AIAA JOURNAL
Vol. 33, No. 3, March 1995

Three-Dimensional Multigrid Reynolds-Averaged Navier–Stokes Solver for Unstructured Meshes

D. J. Mavriplis*
NASA Langley Research Center, Hampton, Virginia 23681

A three-dimensional unstructured mesh Reynolds-averaged Navier–Stokes solver is described. Turbulence is simulated using a single field-equation model. Computational overheads are minimized through the use of a single edge-based data structure, an efficient multigrid solution technique, and the use of multitasking on shared memory multiprocessors. The accuracy and efficiency of the code are evaluated by computing two-dimensional flows in three dimensions and comparing with results from a previously validated two-dimensional code that employs the same solution algorithm. The feasibility of computing three-dimensional turbulent flows on grids of several million points in less than 2 h of wall clock time is demonstrated.

I. Introduction

IN recent years, the use of unstructured meshes has become more widespread for computational fluid dynamics problems. The advantages of unstructured meshes lie in their ability to deal with arbitrarily complex geometries, while providing a natural setting for the use of adaptive mesh enrichment techniques. However, most of the successes of unstructured meshes have been in solving inviscid flows, particularly in three dimensions. Recently, two-dimensional turbulent flow solutions using unstructured meshes have been demonstrated.[1–4] However, few attempts at solving similar three-dimensional flows are known. Although many publications have appeared in the literature concerning three-dimensional unstructured grid Navier–Stokes computations, few if any of these have demonstrated engineering quality solutions for practical aerodynamic flows. The inclusion of viscous terms into an existing unstructured Euler solver is, in fact, a simple proposition. The real challenge is to devise a complete solution strategy capable of resolving complex flows with good accuracy at acceptable cost on highly stretched grids.

The problems associated with unstructured mesh computations of turbulent viscous flows are threefold. Firstly, a suitable mesh with highly stretched elements in the boundary layer and wake regions must be generated. Secondly, a turbulence model capable of operating efficiently on unstructured meshes must be incorporated. Finally, the memory and CPU overheads associated with the solution technique must be low enough to allow for the use of very fine meshes, which are required for meaningful results. Much work has been performed in the two-dimensional setting to alleviate these problems. However, the extension of these ideas to three dimensions has often been hindered by the overwhelming computational overheads incurred by most methodologies.

This paper describes the development of an efficient three-dimensional Navier–Stokes solver. Most of the techniques employed have been developed and demonstrated previously in the two-dimensional setting, and this work involves their extension to three dimensions. By a careful choice of data structures, the use of a rapidly converging multigrid algorithm, and the implementation of parallel processing techniques, a solution technique that incurs acceptable overheads and is capable of dealing with relatively fine grids is obtained.

Received April 2, 1994; presented as Paper 94-1878 at the AIAA 12th Applied Aerodynamics Conference, Colorado Springs, CO, June 20–23, 1994; revision received Nov. 11, 1994; accepted for publication Nov. 21, 1994. Copyright © 1995 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

*Senior Staff Scientist, Institute for Computer Applications in Science and Engineering. Member AIAA.

II. Method Description

A. Discretization and Data Structures

We seek steady-state solutions to the Favre-averaged Navier–Stokes equations. These equations must be closed using a suitable turbulence model to model the Reynolds-stress terms. Spatial discretization of the Navier–Stokes equations is performed using a Galerkin finite element approach. The conserved flow variables are stored at the vertices of the mesh, and the convective fluxes are assumed to vary linearly over each tetrahedral element. For the viscous terms, velocities are assumed to vary linearly over the tetrahedral elements. Velocity gradients can thus be constructed at element centers, which then enables the discretization of the second derivatives contained in the viscous terms. Additional artificial dissipation terms are constructed as a blend of Laplacian and biharmonic operators. The Laplacian dissipation results in local first-order accuracy and is thus triggered only in the vicinity of shock waves, whereas the third-order accurate biharmonic dissipation is employed throughout the flowfield.

The natural data structure that arises from a finite element point of view is the element data structure, in which a list of elements is stored, with pointers for each element identifying the four vertices that constitute that element. It has previously been shown, in the context of inviscid flow calculations, that the convective terms can be assembled using an edge-based data structure, which both is more compact (in terms of memory overheads) and minimizes the amount of gather–scatter required on vector and parallel computer architectures. The basic data structure for assembling the convective terms is thus a list of edges. For each edge we store the addresses of the two endpoints of the edge and three coefficients, which represent the x, y, and z components of the normal of the face of the dual mesh pierced by the edge, as shown in Fig. 1, for the two-dimensional case. In three dimensions, if one considers all tetrahedra that share a given edge, as shown in Fig. 2, the face normal associated with this edge can be computed as the sum of all of the tetrahedral faces that touch only one of the two endpoints of the edge (i.e., sum of all of the F_e in Fig. 2).

When employing a Galerkin finite element discretization, the viscous terms are traditionally thought of as a sequence of two loops: one to construct gradients at triangle or tetrahedron centers and another to form the final residual contributions. However, the final discrete viscous terms obtained in this manner form a nearest neighbor stencil. The viscous terms for a vertex i depend only on values at i and at vertices k, such that k is joined to i by a mesh edge. Thus, an edge-based data structure may also be employed to assemble the viscous terms. This fact has previously been pointed out in several references.[2,5,6] In Ref. 2, a complete derivation of the edge-based coefficients for a Hessian matrix is given. In three dimensions, this would require the storage of nine coefficients per edge, since the

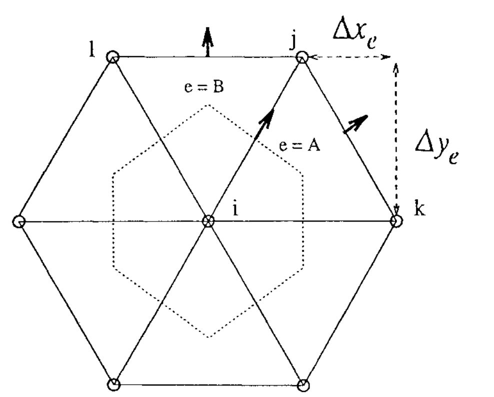

Fig. 1 Internal control volume (dotted line) associated with vertex *i* and face associated with edge *i-j*. The face normal (bold arrow) is one-third of the sum of the two outer edge normals.

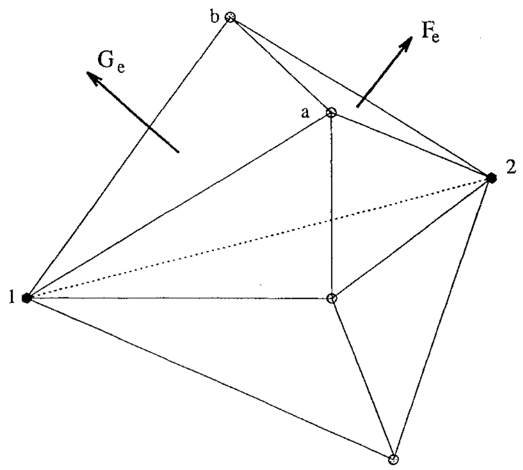

Fig. 2 Illustration of tetrahedra considered for formation of coefficient associated with edge 1-2; F_e and G_e represent face area normals for the two exposed faces of each tetrahedron *e*.

discrete Hessian is written as

$$\begin{bmatrix} u_{xx} & u_{xy} & u_{xz} \\ u_{yx} & u_{yy} & u_{yz} \\ u_{zx} & u_{zy} & u_{zz} \end{bmatrix} = \frac{1}{\mathrm{Vol}_{cv}} \sum_{k=1}^{\text{neighbors}} \begin{bmatrix} \alpha_{xx} & \alpha_{xy} & \alpha_{xz} \\ \alpha_{yx} & \alpha_{yy} & \alpha_{yz} \\ \alpha_{zx} & \alpha_{zy} & \alpha_{zz} \end{bmatrix} (u_i - u_k) \tag{1}$$

where Vol_{cv} represents the volume of the union of tetrahedra that touch vertex i. However, the local edge-based coefficient matrix is symmetric about the diagonal. A proof of this is given in the Appendix. Thus, we need only store six coefficients per edge for the discretization of the viscous terms. If the cross derivative terms in the Navier–Stokes equations are neglected, this can be reduced to three coefficients per edge. Also note that for the discretization of a Laplacian, a single coefficient per edge is required, which is given by the sum of the diagonal terms. Thus, if we adopt the thin-layer form of the Navier–Stokes equations, a single edge coefficient is sufficient to compute the viscous terms.

At this stage, the full Navier–Stokes terms have been included, and thus six edge coefficients are stored. It is useful to examine the amount of storage this represents compared with other approaches. Tetrahedral unstructured meshes of N points contain αN tetrahedra and $(\alpha + 1)N$ edges, neglecting boundary effects. The value of α depends on the mesh but usually lies between 5 and 6. The traditional element data structure requires the storage of the four corners of each tetrahedron. Taking $\alpha = 6$, this corresponds to $24N$. Excessive storage is required if a double loop is employed, with the cell gradients stored as intermediate values (at least nine extra values per cell or $54N$). A single loop over the tetrahedra can, however, be used to assemble the full viscous terms. Within the loop, the gradients may be computed, divided by the volume, multiplied by the appropriate face normal, and then accumulated to one of the four corner vertices of the cell. The last two operations are repeated for each corner vertex of the cell. This approach requires only the storage of the element data structure plus the vertex coordinates (which are not required in the edge-based approach), thus a total of $27N$. The edge-based approach requires a total of $42N$ or $15N$ more than the element-based approach. However, the element approach requires an order of magnitude more operations since the geometric quantities such as cell volume and face normals must be recomputed each time in the loop. If any of these quantities are stored rather than recomputed, the storage requirements quickly exceed those of the edge-based approach. This is a classic example of a memory-CPU-time tradeoff.

Since the viscous terms of the Navier–Stokes equations are not strictly second derivatives of a given quantity, but contain terms such as

$$\nabla_x(\mu \nabla_x u) \tag{2}$$

an exact implementation of a Galerkin discretization requires the evaluation of the viscosity at the tetrahedra centroids in the construction of the edge coefficients. If the viscosity is not constant, the edge coefficients cannot be computed in a preprocessing phase. Thus some approximation must be made to enable the use of preprocessed edge coefficients. The simplest approach is to approximate the viscosity for each edge contribution as the average of the two values at each end of the edge. Thus the viscous terms are computed as

$$\begin{bmatrix} (\mu u_x)_x & (\mu u_x)_y & (\mu u_x)_z \\ (\mu u_y)_x & (\mu u_y)_y & (\mu u_y)_z \\ (\mu u_z)_x & (\mu u_z)_y & (\mu u_z)_z \end{bmatrix} = \frac{1}{\mathrm{Vol}_{cv}} \sum_{k=1}^{\text{neighbors}} \begin{bmatrix} \alpha_{xx} & \alpha_{xy} & \alpha_{xz} \\ \alpha_{yx} & \alpha_{yy} & \alpha_{yz} \\ \alpha_{zx} & \alpha_{zy} & \alpha_{zz} \end{bmatrix} \frac{(\mu_i + \mu_k)}{2}(u_i - u_k) \tag{3}$$

It is important to realize that this discretization does not correspond exactly to the Galerkin discretization. Only under certain conditions do the two formulations become equivalent (e.g., linear viscosity variations and regular triangulations). Another approach is to rewrite the viscous terms as

$$\nabla(\mu \nabla x) = \mu \nabla^2 x + \nabla \mu \nabla x \tag{4}$$

and evaluate the double gradient term at each vertex using a nearest neighbor stencil. Gradient calculations of this type can be performed using the convective edge-based coefficients. Both of these approaches have been tested in two dimensions, and virtually no difference in the final solution could be seen. These modifications are necessary for the viscosity in the momentum equations, the values of μx velocity in the energy equation, as well as for the eddy viscosity in the turbulence equation. The effect of the eddy viscosity variation is expected by far to be the most important of all for practical turbulent flows. In the present work, the formulation of Eq. (3) is employed exclusively.

Finally, another approach that has been suggested for computing the viscous terms[6] consists of forming the gradients of velocity at each vertex, using the nearest neighbor stencil of each vertex, and forming the second derivatives of the viscous terms by reapplying the same integration to the first derivatives. This approach is attractive because it requires little additional storage. All operations can be performed using the edge-based coefficients required for the convective terms. However, it can easily be seen that, on a one-dimensional mesh of spacing h, this scheme reduces to a second difference on a stencil of size $2h$. This will result in lower accuracy and possible odd-even decoupling. Since packing enough points into the viscous layers is generally one of the main difficulties associated with viscous flow computations, a scheme that operates on every other point is highly undesirable. This scheme should therefore be rejected.

B. Turbulence Modeling

The one-equation turbulence model of Spalart and Allmaras[7] has been implemented in the present solver, since this approach avoids the complications involved in implementing algebraic models on unstructured meshes,[8] is reasonably robust and inexpensive, and has been shown to yield favorable results in three-dimensional aerodynamic flows.[9]

The turbulence equation is also discretized using the edge-based data structure. The convective terms are treated using a first-order upwind formulation, and the diffusion terms are treated analogously to the viscous terms in the flow equations. The particular discretization ensures positivity of the discrete solution. The turbulence equation is advanced in time using a point-implicit treatment.[10] For a single equation, this corresponds to a Jacobi iteration. This scheme is attractive since it ensures positivity of the turbulence equation variable, not only at steady state, but throughout the convergence process. The turbulence equation is effectively solved decoupled from the flow equations (using different local time-step values). The convergence of both the turbulence equation and the flowfield equations is accelerated using an unstructured multigrid algorithm. The turbulence and flow equations are only coupled on the finest grid of the multigrid sequence, where the eddy viscosity values from the turbulence model are fed back into the flow solution.

This particular field-equation turbulence model requires information concerning the distance of each grid point to the closest wall boundary. The simplest way of computing this distance function is to compute the normal distance from a given grid point to all boundary faces and preserve the minimum distance found. This results in an $\mathcal{O}(N^{5/3})$ algorithm, where N represents the number of grid points. For fine grids, this becomes excessively costly. Although more sophisticated search techniques exist for reducing this to $\mathcal{O}(N \log N)$, a simple solution is to compute the distance function on the second finest mesh of the multigrid sequence and then to interpolate these values to the finest grid. This alone reduces the cost of computing the distance function by a factor of 32.

C. Solution Technique

The flow equations and turbulence equations are advanced in time to obtain the steady-state solution. The flow equations are advanced using a multistage Runge–Kutta explicit scheme, whereas the turbulence equation is solved as described earlier. Local time stepping and residual averaging are employed to accelerate the convergence of the flow equations.

An unstructured multigrid technique is employed to further accelerate the convergence of both the flow and turbulence equations. This technique, which has previously been demonstrated for the Navier–Stokes equations in two dimensions[1] and for the Euler equations in three dimensions,[5] employs a set of nonnested coarse and fine meshes. In a preprocessing step, the indirection arrays that correspond to the restriction and prolongation operators (interpolation of variables, residuals, and corrections) between each successive pair of grids are constructed, using an efficient search algorithm. On domain boundaries, which may not coincide between the various grids due to the discretization of curved surfaces that constitute the boundaries, the search and interpolation procedures are carried out in the parametric space that defines the surface patches of the boundary geometry. Linear interpolation is used to transfer flow variables, residuals, and corrections between the various meshes of the multigrid sequence. This requires the storage of four addresses and four coefficients per mesh point. Once these operators have been constructed and stored, the intermesh multigrid transfers can be implemented as a simple gather–scatter of array elements within each multigrid cycle.

The unstructured multigrid algorithm incurs approximately a 30% memory overhead, mostly due to the extra storage required for the coarse grids of the multigrid sequence, and requires roughly 90% more CPU time per cycle, but results in convergence rates that are an order of magnitude higher than the single grid solver.

D. Parallel Processing

The use of one basic and simple data structure and the choice of an explicit scheme augmented with multigrid enable a relatively simple and efficient parallel implementation of the solver on shared and distributed memory parallel architectures. Previous experience with distributed memory machines has been less than satisfactory. Such implementations require considerable effort and have not been able to demonstrate superior performance for this class of problems.[11,12] The shared memory vector-parallel architecture of the Cray YMP-C90 has proved to be very effective for the current type of problems. For example, the 16 processor Cray YMP-C90 was found to provide more than double the performance of the Intel Delta machine using 512 processors on a three-dimensional unstructured multigrid Euler solver.[11] Furthermore, the shared memory architecture and the parallel compiler support available on Cray machines enable a relatively simple implementation using standard Fortran 77. Parallelization on shared memory architectures is imperative for large cases, not only to speed up solution time, but also to avoid idling processors in a time-sharing environment when jobs become large enough to fill the main memory of the machine.

The majority of the work in the present solver involves loops over edges. These loops must be both vectorized and parallelized. Since multiple edges meet at each mesh vertex, the loops contain data dependencies that inhibit both vectorization and parallelization. To vectorize these loops, the list of edges is split into subgroups, or colors, such that within each color no vertex dependencies exist. The overall loop is hence transformed into an outer loop over all colors and an inner vectorizable loop within each color. A simple parallelization strategy is to further divide the colored groups into subgroups that can be computed in parallel. This is automatically done at compile time by the autotasking compiler provided the appropriate compiler directive is specified at the beginning of each loop. The subgroups are then distributed over all processors, taking advantage of the complete vector and parallel power of the machine.

For the inviscid version of the present solver, speedups of 13 to 14 on 16 processors have been observed in a dedicated environment, indicating a degree of parallelism of over 99% has been achieved according to Amdahl's law. The viscous cases presented in the results section, however, were run in a time-sharing environment and yielded speedups between 10 and 12 on 16 processors. Benchmarking of the full viscous solver in a dedicated environment is planned for the near future.

III. Results

A. Single Segment Wing

To validate the three-dimensional Navier–Stokes solver, a two-dimensional turbulent flow over a wing geometry with no spanwise variation has been computed, and the results compared with the solution from a two-dimensional unstructured flow solver on an equivalent grid. The particular two dimensional solver employed has been previously described and validated and is routinely used in production environments.[1,13] This two-dimensional solver employs the equivalent discretization, solution technique, and turbulence model as the three-dimensional code described in this paper. Therefore the two codes should give nearly identical results for purely two dimensional cases.

The first test case involves the transonic flow over a wing of aspect ratio 2 with no sweep or spanwise variation. The wing section (independent of span location) is an RAE 2822 airfoil. The three-dimensional grids employed for computing the flow over this wing geometry are displayed in Figs. 3a–3d. They are formed by first constructing a two-dimensional unstructured grid about an RAE 2822 airfoil, using the method described in Ref. 14. The two-dimensional mesh is then stacked in the spanwise direction, thus forming a mesh of spanwise prisms. This prismatic mesh is then converted into a tetrahedral mesh by dividing each prism into three tetrahedra using a variant of the prism division algorithms reported in Refs. 15 and 16. The resulting geometry consists of a wing with a symmetry plane at both ends of the wing. There is thus no wing tip present and no spanwise variation whatsoever. This can be thought of as a typical wing-in-wind-tunnel two-dimensional test.

The finest mesh for this case is depicted in Fig. 3a. The entire mesh contains 1.04 million points and 6 million tetrahedra. The mesh is formed by 33 spanwise stations with 31,571 grid points at each station. The normal mesh spacing at the wing surface is 10^{-5} chords,

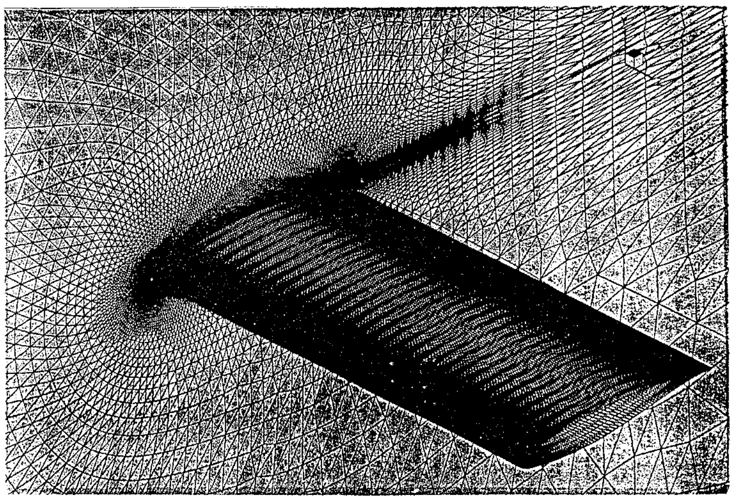
Fig. 3a Fine grid employed for the RAE 2822 wing test case (1.04 million points, 6 million tetrahedra, and wall spacing 10^{-5} chords).

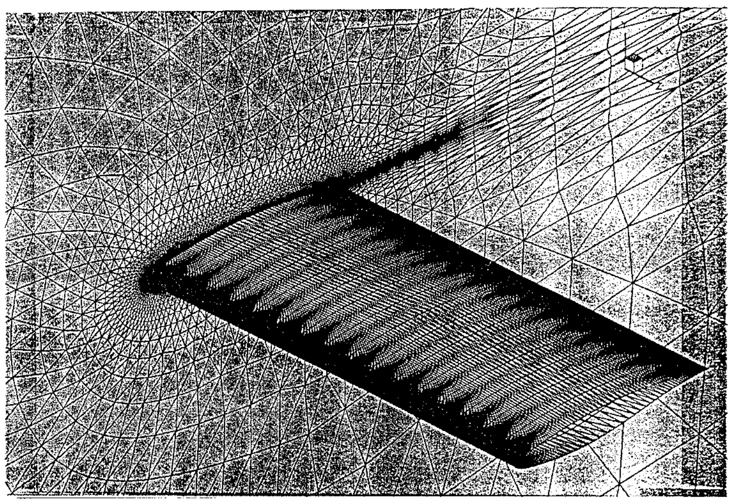
Fig. 3b Second finest grid employed for the RAE 2822 wing test case (135,000 points).

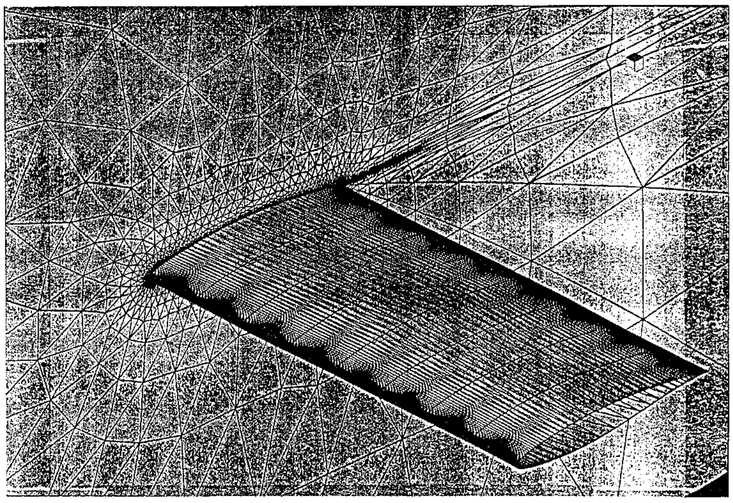
Fig. 3c Third grid employed for the RAE 2822 wing test case (23,000 points).

which results in cell aspect ratios on the order of 500 : 1 in these regions. A total of five meshes were used in the multigrid sequence for this case. Four of these meshes are depicted in Figs. 3a–3d. Each coarser mesh contains a factor of approximately eight fewer points than the previous mesh, and consecutive meshes are generally nonnested.

The freestream Mach number for this case is 0.73, the incidence is 2.79 deg, and the Reynolds number is 6.5×10^6. The solution is depicted qualitatively in Fig. 4 as a plot of density contours on the surface of the wing and symmetry walls. The lack of any spanwise variation of the contours on the wing indicates the presence of purely two-dimensional flow. The flow is transonic, and a normal shock is observed slightly aft of the midchord location. Figure 5 provides a more quantitative picture of this solution. The computed surface pressure at the midspan location is compared with experimental data as well as with the computed results of the two-dimensional code on an equivalent station grid of 31,571 points. Both the two- and three-dimensional codes tend to underpredict the lift compared with the

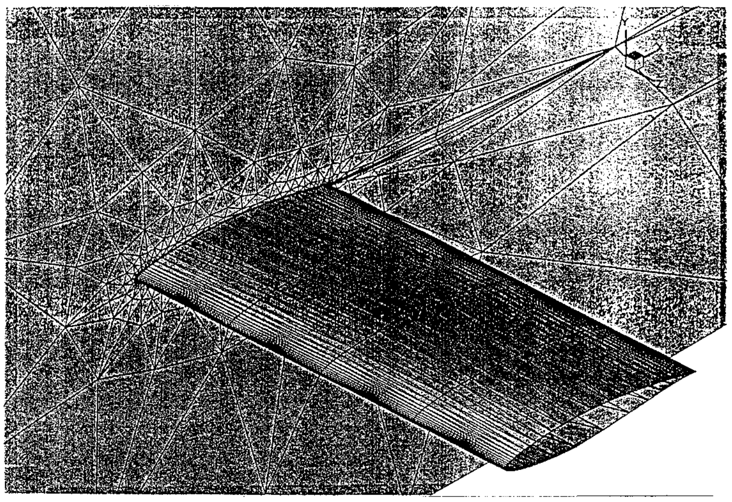

Fig. 3d Fourth grid employed for the RAE 2822 wing test case (2610 points).

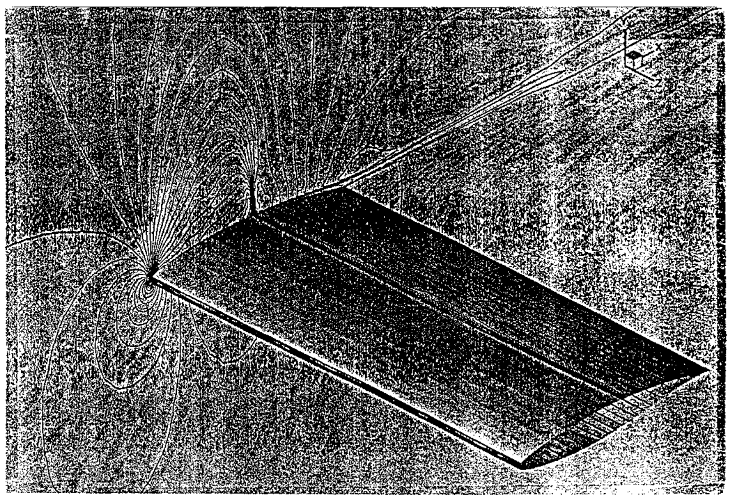

Fig. 4 Computed density contours for the RAE 2822 wing (Mach = 0.73, *Re* = 6.5 × 10^6, and incidence = 2.79 deg).

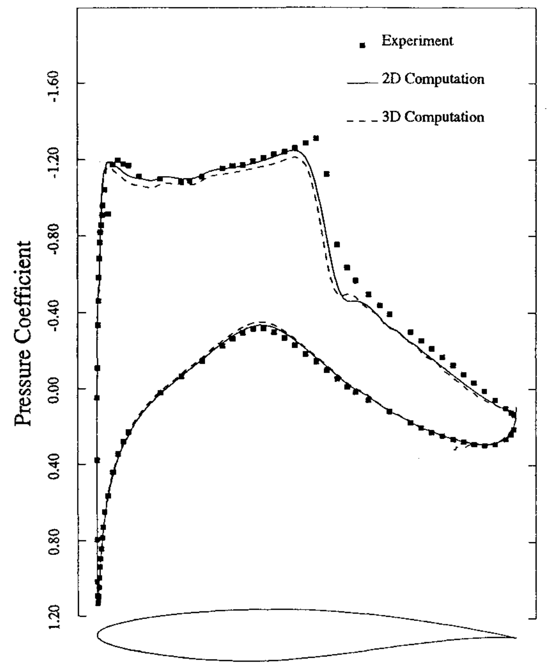

Fig. 5 Comparison of computed and experimental surface pressure for RAE 2822 wing (Mach = 0.73, *Re* = 6.5 × 10^6, and incidence = 2.79 deg).

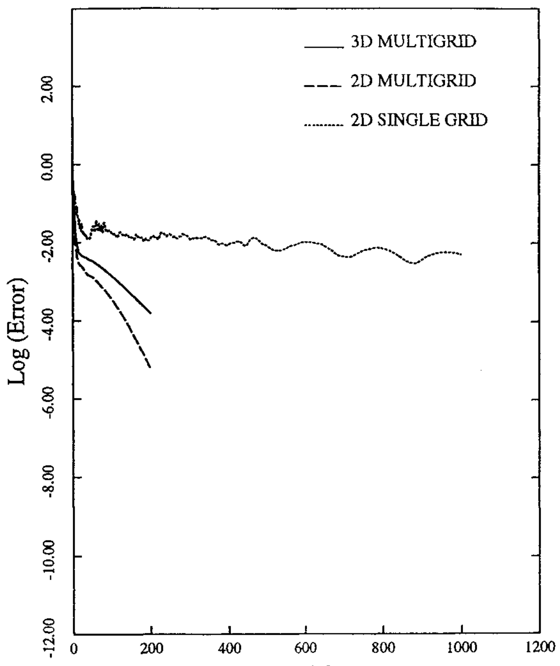

Fig. 6 Computed density contours for the swept and tapered RAE 2822 wing.

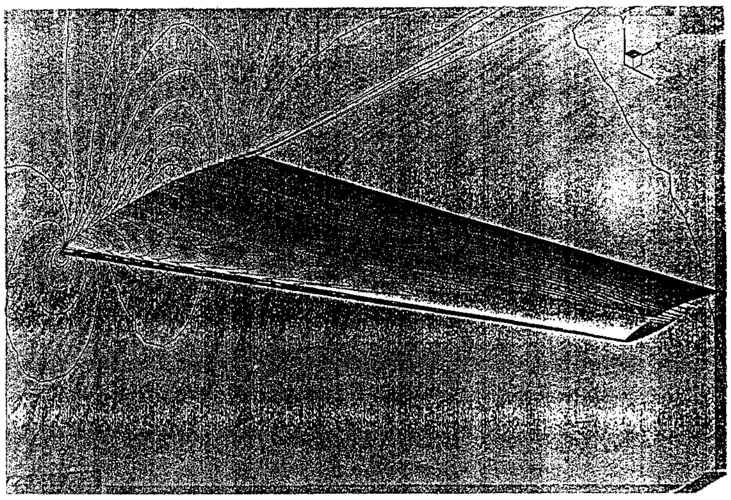

Fig. 7 Comparison of observed convergence rates for the three-dimensional RAE 2822 wing multigrid case and the corresponding two-dimensional case both with and without multigrid acceleration.

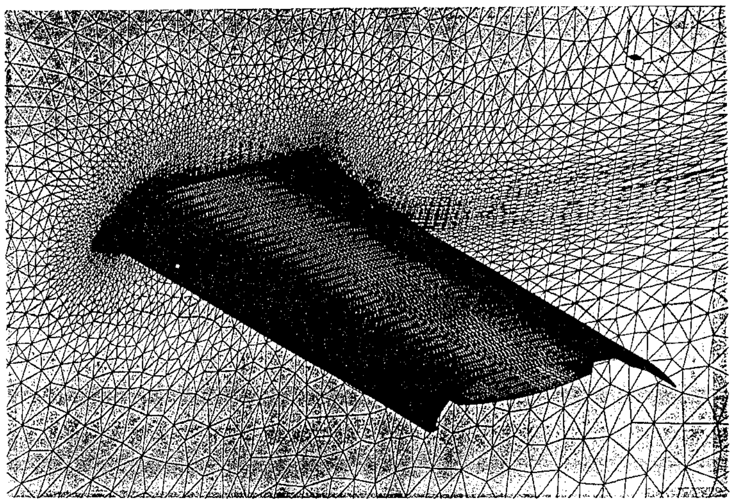

Fig. 8 Fine grid employed for the three-element wing test case (1.84 million points, 10.6 million tetrahedra, and wall spacing 10^{-6} chords).

experimental data, a fact that is attributed to the turbulence model employed. For example, the same two-dimensional code achieves a lift value some 10% higher using the Baldwin–Lomax model. However, the two- and three-dimensional flow solutions agree very well with each other. The three-dimensional solution is slightly more diffusive than the two dimensional solution, which is attributed to the presence of extra spanwise dissipation, which is nonzero even in a two-dimensional flow, due to the presence of diagonal edges in between neighboring spanwise stations.

The convergence rate for this case is plotted in Fig. 6. The residuals are seen to be reduced by almost 4 orders of magnitude over 200 multigrid cycles, for an average residual reduction of 0.957. This is comparable but somewhat slower than the rate of 0.943 achieved by the two-dimensional multigrid code, which is also plotted for comparison purposes in the same figure. Finally, the single grid convergence rate of the two-dimensional code is also shown in the figure. A residual reduction of only 2 orders of magnitude over 1000 cycles is achieved for the single grid approach. Thus the multigrid procedure converges over 10 times faster than the single grid code for the two-dimensional case. Since the three-dimensional multigrid convergence rate is close to that of the two-dimensional case, one can conclude that gains of similar magnitude are afforded by the multigrid algorithm in three dimensions. The three-dimensional single grid convergence is not plotted due to the excessive computer costs required for such a run with obvious conclusions.

To demonstrate the three-dimensional capability of the present code, the wing of the previous case (aspect ratio 2) is given a sweep of 30 deg and a taper of 0.5. This spanwise variation results in fully three-dimensional flow. The flow over the swept and tapered wing at the same conditions is computed using equivalent grids to those described earlier (same point densities). Figure 7 depicts the solution in terms of density contours on the wing and symmetry wall surfaces. The convergence rate for this case is almost identical to that displayed in Fig. 6 for the unswept three-dimensional wing and is therefore not displayed.

For both of these cases a total of 177 Mwords of memory was required. This translates to 170 words per fine-grid vertex. This includes all of the arrays for the coarse grid variables of the multigrid sequence. Both of these cases required 75 s of CPU time per multigrid cycle or a total time of 4.2 single CPU hours on the Cray-YMP-C90 machine. When run in multitasking mode on all 16 processors of the machine, the total aggregate CPU time rose to 5 h but the jobs were executed in 28 min of connect time, as indicated by the batch job log file, in a time-sharing environment in which only 60% of the machine was dedicated to this particular job. In a dedicated environment, these jobs can be expected to execute in jus over 15 min.

B. Three-Element High-Lift Wing

The next test case consists of flow over a high-lift wing configuration with a slat and a single slotted flap. The wing has an aspect ratio of 2 with no sweep or spanwise variation. The wing section (independent of span location) is a Douglas three-element airfoil, whicl has been extensively tested both numerically and experimentally.[1] The three-dimensional fine grid employed for computing the flov over this wing geometry is displayed in Fig. 8. This grid is forme

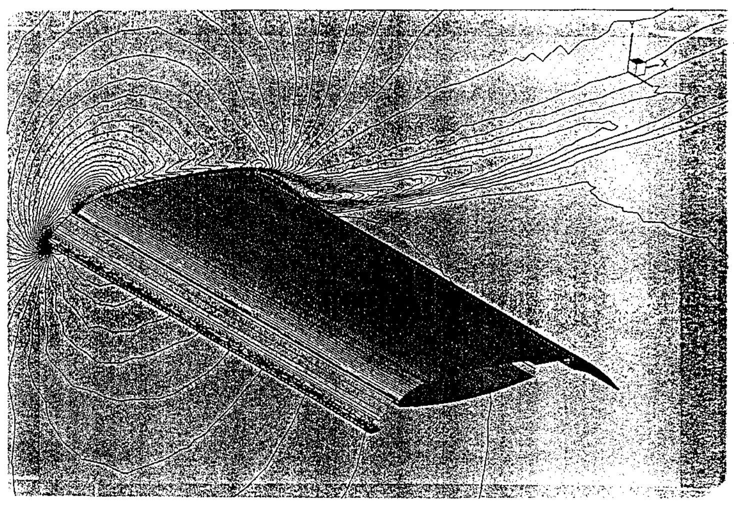

Fig. 9 Computed solution for the three-element wing test case. Mach contours are displayed on the symmetry plane and density contours on the wing surfaces (Mach = 0.2, *Re* = 9 × 10^6, and incidence = 16.21 deg).

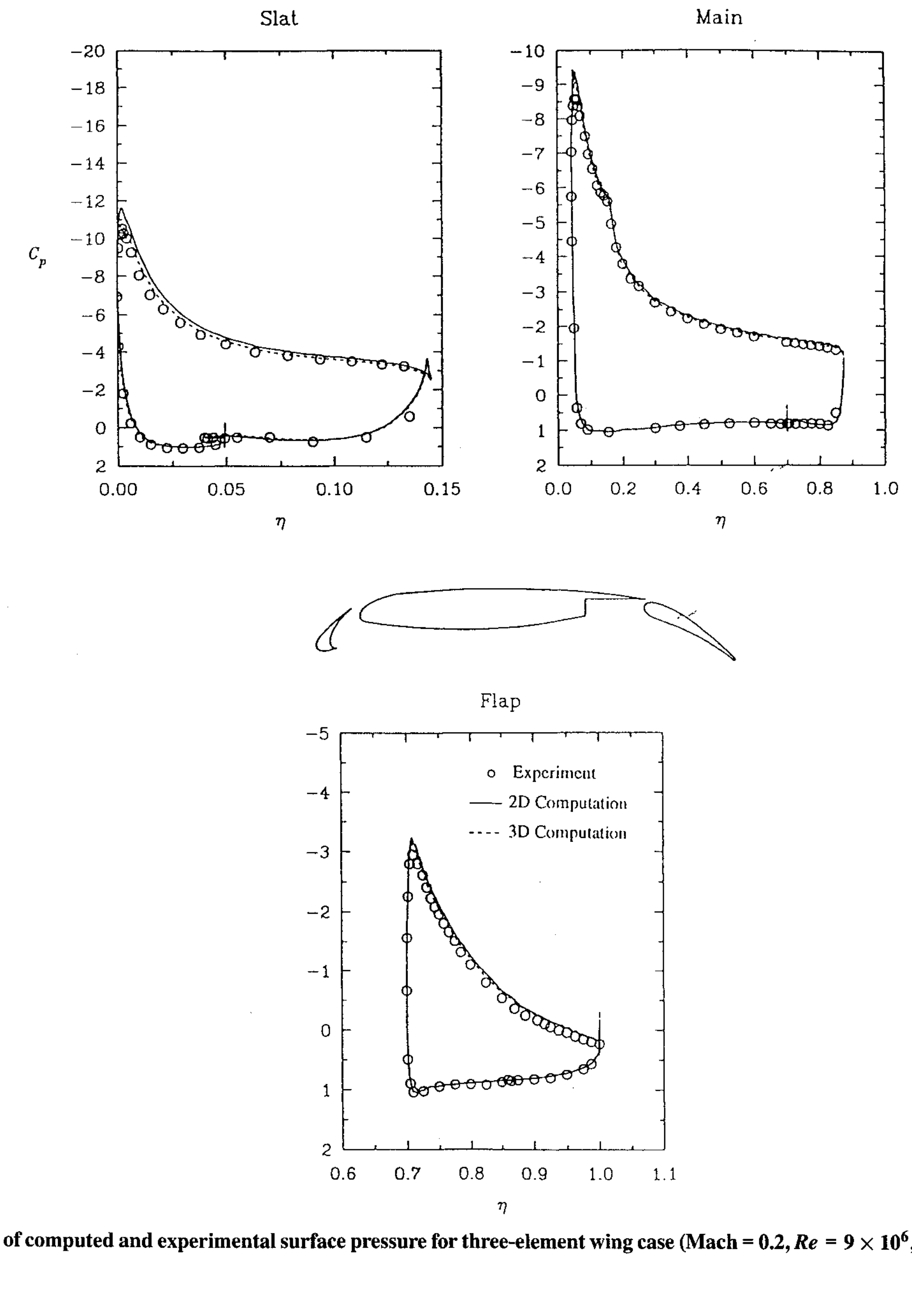

Fig. 10 Comparision of computed and experimental surface pressure for three-element wing case (Mach = 0.2, *Re* = 9 × 10^6, and incidence = 12 deg).

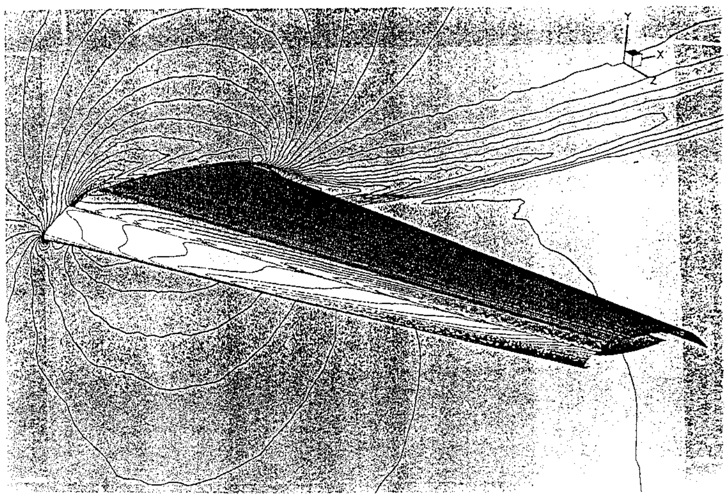

Fig. 11 Computed solution for the three-element swept and tapered wing test case. Mach contours are displayed on the symmetry plane and density contours on the wing surfaces (Mach = 0.2, *Re* = 9 × 10^6, and incidence = 12 deg).

by first constructing a two-dimensional unstructured grid about the three-element airfoil using the method described in Ref. 14 and then stacking the grids and subdividing the resulting prismatic elements, as described previously. The resulting geometry can be thought of as a typical wing-in-wind-tunnel two-dimensional test, with symmetry end walls at both extremities of the wing.

The finest mesh for this case contains 1.84 million points and 10.6 million tetrahedra. The mesh is formed by 33 spanwise stations with 55,865 grid points at each station. The normal mesh spacing at the wing surface is 10^{-6} chords, which results in cell aspect ratios greater than 1000:1 in these regions. A total of five meshes were used in the multigrid sequence for this case. Each coarser mesh contains a factor of approximately 8 fewer points than the previous mesh, and consecutive meshes are generally nonnested. This level of mesh resolution corresponds to the equivalent minimum required resolution for adequate performance prediction using the two-dimensional code, as determined by a grid resolution study.[13]

The freestream Mach number for this case is 0.2, the incidence is 16.21 deg, and the Reynolds number is 9 × 10^6. The flow is assumed to be fully turbulent; thus no transition points are specified. The solution is depicted qualitatively in Fig. 9 as a plot of density contours on the surface of the wing and Mach contours on the surface of the symmetry wall. The lack of any spanwise variation of the contours on the wing indicates the presence of purely two-dimensional flow. The computed surface pressure at the midspan location is compared with experimental data as well as with the computed results of the two dimensional code on an equivalent station grid of 55,865 points in Fig. 10. Excellent agreement between the experimental data and the two- and three-dimensional codes is observed.

To demonstrate the three-dimensional capability of the present code, the segmented wing of the previous case is given a sweep of 30 deg, and a taper of 0.5. This spanwise variation results in fully three-dimensional flow. The freestream Mach number is 0.2 as previously, but the incidence is lowered to 12 deg. The flow over the swept and tapered wing (aspect ratio 2) is computed using equivalent grids to those described earlier (same point densities). Figure 11 depicts the solution in terms of density contours on the wing surfaces and Mach contours on the wall surface. The wall Mach contours are qualitatively similar to those displayed in Fig. 9 for the previous case, and spanwise variation of the wing surface contours is noted, indicating the presence of fully three-dimensional flow.

In Fig. 12, the multigrid convergence rates of the two-dimensional code and the three-dimensional unswept-wing and swept-wing runs are compared. The two-dimensional run and the three-dimensional unswept-wing run converge at very similar rates, achieving a residual reduction of 4.5–5 orders of magnitude over 300 cycles. The three-dimensional swept-wing run converges somewhat slower than the previous two runs but still achieves a similar level of residual

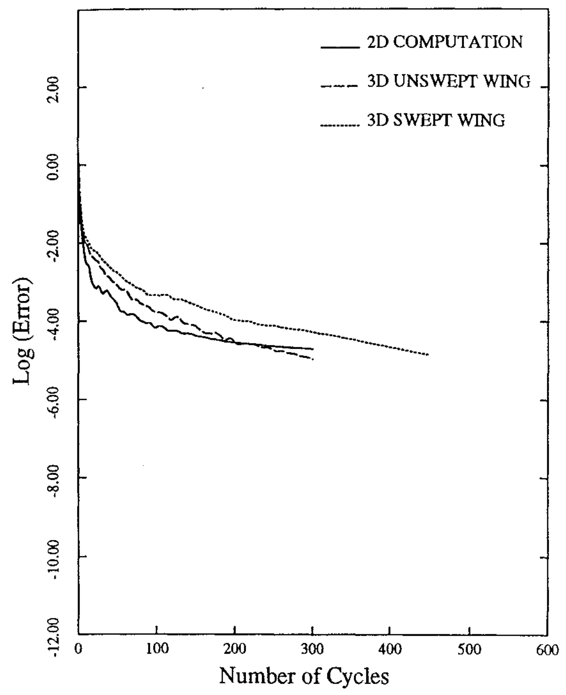

Fig. 12 Comparison of three-dimensional multigrid convergence rates for straight wing and swept wing vs convergence rate of the two-dimensional multigrid solver.

reduction over 450 cycles. This similarity between the two dimensional and three-dimensional convergence rates is a good indication that the full benefit of the multigrid algorithm has been achieved for three-dimensional flows.

Both of the preceding cases required a total of 312 Mwords of memory and 140–175 s of CPU time per multigrid cycle on the Cray-YMP-C90 machine, depending on the amount of concurrency achieved during a particular run in the time-sharing environment. A typical 450 cycle run on all 16 processors required 1.9 h of connect time, as indicated by the batch job log file, in a time-sharing environment in which 65% of the machine was dedicated to this particular job. In a dedicated environment, such a job can be expected to execute in approximately 1.25 h.

IV. Conclusions

The aim of this paper has been to demonstrate the feasibility of computing turbulent viscous flows over complex geometries on fine grids using the unstructured mesh approach. Although it is recognized that this capability is expensive, it is entirely feasible from a technical standpoint. This is made possible by several factors:

1) the reduction of memory overheads through the use of a single efficient data structure, 2) the implementation of a rapidly converging multigrid algorithm, 3) the use of parallel processing, and 4) the availability of a large central memory machine with multiple rapid processors and simple-to-use parallelization tools.

The multigrid convergence rates demonstrated here are comparable to those obtained by the equivalent two-dimensional code,[13] as well as those obtained by structured-grid Navier–Stokes solvers.[17] For inviscid flow solutions, more rapid convergence has been demonstrated with the current algorithm. Residual reductions of 6 orders of magnitude in 100 cycles are generally possible for large inviscid flow calculations (see Ref. 11, for example). The main cause of this convergence degradation for viscous flows relates to the amount of grid stretching present in the meshes required for resolving the viscous layers. In fact, in two dimensions, the convergence rate is observed to degrade progressively as higher grid stretchings are employed on a given geometry. The use of semi-coarsening strategies to alleviate this problem are currently under investigation. Since no relation between the various grids of the multigrid sequence is assumed, more optimal coarse grids may be constructed without modifying the overall multigrid algorithm.

The code currently requires approximately 170 words per fine-grid vertex, which includes all of the coarse grid information. It is estimated that this can be further reduced by 10–15%. The code runs at 320 Mflops on a single Cray YMP-C90 processor and requires 72 μs per vertex per multigrid cycle, with 200–400 cycles usually required for convergence. Predicted performance on a dedicated 16 processor machine is 4 Gflops and 5.8 μs per vertex per cycle, corresponding to a speedup of 13. Thus, on the current maximum configuration of the machine (1 Gword, 16 processors), a case of 6 million grid points could be computed in 2–4 h. Of course, whether such a computation is worth its cost in an industrial environment is another matter.

One of the main obstacles to employing this capability in a production environment remains the grid generation process. The ability to reliably generate very fine highly stretched three-dimensional tetrahedral grids is still under development, and recent progress has been made in this area.[15,16] At this stage, the flow solver has been demonstrated by constructing three-dimensional unstructured grids using a two-dimensional stacking procedure. This grid generation procedure severely limits the types of three-dimensional configurations that can be handled. In this work, validation has been confined to the computation of two-dimensional flows in three dimensions. Although the three-dimensional swept-wing results do little to validate the three-dimensional accuracy of the code, they have been included to demonstrate the performance of the solution strategy for fully three-dimensional cases. In the future, solutions over more complex geometries such as partial flap wings and complete aircraft configurations will be attempted. This will require interfacing with a more general grid generation procedure.

Appendix: Symmetric Edge-Based Coefficients for Viscous Terms

The three-dimensional viscous terms for the Navier–Stokes equations are formed as given by Eq. (3) in Sec. II.A, where the edge-based coefficients are given as

$$\alpha_{ij} = \sum_e \frac{F_i G_j}{\text{vol}_e} \tag{A1}$$

The α_{ij} correspond to the edge coefficients defined in Sec. II.A, where $i, j = 1, 2, 3$ denote the x, y, z subscripts, and the proportionality constant has been absorbed into the α_{ij} coefficients. The summation is over all tetrahedral elements surrounding the edge (i.e., edge 1-2 in Fig. 2). The term F_i denotes the ith component of the normal area vector of the face of tetrahedron e, which touches vertex 2, but does not contain edge 1-2, and G_j the jth component of the face normal of tetrahedron e, which touches vertex 1, but does not contain edge 1-2. Since these coefficients are symmetric in F and G, they can be used to compute the flux contribution to vertex 1 as well as the flux contribution to vertex 2, along edge 1-2. This requires the storage of nine coefficients per edge. If the coefficients are symmetric in the i, j indices, then only six coefficients per edge are required. To prove this property, we form the difference

$$\alpha_{ij} - \alpha_{ji} = \sum_e \frac{F_i G_j}{\text{vol}_e} - \sum_e \frac{F_j G_i}{\text{vol}_e} \tag{A2}$$

The volume of a tetrahedral element can be expressed as

$$\text{vol}_e = \frac{2}{3} \frac{(F_i G_j - F_j G_i)}{\Delta x_k} \tag{A3}$$

where $i \neq j \neq k$ and Δx_k represents the difference $(x_k)_a - (x_k)_b$, where a and b are the two remaining vertices other than the edge endpoints 1, 2 of the tetrahedron e, as shown in Fig. 2. Inserting this expression into Eq. (2), we obtain

$$\sum_e \frac{F_i G_j}{(F_i G_j - F_j G_i)/\Delta x_k} - \sum_e \frac{F_j G_i}{(F_i G_j - F_j G_i)/\Delta x_k} = \sum_e \Delta x_k \tag{A4}$$

Since the summation is over all tetrahedra surrounding the edge 1-2, the sum of all such Δx_k forms a closed polygon encircling the edge and thus vanishes. The coefficients are thus symmetric in ij for all values of i and j, and thus only six coefficients per edge need to be stored.

Acknowledgments

This work was made possible in large part by the computational resources provided by the NASA Numerical Aerodynamic Simulation Program (NAS).

References

[1]Mavriplis, D. J., "Turbulent Flow Calculations Using Unstructured and Adaptive Meshes," *International Journal for Numerical Methods in Fluids*, Vol. 13, No. 9, 1991, pp. 1131–1152.

[2]Barth, T. J., "Numerical Aspects of Computing Viscous High-Reynolds Number Flows on Unstructured Meshes," AIAA Paper 91-0721, Jan. 1991.

[3]Anderson, W. K., and Bonhaus, D. L., "Navier–Stokes Computations and Experimental Comparisons for Multielement Airfoil Configurations," AIAA Paper 93-0645, Jan. 1993.

[4]Davis, W. H., and Matus, R. J., "High-Lift Multiple Element Airfoil Analysis with Unstructured Grids," AIAA Paper 93-3478, Aug. 1993.

[5]Mavriplis, D. J., "Three Dimensional Unstructured Multigrid for the Euler Equations," *AIAA Journal*, Vol, 30, No. 7, 1992, pp. 1753–1761.

[6]Luo, H., Baum, J. D., Lohner, R., and Cabello, J., "Adaptive Edge-Based Finite Element-Schemes for the Euler and Navier–Stokes Equations on Unstructured Grids," AIAA Paper 93-0336, Jan. 1993.

[7]Spalart, P. R., and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," AIAA Paper 92-0439, Jan. 1992.

[8]Mavriplis, D. J., "Algebraic Turbulence Modeling for Unstructured and Adaptive Meshes," *AIAA Journal*, Vol. 29, No. 12, 1991, pp. 2086–2093.

[9]Rumsey, C. L., "A Comparison of the Predictive Capabilities of Several Turbulence Models Using Upwind and Central-Difference Computer Codes," AIAA Paper 93-0192, Jan. 1993.

[10]Mavriplis, D. J., and Martinelli, L., "Multigrid Solution of Compressible Turbulent Flow on Unstructured Meshes Using a Two-Equation Model," AIAA Paper 91-0237, Jan. 1991.

[11]Mavriplis, D. J., Vermeland, R. E., Das, R., and Saltz, J., "Implementation of a Parallel Unstructured Euler Solver on Shared and Distributed Memory Architectures," *Proceedings of the SuperComputing '92 Conference*, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 132–141.

[12]Morano, E. M., and Mavriplis, D. J., "Implementation of a Parallel Unstructured Euler Solver on the CM-5," AIAA Paper 94-0755, Jan. 1994.

[13]Valarezo, W. O., and Mavriplis, D. J., "Navier–Stokes Applications to High-Lift Airfoil Analysis," AIAA Paper 93-3534, Aug. 1993.

[14]Mavriplis, D. J., "Unstructured and Adaptive Mesh Generation for High Reynolds Number Viscous Flows," *Proceedings of the 3rd International Conference on Numerical Grid Generation in Computational Fluid Dynamics*, edited by A. S. Arcilla, J. Hauser, P. R. Eisman, and J. F. Thompson, Pineridge, Swansea, Wales, UK, 1991, pp. 79–91.

[15]Lohner, R., "Matching Semistructured and Unstructured Grids for Navier–Stokes Calculations," AIAA Paper 93-3348, July 1993.

[16]Pirzadeh, S., "Unstructured Viscous Grid Generation by Advancing Layers Method," AIAA Paper 93-3453, Aug. 1993.

[17]Vatsa, V. N., and Wedan, B. W., "Development of a Multigrid Code for 3D Navier–Stokes Equations and Its Application to a Grid Refinement Study," *Computers and Fluids*, Vol. 18, No. 4, 1990, pp. 391–403.