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 23681-0001

Abstract

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.

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.

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 straight-forward manner either through Delaunay point insertion methods [1], subdivision with local reconnection methods [2, 3, 4], or hierarchical-based element subdivision approaches [5, 6, 7, 8, 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. 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 [10] by the author. 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 was 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 which are no larger than that required by the flow solver alone, otherwise the technique would be self-defeating. Adaptive meshing requires additional data-structures which 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 subdivision 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 straight-forward

element subdivision is employed, without any subsequent optimization procedure, strict hierarchical subdivision 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 subdivision 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 subdivision techniques, particularly for high stretching in the presence of curvature. Local optimization techniques and hierarchical subdivision 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 subdivision. 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 which is different from that of regular vertices (see [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 Subdivision

In the interest of developing a single strategy for adapting simplicial as well as mixed element meshes, a hierarchical element subdivision 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 [10] without modification.

In order to implement this technique on mixed element meshes, the various allowable subdivision 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 subdivision rules have already been well formulated in the literature [15]. We allow only three basic subdivision types, as depicted in Figure 1: A tetrahedron may be divided into 2 children, 4 children, or 8 children. The two former cases result in anisotropic refinement, while the last case produces an isotropic refinement. 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 which contain the edge, the process is applied iteratively, until all resulting element refinement patterns are valid and no further points are required.

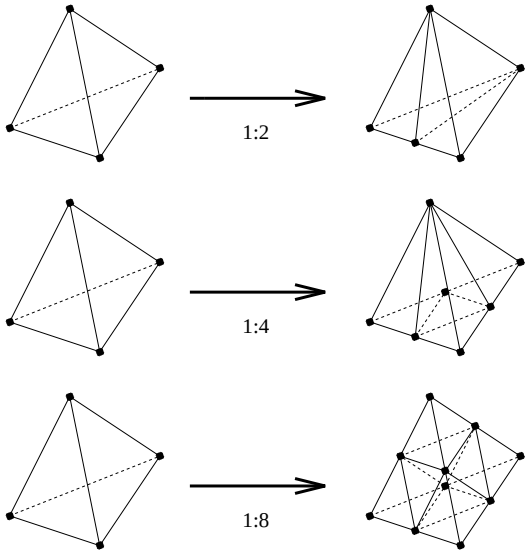


Figure 1: Permitted subdivision types for tetrahedral elements

The isotropic refinement of a hexahedral element results in eight similar but smaller hexahedral elements. However, anisotropic refinement of a hexahedral element results in children which 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 which differ by one refinement level.

The task of implementing adaptive mesh subdivision for elements other than tetrahedra, consists in defining the minimum number of allowable subdivision types. On the one hand, it is desirable to limit the number of subdivision types for complexity reasons. On the other hand, a minimum number of subdivision types must be implemented to allow for compatible subdivision 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 8 hexahedral refinement types have been implemented. These types are illustrated in Figure 2.

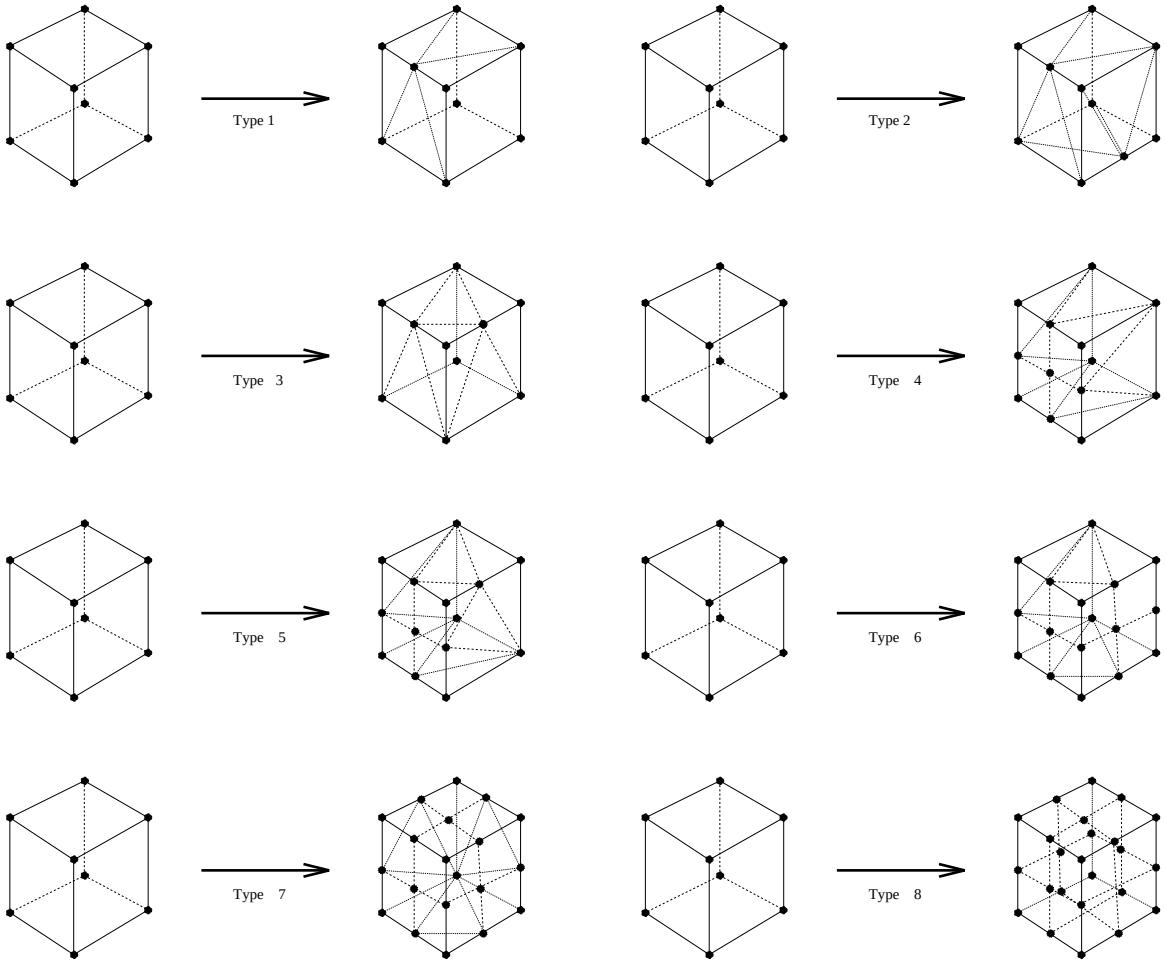
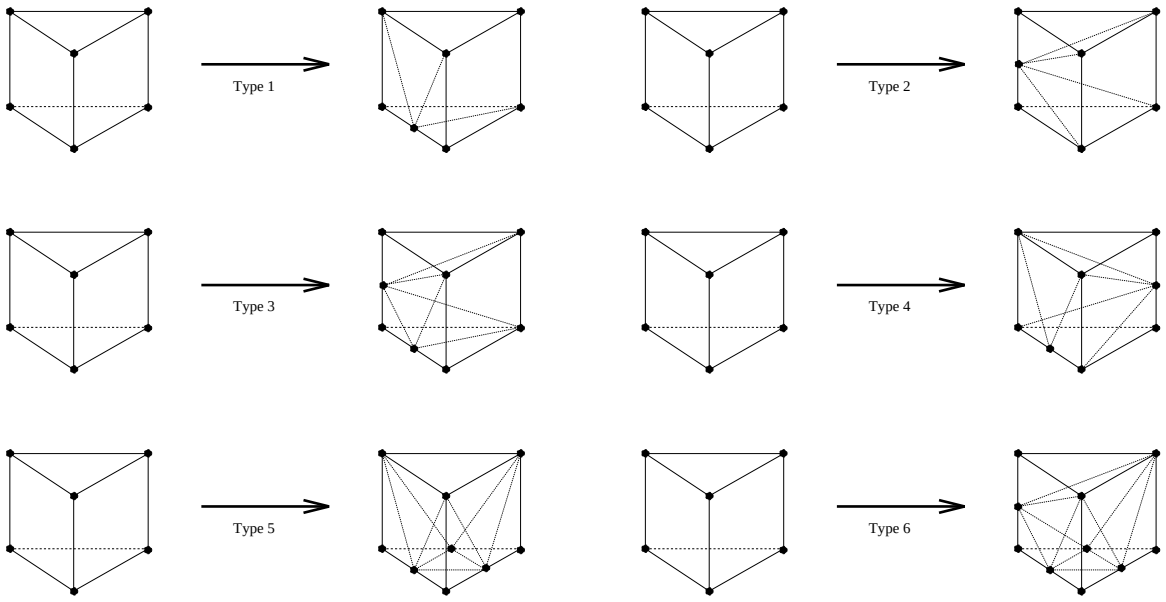


Figure 2: Permitted subdivision types for hexahedral elements (internal edges omitted for clarity)



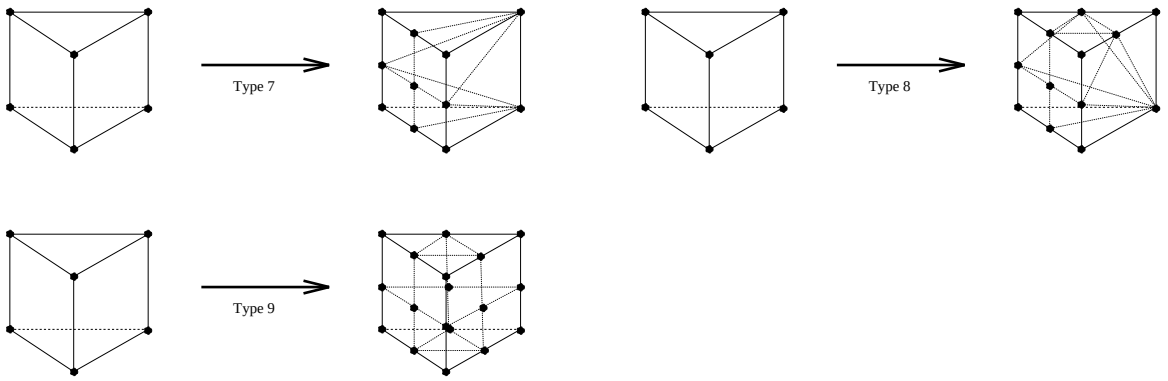


Figure 3: Permitted subdivision types for prismatic elements (internal edges omitted for clarity)

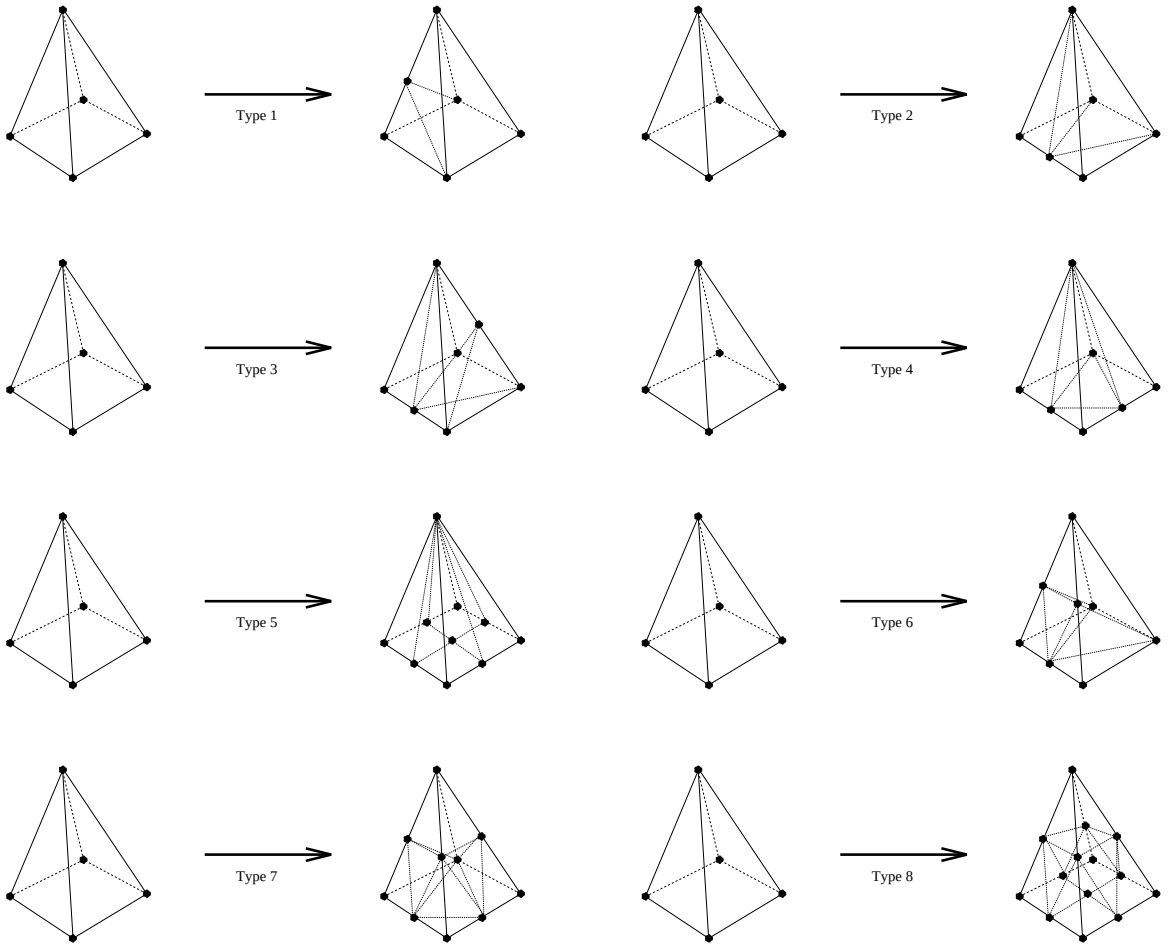


Figure 4: Permitted subdivision types for pyramidal elements (internal edges omitted for clarity)

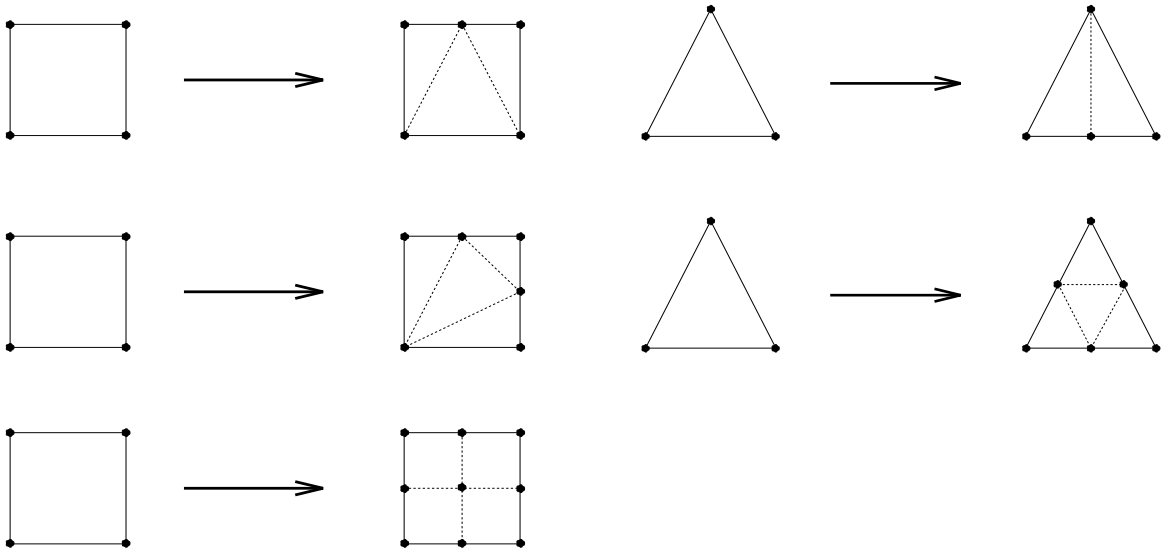


Figure 5: Permitted subdivision types for triangular and quadrilateral faces

Hex Ref 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 1: Resulting Children Cell Types for Various Hexahedral Refinement Patterns

A total of 9 prismatic and 8 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 1 for hexahedral subdivisions, Table 2 for prismatic subdivisions, and Table 3 for pyramidal subdivisions. The required refinement types are

Prism Ref 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 2: Resulting Children Cell Types for Various Prismatic Refinement Patterns

Pyramid Ref 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

Table 3: Resulting Children Cell Types for Various Pyramidal Refinement Patterns

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 in Figure 5. By considering all possible face subdivision patterns for all faces of a given element, a list of all possible cell subdivision types can be constructed. Many of these subdivision 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 which must be taken into account. These are implemented by coding one canonical subdivision routine for each type, and using an orientation mask to reorder the element indices according to the orientation of the subdivision. Isotropic subdivision of all cell types results in 8 smaller cells of the same type as the parent cell, except in the case of pyramidal cells, were isotropic refinement yields 6 pyramidal children, and 4 tetrahedral children.

This implementation differs significantly from other non-tetrahedral adaptive meshing strategies [16, 17]. For example, in [17], a limited number of subdivision types is 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 [16, 17, 18] has not been implemented in the present context.

4 Hanging Edges

The process of defining the various element subdivision 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 subdivision patterns of the parent faces. This is achieved for all subdivision 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 which respect the face subdivision 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 subdividing the parent hexahedron into 3 prisms, 8 pyramids and 2 tetrahedra, which results in the face subdivision pattern shown in Figure 6. This face pattern differs from the desired one by 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 which shares this quadrilateral face.

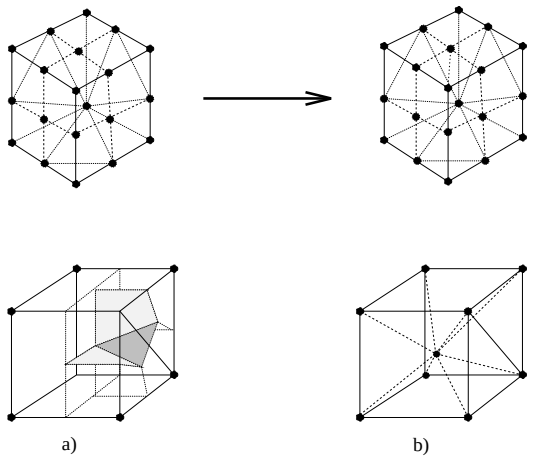


Figure 6: Hanging edge resulting from Type 7 hexahedral refinement and two possible treatments of hanging edge: a) modified dual control-volume, b) insertion of extra vertex into hexahedron with hanging edge

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 which border on a quadrilateral face with a hanging edge. For

example, if the neighboring element is a hexahedron, this cell can be subdivided 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 subdivision pattern. This is similar to the point insertion procedure used in [17] 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 forms the beginning state for subsequent refinements, since 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 this work, both approaches have been implemented. However, the examples shown below have all been performed using the additional point isertion method.

5 Implementation Aspects

Throughout the refinement procedure, new vertices are created either at the center of an edge when an edge is split into two children edges, at the centroid of a quadrilateral face when this face is refined into four smaller quadrilateral faces (c.f. 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 is 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 which 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, but the subdivision 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 to 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 subdivision 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 which 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 which 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% 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 which 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 which 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 coordinate directions. For viscous flow cases, turbulence effects are accounted for using the one-equation model of Spalart and Allmaras [19]. 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 to 4 at each refinement pass, and thus employ 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.

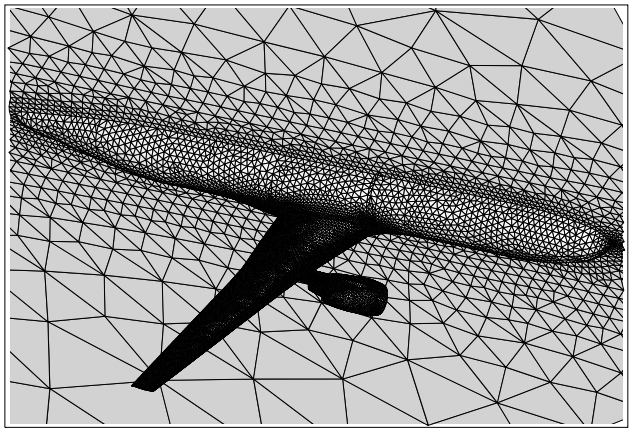


Figure 7: Initial tetrahedral mesh about transport aircraft configuration (number of vertices = 106,000)

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.

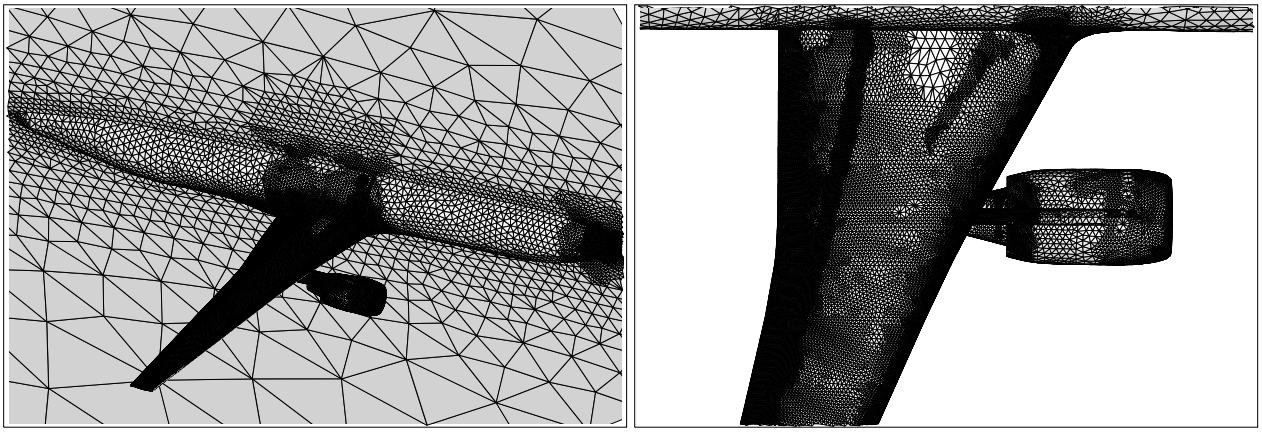


Figure 8: Adaptively refined tetrahedral mesh about transport aircraft configuration after two levels of refinement (number of vertices = 1.3 million)

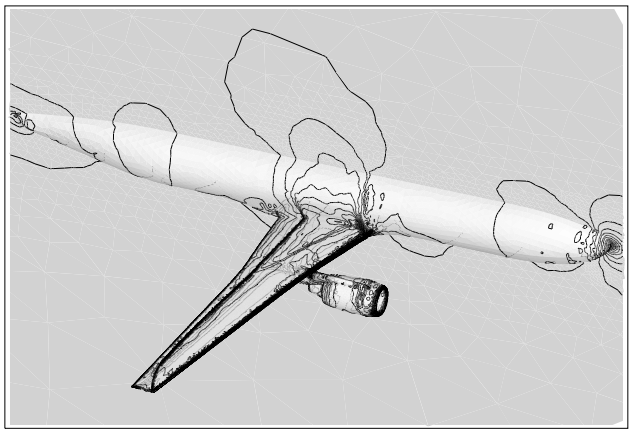


Figure 9: Computed Mach contours on adaptively refined tetrahedral mesh for transport aircraft configuration (Mach = 0.768, incidence = 1.116 degrees)

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 advancing-front method of Pirzadeh [20] 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 degrees. 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 minutes to generate the final mesh, while the flow solver requires 1 Gbyte of memory and 11 hours 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 subset of these cases, block structured meshes can also be handled by treating them as a set of unstructured hexahedra.

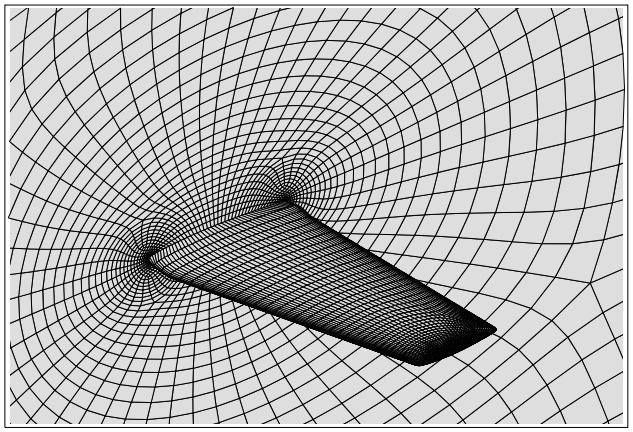


Figure 10: Initial block structured mesh about ONERA M6 wing (number of vertices = 129,187)

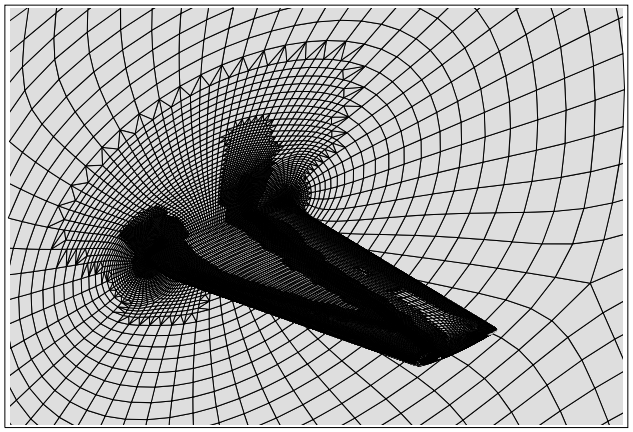


Figure 11: Adapted hexahedral mesh about ONERA M6 wing (number of vertices = 3 million)

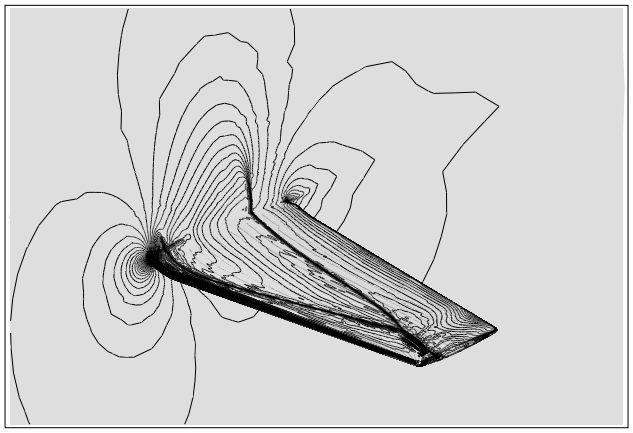


Figure 12: Computed Mach contours on adapted hexahedral mesh (Mach = 0.84, Incidence = 3.06 degrees)

Figure 10 depicts a fully hexahedral block structured mesh about an ONERA M6 wing. This grid was provided by 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 degrees 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 which 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 minutes of CPU time. The flow solver required 1.6 Gbytes of memory and was therefore run on the 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 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 [20]. The tetrahedra in the boundary layer region of this mesh were then merged into prisms using the mesh merging algorithm described in [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 closeup 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 turbulent flow is computed on this mesh at a Mach number of 0.2, a Reynolds number of 3.7 million, and 10 degrees incidence.

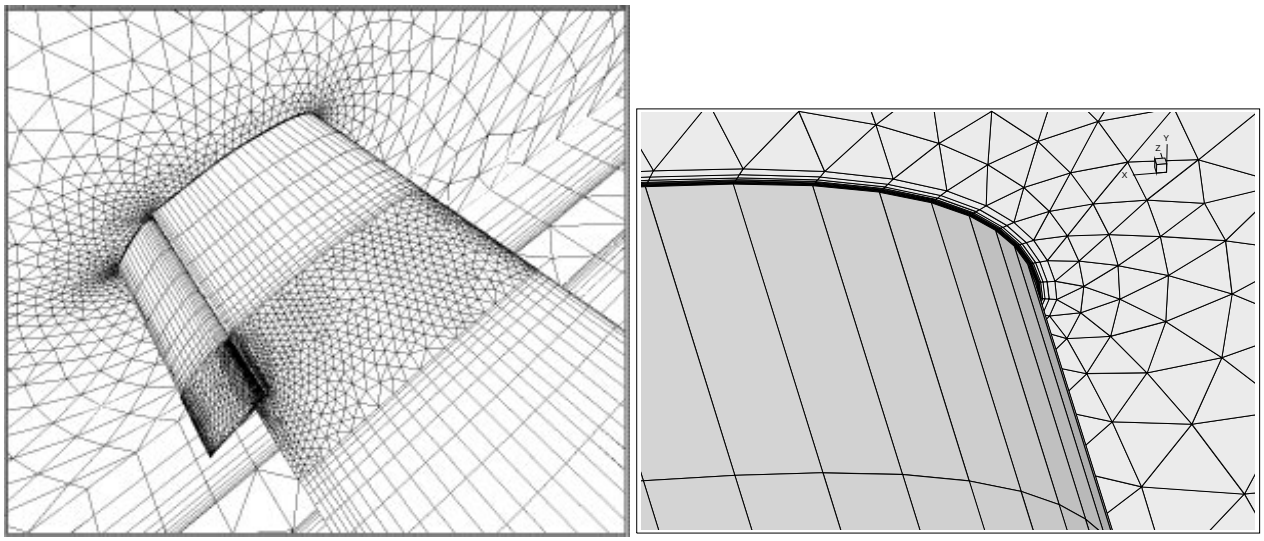


Figure 13: Initial mixed-element mesh for partial-span flap configuration showing hexahedral and prismatic cells in near wall region (number of vertices = 165,000)

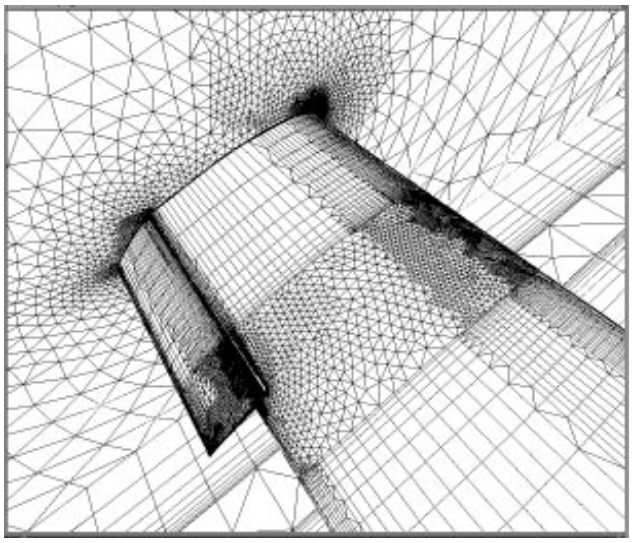


Figure 14: Adaptively refined mixed element mesh for partial-span flap configuration after two levels of refinement (number of vertices = 1.8 million)

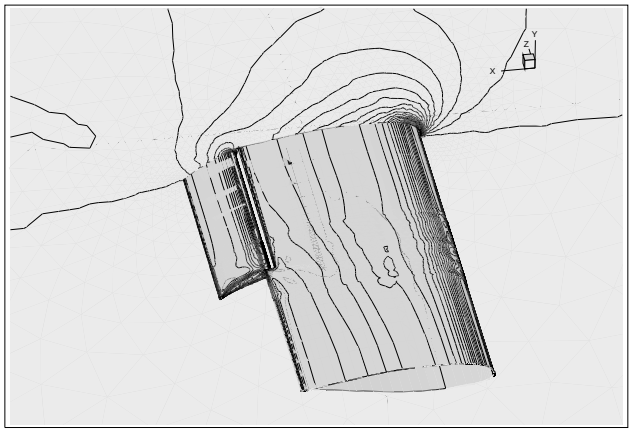


Figure 15: Computed density contours on adapted mixed-element mesh for partial span flap configuration (Mach = 0.2, Reynolds number= 3.7 million, Incidence = 10 degrees)

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

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 which receive additional refinement at each pass, and then examining only those cells and their neighbors at each additional iteration. Derefinement by retracing the hierarchical subdivision 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 in 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 which 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 C-90, 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.

References

[1] D. J. Mavriplis. Three-dimensional multigrid for the Euler equations. *AIAA J.*, 30(7):1753–1761, July 1992.

[2] D. L. Marcum. Generation of unstructured grids for viscous flow applications. AIAA paper 95-0212, January 1995.

[3] C. L. Bottasso, H. L. De Cougny, J. E. Flahery, C. Ozturan, Z. Rusak, and M. S. Shephard. Compresible aerodynamics using a parallel adaptive time-discontinous Galerkin least-squares finite element method. AIAA paper 94-1888, June 1994.

[4] T. J. Barth. Aspects of unstructured grids and finite-element volume solvers for the Euler and Navier-Stokes equations. In *von Karman Institute Lecture Series*, AGARD Pub. R-787, 1992.

[5] R. Löhner and J. D. Baum. Adaptive H-refinement on 3-D unstructured grids for transient problems. *Int. J. Num. Meth. Fluids*, 14:1407–1419, 1992.

[6] R. D. Rausch, J. T. Batina, and H. T. Y. Yang. Spatial adaptation of unstructured meshes for unsteady aerodynamic flow computations. *AIAA J.*, 30(5):1243–1251, 1992.

[7] S. D. Connell and D. G. Holmes. A 3D unstructured adaptive multigrid scheme for the Euler equations. *AIAA J.*, 32(8):1626–1632, 1994.

[8] Y. Kallinderis and P. Vijayan. Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. *AIAA J.*, 31(8):1440–1447, 1993.

[9] R. Biswas and R. Strawn. A dynamic mesh adaptation procedure for unstructured hexahedral meshes. AIAA paper 96-0027, January 1996.

[10] D. J. Mavriplis and V. Venkatakrishnan. A unified multigrid solver for the Navier-Stokes equations on mixed element meshes. AIAA Paper 95-1666, June 1995.

[11] A. Bowyer. Computing Dirichlet tessalations. *The Computer Journal*, 24(2):162–166, 1981.

[12] D. L. Marcum and N. P. Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. AIAA paper 94-1926, June 1994.

[13] G. Spragle and W. A. Smith. Hanging node solution adaptation on hybrid grids. In *Proc. of the Fifth International Conference on Numerical Grid Generation in Computational Field Simulations*, pages 1221–1230, Starkville, MS, April 1996. Mississippi State University. eds. B. K. Soni, J. F. Thompson, J. Hauser, and P. R. Eisman.

[14] M. Aftosmis. Upwind method for simulation of viscous flow on adaptively refined meshes. *AIAA J.*, 32(2):268–277, 1994.

[15] R. Lohner. Finite-element methods in CFD: Grid generation, adaptivity and parallelization. In *von Karman Institute Lecture Series*, AGARD Pub. R-787, 1992.

[16] V. Parthasarathy, Y. Kallinderis, and K. Nakajima. Hybrid adaptation method and directional viscous multigrid with prismatic-tetrahedral meshes. AIAA Paper 95-0670, 1995.

[17] R. Biswas and R. Strawn. Tetrahedral and hexahedral mesh adaptation for CFD problems. *Journal of Applied Numerical Mathematics*, 1997. To appear.

[18] J. van der Vegt. Anisotropic grid refinement using an unstructured discontinuous Galerkin method for the three-dimensional Euler equations of gas dynamics. AIAA paper 95-1657, June 1995.

[19] P. R. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. AIAA Paper 92-0439, January 1992.

[20] S. Pirzadeh. Viscous unstructured three-dimensional grids by the advancing-layers method. AIAA paper 94-0417, January 1994.