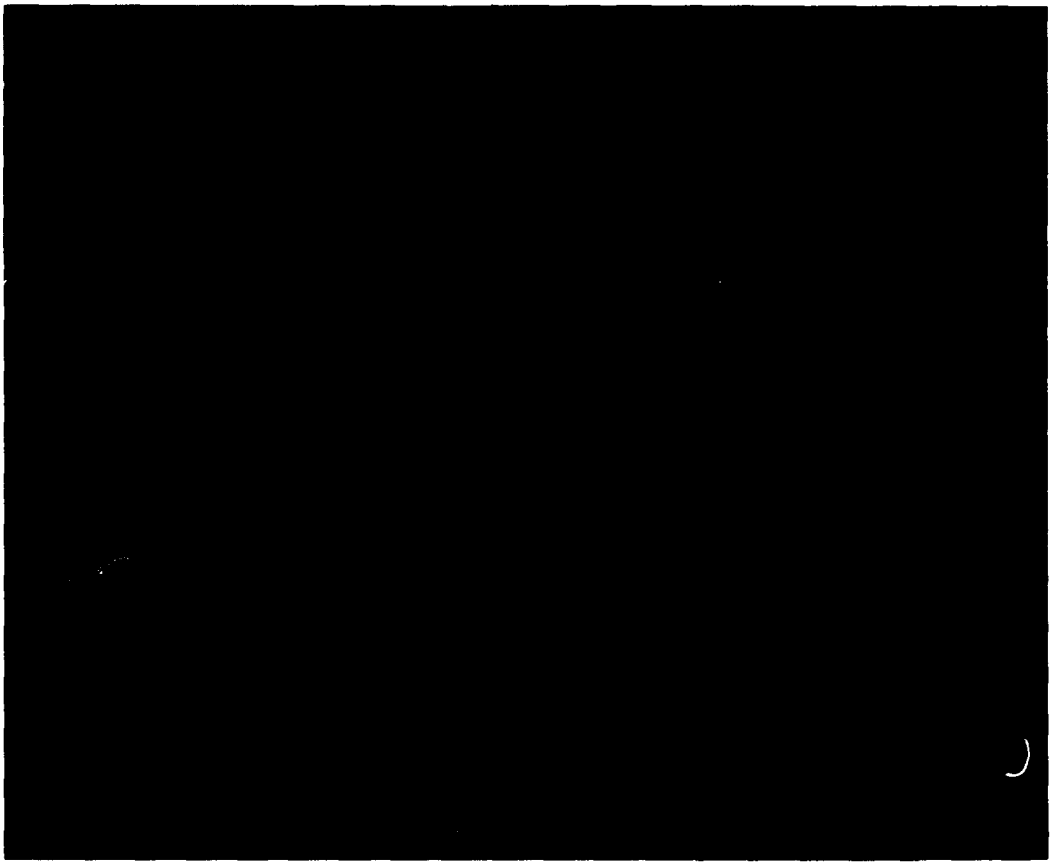


Fig. 5.2. Single store separation. Surface mesh at $T = 0.0$.

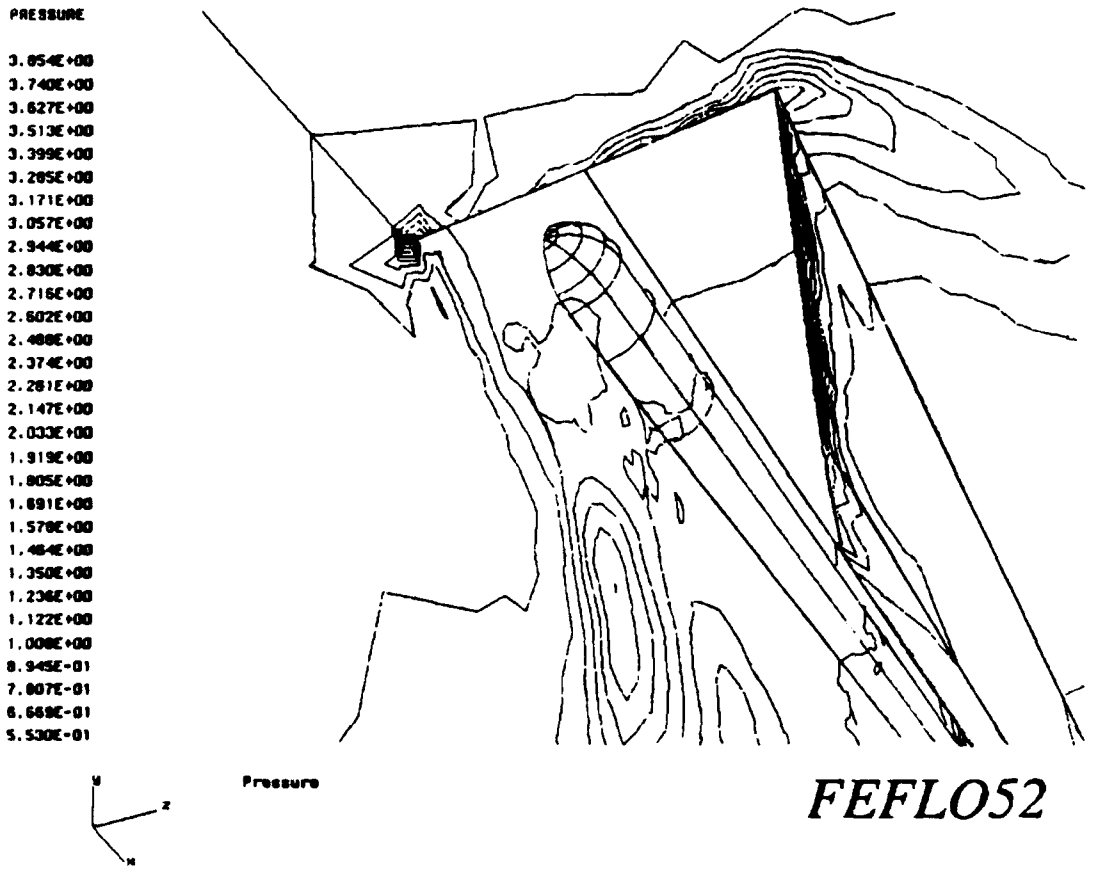


Fig. 5.3. Single store separation. Surface pressure at $T = 0.0$.

6.1. *Single object falling into supersonic free stream*

The computational domain is shown in Figs 5.1 and 5.2. Observe that the doors of the bay simulated are somewhat thicker than in real life. The store is a slender object, resembling a missile. Figures 5.1 and 5.2 show the mesh on the surface of the computational domain at time $T = 0.0$. The corresponding pressure contours (30) are shown in Fig. 5.3. Figures 5.4–5.6 show the surface mesh and the pressure contours (60) at time $T = 8.5$. One can clearly discern the extent of mesh adaptation, as well as the considerable change in shock-strengths and shock-positions that occurred due to body motion.

6.2. *Multiple objects falling into supersonic free stream*

The computational domain is the same as before. The two stores resemble bombs. The store at the back of the cavity is ejected first, followed by the store in the front of the cavity. Figures 6.1 and 6.2 show the mesh on the surface of the computational domain at time $T = 0.0$. The corresponding pressure contours (60) are shown in Fig. 6.3. Figures 6.4–6.6 show the surface mesh and the pressure contours (60) at time $T = 2.62$. While the store at the back of the cavity has already moved a considerable distance, the store in the front has just begun to move. As before, one can clearly discern the extent of mesh adaptation, as well as the considerable change in shock-strengths and shock-positions that occurred due to body motion.

7. CONCLUSIONS

We have demonstrated how the combination of adaptive remeshing techniques, flow solvers for transient problems with moving grids, and integrators for rigid body motion allows the simulation of fully coupled fluid–rigid body interaction problems of arbitrary geometric complexity in three dimensions. The overall reduction in CPU times as compared to those of uniformly fine grids depends strongly on the stretching ratios allowed by the physics of the problem, but typically lies between 10 and 50. Areas that deserve further study are as follows:

—the diffusive effect of interpolation while remeshing

—extension to Navier–Stokes problems

—treatment of multifluid interactions

—extension to flexible bodies and structures.

Fig. 5.4. Single store separation. Surface mesh at $T = 8.5$.

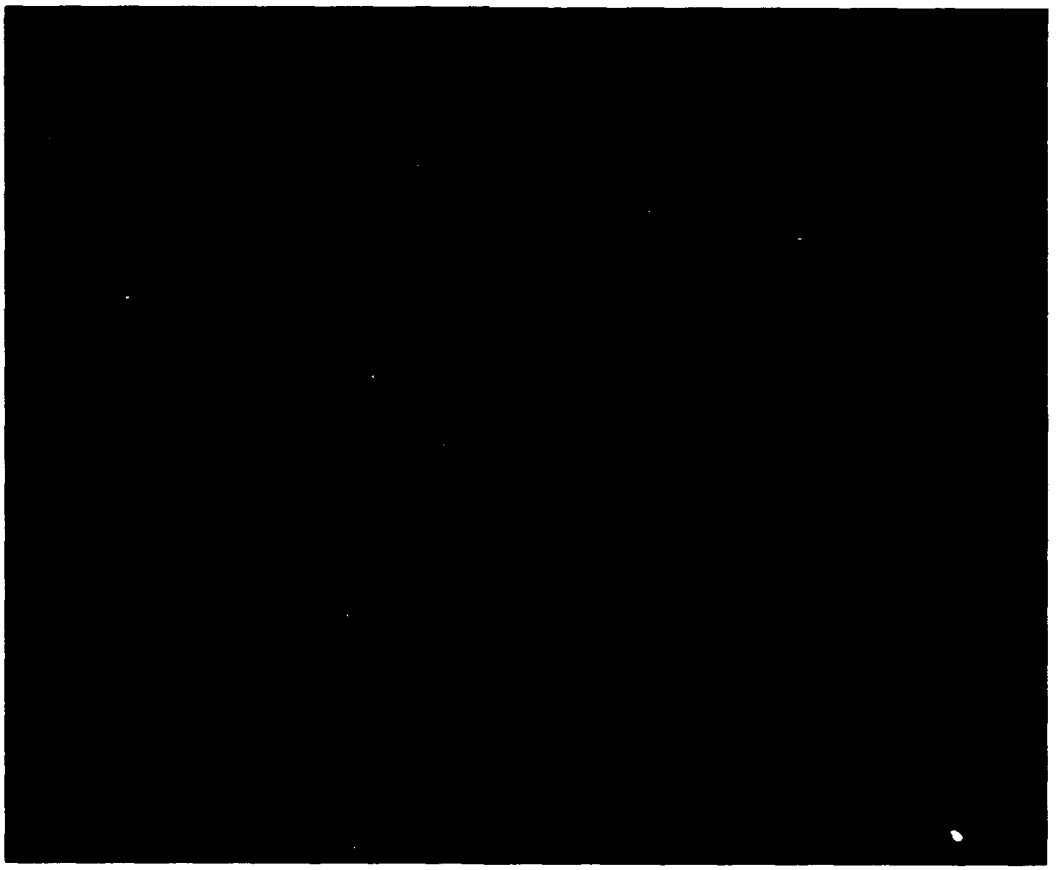


Fig. 5.5. Single store separation. Surface mesh at $T = 8.5$.

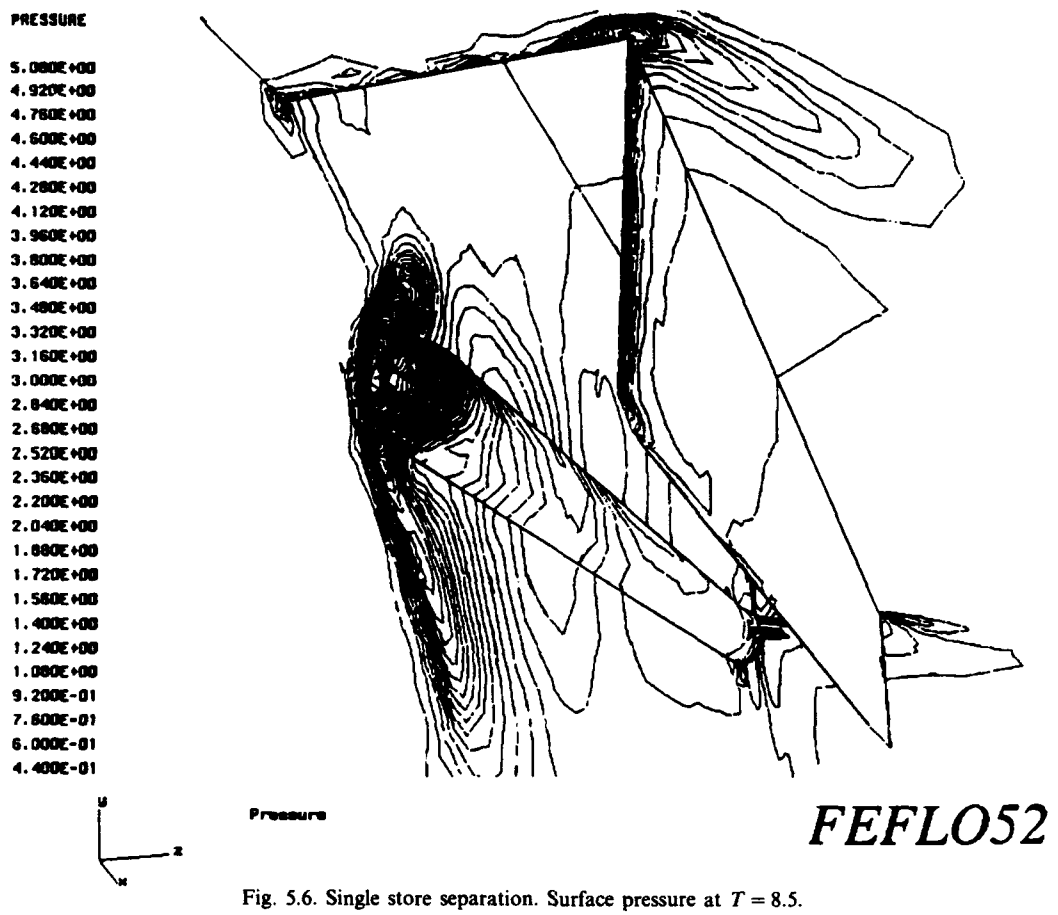


Fig. 5.6. Single store separation. Surface pressure at $T = 8.5$.

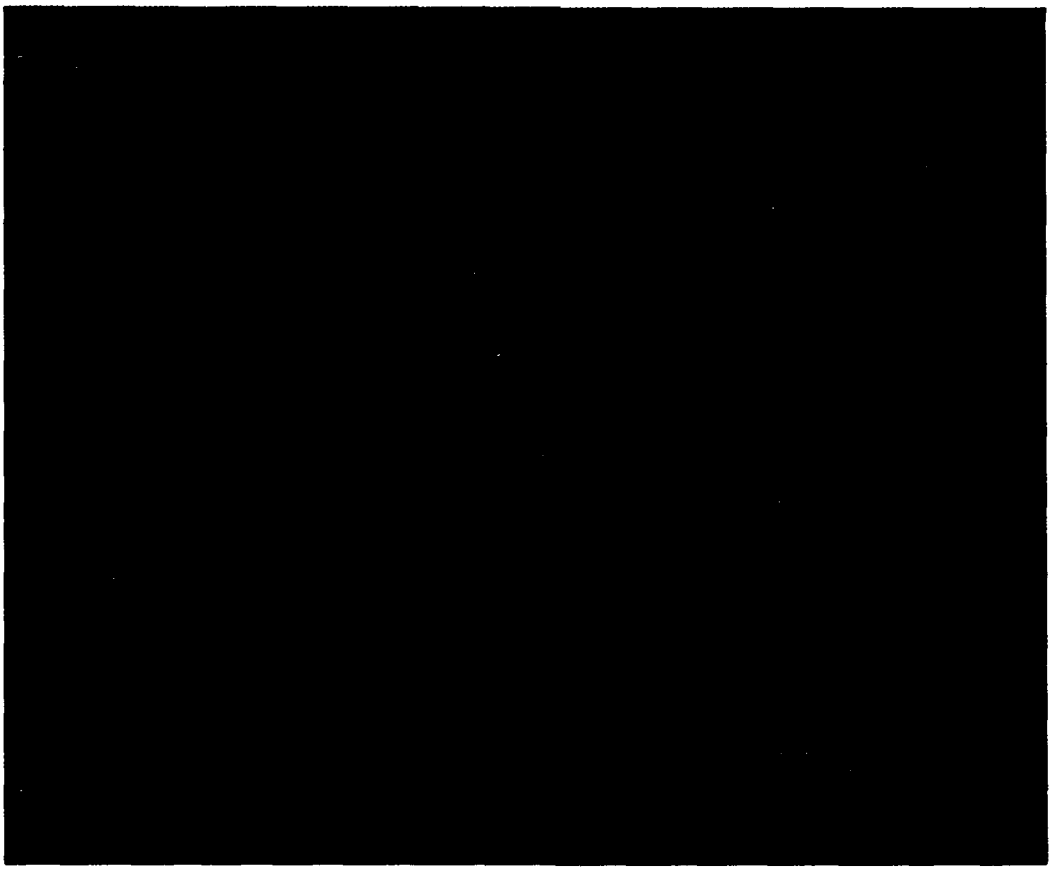


Fig. 6.1. Multiple store separation. Surface mesh at $T = 0.0$.

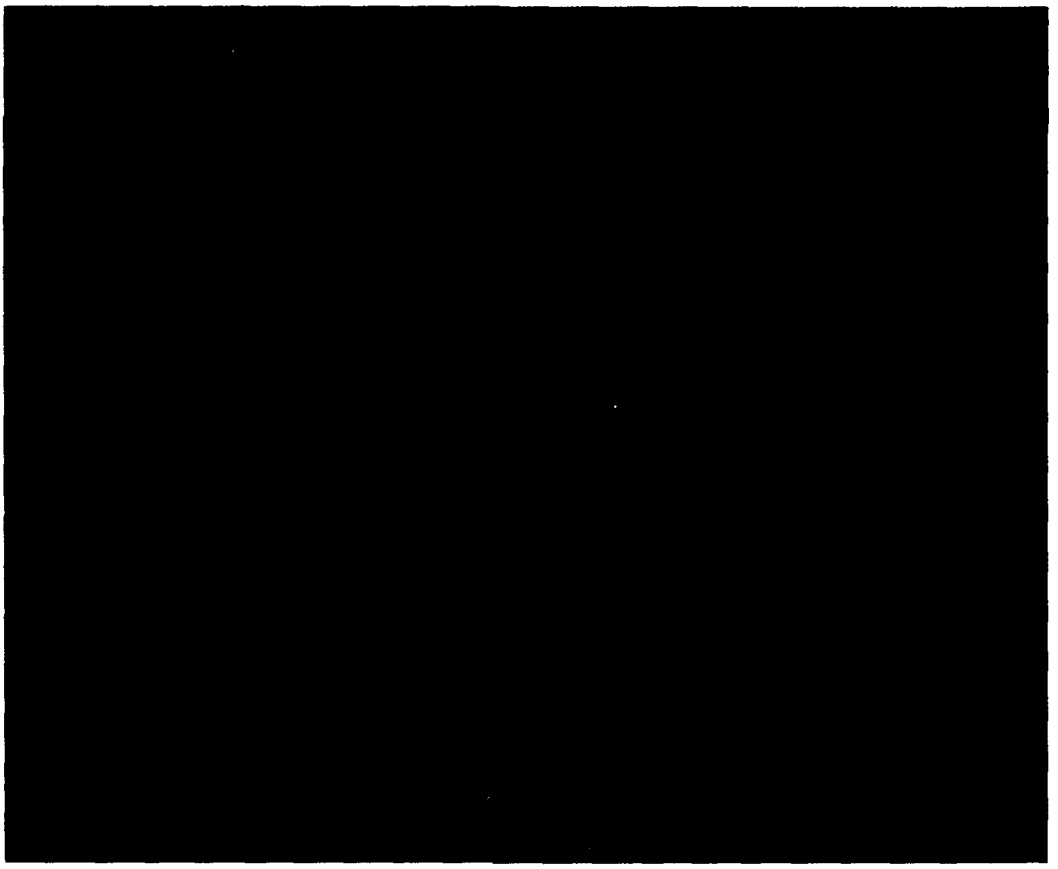


Fig. 6.2. Multiple store separation. Surface mesh at $T = 0.0$.

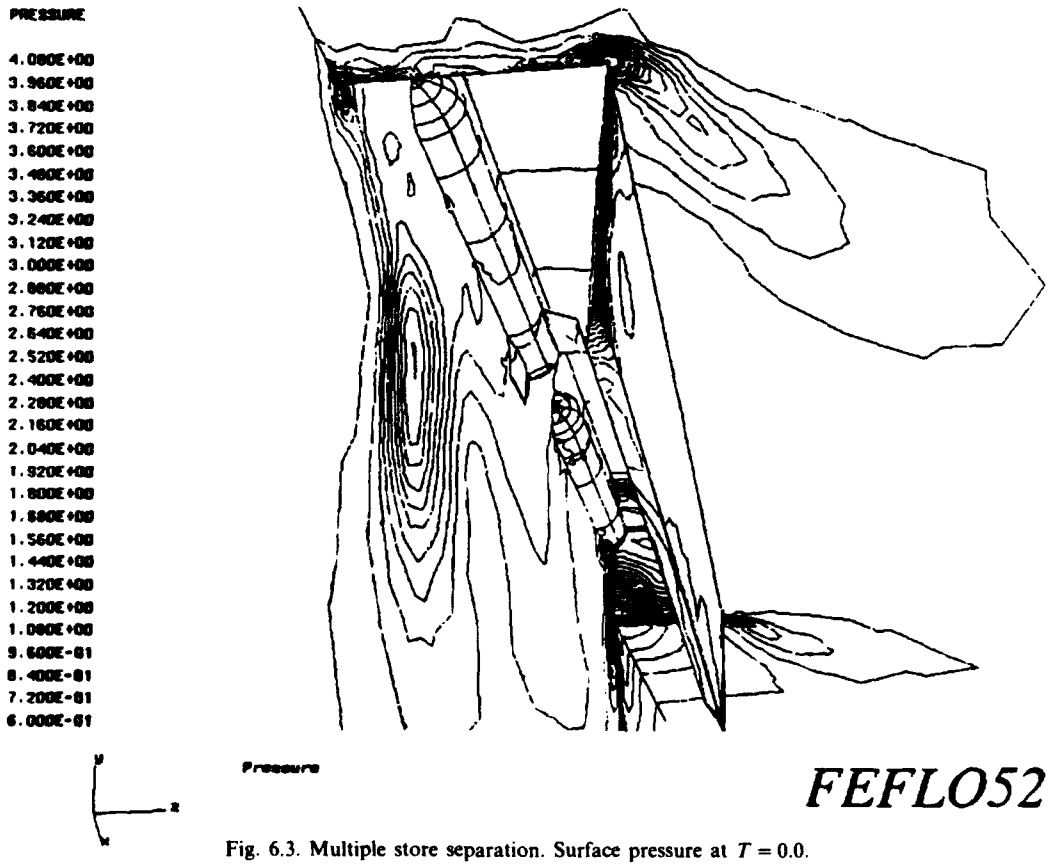


Fig. 6.3. Multiple store separation. Surface pressure at $T = 0.0$.

Fig. 6.4. Multiple store separation. Surface mesh at $T = 2.62$.

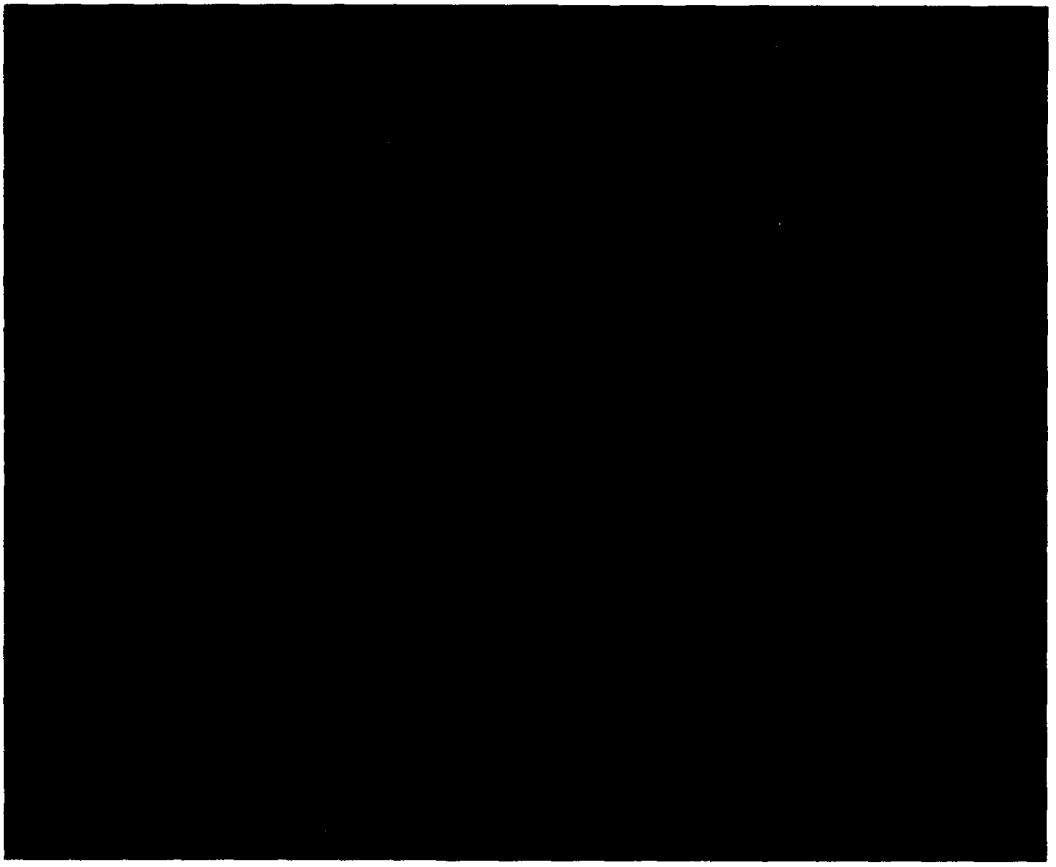


Fig. 6.5. Multiple store separation. Surface mesh at $T = 2.62$.

Fig. 6.6. Multiple store separation. Surface pressure at $T = 2.62$.

Acknowledgements—This work was partially funded by AFOSR under contract AFOSR-89-0540. Dr Leonidas Sakell was the technical monitor. The generous support of CRAY Research, Inc. in the form of ample CPU-time on a CRAY-2S is also gratefully acknowledged.

REFERENCES

1. B. Edney, "Anomalous heat transfer and pressure distribution on blunt bodies at hypersonic speeds in the presence of an impinging shock," FAA Report No. 115, Aero. Research Institute, Sweden, 1986.
2. R. Thareja, J. R. Steward, O. Hassan, K. Morgan and J. Peraire, "A point-implicit unstructured grid solver for the Euler and Navier–Stokes equations," *International Journal of Numerical Methods in Fluids* **9**, 405–425 (1989).
3. R. Löhner, "An adaptive finite element scheme for transient problems in CFD," *Computer Methods in Applied Mechanics and Engineering* **61**, 323–338 (1987).
4. J. D. Baum and R. Löhner, "Numerical simulation of shock-elevated box interaction using an adaptive finite element shock capturing scheme," AIAA paper 89-0653, 1989.
5. R. Löhner, "An adaptive finite element solver for transient problems with moving bodies," *Computers & Structures* **30**, 303–317 (1988).
6. R. Löhner, "Adaptive remeshing for transient problems," *Computer Methods in Applied Mechanics and Engineering* **75**, 195–214 (1989).
7. J. Peraire, M. Vahdati, K. Morgan and O. C. Zienkiewicz, "Adaptive remeshing for compressible flow computations," *Journal of Computational Physics* **72**, 449–466 (1987).
8. F. C. Dougherty and J. Kuan, "Transonic store separation using a three-dimensional chimera grid scheme, AIAA paper 89-0637, 1989.
9. L. Formaggia, J. Peraire and K. Morgan, "Simulation of a store separation using the finite element method," *Applied Mathematical Modelling* **12**, 175–181 (1988).
10. J. Donea, "An arbitrary Lagrangian–Eulerian finite element method for transient dynamic fluid–structure interactions," *Computer Methods in Applied Mechanics and Engineering* **33**, 689–723 (1982).
11. R. Löhner, K. Morgan, J. Peraire and M. Vahdati, "Finite element flux-corrected transport (FEM–FCT) for the Euler and Navier–Stokes equations," *International Journal for Numerical Methods in Fluids* **7**, 1093–1109 (1987).
12. A. Sommerfeld, *Vorlesungen über Theoretische Mechanik*, Harri Deutsch, Frankfurt (1976).
13. R. Löhner, "Some useful data structures for the generation of unstructured grids," *Communications in Applied Numerical Methods* **4**, 123–135 (1988).
14. R. Löhner and P. Parikh, "A three-dimensional grid generation by the advancing front method," *International Journal of Numerical Methods in Fluids* **8**, 1135–1149 (1988).
15. J. Peraire, K. Morgan and J. Peiro, "Unstructured finite element mesh generation and adaptive procedures for CFD," AGARD-CP-464, 18, 1990.