INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 14, 1407–1419 (1992)

ADAPTIVE *h*-REFINEMENT ON 3D UNSTRUCTURED GRIDS FOR TRANSIENT PROBLEMS

RAINALD LÖHNER

CMEE, School of Engineering and Applied Science, The George Washington University, Washington, DC 20052, U.SA.

AND

JOSEPH D. BAUM

SAIC, McLean, VA 22102, U.S.A.

SUMMARY

An adaptive finite element scheme for transient problems is presented. The classic *h*-enrichment/coarsening is employed in conjunction with a tetrahedral finite element discretization in three dimensions. A mesh change is performed every *n* time steps, depending on the Courant number employed and the number of 'protective layers' added ahead of the refined region. In order to simplify the refinement/coarsening logic and to be as fast as possible, only one level of refinement/coarsening is allowed per mesh change. A high degree of vectorizability has been achieved by pre-sorting the elements and then performing the refinement/coarsening groupwise according to the case at hand. Further reductions in CPU requirements are realized by optimizing the identification and sorting of elements for refinement and deletion. The developed technology has been used extensively for shock–shock and shock–object interaction runs in a production mode. A typical example of this class of problems is given.

KEY WORDS Adaptive mesh refinement Finite elements Compressible flow Transient problems

1. INTRODUCTION

The solution of large-scale transient problems around complex geometries is a problem common to many fields of computational fluid dynamics. In the present case we desire to simulate efficiently shock–shock and shock–structure interactions for realistic, complex, three-dimensional, engineering-type geometries. Therefore from the onset we used *unstructured grids.*[1] At the same time we developed monotonicity-preserving schemes that operate on unstructured grids.[2,3] With these schemes, moving and standing shocks are captured within two or three grid points without the over- and undershoots that appear in ordinary linear schemes. Besides their ability to discretize accurately complex geometries, a second very attractive feature of unstructured grids is the ease with which adaptive refinement can be incorporated into them. The addition of further degrees of freedom does not destroy any previous structure. Thus the flow solver requires no further modification when operating on an adapted grid. For many practical problems the regions that need to be refined are extremely small as compared to the overall domain. Therefore the savings in storage and CPU requirements typically range between 10 and 100 as compared to an overall fine mesh.[4,5] Our experience in 2D,[5,6] as well as that of other researchers working in

0271–2091/92/121407–13$06.50

Received 21 December 1990
Revised 11 October 1991

this area,[7-9] indicates that for the majority of the daily production-type runs, adaptive refinement makes the difference between being or not being able to run the problems to an acceptable accuracy in a reasonable time. Without it we would be forced to use much coarser grids, with lower accuracy for the same expense. The present paper extends to 3D the capabilities developed in 2D several years ago.[4] Though conceptually the same, this extension to 3D required the advent of powerful graphics workstations for effective debugging as well as access to a large-memory supercomputer for testing and optimization.

Any adaptive refinement scheme is composed of three main ingredients. These are

(1) an optimal mesh criterion
(2) an error indicator
(3) a method to refine and coarsen the mesh.

They give answers to the questions

(1) how should the optimal mesh be?
(2) where is refinement/coarsening required?
(3) how should the refinement/coarsening be accomplished?

Many variants of each of these subtopics have been explored and shown to be useful for a certain class of problems.[4-13] Here we seek a method that is efficient and reliable for *transient* compressible flow problems, where a mesh change may be required every 5–10 time steps. This leads to the following design criteria for the error indicator.

(a) the error indicator should be fast.
(b) The error indicator should be dimensionless so that several 'key variables' can be monitored at the same time.
(c) The error indicator should be bounded so that no further user intervention becomes necessary as the solution evolves.
(d) The error indicator should not only mark the regions with strong shocks to be refined, but also weak shocks, contact discontinuities and other 'weak features' in the flow.

For the refinement method the design criteria are as follows.

(e) The method should be conservative, i.e. a mesh change should not result in the production or loss of mass, momentum or energy.
(f) The method should have a minimal amount of numerical dissipation since many mesh changes are required during the course of one simulation.
(g) The method should not produce elements that are too small since this would reduce too severely the allowable time step of the explicit flow solvers employed.
(h) The method should be fast; in particular it should lend itself to a high degree of parallelism.
(i) The method should not involve a major storage overhead.

2. THE ERROR INDICATOR

An error indicator that meets the design criteria (a)–(d) was proposed in Reference 4. In general terms it is of the form

$$\text{error} = \frac{h^2 |\text{second derivatives}|}{h|\text{first derivatives}| + \varepsilon |\text{mean value}|}. \tag{1}$$

By dividing the second derivatives by the absolute value of the first derivatives, the error indicator

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

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

where N^I denotes the shape function of node I. This error indicator has performed very well in 2D over the years.[4-6] However, when first used in 3D, it proved unreliable. The source for this seemingly inconsistent behaviour was found to stem from the large *local* variations in element size and shape as well as the number of elements surrounding a point encountered in typical 3D unstructured grids. These will produce large variations in the second term in the denominator which are not based on physics but on the mesh structure itself. The solution was to modify this error indicator as follows:

$$E^I = \sqrt{\left(\frac{\sum_{k,l} \left(\int_\Omega N^I_{,k} N^J_{,l} \, d\Omega \cdot U_J \right)^2}{\sum_{k,l} \left(\int_\Omega |N^I_{,k}| |N^J_{,l} U_J| \, d\Omega \right)^2 + \varepsilon M M_I h_I^{-2} |U_I|} \right)}, \tag{3}$$

where MM_I is the lumped mass matrix at point I and h_I is the average element length at point I. This error indicator proved to be remarkably insensitive to local variations in element size and shape while still yielding the correct indicator values for physical phenomena of interest. We attribute this good performance to the smoothing effects of two averaging operations working simultaneously: the lumped mass matrix and the point lengths.

After having determined the values of the error indicators in the elements, all elements lying above a preset threshold value CTORE are refined while all elements lying below a preset threshold value CTODE are coarsened. Protective layers of elements are added to surround the elements to be refined, so that the 'feature' (e.g. a shock) always travels into an already refined region. The number of protective layers that are added depends on the Courant number employed and the number of time steps taken between grid modifications. Usually the refinement is performed every 5–10 time steps, so that for a Courant number $C = 0{\cdot}2$–$0{\cdot}4$, zero to two protective layers are sufficient.

3. ADAPTIVE REFINEMENT METHOD

Our previous experience indicates that the only two refinement methods that are truly general and efficient for the class of problems considered here are *h*-refinement[4-10] and remeshing.[11-14] However, for *strongly unsteady problems*, where a new grid is required every 5–10 timesteps, local *h*-refinement seems to be preferable. Several reasons can be given for this choice.

(a) Conservation presents no problem for *h*-refinement.
(b) No interpolations other than the ones naturally given by the element shape functions are required. Therefore no numerical diffusion is introduced by the adaptive refinement procedure. This is in contrast to adaptive remeshing, where the grids before and after a mesh change may not have the same points in common. The required interpolations of the unknowns will result in an increased amount of numerical diffusion.[13]

(c) *H*-refinement is very well suited to vector and parallel processors. This is of particular importance in the present context, where a mesh change is performed every 5–10 timesteps and a large percentage of mesh points are affected in each mesh change.
(d) *H*-refinement is more robust than remeshing. The number of things that can go wrong seems to be much less than when remeshing.

As described above, we limit the number of refinement/coarsening levels per mesh change to one. Moreover, we only allow refinement of a tetrahedron into two (along a side), four (along a face) or eight new tetrahedra. We call these tetrahedra 1:2, 1:4 and 1:8 tetrahedra or refinement cases respectively. At the same time a 1:2 or 1:4 tetrahedron can only be refined further to a 1:4 tetrahedron, or by first going back to a 1:8 tetrahedron with subsequent further refinement of the 8 subelements. We call these the 2:4, 2:8+ and 4:8+ refinement cases. The refinement cases are summarized in Figure 1. This restrictive set of refinement rules is necessary to avoid the appearance of ill-deformed elements. At the same time it considerably simplifies the refinement/coarsening logic. An interesting phenomenon that does not appear in 2D is the apparently free choice of the inner diagonal for the 1:8 refinement case. As shown in Figure 2, we can place the inner four elements around the inner diagonals 5–10, 6–8, or 7–9. In the present case the shortest inner diagonal was chosen. This choice produces the smallest number of distorted tetrahedra in the refined grid. When coarsening, we again only allow a limited number of cases that are compatible with the refinement. Thus the coarsening cases become 8:4, 8:2, 8:1, 4:2, 4:1 and 2:1. These coarsening cases are summarized in Figure 3.

When constructing the algorithm to refine or coarsen the grid, one faces the usual decision of speed versus storage. The more information from the previous grid that is stored, the faster the new grid may be constructed. Because storage requirement minimization was one of the goals of the present research, we tried to keep only the essential information needed between mesh changes without sacrificing an excessive amount of CPU time. This was accomplished by a modified tree structure which requires 12 integer locations per element in order to identify the 'parent' and 'son' elements of any element as well as the element type.

The first seven integers store the new elements ('sons') of an element that has been subdivided into eight (1:8). For the 1:4 and 1:2 cases the sons are also stored in this allocated space and the remaining integer locations are set to zero.

In the eighth integer the element from which the present element originated (the 'parent' element) is stored.

The ninth integer denotes the position number in the parent element that this element came from.

The 10th integer denotes the element type. We can either have parents or sons of 1:8, 1:4 or 1:2 tetrahedra. We mark these by a positive value of the element type for the parents and a negative value for the sons. Thus, for example, the son of a 1:8 element would be marked as −8.

Finally, in the 11th and 12th integer locations the local and global refinement levels are remembered.

These 12 integer locations per element are sufficient to construct further refinements or to reconstruct the original grid. It is clear that in these 12 integers a certain degree of redundancy is present. For example, the information stored in the 10th integer could be recovered from the data stored in locations 1:8 and 11:12. However, this would require a number of non-vectorizable loops with many IF tests. Therefore it was decided to store this value at the time of creation of new elements instead of recomputing it at a later time. Similarly, the 11th integer can be recovered from the information stored in locations 1:8 and 12. As is the case with the 10th integer, storage was traded for CPU time.

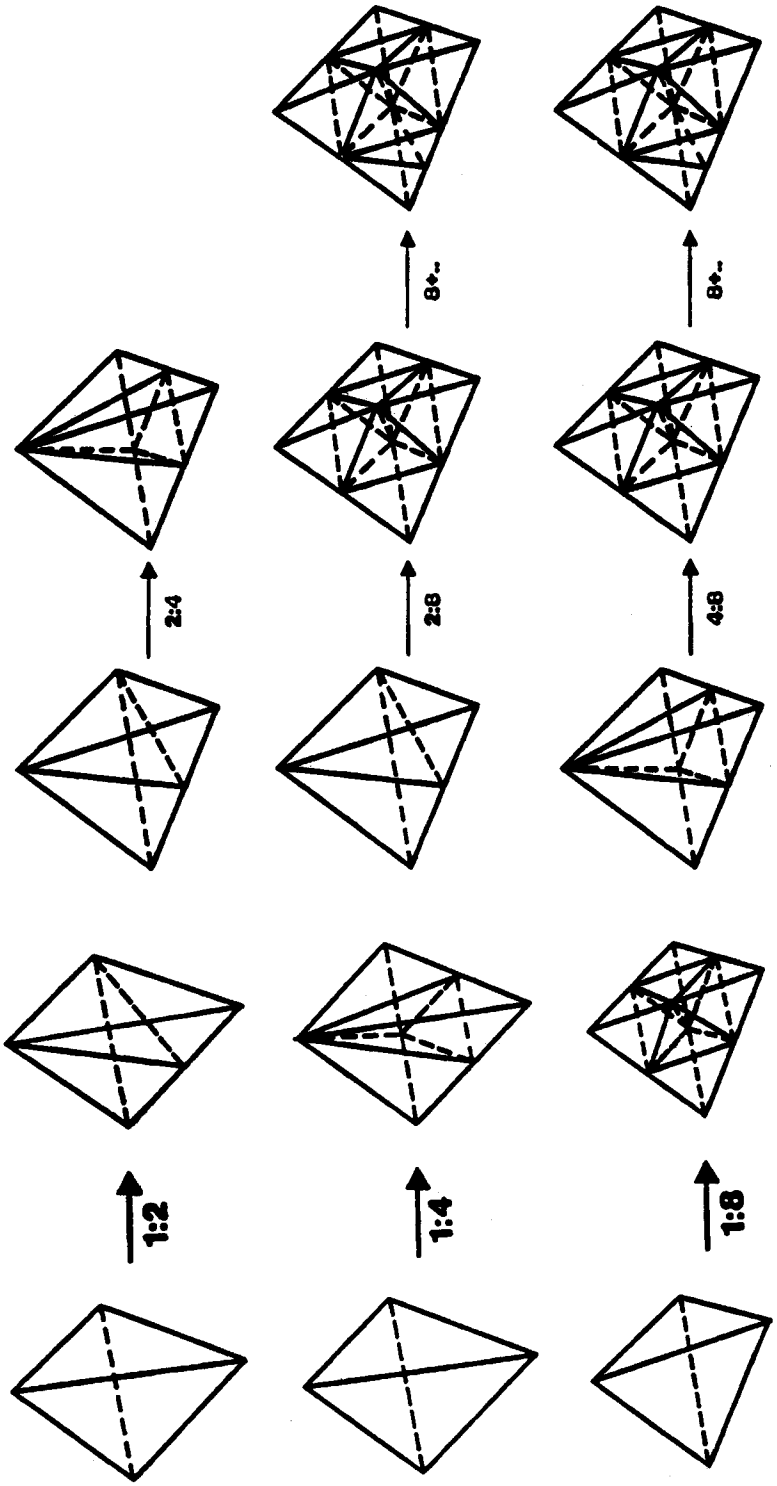


Figure 1. Refinement cases

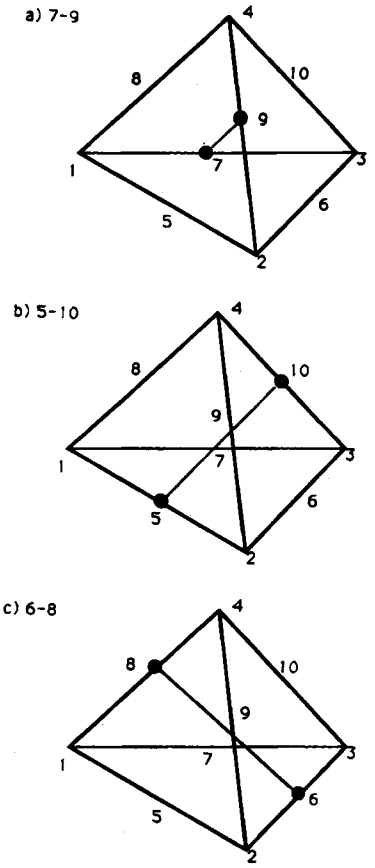


Figure 2. Possible choices for the inner diagonals

4. ALGORITHMIC IMPLEMENTATION

Having outlined the basic refinement/coarsening strategy, we can now describe in more depth its algorithmic implementation. One complete grid change requires algorithmically the following five steps.

1. Construction of the missing grid information needed for a mesh change (basically the sides of the mesh and the sides adjoining each element).
2. Identification of the elements to be refined.
3. Identification of the elements to be deleted.
4. Refinement of the grid where needed.
5. Coarsening of the grid where needed.

4.1. Construction of missing grid information

The missing information consists of the sides of the mesh and the sides belonging to each element. The sides are dynamically stored in two arrays, one containing the two points each side connects and the other (a pointer array) containing the lowest side number reaching out of a

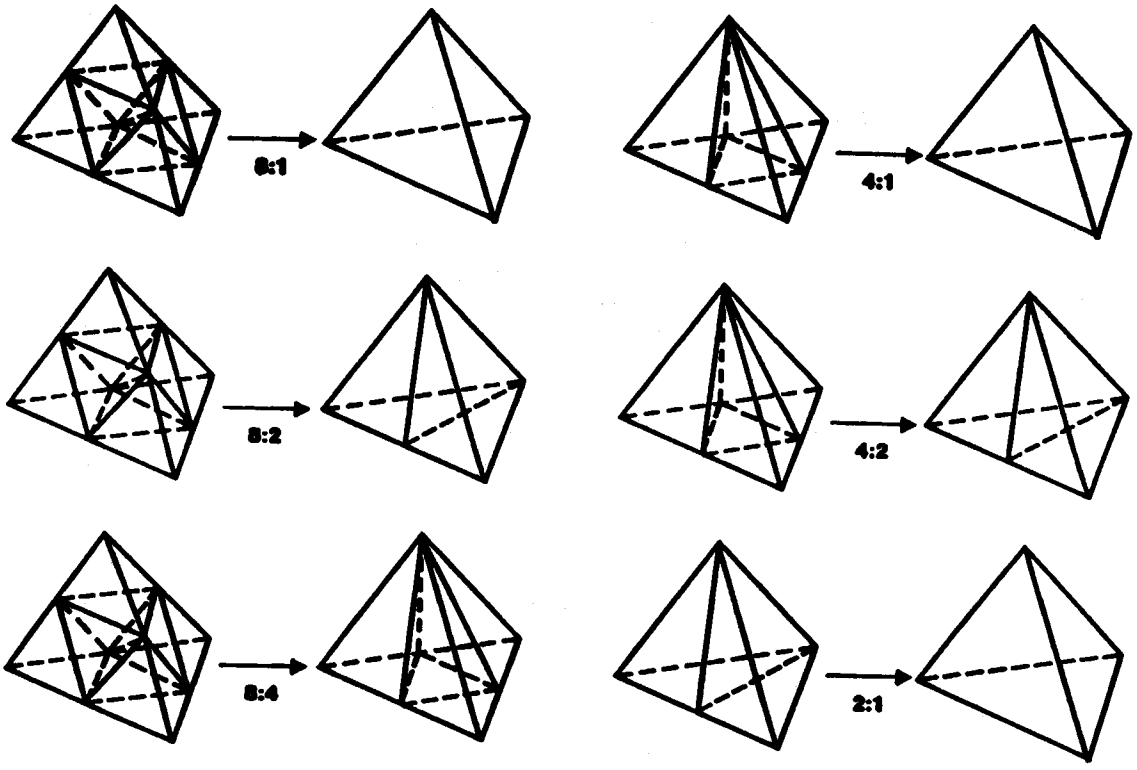


Figure 3. De-refinement cases

point. The formation of these two arrays is accomplished in three main loops over the elements, which are partially vectorizable. After having formed these two side arrays, a further loop over the elements is performed, identifying which sides belong to each element.

4.2. *Identification of elements to be refined*

The aim of this substep is to determine on which *sides* further grid points need to be introduced so that the resulting refinement patterns on an element level belong to the allowed cases listed above, thus producing a compatible, valid new mesh. Five main steps are necessary to achieve this goal.

(*a*) *Mark elements that require refinement.* Using the modified error indicator given by equation (3) and the prescribed refinement tolerance CTORE, those elements that require further refinement are marked. In FORTRAN this may be achieved by having an array over the elements LELEM(1 : NELEM) which is marked as follows: LELEM(IE) = 1 ⇒element is to be refined; LELEM(IE) = 0⇒element is not to be refined.

(*b*) *Add protective layers of elements to be refined.* If protective layers of elements are to be added ahead of the feature to be refined, we perform for each additional layer the following set of operations.

(i) Zero an integer point array (e.g. LPOIN(1; NPOIN) = 0).
(ii) Loop over the elements to be refined, marking (e.g. LPOIN(IP) = 1) all points connected to these elements.
(iii) Zero the integer element array (e.g. LELEM(1; NELEM) = 0).
(iv) Loop over all elements; if at least one point of a given element has been marked, refine this element (e.g. LELEM(IE) = 1).

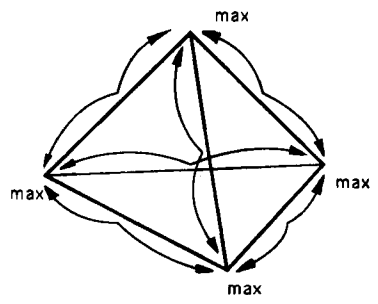
(*c*) *Avoid elements that become too small or that have been refined too often.* A sharp feature in the flow domain, e.g. a shock, will tend to produce error indicator values that always lie above the refinement tolerance CTORE. As a consequence, elements close to such a feature will be refined every time the mesh is adapted. In order to avoid this 'refinement *ad infinitum*', one has to impose either a maximum permissible number of refinement levels per element or a minimum allowable element size. Given these constraints, those elements which are already too small (if a minimum allowed element size has been given) or have already been refined too many times (if a maximum allowed number of refinement levels has been prescribed) are deleted from the list of elements to be refined.

(*d*) *Obtain preliminary list of sides for new points.* Given the side/element information obtained in substep 4.1, we can now determine a first set of sides on which new grid points need to be introduced. This set of sides is still preliminary since we only allow certain types of refinement.

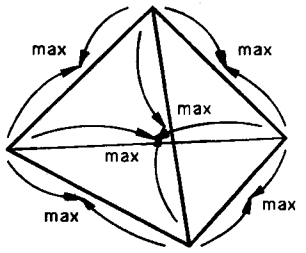
(*e*) *Add further sides to this list until an admissible refinement pattern is achieved.* The list of sides marked for the introduction of new points is still preliminary at this point. In most cases it will not lead to an admissible refinement pattern to construct a new mesh. Therefore further sides are marked for the introduction of new points until an admissible refinement pattern is reached. This is accomplished by looping several times over the elements, checking on an element level whether the set of sides marked can lead to an admissible new set of subelements. The algorithm used is based on the observation that the admissible cases are based on the introduction of new points along one side (1 : 2), three contiguous sides (1 : 4) or six contiguous sides (1 : 8). These admissible cases can be obtained from the following element-by-element algorithm (see Figure 4).

(i) Set the node array LNODE(1 : 4) = 0.
(ii) Loop over the sides of the element: if the side has been marked for the introduction of a new point, set LNODE(IP1) = 1 and LNODE(IP2) = 1, where IP1 and IP2 are the endnodes corresponding to this side.
(iii) Loop over the sides of the element: if LNODE(IP1) = 1 and LNODE(IP2) = 1, mark the side marked for the introduction of a new point.

Practical calculations with several admissible layers of refinement and large grids revealed that sometimes up to 15 passes over the mesh were required to obtain an admissible set of sides. This relatively high number of passes can occur when the mesh exhibits regions where the refinement criterion is just met by the elements. Then the list of sides originally marked for refinement will be far from an admissible one. In each pass over the mesh a further 'layer' of elements with admissible sides marked for refinement will be added. Moreover, since an element can be refined is six possible ways, in some cases it may take three passes to go from a 1 : 2 to a 1 : 8 case. Thus the 'front' of elements with an admissible set of sides marked for refinement may advance slowly, resulting in many passes over the mesh. A considerable reduction in CPU time was achieved by pre-sorting the elements as follows.



a) From Sides to Points



b) From Points to Sides

Figure 4. Algorithm to screen for admissible refinement cases

(i) Add up all the sides marked for refinement in an element.
(ii) If zero, one or six sides were marked: do not consider further.
(iii) If four or five sides were marked: mark all sides of this element to be refined.
(iv) If two or three sides were marked: analyse in depth as described above.

This then yields the final set of sides on which new grid points are introduced.

4.3. Identification of elements to be deleted

The aim of this substep is to determine which *points* are to be deleted so that the resulting coarsening patterns on an element level belong to the allowed cases listed above, thus producing a compatible, valid new mesh. Four main steps are necessary to achieve this goal.

(*a*) *Mark elements to be deleted.* As before, we start by determining—using the modified error indicator given by equation (3) and the prescribed deletion tolerance CTODE—those elements that should be coarsened. Thus we mark an element array LELEM(1 : NELEM) as follows: LELEM(IE) $= -1 \Rightarrow$ element is to be deleted; LELEM(IE) $= 0 \Rightarrow$ element is not to be deleted.

(*b*) *Filter out elements where father and all sons are to be deleted.* It is clear that only those elements should be deleted for which both the father as well as all its sons have been marked for deletion. Therefore only the parent elements to be coarsened are considered further. For these elements a check is performed whether their respective son elements have also been marked for

deletion. If any of the sons have not been marked for deletion, neither the parent element nor any of its sons are considered further.

(*c*) *Obtain preliminary list of points to be deleted.* Given the list of parent elements to be coarsened, we can now determine a preliminary list of points to be deleted. Thus all the points that would be deleted if all the elements contained in this list were coarsened are marked as 'total deletion points'.

(*d*) *Delete points from this list until an admissible coarsening pattern is achieved.* The list of total deletion points obtained in the previous step is only preliminary, since unallowed coarsening cases may appear on an element level. We therefore perform loops over the elements, deleting all those total deletion points which would result in unallowed coarsening cases for the elements adjoining them. This process is stopped when no incompatible total deletion points are left. As before, this process may be made considerably faster by grouping together and treating differently the parent elements with zero, one, two, three, four, five or six total deletion points.

4.4 Refinement of the grid where needed

The introduction of further points and elements is performed in two independent steps, which in principle could be performed in parallel.

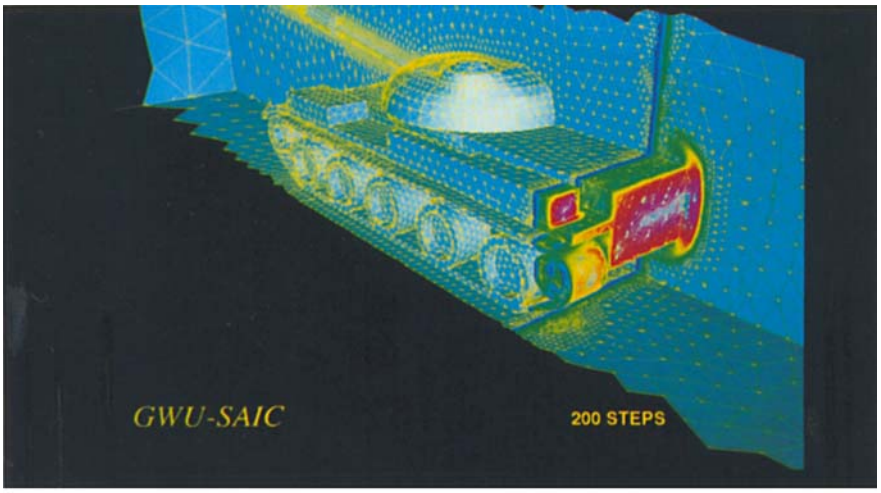
(*a*) *Points.* To add further points, the sides marked for refinement in substep 4.2 are grouped together. For each of these sides a new gridpoint will be introduced. The interpolation of the co-ordinates and unknowns is then performed using the side/point information obtained in substep 4.1. These new co-ordinates and unknowns are added to their respective arrays. In the same way new boundary conditions are introduced where required, and the location of new boundary points is adjusted using the CADCAM data defining the computational domain.

(*b*) *Elements.* In order to add further elements, the sides marked for refinement are labelled with their new grid point number. Thereafter the element/side information obtained in substep 4.1 above is employed to add the new elements. The elements to be refined are grouped together according to the refinement cases shown in Figure 1. Each case is treated in block fashion in a separate subroutine. Perhaps the major breakthrough of the present work was the reduction of the many possible refinement cases to only six. In order to accomplish this, some information for the 2:8+ and 4:8+ cases is stored ahead in scratch arrays. After these elements have been refined according to the 2:8 and 4:8 cases, their sons are screened for further refinement using this information. All sons that require further refinement are then grouped together as 1:2 or 1:4 cases and processed in turn.

Since the original description of all variables was performed using linear elements, the linear interpolation of the unknowns to the new points will be conservative. However, small conservation losses will occur at curved surfaces. We consider these losses to be both unavoidable and small.

4.5. Coarsening of the grid where needed

The deletion of points and elements is again performed in two independent steps, which in principle could be performed in parallel.



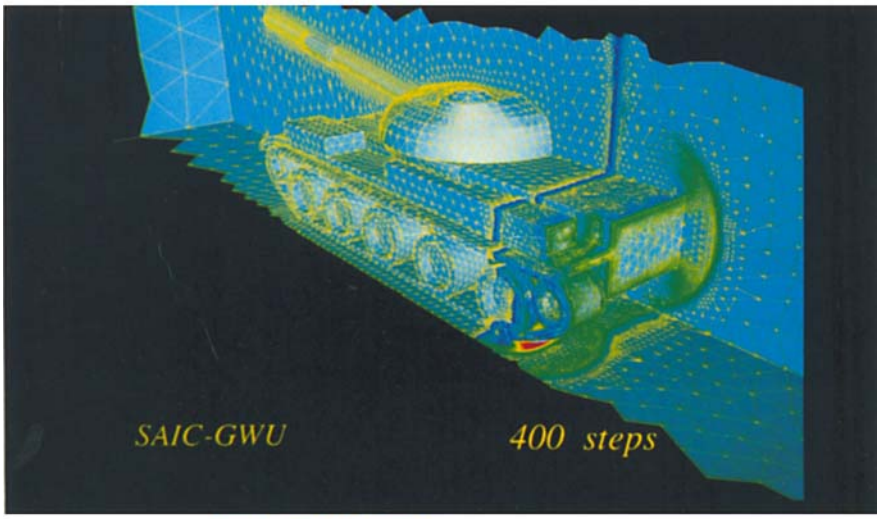
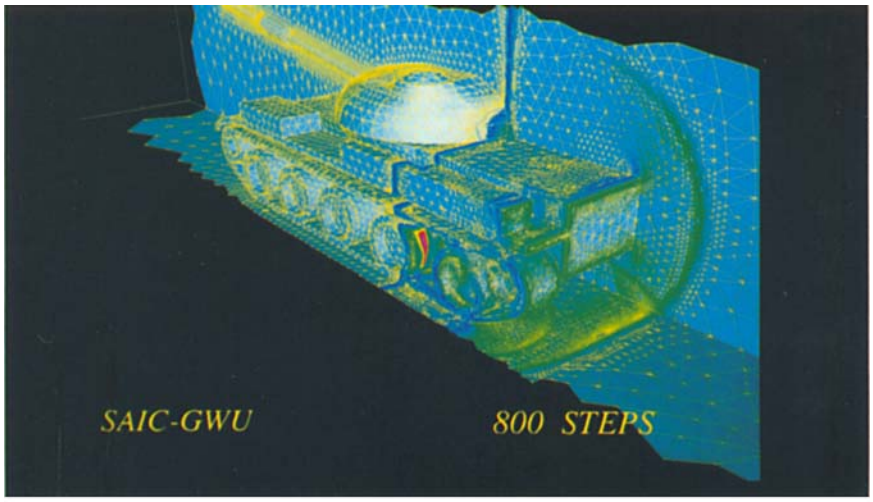
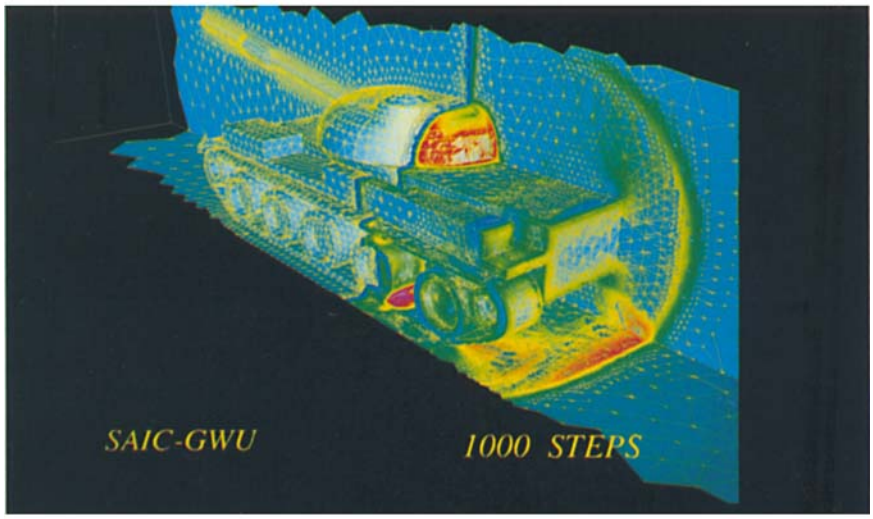


Plate 1 (a – c)





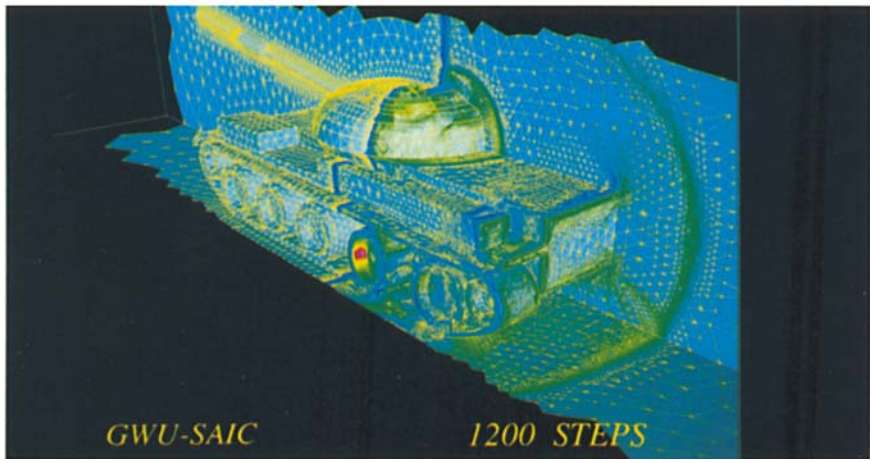


Plate 1 (d – f)

(*a*) *Points.* The points to be deleted having been marked in substep 4.3 above, all that remains to be done is to fill up the voids in the co-ordinate, unknown and boundary condition arrays by renumbering points and boundary conditions.

(*b*) *Elements.* The deletion of elements is again performed blockwise by grouping together all elements corresponding to the coarsening cases shown in Figure 3. Thereafter the elements are also renumbered (in order to fill up the gaps left by the deleted elements) and the point renumbering is taken into consideration within the connectivity arrays.

It is clear that the coarsening procedure is non-conservative. However, we have never observed any physical or numerical problems using it. This may be explained by the fact that the coarsening is done in those regions where the solution is smooth. Thus the coarsened grid represents the solution very well and consequently the conservation losses are small. Moreover, those regions where the maintenance of conservation is important (e.g. discontinuities) are never affected.

5. NUMERICAL EXAMPLE

We demonstrate the performance of the method on a typical production run. The example simulates shock impact on a generic tank. Although some comparison with experimental data has been done,[5] the aim here is to demonstrate the developed adaptive refinement/coarsening algorithm. As the basic hydrodynamics solver we employ the FEM-FCT code of Löhner *et al.*,[2,3] which is capable of reproducing moving and stationary shocks over two elements without loss of monotonicity. For this class of problems and the algorithm employed it was found that the following choice of refinement/coarsening parameters produced acceptable results:

(i) refinement tolerance: CTORE = 0·15
(ii) coarsening tolerance: CTODE = 0·07
(iii) noise parameter: $\varepsilon = 0{\cdot}30$
(iv) key variable: density.

Other relevant parameters that were found useful for production runs such as the one shown include:

(i) refinement frequency: every seven time steps
(ii) number of protective layers: none
(iii) Courant number of the hydro-solver: $C = 0{\cdot}4$.

We have found these parameters to be useful not only for the present run but for a whole class of similar runs involving shock–object interactions. Thus they seem to be rather general in their applicability. A similar behaviour was observed previously in 2D.[4-6]

5.1. Shock impact on a generic tank

The problem statement as well as the solutions obtained are shown in Plate 1. A very strong shock impacts the rear of the generic main battlefield tank. A maximum of two layers of refinement were specified close to the tank, whereas only one level of refinement was employed further away. The original, unrefined, but strongly graded mesh consisted of approximately NELEM = 100,000 elements and NPOIN = 20,000 points. During the run the mesh size increased to approximately NELEM = 1,600,000 elements and NPOIN = 280,000 points. This represents an increase factor of 1:16. Although seemingly high, the corresponding global *h*-

refinement would have resulted in a 1 : 64 size increase. A second important factor is that most of the elements of the original mesh are close to the body, where most of the refinement is going to take place. Plates 1(a)–1(f) show surface gridding and pressure contours at selected times during the run. The extent of mesh refinement is clearly discernible, as well as the location and interaction of shocks.

A complete run such as the one shown here takes approximately 35 h of single-processor CRAY-YMP time and 110 Mwords of memory. Given that it takes 3–5 days to grid up a configuration such as the one shown and another 3–5 days to plot and understand the data, these CPU and memory requirements are not deemed excessive. Moreover, given the DO loop lengths encountered in a run like the one shown, microtasking works very well. On an eight-processor CRAY-YMP, speed-ups in excess of 1 : 6 have been obtained. This reduces run times to 6 h, i.e. an overnight run. From these figures it also becomes apparent that without the adaptive mesh refinement/coarsening scheme developed, a run like this would be impossible on today's hardware.

6. CONCLUSIONS

An adaptive finite element scheme for transient problems has been presented. The classic *h*-enrichment/coarsening is employed in conjunction with a tetrahedral finite element discretization in three dimensions. The grid is adapted every *n* time steps, depending on the Courant number employed and the number of 'protective layers' added ahead of the refined region. Particular emphasis was placed on speed and low storage requirements from the outset. Therefore only one level of refinement/coarsening was allowed per mesh change and the number of possible refinement/coarsening patterns was reduced. This avoided badly deformed elements and simplified the grid logic considerably without loss of generality. It has been demonstrated that with these restrictions in mind a high degree of vectorizability can be achieved on modern supercomputers.

A numerical example taken from practical shock–shock and shock–structure interaction problems indicates that with the present approach, saving factors in both CPU and storage requirements of more than an order of magnitude as compared to uniform refinement are attainable without deteriorating the accuracy of the solutions.

ACKNOWLEDGEMENTS

This work was partially funded by the Defense Nuclear Agency and the Air Force Ballistic Missile Office through SAIC.

The generous support of CRAY Research, Inc., and in particular of Steven Perry, in the form of ample CPU time on a CRAY-2S during the debugging stages of the code, is also gratefully acknowledged.

REFERENCES

1. R. Löhner, K. Morgan, J. Peraire and O. C. Zienkiewicz, 'Finite element methods for high speed flows', *AIAA-85-1531-CP*, 1985.
2. R. Löhner, K. Morgan, J. Peraire and M. Vahdati, 'Finite element flux-corrected transport (FEM-FCT) for the Euler and Navier–Stokes equations', *ICASE Rep. 87-4*, 1987; *Int. j. numer. methods fluids*, **7**, 1093–1109 (1987).
3. R. Löhner, K. Morgan, M. Vahdati, J. P. Boris and D. L. Book, 'FEM-FCT: combining unstructured grids with high resolution'; *Commun. Appl. Numer. Methods*, **4**, 717–730 (1988).
4. R. Löhner, 'An adaptive finite element scheme for transient problems in CFD', *Comput. Methods Appl. Mech. Eng.*, **61**, 323–338 (1987).

5. J. D. Baum and R. Löhner, 'Numerical simulation of shock-elevated box interaction using an adaptive finite element shock capturing scheme', *AIAA-89-0653*, 1989.
6. J. D. Baum, E. Loth and R. Löhner, 'Numerical simulation of shock interaction with complex geometry canisters', in Y. Kim (ed.), *Current Topics in Shock Waves*, American Institute of Physics, New York, 1989.
7. J. T. Oden, P. Devloo and T. Strouboulis, 'Adaptive finite element methods for the analysis of inviscid compressible flow: I. Fast refinement/unrefinement and moving mesh methods for unstructured meshes', *Comput. Methods Appl. Mech. Eng.*, **59**, 327–362 (1986).
8. M. J. Berger and J. Oliger, 'Adaptive mesh refinement for hyperbolic partial differential equations', *J. Comput. Phys.*, **53**, 484–512 (1984).
9. J. Bell, P. Colella, J. Trangenstein and M. Welcome, 'Adaptive mesh refinement on moving quadrilateral grids', *AIAA-89-1979-CP*, 1989.
10. M. M. Pervaiz and J. R. Baron, 'Spatio-temporal adaptation algorithm for two-dimensional reacting flows', *AIAA-88-0510*, 1988.
11. J. Peraire, M. Vahdati, K. Morgan and O. C. Zienkiewicz, 'Adaptive remeshing for compressible flow computations', *J. Comput. Phys.*, **72**, 449–466 (1987).
12. J. Peraire, J. Peiro, L. Formaggia, K. Morgan and O. C. Zienkiewicz, 'Finite element Euler computations in three dimensions', *AIAA-88-0032*, 1988.
13. R. Löhner, 'Adaptive remeshing for transient problems', *Comput. Methods Appl. Mech. Eng.*, **75**, 195–214 (1989).
14. R. Löhner, 'Three-dimensional fluid–structure interaction using a finite element solver and adaptive remeshing', *Comput. Syst. Eng.*, **1**, 257–272 (1990).