AIAA JOURNAL
Vol. 32, No. 8, August 1994

Three-Dimensional Unstructured Adaptive Multigrid Scheme for the Euler Equations

Stuart D. Connell* and D. Graham Holmes*
General Electric Research and Development Center, Schenectady, New York 12301

This paper presents a solution adaptive multigrid scheme for the three-dimensional Euler equations. The procedure begins with an initial coarse mesh and enriches this mesh by h-refinement in regions of high solution gradient. This adaptive procedure is used to generate the multigrid levels required by the solver. New boundary nodes are added in a way to reflect the underlying surface representation of the geometry. The h-refinement procedure is extremely fast and results in a scheme where only a small percentage of the overall CPU time is spent in the meshing and refinement part of the algorithm. The multigrid flow solver typically reduces the CPU time by an order of magnitude over the same case without multigrid. Initial mesh generation is achieved through the destructuring of a structured mesh or use of the advancing front method. The algorithm is demonstrated on a range of cases.

I. Introduction

THE use of an unstructured mesh for solving the Navier-Stokes and Euler equations in two- and three-dimensional geometries has proven very successful.[1-5] The reasons for employing an unstructured mesh approach to fluid flow problems are twofold: 1) complex geometries can be meshed efficiently and with little user intervention and 2) solution adaption is straightforward. The adaption can be performed either through adaptively remeshing the geometry where the meshing parameters are based on the current solution[4] or through reinforcing the mesh by locally adding new nodes (h-refinement).[5] The latter approach is in general many times faster and more robust than adaptive remeshing.

The multigrid solution acceleration scheme has been widely used for structured meshes[6] and has also proven successful for unstructured meshes.[2,7] With these techniques the multigrid levels are obtained through repeatedly remeshing the entire geometry to obtain successively finer meshes.

Multigrid schemes based on adaptive remeshing have three disadvantages. The first is that the mesh generation algorithm needs to be robust and fast. If the scheme is not fast, the mesh generation time can quickly exceed the solution time, and a lot of the benefit of multigrid is lost. The second disadvantage relates to the difficulty in computing transfer operators for the unstructured mesh. As the meshes bear no relation to each other, this is a nontrivial task. Complex coding is required to avoid an $\mathcal{O}(n^2)$ search. Finally, unless the meshes are generated through solution adaption then a large percentage of the nodes on the finer meshes may serve no useful purpose.

The multigrid scheme presented in this paper overcomes the cited disadvantages and is similar to the two-dimensional scheme described in Ref. 8. The process begins with an initial coarse grid. An initial solution is obtained on this grid. This solution is then examined to determine if the grid is sufficiently fine to resolve features of interest. In regions where more resolution is required the mesh is enhanced through h-refinement. This h-refinement is described in Sec. III. A solution is obtained on this new grid. The solve and refine process is repeated until some desired level of accuracy is achieved. When using this approach the multigrid levels are formed naturally by the refinement procedure. The construction of the transfer operators is a simple task requiring a single pass through the data. Full details of the multigrid scheme are presented in Sec. VII.

The h-refinement procedure is orders of magnitude faster and more robust than remeshing. Typically, the h-refinement procedure generates tetrahedra at the rate of 12,000 per CPU second on an HP720 workstation using the current scheme. No attempt has been made to optimize the algorithm. In contrast the widely used advancing front grid generation method[13] produces tetrahedra at the rate of 33 per CPU second on the same workstation. These CPU times are for a mesh of approximately 100,000 tetrahedra. It is also worth noting that the CPU times will be of $\mathcal{O}(n)$ for h-refinement and at best $\mathcal{O}[n \log(n)]$ for the advancing front scheme.

Results for a range of test cases are presented in Sec. VIII demonstrating a reduction in CPU time by up to a factor of 10 over the same case without multigrid. It is shown that the CPU time employed by the grid generator and refinement algorithm are small in comparison with the overall CPU time.

II. Data Structures

Data structures are used to define the connectivity between various geometrical entities in the computational grid. Four distinct entities are considered: cells, faces, edges and nodes. A cell is the tetrahedral element formed by four nodes. A cell has four triangular faces each of which has three edges. Each edge is associated with the two nodes located at the ends of the edge.

Unfortunately the data structures required by the refinement algorithm, flow solver, and postprocessor differ significantly. For example, in the flow solver an edge-to-node pointer is used. In the postprocessor a cell-to-node pointer is needed. In the refinement algorithm a cell-to-face and a face-to-edge pointer is required. This connectivity is needed because the decision to refine a cell or face is based on the status of their component entities.

Obviously, it would be extremely inefficient to store all possible connectivities. Instead the following hierarchical structure is adopted:

cells	point to	4 faces
faces	point to	3 edges
edges	point to	2 nodes

If the connectivity between cells and nodes is required (for the postprocessor) then the data structure is traversed down through the hierarchy first to faces, then to edges, and finally to nodes. By adopting these hierarchical data structures it is a simple matter to derive the connectivity between any two entities in the grid; all that is required is a single pass through the data structures.

Received June 14, 1993; presented as Paper 93-3339 at the AIAA 11th Computational Fluid Dynamics Conference, Orlando, FL, July 6–9, 1993; revision received March 28, 1994; accepted for publication March 30, 1994.

*Mechanical Engineer. Member AIAA.

III. Adaptive Refinement

Refinement is invoked after an error measure has fallen below some threshold. The two error measures used are the nodal values of the maximum and average change in velocity.

An edge is tagged for refinement if the normalized first difference of the static pressure, total pressure, or velocity between the nodes at the ends of the edge exceeds some user defined threshold. The threshold values are typically between 1 and 15%. To prevent endless refinement at shocks a lower limit on edge length is set. Any edge whose length falls below this threshold cannot be refined further.

Before describing the three-dimensional refinement algorithm it is instructive to consider the process in two dimensions.

A. Two-Dimensional Refinement

The refinement process begins with a computational grid and a list of edges tagged for refinement. For a triangular face there are three possibilities: one, two, or three edges may be tagged. Figure 1 shows how the face is divided according to how many edges are tagged. The new nodes are inserted at the midpoint of the edges which are refined. In Fig. 1a the face is said to be partially refined. In Figs. 1b and 1c the face is said to be fully refined. It should be noted that when two edges are refined then refinement of the third is forced. This is done to reduce the number of possible face divisions to two. This choice suggests the first rule of refinement.

Rule 1: If two edges of a face have been refined then refine the third.

This rule significantly reduces the number of possible cell divisions in three dimensions.

If only one edge is tagged, then the face is split into two as seen in Fig. 2a. Therefore, it is possible for successive refinement steps to increase the aspect ratio of the new faces as illustrated in Figs. 2b and 2c. Further refinement of these faces can lead to a chaotic irregular mesh. For many numerical schemes this chaotic mesh will have an undesirable influence on solution quality. To avoid this property, where the solution quality would diminish as refinement progresses, the first split of the face (Fig. 2a) is considered temporary and is only made to provide a valid grid for use by the solver. It is not made by the refinement algorithm. The refinement algorithm is now modified so as to force the full refinement of a face if any edge has been refined twice. The effect of this modification on Fig. 2 is illustrated in Fig. 3.

To implement this change, the concept of parents and children is introduced. An edge on refinement has two children; if either of these child edges are refined, then the original edge has grandchildren. This idea of parents and children leads to the second rule of refinement which may be stated as follows.

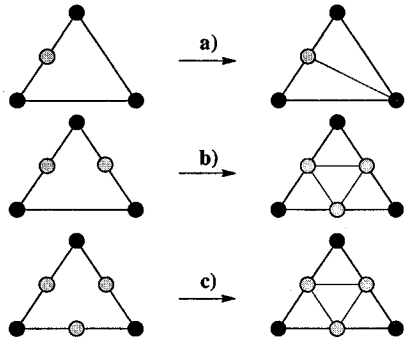

Fig. 1 Possible face divisions.

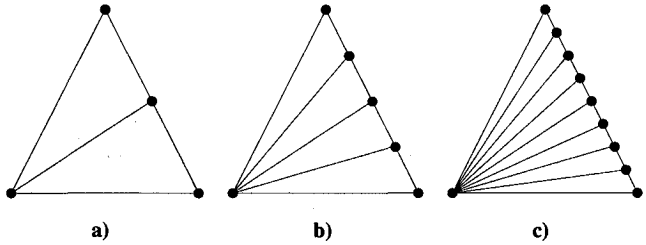

Fig. 2 Possible face refinements.

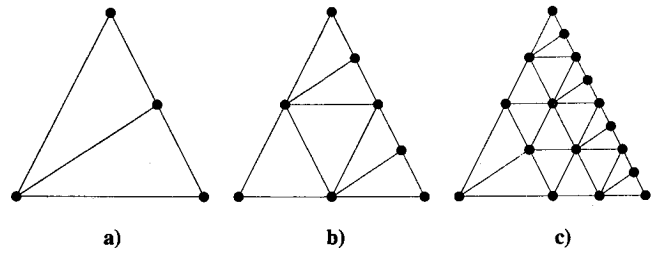

Fig. 3 Modified refinement scheme.

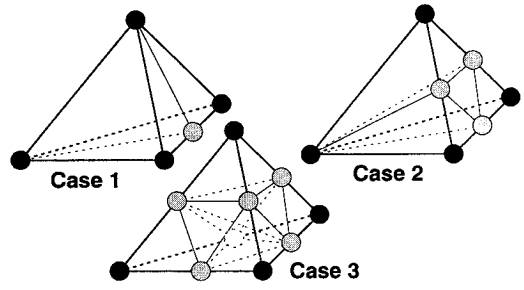

Fig. 4 Three possible tetrahedral refinement cases.

Rule 2: If any edge of a face has grandchildren then refine all edges of the face.

In a typical grid the result of applying the refinement rules will be to refine edges and faces that were not initially tagged for refinement. To ensure that all of the desired refinement takes place, repeated loops through the data structures are made applying the rules until no further change takes place.

B. Three-Dimensional Refinement

In three dimensions the refinement strategy remains the same. The procedure begins with a computational grid of tetrahedra and a list of edges tagged for refinement. For each tetrahedral cell between one and six edges may be tagged leading to many possible ways of dividing the cell. This number can be reduced to manageable levels by the repeated application of rule 1 to each face of the cell. The number of possible cell splits is then reduced to the three cases illustrated in Fig. 4. In case 1 only one edge has been refined, and a new node added at the center of the edge. The cell is split into two. In case 2 one face has been fully refined, and the cell is split into four. Finally, in case 3 all faces of the cell have been fully refined, and the cell is split into eight new cells. It is arbitrary how to place the diagonal at the center of the refined cell in case 3, so as to maintain the highest quality grid the shortest of the three possibilities is chosen. References 9 and 10 use identical cell splits but obtain them by different methods.

The cell aspect ratio may be defined to be the ratio of the circumsphere radius defined by the nodes of the cell to the inner sphere radius. The inner sphere is the largest sphere which will fit inside the tetrahedron. This ratio is normalized so that an equilateral tetrahedron has an aspect ratio of unity.

With the case 3 split, the four inner children cells will have a different and possibly worse aspect ratio than their parent cell. The other children cells will have an aspect ratio identical to their parent. Initially, it may appear that successive refinement could cause the aspect ratios to deteriorate. However, numerical experimentation with a variety of tetrahedra leads to the conclusion that after any number of refinements the aspect ratios of the tetrahedra do not deteriorate and get no worse than the worst aspect ratio of the eight tetrahedra after the first refinement.

There is one case which will not be removed by repeated application of rule 1. In this case two opposite edges have been tagged. To reduce this case to one of the three allowable splits shown in Fig. 4, full refinement of the cell (case 3) is forced. The removal of this case becomes refinement rule 3.

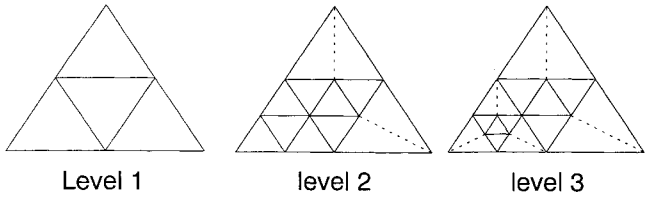

Fig. 5 Multigrid levels using clipping.

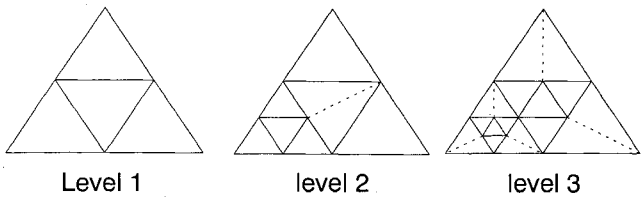

Fig. 6 Multigrid levels using push down.

Rule 3: If two opposite edges of a cell have been refined then refine the cell.

As with two-dimensional face refinement, it is undesirable to have high aspect ratio cells appearing in the grid as a result of the refinement process. To prevent this occurring a similar procedure is adopted to that used in two dimensions. The case 1 and case 2 splits shown in Fig. 4 are regarded as being temporary and are only made prior to producing a valid grid for use by the solver. The associated cells and faces from these temporary splits do not exist in the structures used by the refinement algorithm. Again the idea of parents and children is introduced. If the refined edge in case 1 was to be refined again, then initially it may appear that high aspect ratio cells would result. However, application of rule 2 will force the refinement of the two faces adjacent to this edge and the subsequent application of rule 1 will fully refine the cell. Considering case 2, if any child edge of the refined face is itself tagged for refinement, then high aspect ratio cells will result. Refinement rule 2 will prevent the occurrence of high aspect ratio cells if any edge of the parent face has grandchildren. However, if any of the edges surrounding the central child face are refined none of the current refinement rules will force full refinement of the cell. To ensure full refinement of the cell the final rule may be written as follows.

Rule 4: If any face of the cell has a child face that points to a refined edge then fully refine the cell.

It will be noted that due to the hierarchical data structures employed it is straightforward to apply the four refinement rules.

As the refinement of the grid takes place, new entities are created. These are added to the existing data structures described in Sec. II. When the refinement process is completed the data structures contain the original grid together with all of the new entities. This approach of keeping all of the meshes is crucial for the efficient implementation of multigrid described in Sec. VII and derefinement described in Sec. V.

IV. Surface Representation

If the boundaries of the computational domain are nonplanar, it is not sufficient to place the new nodes at the center of the parent edge as is done with all other new nodes. If this is done, a faceted representation of the original geometry will result. What must be done instead is to move the node from this approximate location onto the boundary.

Three-dimensional solid modelers typically represent an object as a series of geometric surfaces bounded by geometric edges. These geometric entities have an associated integer identifier. It is assumed that both nodes of a boundary edge of the mesh lie in the same geometric edge or surface. In addition, it is assumed that the geometric edges and surfaces may be represented in parametric form.

To make the snapping procedure efficient the parametric locations of the nodes at the ends of the edge in the geometric surface and/or edge are stored together with the surface and/or edge identifier. It is then a simple matter to average these parametric coordinates and interrogate the surface modeler to obtain the physical location of the new node.

V. Derefinement

The ability to derefine a computational grid can be advantageous for a variety of reasons. Consider a case where a solution feature appears initially in an incorrect location due to truncation error caused by an insufficiently fine grid. The adaption process will immediately begin to place nodes in that location. However, as the solution develops this feature will move to its correct location, and the initial points serve no useful purpose and may be removed from the grid. Another use for a derefinement procedure is to use it to generate the multigrid levels required by the flow solver. This procedure is described in Sec. VI.B.

The process of derefinement begins by tagging entities as candidates for removal (derefinement) from the grid. An edge is tagged for derefinement if the normalized first differences (described in Sec. III) fall significantly below the refinement threshold, typically if the difference is less than 25% of the threshold. Refinement tags always override derefinement tags. This tagging can be done concurrently with the refinement tagging. The next step is to tag entities for deletion from the grid. A cell or face is tagged as deletable if all of its siblings are tagged for derefinement. The parent entity is then said to be provisionally refined.

The refinement process described in Sec. III now takes place. During this procedure the conflicting situation may arise where a refinement rule would force refinement of a provisionally refined entity. Refinement is deemed to take priority over derefinement, and the provisionally refined tag on the parent entity and the deletable tags on the children entities are removed. The final step is to purge all cells (together with their edges faces and nodes) that remain tagged as deletable from the grid.

VI. Multilevel Data Structures

During the multigrid procedure it is required that time steps be performed on a variety of meshes of differing resolution. This section deals with the extraction of the various multigrid levels from the data structure. The movement of the residuals and solution between the meshes is described in Sec. VII.

Obviously, the coarsest mesh is the starting mesh prior to any refinement. The finest mesh can be quickly derived by only considering those entities with no children after any temporary splits have been made. The question of how the intermediate levels are derived is now addressed. Two possible algorithms are considered: clipping and push down.

A. Clipping

With this approach every entity is assigned a level. Cells are assigned levels as follows: cells on the initial grid are at level 1; their children are at level 2; their grandchildren are at level 3, and so on.

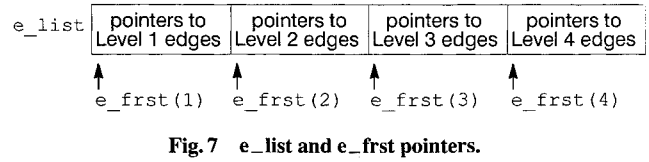

Fig. 7 e_list and e_frst pointers.

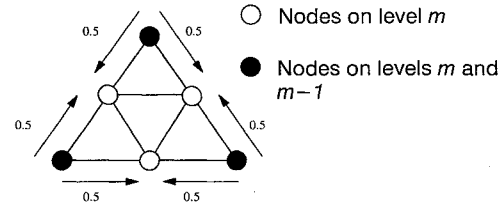

Fig. 8 Interpolation of the changes from level $m-1$ to level m.

Faces and edges are assigned the level of the lowest level of the entity that points to it. Note that the finest mesh may contain entities with levels between 1 and m, where m is the highest level entity in the mesh.

To compute the various multigrid levels the algorithm begins with the finest mesh. To compute the next coarser mesh all entities with level m are removed. This clipping procedure is repeated until the initial mesh is recovered. Figure 5 illustrates the various multigrid levels for a simple two-dimensional mesh.

B. Push Down

An alternative procedure for generating the multigrid levels uses the derefinement algorithm described in Sec. V. Again the algorithm begins with the finest level mesh. Every entity is then tagged for derefinement, and the derefinement algorithm called. This method will coarsen the mesh as quickly as possible. Figure 6 illustrates the push-down algorithm on the two-dimensional case shown in Fig. 5. Note the change in multigrid level 2. The advantage of using this method over clipping is that the lower multigrid levels will have significantly fewer nodes and, hence, the time stepping will require less CPU time. Thus, the multigrid scheme on a push-down mesh will require less CPU time than a clipping mesh for the same level of convergence. This will be demonstrated in Sec. VIII.

C. Data Structures

Having decided which entities comprise a given level, a suitable data structure is required to store this information. A pointer is employed. The pseudocode to obtain the edges ie and their nodes j1, j2 of level m would look like

```
do iep = e_frst (m), e_frst (m + 1) 2 1
    ie = e_list (iep)
    j1 = ntabe(1, ie)
    j2 = ntabe(2, ie)
end do
```

where ntabe is the edge-to-node pointer. The structure of the pointers e_list and e_frst is illustrated in Fig. 7 for a grid with four levels. Note that the same edge may appear at different levels. A similar structure is employed for cells and faces.

Using this pointer structure, there is no duplication of information. For example, if a cell exists at all multigrid levels, then the data for that cell is stored only once.

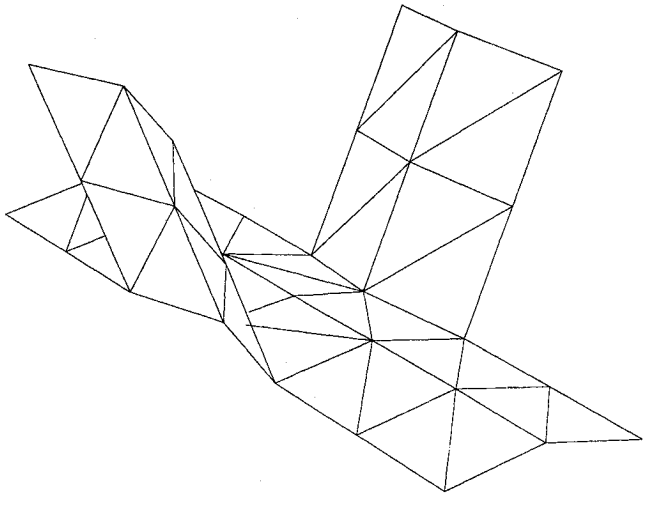

Fig. 9 Initial mesh for biconvex case.

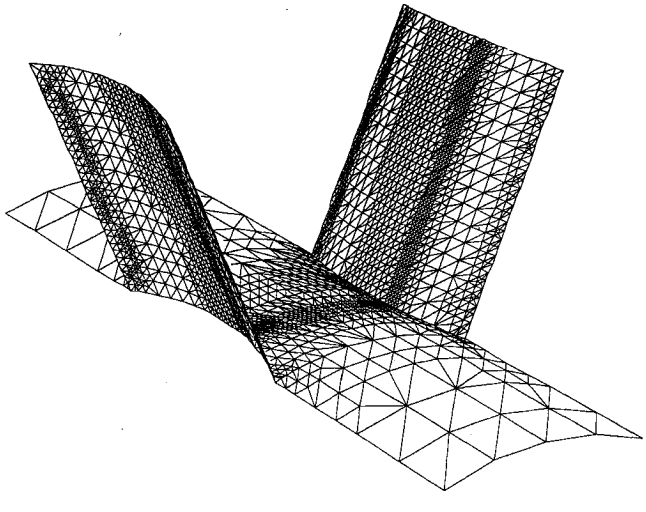

Fig. 10 Final mesh for biconvex case.

Table 1 Comparison of mesh sizes for push down and clipping

Level	Push down	Clipping
1	63	63
2	179	325
3	517	1,287
4	1,535	4,650
5	4,431	12,061
6	19,414	19,414

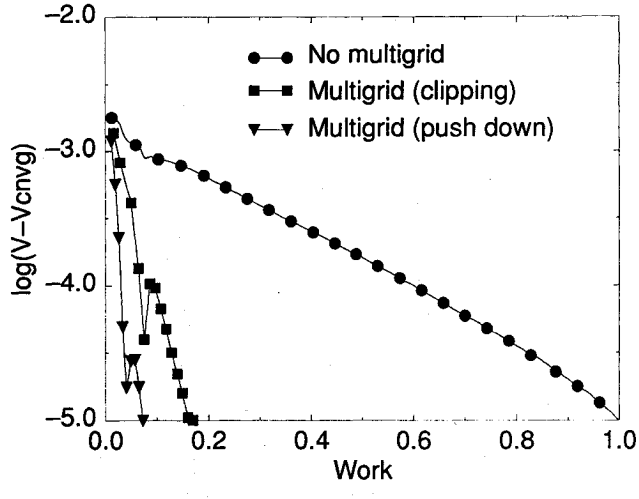

Fig. 11 Convergence rates for biconvex case.

VII. Multigrid Flow Solver

The multigrid flow solver uses a Runge-Kutta time marching procedure with an adaptive blend of second- and fourth-order smoothing similar to that of Ref. 1. The solution variables are stored at the vertices of the mesh. Residual averaging and enthalpy damping were not used.

Usually, a face based scheme is used for the residual calculation.[1,3] With this scheme the control volumes surrounding each node are comprised of all of the tetrahedra containing that node. These control volumes overlap. The residual for the control volume is then evaluated by computing the flux through each face in the mesh and accumulating this flux to the two nodes on either side of the face. An alternative and more computationally efficient scheme is one based on edges.[2] Here, an area is associated with each edge and the flux computed through this area is accumulated to the nodes at either end of the edge. It can be shown that this scheme yields an identical flux balance to the face-based scheme if suitable modifications are made near boundaries. The smoothing residuals and time steps may also be computed using this edge-based data structure. It is this edge-based scheme which is adopted in the current work.

The multigrid operations that are needed are restriction of the net fluxes and interpolation of the changes. Interpolation of the

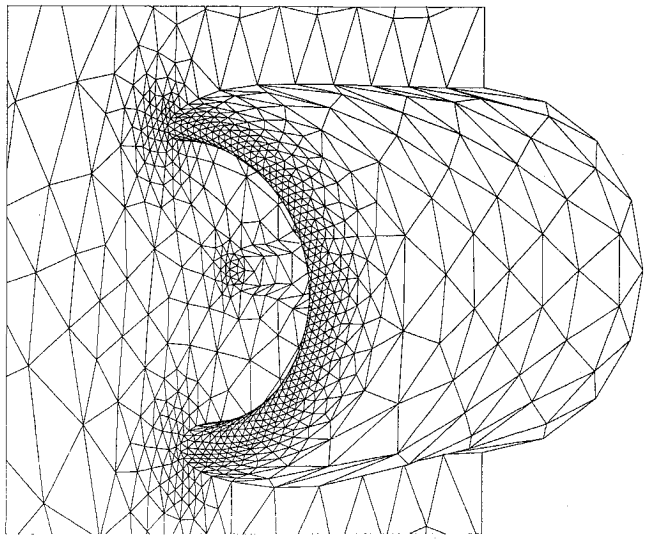

Fig. 12 Initial inlet mesh.

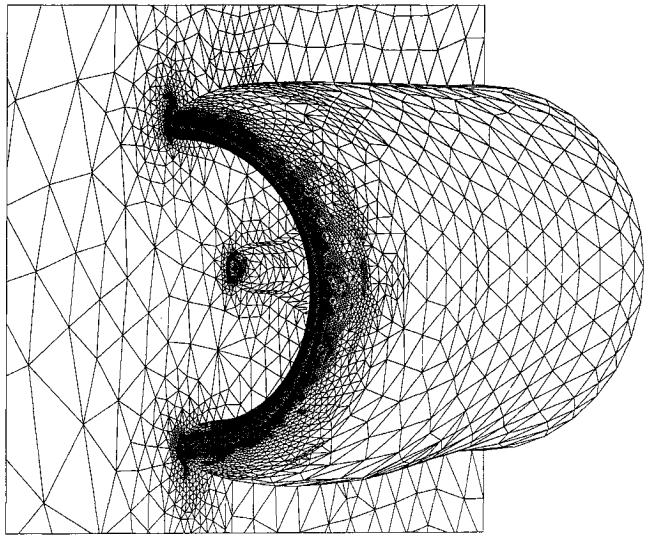

Fig. 13 Final inlet mesh.

changes is simple. Sets of parent/child edges are identified in the data structure for which the parent is on grid $m-1$, and the two children are on grid m. Changes are then interpolated from the control volumes on level $m-1$ to control volumes on level m as shown in Fig. 8. No interpolation coefficients need be stored, and only simple integer pointer tables, easily derived from the data structure, are required.

The algorithm for restricting the net fluxes from grid m to grid $m-1$ should have the following properties:

1) If the fluxes on grid m are zero, the restricted fluxes on grid $m-1$ should be zero.

2) The restriction should be conservative.

3) If the solution is sufficiently smooth, the fluxes restricted from grid m to grid $m-1$ should be approximately equal to the fluxes computed on grid $m-1$.

Properties 1 and 2 are achieved if the scheme used for interpolating the changes is employed. All that is required is to reverse the direction of the arrows in Fig. 8 and to distribute the fluxes from the control volumes on level m to the control volumes on level $m-1$. It is also necessary to send the flux from the control volume on level m to the control volume on level $m-1$ that shares the same node, with a weight of unity. Property 3 is not always achieved. It is satisfied if the grid at level m is a full refinement of the grid at level $m-1$. The price that is paid for failing to achieve property 3 on a general grid is unknown.

A simple V cycle is used. Single time steps are performed both on the way down from fine to coarse grids, and on the way back up, as the coarse grid changes are interpolated. A three-step scheme (coefficients 0.6, 0.6, and 1) is used, rather than a four- or five-step scheme, because of the higher temporal damping of the three-step scheme. Since a vertex scheme is used, in which the boundary conditions partly override the results of the flux calculations at the half-control volumes on the boundary, care has to be taken that the forcing functions are computed on the boundary in such a way that the multigrid solution converges to within round-off to the solution obtained on the fine grid alone. The overall storage requirements for the solver is approximately 100 words per node.

VIII. Results

In this section results from two different test cases are presented. The first is a simple annular cascade of biconvex airfoils and serves to demonstrate the multigrid procedure. The second is the more complex case of flow around an engine inlet and demonstrates the applicability of the algorithm to complex three-dimensional geometries.

A. Biconvex Cascade

The first case is a annular cascade of constant section biconvex airfoils. For an inlet Mach number of 0.6 the flow is transonic with a shock running from hub to tip on the blade surface. Periodic boundary conditions are applied upstream and downstream of the blade. Surface representation was achieved through the use of biquadratic patches.

The initial mesh was generated by destructuring an existing structured mesh. The coarsest level was $3 \times 3 \times 7$. This coarse mesh was chosen to provide the maximum possible benefit from the multigrid algorithm. This mesh was then globally refined twice to a mesh containing 2025 nodes. The solve and refine cycle was then begun on this mesh. Three refinements were made to give a final mesh of 19,414 nodes (102,526 tetrahedra) which has six multigrid levels. Figures 9 and 10 show the initial and final meshes

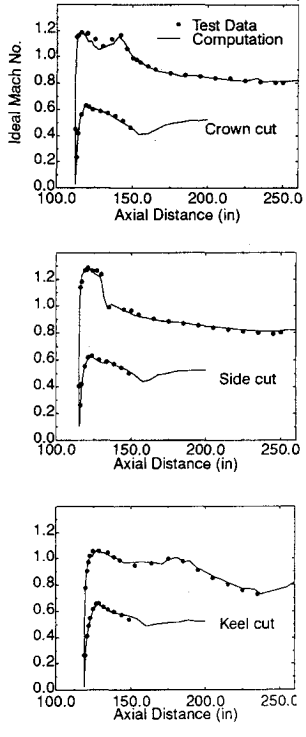

Fig. 14 Comparison of surface Mach number with test data at cruise $\alpha = 4.0$.

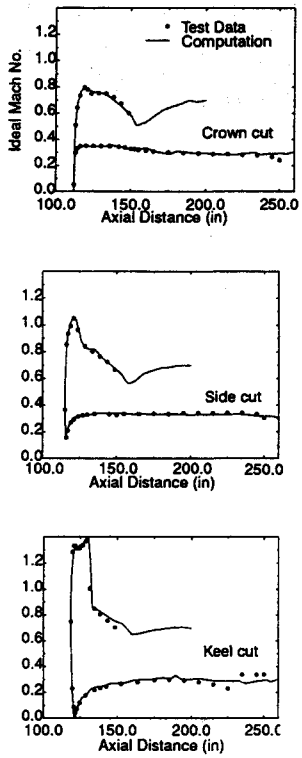

Fig. 15 Comparison of surface Mach number with test data at takeoff $\alpha = 20.0$.

for this case. It can be clearly seen that refinement has taken place in the region of the shock.

To demonstrate the effectiveness of the multigrid algorithm the flow solver was run on the final mesh with and without multigrid. When using multigrid the effect of using the push-down or clipping algorithm to generate the levels was also investigated. Table 1 illustrates the differing mesh sizes at the various levels using these two techniques. As expected the push-down procedure coarsens the mesh more rapidly and results in a scheme requiring the least CPU time as shown in Fig. 11. The push-down and clipping schemes reduce the CPU time by factors of 13.7 and 5.9, respectively, over the same case without multigrid.

For push down or clipping approximately 100 multigrid cycles are required for convergence. This is because the convergence rate in both cases is governed by that on the coarsest grid. The difference in CPU time arises from the additional work required on the coarser levels for clipping.

The mesh refinement part of the algorithm consumes less than 2% of the flow solver CPU time of 1020 s (HP720). An estimate may be made of the grid generation time if the advancing front algorithm was used to generate each multigrid level. Experience with the advancing front algorithm indicates a tetrahedral generation rate of 33 per CPU second. If this is assumed for all of the meshes, then the mesh generation time is estimated to be almost four times the current flow solver CPU time! The meshes have 144, 636, 2,085, 6,868, 21,251, and 102,526 tetrahedra.

B. Isolated Engine Inlet

The next case is an isolated inlet with a center body. This case was run at two angles of attack representing takeoff and cruise conditions. The initial mesh for this case was generated by the advancing front algorithm[13] and is shown in Fig. 12. This mesh contained 3536 nodes. The mesh after three refinement cycles for the cruise condition is shown in Fig. 13. This mesh contained 61,002 nodes. The surface modeler, which was supplied with the mesh generator, was used to correctly place the new boundary nodes.

Through testing the advancing front algorithm[13] on a variety of geometries it has been found that it has a tendency to produce a few sliver tetrahedra.[11] During the refinement process these flat cells cause no problem in the interior of the mesh. However, for cells with two or more boundary nodes it is possible for the children cells to have a negative volume due to the node snapping procedure. These sliver cells were purged from the mesh and some reconnecting done in their vicinity to produce a valid mesh. For the initial mesh in Fig. 12, 14 cells were removed.

Figures 14 and 15 show the comparison of ideal Mach number with experimental data for three cuts through the geometry for the two different angles of attack. The agreement with test data is excellent with the shocks on the side cut ($\alpha = 4.0$) and keel cut ($\alpha = 20.0$) well resolved by the refinement procedure. The push-down procedure was used to generate the multigrid levels.

To ensure that a relatively mesh independent solution had been achieved further mesh refinement was allowed. No significant change in the numerical solution were observed.

Total solver CPU time for this case was 7300 s (HP720) with the initial mesh generation taking 210 s and the refinement process 83 s. Again, an estimate may be made of the grid generation time if the advancing front algorithm was used to generate each multigrid level. This estimate leads to a mesh generation time of 20,000 s. The meshes have 17,166, 84,432, 257,431, and 334,088 tetrahedra.

IX. Conclusions

A multigrid numerical scheme for solving the Euler equations has been presented. The multigrid levels evolve naturally from the solution adaptive h-refinement procedure used. The use of an h-refinement procedure removes much of the mesh generation overhead associated with an adaptive remeshing approach. In the current scheme the initial mesh generation and refinement parts of the algorithm account for only a few percent of the overall CPU time. In contrast with adaptive remeshing, the mesh generation time can account for the majority of the overall CPU time.

The refinement and multigrid algorithm are efficient on both CPU time and memory. This makes solutions to three-dimensional geometries tractable on a workstation.

The solution adaptive multigrid scheme has been demonstrated on two geometries and gives CPU reductions comparable to a structured multigrid approach.

X. Future Work

The obvious next step is to include the viscous terms and a suitable turbulence model in the algorithm. This has already been done for a similar algorithm in two dimensions[12] using the *k-e* and Baldwin-Lomax turbulence models. The key to obtaining efficient high-quality viscous solutions is the use of a quadrilateral mesh (in two dimensions) and a prismatic mesh (in three dimensions) in the near wall region. These mesh types allow the one-dimensional refinement necessary to efficiently resolve the one-dimensional gradients in boundary layers.

The initial mesh has to be fine enough to resolve all of the features of the geometry. If this is not so, it is possible for the node snapping procedure to produce a crossed over mesh. This restriction prevents the full benefit of multigrid being realized on complex geometries as a relatively fine initial mesh has to be used. Work in this area to allow a much coarser initial mesh would be beneficial.

In the leading-edge region of the inlet the flow gradients are essentially one dimensional. With the advancing front mesh generation technique the tetrahedra are formed to be as close to equilateral as possible. This results in a mesh which is very uniform in the leading-edge region. The current refinement algorithm preserves this uniform mesh and, hence, is rather wasteful on points in this region. Work on producing a initial mesh which is stretched to align with regions such as leading edges would result in a more efficient algorithm.

Acknowledgments

The authors wish to acknowledge the General Electric Company for permission to publish this paper and Rajiv Thareja of NASA Langley for supplying the advancing front mesh generation program. In addition, the authors would like to thank Richard Cedar

of General Electric Aircraft Engines for supplying the geometry and test data for the inlet case.

References

[1]Jameson, A., and Baker, T. J., "Improvements to the Aircraft Euler Method," AIAA Paper 87-0452, Jan. 1987.

[2]Mavripilis, D. J., "Three Dimensional Unstructured Multigrid for the Euler Equations," AIAA Paper 91-1549, June 1991.

[3]Dawes, W. N., "The Extension of a Solution Adaptive 3D Navier-Stokes Solver Towards Geometries of Arbitrary Complexity," American Society of Mechanical Engineers, ASME Paper 92-GT-363, June 1992.

[4]Löhner, R., "Adaptive Remeshing for Transient Problems With Moving Bodies," AIAA Paper 88-3737, Jan. 1988.

[5]Holmes, D. G., and Connell, S. D., "Solution of the 2D Navier-Stokes Equations on Unstructured Adaptive Grids," AIAA Paper 89-1932, June 1989.

[6]Martinelli, L., "Calculation of Viscous flows with a Multigrid Method," Ph.D. Thesis, Dept. of Mechanical and Aerospace Engineering, Princeton Univ., Princeton, NJ, Oct. 1987.

[7]Morgan, K., Peraire, J., and Peiro, J., "Unstructured Grid Methods for Compressible Flows," AGARD Rept. R-787, March 1992.

[8]Holmes, D. G., and Connell, S. D., "Unstructured, Adaptive, Finite Volume Solution Methods for Fluid Dynamics," 7th IMACS International Conf. on Computer Methods for Partial Differential Equations, IMACS PDE-7, Rutgers Univ., June 22–24, 1992.

[9]Dawes, W. N., "The Simulation of Three-Dimensional Viscous Flow in Turbomachinery Geometries Using a Solution-Adaptive Unstructured Mesh Methodology," American Society of Mechanical Engineers, ASME Paper 91-GT-124, June 1991.

[10]Löhner, R. "Adaptive H-Refinement on 3-D Unstructured Grids for Transient Problems," AIAA Paper 89-0365, Jan. 1989.

[11]Haimes, R., Connell, S. D., and Sabine, A. V., "Visual Grid Quality Assessment for 3D Unstructured Meshes," AIAA Paper 93-3352, July 1993.

[12]Connell, S. D., Holmes, D. G., and Braaten, M. E., "Adaptive Unstructured 2D Navier-Stokes Solutions on Mixed Quadrilateral/Triangular Meshes," American Society of Mechanical Engineers, ASME Paper 93-GT-99, May 1993.

[13]Peraire, J., Morgan, K., and Peiro, J., "Unstructured Finite Element Mesh Generation and Adaptive Procedures for CFD," AGARD Specialists Meeting (Loen, Norway), May 1989.