INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2000; **34**: 93–111

Adaptive meshing techniques for viscous flow calculations on mixed element unstructured meshes

D. J. Mavriplis*

Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA, U.S.A.

SUMMARY

An adaptive refinement strategy based on hierarchical element subdivision is formulated and implemented for meshes containing arbitrary mixtures of tetrahedra, hexahedra, prisms, and pyramids. Special attention is given to keeping memory overheads as low as possible. This procedure is coupled with an algebraic multigrid flow solver, which operates on mixed element meshes. Inviscid flows, as well as viscous flows, are computed on adaptively refined tetrahedral, hexahedral, and hybrid meshes. The efficiency of the method is demonstrated by generating an adapted hexahedral mesh containing 3 million vertices on a relatively inexpensive workstation. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: adaptive meshing; multigrid solver; unstructured meshes

1. INTRODUCTION

The ability to easily incorporate adaptive meshing techniques represents one of the major advantages of unstructured grid solution strategies. For tetrahedral meshes, these may be incorporated in a rather straightforward manner, either through Delaunay point insertion methods [1], sub-division with local reconnection methods [2–4], or hierarchical-based element sub-division approaches [5–9].

Recently, there has been renewed interest in hybrid structured–unstructured grids, or mixed element unstructured grids, which contain elements other than simplices (i.e., tetrahedra). Such grids offer the advantages of reduced complexity and possibly increased accuracy compared with equivalent fully tetrahedral meshes. This is due partly to the reduced connectivity of elements, such as hexahedra, over simplicial elements, and the more regular tessalation of three-dimensional space which they afford. Particularly for viscous flow calculations, the use of prismatic elements in the boundary layer regions and tetrahedra in the inviscid flow regions has been demonstrated as a viable technique for reducing computational overheads.

* Correspondence to: Institute for Computer Applications in Science and Engineering, Mail Stop 403, NASA Langley Research Center, Hampton, VA 23681-0001, U.S.A.

Received 12 February 1997
Revised 3 November 1999

Additionally, mixed element mesh solvers provide added flexibility by enabling the use of a single flow solver on meshes of almost any construction, including block-structured meshes.

A viscous flow solver for mixed element meshes has been previously described in Reference [10]. By using a single edge-based data structure and an algebraic multigrid technique, a unified discretization and solution strategy for meshes of arbitrary element types were demonstrated. This paper represents a continuation of this philosophy, by extending adaptive meshing techniques for unstructured simplicial meshes to meshes of mixed element types, and computing the flow on these adapted meshes with the previously developed solver. In order to retain the flexibility and simplicity of the approach, we require that the adaptation process introduce no new complexities such as 'hanging nodes', which would subsequently require modifications to the flow solver and post-processing modules.

Unstructured mesh computations are currently memory limited. The incorporation of adaptive meshing represents but one of several techniques employed to overcome the large overheads incurred by such computations. It is, therefore, imperative that the adaptive meshing module incur memory overheads that are no larger than those required by the flow solver alone; otherwise the technique would be self-defeating. Adaptive meshing requires additional data structures that are not present in the flow solver. The implementation must, therefore, be executed with care to avoid excessive overheads.

2. ADAPTIVE MESHING APPROACHES

For tetrahedral meshes, various adaptive meshing techniques may be employed. New refinement points may be created and inserted into the mesh using the Delaunay point insertion algorithm of Bowyer [11]. Similarly, points may also be inserted using an initial forced connectivity, and then locally optimizing the mesh through edge-face swapping techniques, which can be used to recover the Delaunay property of the mesh [4,12]. Alternatively, an element sub-division procedure may be adopted, followed by a local edge-face swapping phase where the mesh is optimized to the Delaunay or some other criterion [3]. If straightforward element sub-division is employed, without any subsequent optimization procedure, strict hierarchical sub-division rules must be followed, otherwise mesh quality degrades rapidly with each subsequent adaptation phase [5,6].

The advantages of point insertion and optimization methods are potentially smoother adapted mesh point distributions with higher quality connectivity, while sub-division techniques benefit from efficiency and the ability to easily incorporate de-refinement. For highly stretched meshes typically employed in viscous flow regions, the Delaunay construction is no longer optimal, and face-edge swapping based on some more appropriate criterion must be employed. Although optimization based on the minimum–maximum angle has often been advocated for such cases, it has been found to be much less reliable than the use of a forced connectivity, such as one obtains through sub-division techniques, particularly for high stretching in the presence of curvature. Local optimization techniques and hierarchical sub-division methods are in some sense mutually exclusive, since the edge-face swapping operation destroys the hierarchical relationship with previous grid levels.

For hexahedral meshes, adaptive meshing must be achieved through element sub-division. Interfaces delimiting refined and non-refined regions usually require special consideration. These regions result in 'hanging nodes' if the mesh is required to contain only hexahedral elements. For cell-centered discretizations, using a data structure based on the faces of the cells, these regions can be handled without any additional complications to the flow solver [13]. However, for vertex-based schemes, these 'hanging node' control volumes must be constructed in a manner that is different from that of regular vertices (see Reference [14] for example). On the other hand, if different types of elements are allowed in the mesh, these can be used to transition the mesh from the refined to the unrefined regions, and the flow solver may operate on the resulting mixed element mesh without any modifications.

3. ADAPTIVE MESHING BY SUB-DIVISION

In the interest of developing a single strategy for adapting simplicial as well as mixed element meshes, a hierarchical element sub-division approach has been adopted. This technique can be applied to fully tetrahedral meshes, as well as to any hybrid mesh containing mixtures of tetrahedra, pyramids, prisms, and hexahedra. The resulting meshes can be employed by the multigrid solver described in Reference [10] without modification.

In order to implement this technique on mixed element meshes, the various allowable sub-division types for each element type must be defined. The hierarchical rules required to prevent the degeneration of the grid quality with successive adaptation levels must also be constructed.

For tetrahedral elements, the sub-division rules have already been well formulated in the literature [15,16]. We allow only three basic sub-division types, as depicted in Figure 1. A tetrahedron may be divided into two children, four children, or eight children. The two former cases result in anisotropic refinement, while the last case produces an isotropic refinement. For simplicity, the case where a pair of opposite edges are split is not considered. In this case, the tetrahdron must be fully refined into eight children. In the case of full refinement into eight sub-tetrahedra, there are three possible ways to split the center octahedron into four new tetrahedra. One of these sub-division patterns has been (randomly) chosen and is used uniquely for all elements. In order to prevent the degeneration of grid quality, any anisotropic children may not be refined further. If any such cells require refinement, they are removed, the parent cell is isotropically refined, and the resulting isotropic children may then be further refined. When limiting the possible refinement types as described above, one must ensure that a compatible refinement pattern is obtained on all elements of the mesh if a valid refined mesh is to be obtained. This is achieved by adding refinement points along the appropriate edges on all elements, which are flagged as having a non-valid refinement pattern. Since the addition of a refinement point to an edge affects all elements that contain the edge, the process is applied iteratively until all resulting element refinement patterns are valid and no further points are required.

The isotropic refinement of a hexahedral element results in eight similar but smaller hexahedral elements. However, anisotropic refinement of a hexahedral element results in

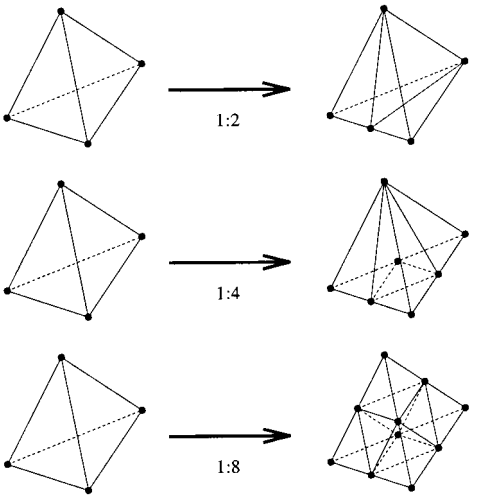

Figure 1. Permitted sub-division types for tetrahedral elements.

children that may consist of hexahedra, pyramids, prisms, and tetrahedra. By applying the same hierarchical rules as described for tetrahedral meshes, we can ensure that these elements will never be refined further. Instead, if further refinement in these regions is desired, such elements are deleted and their parents refined into eight smaller hexahedra. Thus, for fully hexahedral meshes, additional element types may only appear at the boundaries between refined and non-refined regions; or more generally, between two regions that differ by one refinement level.

The task of implementing adaptive mesh sub-division for elements other than tetrahedra consists in defining the minimum number of allowable sub-division types. On the one hand, it is desirable to limit the number of sub-division types for complexity reasons. On the other hand, a minimum number of sub-division types must be implemented to allow for compatible sub-division types to be attained on all elements without incurring excessive additional refinement. For example, if we do not allow for hexahedra with three fully refined quad faces (e.g., Type 7 in Figure 2), refinement in concave regions cannot be made to terminate, and propagates throughout large portions of the domain during the iterative addition of refinement points. A total of eight hexahedral refinement types have been implemented. These types axe illustrated in Figure 2.

A total of nine prismatic and eight pyramidal refinement types have also been implemented. These are illustrated in Figures 3 and 4. The resulting children cell types for each refinement pattern are documented in Table I for hexahedral sub-divisions, Table II for prismatic sub-divisions, and Table III for pyramidal sub-divisions. The required refinement types are determined by first considering the various possible refinement patterns of a triangular and a quadrilateral face. A triangular face may be divided into two or four triangles, while a quadrilateral face may be divided into three or four triangles, or four quadrilaterals, as shown

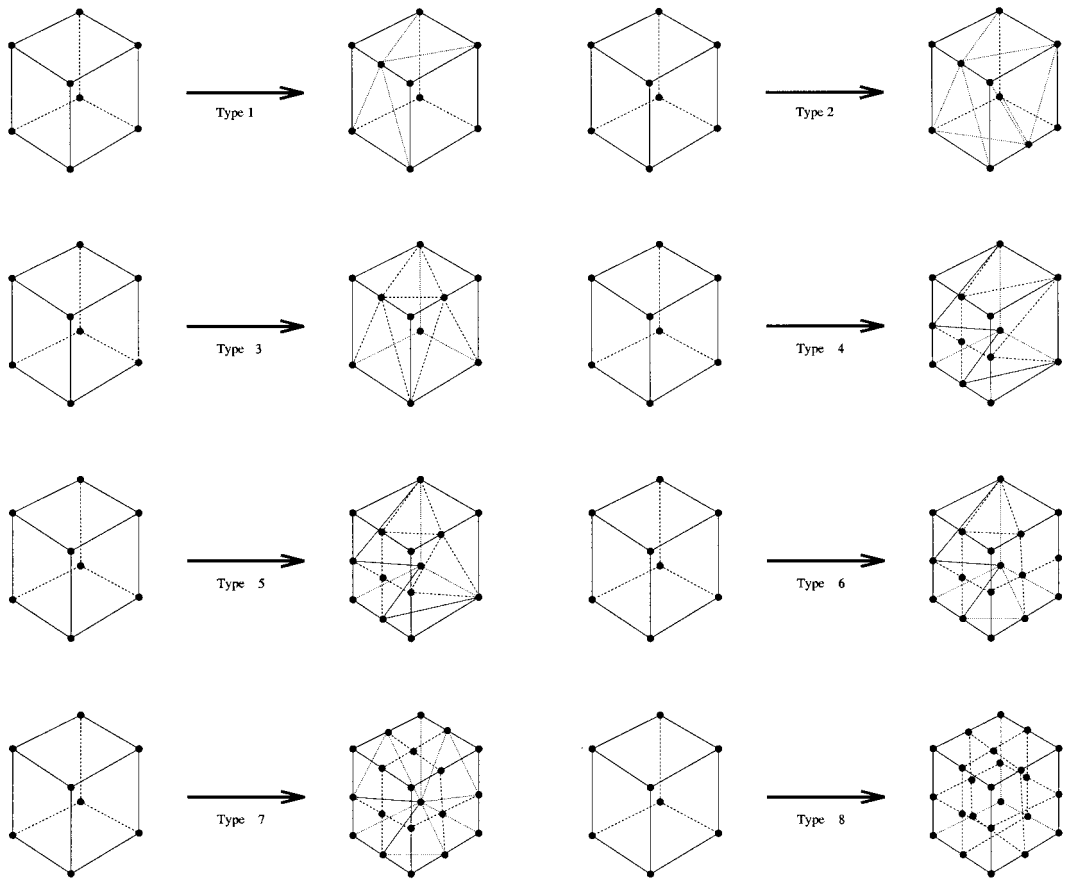

Figure 2. Permitted sub-division types for hexahedral elements (internal edges omitted for clarity).

in Figure 5. By considering all possible face sub-division patterns for all faces of a given element, a list of all possible cell sub-division types can be constructed. Many of these sub-division types are equivalent and merely correspond to a different orientation of the cell. For example, in case 1 in Figure 2, where a single edge of a hexahedral cell is refined, there are 12 possible orientations, corresponding to the 12 edges of the hexahedron. In total, when the number of orientations for each refinement type is considered, there is a total of 90 hexahedral, 45 prismatic, and 30 pyramidal configurations that must be taken into account. These are implemented by coding one canonical sub-division routine for each type, and using an orientation mask to reorder the element indices according to the orientation of the sub-division. Isotropic sub-division of all cell types results in eight smaller cells of the same type as the parent cell, except in the case of pyramidal cells, where isotropic refinement yields six pyramidal children and four tetrahedral children.

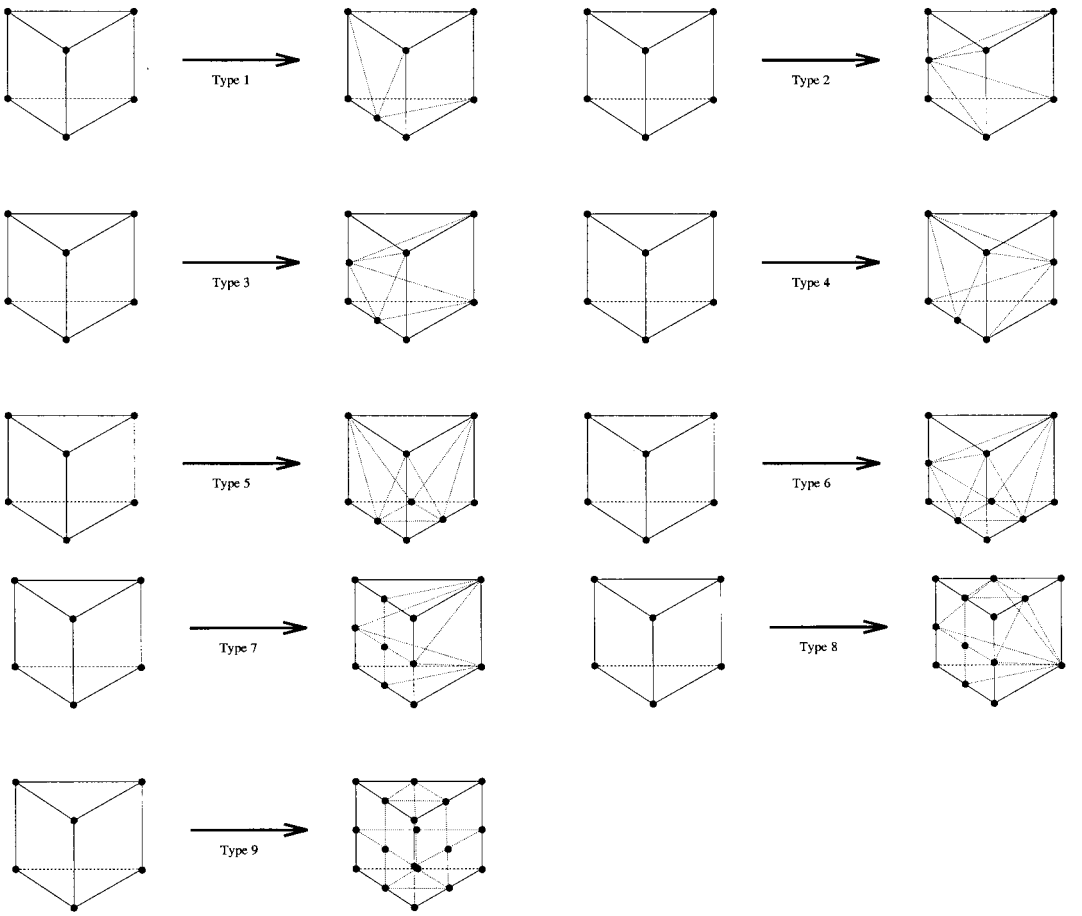

Figure 3. Permitted sub-division types for prismatic elements (internal edges omitted for clarity).

This implementation differs significantly from other non-tetrahedral adaptive meshing strategies [17,18]. For example, in Reference [18], a limited number of sub-division types are used, and refinement compatibility is ensured through the use of additional points inserted at neighboring cell centroids. Instead, we rely on the use of many possible anisotropic refinement patterns to generate compatible refinement patterns. These refinement types have been implemented for tetrahedra and hexahedra, as well as for prisms and pyramids. However, directional refinement [17–19] has not been implemented in the present context.

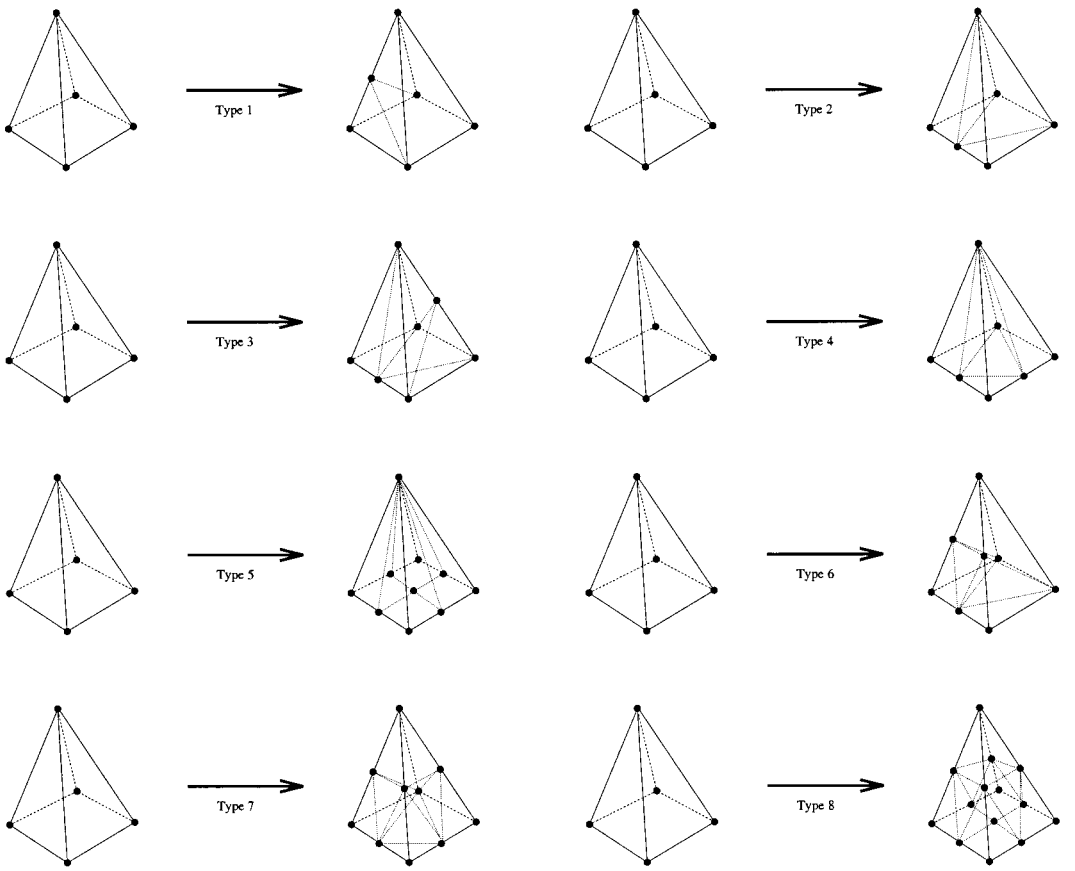

Figure 4. Permitted sub-division types for pyramidal elements (internal edges omitted for clarity).

Table I. Resulting children cell types for various hexahedral refinement patterns.

Hexahedral reference type	Hexahedra	Prisms	Pyramids	Tetrahedra
Type 1	0	0	4	0
Type 2	0	0	2	6
Type 3	0	0	3	4
Type 4	0	0	5	4
Type 5	0	0	5	6
Type 6	0	2	6	3
Type 7	0	3	8	2
Type 8	8	0	0	0

Table II. Resulting children cell types for various prismatic refinement patterns.

Prism reference type	Prisms	Pyramids	Tetrahedra
Type 1	0	2	1
Type 2	0	1	2
Type 3	0	1	4
Type 4	0	0	6
Type 5	0	0	7
Type 6	0	2	4
Type 7	0	4	2
Type 8	0	4	5
Type 9	8	0	0

Table III. Resulting children cell types for various pyramidal refinement patterns.

Pyramid reference type	Pyramids	Tetrahedra
Type 1	1	2
Type 2	0	3
Type 3	0	5
Type 4	0	4
Type 5	4	0
Type 6	2	2
Type 7	0	8
Type 8	6	4

4. HANGING EDGES

The process of defining the various element sub-division types consists in constructing combinations of children hexahedra, tetrahedra, prisms, and pyramids, which are non-overlapping, completely fill the volume of the parent element, and are compatible with the sub-division patterns of the parent faces. This is achieved for all sub-division types of all elements except for one, a hexahedral Type 7 refinement, i.e., three fully refined hexahedral faces.

In this case, it is not possible to fill the interior of the hexahedron with combinations of other elements that respect the face sub-division patterns. This type of refinement cannot be omitted, however, since this obviates the possibility of creating convex refined regions on hexahedral meshes, as described earlier. This refinement type is constructed by sub-dividing the parent hexahedron into three prisms, eight pyramids, and two tetrahedra, which results in the face sub-division pattern shown in Figure 6. This face pattern differs from the desired one by

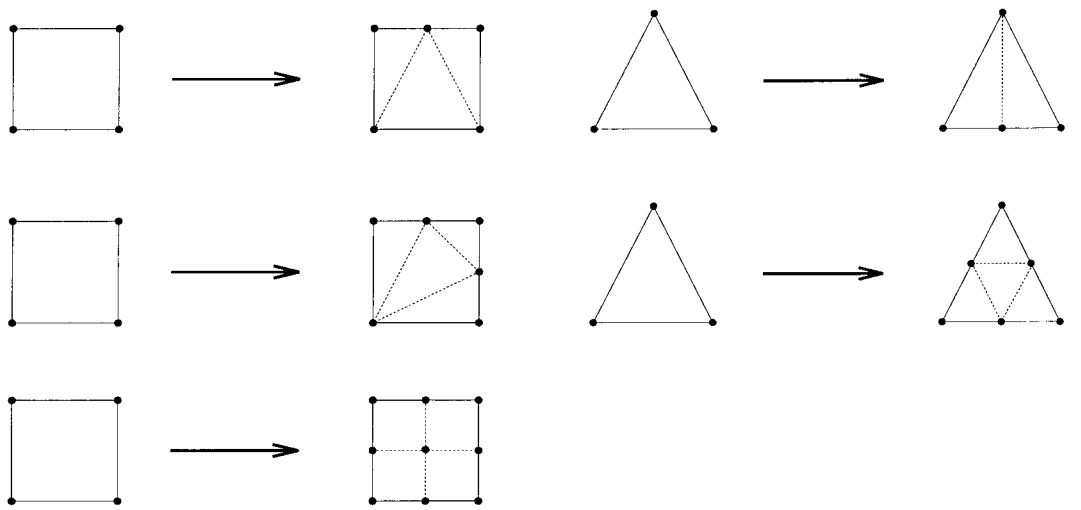

Figure 5. Permitted sub-division types for triangular and quadrilateral faces.

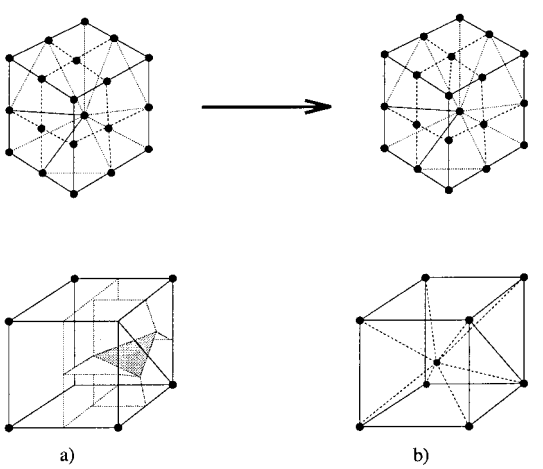

Figure 6. Hanging edge resulting from Type 7 hexahedral refinement and two possible treatments of hanging edge: (a) modified dual control volume; (b) insertion of extra vertex into hexahedron with hanging edge.

the addition of a diagonal edge on one of the children quadrilateral faces. The result is a 'hanging diagonal' edge, which may not be present in the neighboring element that shares this quadrilateral face.

There are various possibilities for treating this situation. The first option is to allow for hanging edges in the final mesh, and modify the control volumes employed by the flow solver

in the vicinity of these edges. The dual control volume graph for a hexahedron with a hanging edge is depicted in Figure 6(a). The faces of the dual control volumes are constructed by considering the centroids of each neighboring element as well as the centroids of all quadrilateral and triangular faces. This implementation corresponds to a simple modification of the edge weights in the flow solver.

Another possibility is to ensure compatible face patterns throughout the mesh by inserting an additional point into cells that border on a quadrilateral face with a hanging edge. For example, if the neighboring element is a hexahedron, this cell can be sub-divided into six pyramids by inserting a vertex at its centroid, and the appropriate pyramid can then be split into two tetrahedra in order to conform to the face sub-division pattern. This is similar to the point insertion procedure used in Reference [18] to prevent propagation of hexahedral refinement.

In both cases, the modifications to the mesh are performed at the end of the adaptation phase prior to the flow solution phase. These modifications are not included in the hierarchical refinement description of the mesh, which may allow hanging edges, and which form the beginning state for subsequent refinements, as any further refinement of a cell with a hanging edge will result in isotropic refinement of its parent and thus removal of the hanging edge.

In the present work, both approaches have been implemented. However, the examples shown below have all been performed using the additional point insertion method.

5. IMPLEMENTATION ASPECTS

Throughout the refinement procedure, new vertices are created, either at the center of an edge when the edge is split into two children edges, at the centroid of a quadrilateral face when this face is refined into four smaller quadrilateral faces (cf., Figure 5), or at the centroid of a hexahedral cell when the hexahedron is isotropically refined. The creation of vertices at the centroids of quadrilateral faces requires the storage of the quadrilateral faces of the mesh in order to uniquely flag such new points, in addition to the storage of edges and cells, which is required for all element types.

Although refinement criteria can be evaluated using only edges of the mesh, refinement decisions are element-based. In the initial refinement pass, the refinement criterion is evaluated along each edge of each element. When the refinement criterion is satisfied along any edge of a given element, all edges of that element are flagged for refinement, thus creating an isotropic refinement pattern for that element.

An iterative procedure is subsequently employed to generate suitable refinement patterns for all element types in the mesh. In general, a small number of iterations are required to achieve convergence. When anisotropically refined elements are to be re-refined, they must first be removed and their parents must be isotropically refined. Since these new children are not present in the initial mesh, it is possible that after they are formed, they contain edges that have been previously flagged for refinement, and thus subsequent refinement is required. Additionally, one must ensure that compatible refinement patterns are now obtained on these new elements. Although it is possible to predict the entire refinement patterns for all elements, including anisotropically refined elements, the complexity of such a task for mixed element meshes becomes overwhelming. Therefore, an outer iteration loop is employed. In the first

phase, the compatible refinement patterns on all existing cells is determined iteratively. The newly refined cells are then formed, and the process is repeated on the new set of cells, using the same refinement flags. The entire process is repeated until no further refinement is detected. Usually, the outer iterative loop terminates in one or two passes.

As mentioned previously, an important aspect in the implementation of any adaptive meshing module is that the resources required by the adaptive meshing module be no larger than those required by the flow solver. The flow solver requires only a single data structure to compute the discretization on mixed element meshes, i.e., the edge data structure. On the other hand, the adaptive meshing module requires the edge data structure as well, but also requires the cell-to-vertex data structure, as well as the cell-to-edge data structures, since refinement is achieved by flagging edges for refinement. However, the sub-division decisions are performed on an element basis. In addition, all the hierarchical information must be stored.

The flow solver of Reference [10] can operate at approximately 100–150 words per vertex. For tetrahedral meshes, the cell-to-vertex and cell-to-edge data structures alone consume approximately 60 words per vertex. These additional data structures thus require a substantial amount of memory, and it is a non-trivial task to ensure that the adaptive meshing module memory requirements do not exceed those of the flow solver.

Although a hierarchical storage scheme, where the cells point to the faces, the faces point to the edges, and the edges point to the vertices, simplifies the implementation of sub-division schemes, such implementations incur excessive memory overheads. In our implementation, we make use of cell-to-vertex, cell-to-edge, and cell-to-quadrilateral face information, as well as quadrilateral face-to-edge information. Of particular importance is the need to avoid storing the triangular faces of the mesh. For a tetrahedral mesh of N vertices, the number of triangular faces is of the order of $12N$. To identify the three forming vertices or edges for each triangular face, as well as the two tetrahedra that share the face, requires a total of $60N$ storage. Similarly, if one chooses to store a list of all eight potential children for each element, this amounts to $48N$ storage locations for a tetrahedral mesh of N vertices. Instead, we use a linked list to store the children of all elements. This can be achieved with an array that is twice as large as the overall number of elements for mixed element meshes.

Overall memory requirements depend on the types of elements in the mesh. For tetrahedral meshes, approximately 120 words per vertex are required, while for hexahedral meshes 74 words per vertex are used. Requirements for prismatic, pyramidal, and of course hybrid meshes fall in between these two extremes. These estimates do not include associated boundary face and hierarchical information, which can add from 10 to 25 per cent more storage.

The adaptive refinement procedure begins by iteratively determining a compatible refinement pattern on all elements of the input mesh. Once this has been achieved, all children cells of anisotropically refined parent cells, which are to be further refined, are deleted, as well as all faces and edges that are only accessed by these removed cells. All corresponding lists are then shifted to occupy contiguous space in memory. The size of the new mesh is then predicted by looping over all cells and counting the number of children cells, faces, and edges that will result in the refinement phase. The predicted size of the new mesh is then used to allocate all the required memory for the refinement operation in a single step, or to terminate the program if not enough memory is available. This procedure reduces overall memory requirements by first removing any memory associated with data structure elements to be deleted, and enables

one to examine the characteristics of the refined mesh before actually generating this new mesh. This should also prove useful for distributed memory applications, where it may be more efficient to partition the unrefined mesh based on the predicted refinement distribution, particularly in cases where the amount of desired refinement local to a given processor exceeds the memory resources of that processor.

6. FLOW SOLVER DESCRIPTION

The Reynolds-averaged Navier–Stokes flow solver consists of a finite volume central difference discretization, with added artificial dissipation for stability. The variables are stored at the vertices of the mesh, and the solver is capable of operating on meshes containing any mixture of tetrahedra, pyramids, prisms, and hexahedra. On tetrahedral elements, the full Navier–Stokes viscous terms are employed, while on other types of elements, the thin-layer version of these terms is employed, albeit in all three co-ordinate directions. For viscous flow cases, turbulence effects are accounted for using the one-equation model of Spalart and Allmaras [20]. The code employs a single edge-based data structure, which enables the fluxes to be computed over all various element types of the mesh in a single loop over the edges.

Explicit time stepping is employed to integrate the discretized equations to steady state, and an algebraic or agglomeration multigrid procedure is employed to accelerate convergence. The agglomeration procedure automatically constructs coarse levels for the multigrid algorithm by fusing together or agglomerating neighboring fine grid control volumes. These control volumes are based on the dual of the mesh, and can be constructed using only the edges of the mesh. The agglomeration procedure can thus be employed without modification on meshes of mixed element types. The resulting coarse meshes contain large polyhedral cells, which in general are not tetrahedral, hexahedral, pyramidal, or prismatic. The edge-based discretization routines enable the flow solver to operate on these coarse levels as well.

7. RESULTS

A 'conservative' adaptive refinement strategy is adopted in the following cases. We begin with relatively well-resolved initial grids, and set the refinement tolerances to produce liberal amounts of refinement, generally increasing the number of mesh points by a factor of 3–4 at each refinement pass, and thus employing a small number of refinement levels. This is in contrast to the often advocated approach of using a large number of refinement levels in conjunction with a very coarse initial grid to obtain a so-called 'optimal' final mesh.

Our conservative approach is dictated by the lack of reliable error estimators. In the following examples, the undivided gradient of density is used as a refinement criterion. A threshold value is set as an input parameter, and any cells whose values exceed the threshold are flagged for refinement. Additional refinement is then determined by the iterations required to obtain compatible refinement patterns.

The first example consists of the inviscid flow over an aircraft configuration using a fully tetrahedral mesh. The initial mesh is depicted in Figure 7. This mesh was generated using the

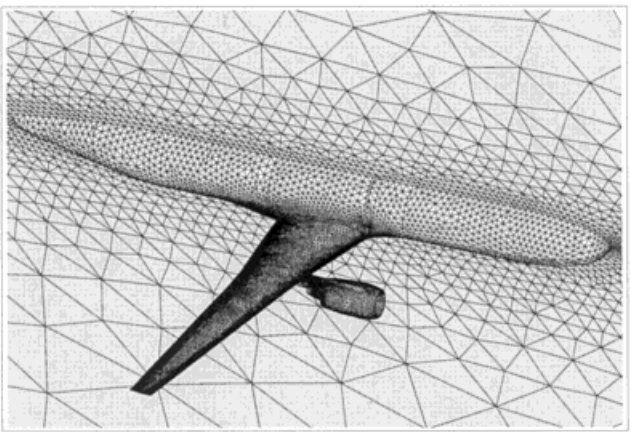

Figure 7. Initial tetrahedral mesh about transport aircraft configuration (number of vertices = 106 000).

advancing-front method of Pirzadeh [21] and contains approximately 106 000 points and 576 000 cells. The mesh is adapted twice throughout the calculation, and the final mesh is depicted in Figure 8. This mesh contains 1.3 million points and 8 million tetrahedra. Most of the refinement occurs along the wing, in particular at the leading and trailing edges and in the vicinity of the shock for the last refinement level. The computed Mach contours on the final adapted mesh are displayed in Figure 9. The Mach number for this case is 0.768 and the incidence is 1.116°. The adaptive meshing module, as well as the flow solution module, were both run on a SUN ULTRA workstation with 1 Gbyte of memory. The adaptive meshing module requires 750 Mbytes of memory and approximately 15 min to generate the final mesh,

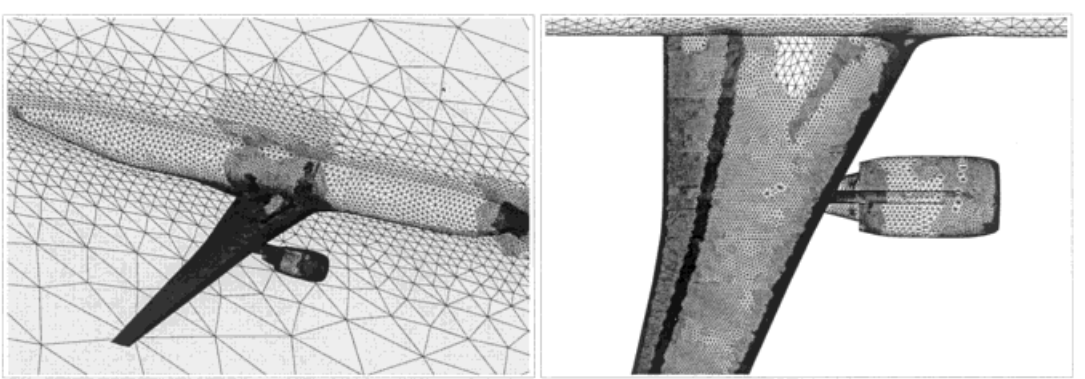

Figure 8. Adaptively refined tetrahedral mesh about transport aircraft configuration after two levels of refinement (number of vertices = 1.3 million).

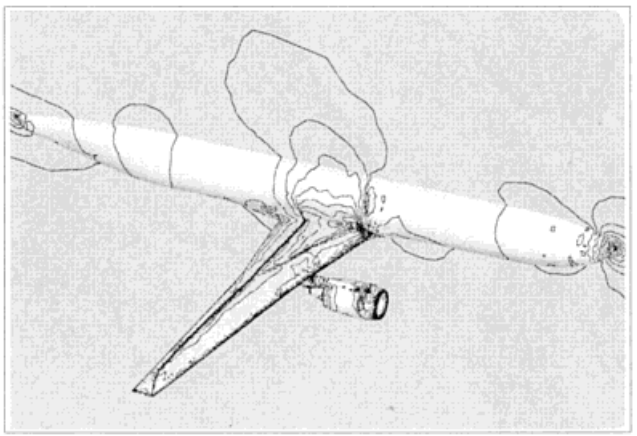

Figure 9. Computed Mach contours on adaptively refined tetrahedral mesh for transport aircraft configuration (Mach = 0.768, incidence = 1.116°).

while the flow solver requires 1 Gbyte of memory and 11 h to generate the solution displayed in Figure 9, using 100 multigrid cycles. Both codes were compiled using 64 bit real precision and 32 bit integer precision.

In its present form, the mesh adaptation module may be applied to fully tetrahedral meshes as well as mixed element meshes. As a sub-set of these cases, block structured meshes can also be handled by treating them as a set of unstructured hexahedra. Figure 10 depicts a fully hexahedral block structured mesh about an ONERA M6 wing. This grid was provided by

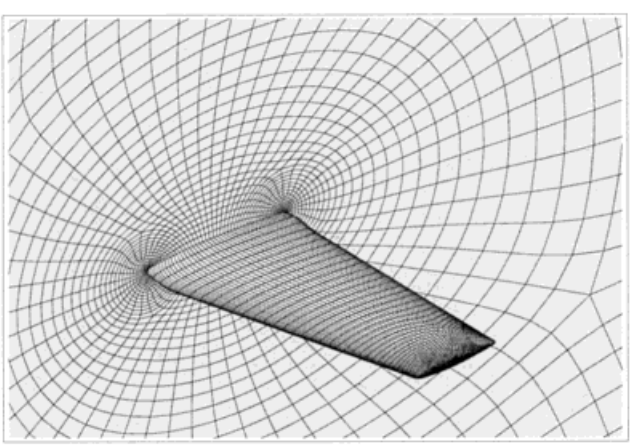

Figure 10. Initial block structured mesh about ONERA M6 wing (number of vertices 129 187).

P. Eiseman of the Program Development Corporation. It contains 129 187 points and 141 blocks. The inviscid flow over this configuration has been computed at a Mach number of 0.84 and 3.06° incidence, and the mesh has been adaptively refined three times. The resulting refined mesh is depicted in Figure 11. This mesh contains a total of 3 million points, from which are formed approximately 3.1 million hexahedra, 300 000 tetrahedra, 500 000 prisms, and 47 000 pyramids. The refinement pattern is seen to follow the double shock pattern that is associated with the flow at these conditions, as well as the regions of rapid expansion and compression near the leading and trailing edges of the wing. The computed Mach contours on this mesh are displayed in Figure 12. The refinement operation was run on a SUN ULTRA workstation and required 1.2 Gbytes of memory (swap space was used for the memory requirements over the 1 Gbyte core memory of the workstation) and about 45 min of CPU time. The flow solver required 1.6 Gbytes of memory and was therefore run on a CRAY C90. While the use of 3 million grid points to resolve the inviscid flow over a simple swept wing may appear excessive, it is nevertheless useful to demonstrate the efficiency attainable on a relatively inexpensive workstation by the combination of the low memory implementation of the adaptive meshing module, and the use of hexahedral meshes which contain less than half the number of edges of an equivalent tetrahedral mesh.

The final example involves the viscous flow over a partial span flap wing configuration using a mixed element mesh. The initial mesh is depicted in Figure 13. The center section of this mesh was constructed using the advancing-front method of Pirzadeh [21]. The tetrahedra in the boundary layer region of this mesh were then merged into prisms using the mesh-merging algorithm described in Reference [10]. This combined prismatic–tetrahedral center mesh section was then extruded in both spanwise directions, resulting in a mesh containing a mixture of hexahedra, prisms, and tetrahedra. A close-up of the near wall region is depicted in Figure 13, illustrating the hexahedral and prismatic elements in this region. The mesh contains a total of 165 000 vertices, 228 000 tetrahedra, 184 000 prisms, and 28 000 hexahedra. The viscous

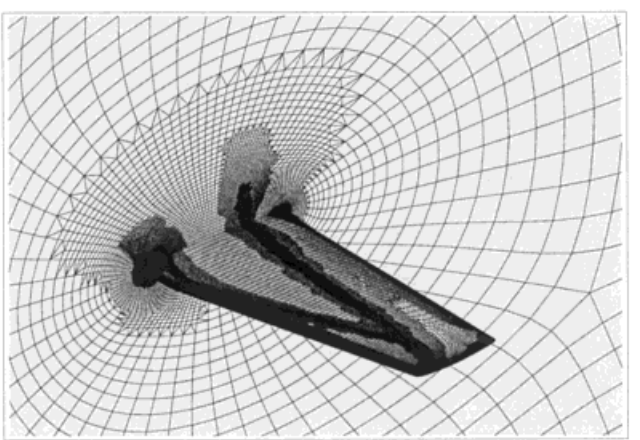

Figure 11. Adapted hexahedral mesh about ONERA M6 wing (number of vertices = 3 million).

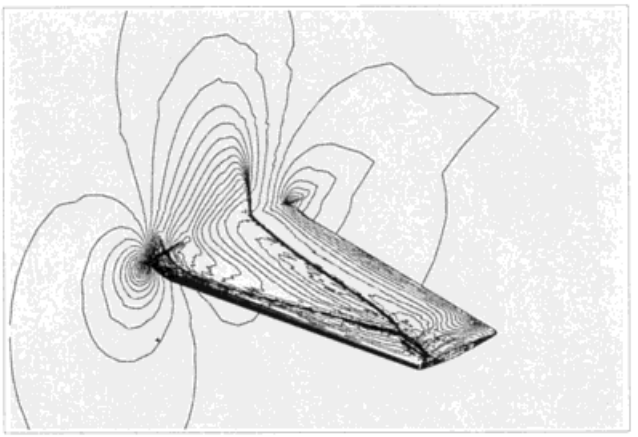

Figure 12. Computed Mach contours on adapted hexahedral mesh (Mach = 0.84, incidence = 3.06°).

turbulent flow is computed on this mesh at a Mach number of 0.2, a Reynolds number of 3.7 million, and 10° incidence.

The mesh is adapted twice during the solution process, resulting in the final mesh depicted in Figure 14. This mesh contains a total of 1.8 million points, with approximately 2.2 million tetrahedra, 2 million prisms, 260 000 pyramids, and 312 000 hexahedra. The adaptive meshing

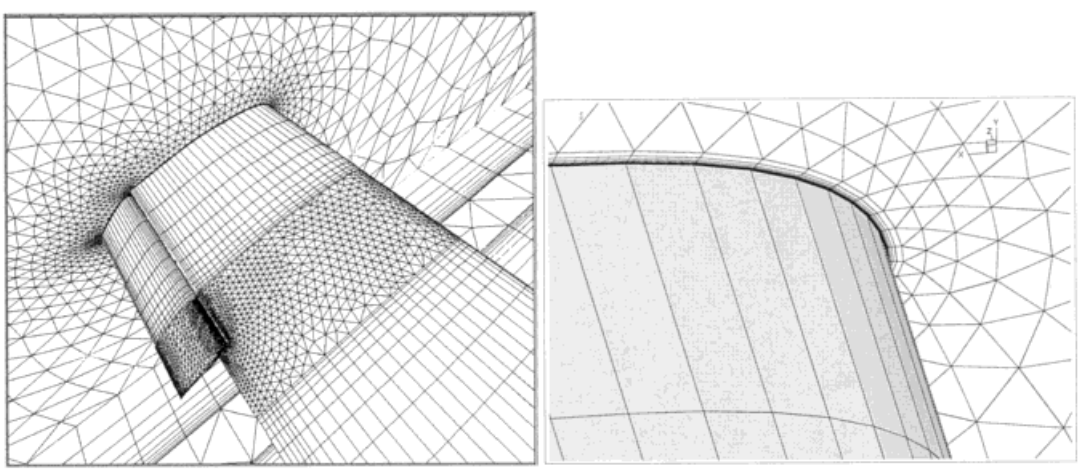

Figure 13. Initial mixed element mesh for partial span flap configuration showing hexahedral and prismatic cells in near wall region (number of vertices = 165 000).

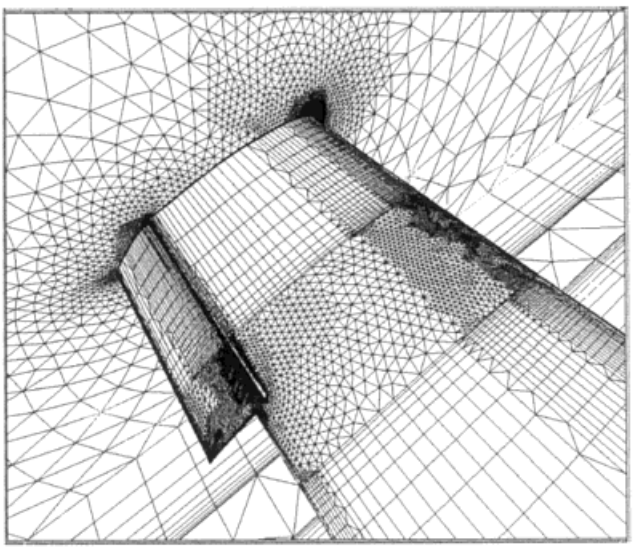

Figure 14. Adaptively refined mixed element mesh for partial span flap configuration after two levels of refinement (number of vertices = 1.8 million).

module was run on a SUN ULTRA workstation and required 1 Gbyte of memory and 30 min of CPU time for the final refinement phase. The computed density contours on this adapted mesh are depicted in Figure 15.

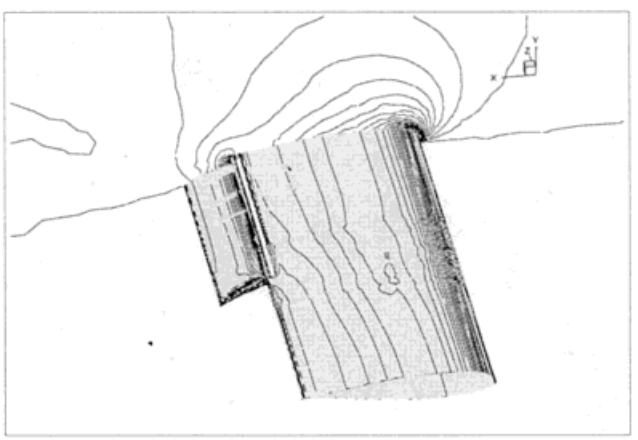

Figure 15. Computed density contours on adapted mixed element mesh for partial span flap configuration (Mach = 0.2, Reynolds number = 3.7 million, incidence = 10°).

8. CONCLUSIONS AND FUTURE WORK

While the memory and CPU requirements of the adaptive meshing module are reasonable for large steady state calculations, reduced CPU requirements would be desirable for unsteady calculations, where adaptation may be required every several time steps. Much of the CPU time is spent in the iterative determination of compatible refinement patterns. This phase can be greatly accelerated, albeit with added memory requirements, by flagging only those cells that receive additional refinement at each pass, and then examining only those cells and their neighbors at each additional iteration. Derefinement by retracing the hierarchical sub-division tree has been implemented for tetrahedral meshes and should also be extended to other types of elements.

The adaptive meshing module was constructed as a separate code from the flow solver. The adaptive mesher reads a mesh file and a solution file generated by the flow solver and outputs a refined mesh and interpolated solution file, which is then read in by the flow solver for the next solution phase. Integrating the two modules into a single code would require all flow solver and mesh refinement data structures to be held simultaneously in core memory, which would in turn greatly reduce the size of problems that can be handled. In the present procedure, only the required data structures for each module are resident in memory and the two modules communicate through file I/O. This procedure can easily be automated using a scripting language. On hierarchical memory machines, such as the CRAY C90, further acceleration could be achieved by making use of the solid state disk (SSD).

The use of hexahedral and mixed element meshes for transient calculations with relative body motion and deforming grids remains an open area for research.

ACKNOWLEDGMENTS

This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001, U.S.A.

REFERENCES

1. Mavriplis DJ. Three-dimensional multigrid for the Euler equations. *AIAA Journal* 1992; **30**(7): 1753–1761.
2. Marcum DL. Generation of unstructured grids for viscous flow applications. AIAA Paper No. 95-0212, 1995.
3. Bottasso CL, De Cougny HL, Flahery JE, Özturan C, Rusak Z, Shephard MS. Compressible aerodynamics using a parallel adaptive time-discontinuous Galerkin least-squares finite element method. AIAA Paper No. 94-1888, 1994.
4. Barth TJ. Aspects of unstructured grids and finite-element volume solvers for the Euler and Navier–Stokes equations. In *von Karman Institute Lecture Series*, AGARD Publications R-787, 1992.
5. Löhner R, Baum JD. Adaptive H-refinement on 3D unstructured grids for transient problems. *International Journal for Numerical Methods in Fluids* 1992; **14**: 1407–1419.
6. Rausch RD, Batina JT, Yang HTY. Spatial adaptation of unstructured meshes for unsteady aerodynamic flow computations. *AIAA Journal* 1992; **30**(5): 1243–1251.
7. Connell SD, Holmes DG. A 3D unstructured adaptive multigrid scheme for the Euler equations. *AIAA Journal* 1994; **32**(8): 1626–1632.
8. Kallinderis Y, Vijayan P. Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. *AIAA Journal* 1993; **31**(8): 1440–1447.

9. Biswas R, Strawn R. A dynamic mesh adaptation procedure for unstructured hexahedral meshes. AIAA Paper No. 96-0027, 1996.
10. Mavriplis DJ, Venkatakrishnan V. A unified multigrid solver for the Navier–Stokes equations on mixed element meshes. AIAA Paper No. 95-1666, 1995.
11. Bowyer A. Computing Dirichlet tessalations. *The Computer Journal* 1981; **24**(2): 162–166.
12. Marcum DL, Weatherill NP. Unstructured grid generation using iterative point insertion and local reconnection. AIAA Paper No. 94-1926, 1994.
13. Spragle G, Smith WA. Hanging node solution adaptation on hybrid grids. In *Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Field Simulations*, Starkville, MS, Soni BK, Thompson JF, Hauser J, Eisman PR (eds). Mississippi State University, 1996; 1221–1230.
14. Aftosmis M. Upwind method for simulation of viscous flow on adaptively refined meshes. *AIAA Journal* 1994; **32**(2): 268–277.
15. Löhner R. Finite-element methods in CFD: grid generation, adaptivity and parallelization. In *von Karman Institute Lecture Series*, AGARD Publications R-787, 1992.
16. Liu A, Joe B. Quality local refinement of tetrahedral meshes based on 8-sub-tetrahedron sub-division. *Mathematics of Computation* 1996; **65**(215): 1183–1200.
17. Parthasarathy V, Kallinderis Y, Nakajima K. Hybrid adaptation method and directional viscous multigrid with prismatic–tetrahedral meshes. AIAA Paper No. 95-0670, 1995.
18. Biswas R, Strawn R. A dynamic mesh adaptation procedure for unstructured hexagonal grids. AIAA Paper 96-0027. Presented at the 34th Aerospace Sciences Meeting, Reno, NV, 15–18 January 1996.
19. van der Vegt J. Anisotropic grid refinement using an unstructured discontinuous Galerkin method for the three-dimensional Euler equations of gas dynamics. AIAA Paper No. 95-1657, 1995.
20. Spalart PR, Allmaras SR. A one-equation turbulence model for aerodynamic flows. AIAA Paper No. 92-0439, 1992.
21. Pirzadeh S. Viscous unstructured three-dimensional grids by the advancing-layers method. AIAA Paper No. 94-0417, 1994.

