Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 206 (2005) 208–226
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Selective edge removal for unstructured grids with Cartesian cores

Rainald Löhner [a,*], Hong Luo [b], Joseph D. Baum [b]

[a] Department of Science and Technology, School of Computational Sciences, M.S. 4C7, George Mason University, 4400 University Drive, Fairfax, VA 22030-4444, USA
[b] Advanced Technology Group, Science Applications International Corp., McLean, VA 22102, USA

Received 28 May 2004; received in revised form 26 November 2004; accepted 26 November 2004

Abstract

Several rules for redistributing geometric edge-coefficient obtained for grids of linear elements derived from the subdivision of rectangles, cubes or prisms are presented. By redistributing the geometric edge-coefficient, no work is carried out on approximately half of all the edges of such grids. The redistribution rule for triangles obtained from rectangles is generalized to arbitrary situations in 3-D, and implemented in a typical 3-D edge-based flow solver. The results indicate that without degradation of accuracy, CPU requirements can be cut considerably for typical large-scale grids. This allows a seamless integration of unstructured grids near boundaries with efficient Cartesian grids in the core regions of the domain.

Keywords: CFD; FEM; Cartesian grids; Edge-based solvers

1. Introduction

Many applications with complex geometries require large regions of uniform grids. Examples are wave propagation (acoustics, electromagnetics) and large-eddy simulation of flows. It can be argued that in these regions, where more than 90% of all elements reside, a uniform, Cartesian grid represents the optimal discretization. Furthermore, due to the uniformity of the mesh, the traditional 27-point stencil obtained for trilinear hexahedral elements may be replaced by the more efficient 7-point stencil while still retaining second order accuracy in space. Traditional Cartesian grid solvers require special treatment of stencils or

* Corresponding author. Tel.: +1 703 993 1990.
E-mail address: rlohner@gmu.edu (R. Löhner).

0021-9991/$ - see front matter
doi:10.1016/j.jcp.2004.11.034

volumes close to boundaries [4,20,11,2,12,5,10]. This is not the case for solvers based on unstructured grids, which have found widespread use for complex geometries. A seamless combination of traditional unstructured grids close to boundaries (comprising a small percentage of the total mesh) with highly efficient Cartesian grids in the core regions would thus seem very promising. We remark, in passing, that many unstructured grid generators utilize point distributions from Cartesian [3] or adaptive Cartesian grids [25,9] for the regions where isotropic elements are required, and that for electromagnetics the combination of unstructured grids neas boundaries and structured grids in the core regions has been found to be advantageous [6].

For problems with boundary layers, semi-structured grids obtained by lofting surface triangulations are commonly used [15,22,24]. A number of researchers have reported the degradation of accuracy that occurs when solving compressible flow problems using traditional finite volume schemes in regions where elements are highly stretched [1,27,7,14]. The reason for this degradation is that the normals of the finite volume faces associated with the edges of the mesh are misaligned with the direction of the edge. This can be seen from Fig. 1, where a rectangular triangle, typical of boundary layer grids, is depicted. The normal of face D,G belonging to edge A,B, which is aligned with the x-direction, will tend to be in the y-direction for $h_x/h_y \gg 1$.

The solution advocated so far for grids of this type is to construct finite volumes using the circumcenter/sphere of the points comprising the triangle/tetrahedron instead of the geometric center. The normals of these so-called containment duals tend to be well aligned with their corresponding edges, and remove the dependency of edges that are diagonal. Fig. 1 shows this effect for a stretched triangle. The success of finite volume schemes based on the containment dual concept for highly stretched elements has led to the search for an equivalent modification of edge-based schemes stemming from finite element discretizations. Attempts have been made to modify quadrature rules or shape-functions [27]. Some of these schemes are extremely elaborate.

The decoupling of diagonal connections, which can be interpreted as a selective edge removal, is not limited to stretched elements. For any mesh with *rectangular triangles* this type of edge removal via containment duals is possible. Given that most of the CPU-intensive operations occur at the edge-level (fluxes, limiting, Riemann-solvers, etc.), removal of edges without degradation of accuracy should have an immediate benefit.

This has led to the following overall procedure:

- Generate as large a portion of the volume as possible with *adaptive Cartesian* grids (for the inviscid/wave propagation domain) or *semi-structured* grids that are *split into tetrahedra*.
- Fill in the remaining regions close to boundaries with an *unstructured grid of tetrahedra*.
- Build all geometrical coefficients as if an unstructured grid is used.
- Where possible, selectively remove the edges.

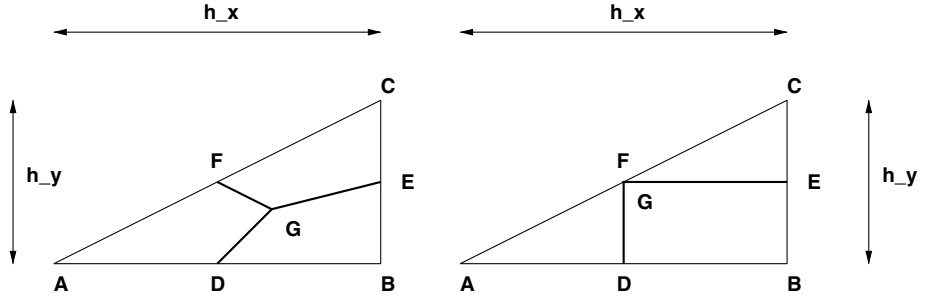

Fig. 1. Triangular element.

The expected savings as compared to a traditional unstructured grid are approximately a factor of 2, i.e., not spectacular but worth the effort. As will be shown below, these savings are indeed realized. It may be argued that special solvers should be used in the interior, Cartesian (or tensor-product) region. Such an approach, which has been proposed by Pflaum [21], precludes the use of adaptive Cartesian grids, and requires the development and maintenance of two separate solvers. The aim of the present work was to modify the general, unstructured grid solver as little as possible, so that a seamless transition from fully unstructured to special-purpose grids could be accomplished.

The remainder of the paper is organized as follows: Sections 2 and 3 present the redistribution schemes for edges of triangles and tetrahedra. Section 4 treats the general case. Section 5 discusses the generation of adaptive Cartesian cores. Examples are given in Section 6, and some conclusions and an outlook for future work complete the paper in Section 7.

2. Redistribution of weights: triangles

2.1. Advective coefficients

Consider again the element shown in Fig. 1. For any given point i, the discretization of a conservation law of the form

$$u,t + \nabla \cdot \mathbf{f} = 0 \tag{1}$$

will result in a sum of fluxes

$$M^i \hat{u}^i_{,t} = r^i = C^{ij} \hat{f}_{ij}, \tag{2}$$

where $\hat{f}_{ij}$ denotes a consistent numerical flux, e.g. that obtained from an approximate Riemann [8] solver and C^{ij} is a geometric coefficient (typically an area normal) associated with the edge ij [13,17]. The sum over the edges ij can be split into a sum of element contributions

$$r^i = \sum_{\text{el}} r^i_{\text{el}} = \sum_{\text{el}} C^{ij}_{\text{el}} \hat{f}_{ij}. \tag{3}$$

With reference to Fig. 1, we see that for point A, the edge-contributions are:

$$r^A_{\text{el}} = C^{AB}_{\text{el}} \hat{f}_{AB} + C^{AC}_{\text{el}} \hat{f}_{AC}. \tag{4}$$

Going from a median dual to a containment dual finite volume representation may now be interpreted in a variety of ways:

(a) Move the centroid G to the circumcenter F and recompute all finite volumes.
(b) Add the face (edge) G, F to D, G and remove the dependency A–C.
(c) In Eq. (4), assume that $u_C \approx u_B$, and use only u_B.

The end result is the same:

$$r^A_{\text{el}} = [C^{AB}_{\text{el}} + C^{AC}_{\text{el}}] \hat{f}_{AB}, \tag{5}$$

i.e., the edge-coefficient from edge A–C is simply added to edge A–B. In the same manner, for point C, the contribution of edge AC must be accounted for. As before, this can be interpreted in a variety of ways:

(a) Move the centroid G to the circumcenter F and recompute all finite volumes.
(b) Add the face (edge) G, F to E, G and remove the dependency A–C.
(c) Assume that $u_A \approx u_B$, and use only u_B.

The end result is the same:

$$r_{\mathrm{el}}^{C} = [C_{\mathrm{el}}^{CA} + C_{\mathrm{el}}^{CB}]\hat{f}_{CB}, \tag{6}$$

i.e., the edge-coefficient from edge A–C is simply added to edge B–C.

2.2. Mass-matrix (areas)

A change in the finite volume surrounding a node must also be accounted for in the mass (or area) associated to the node. From Fig. 2, one can see that the change is not negligible. From geometrical considerations, the area of sub-triangle D, G, F is given by

$$A_{D,G,F} = \tfrac{1}{12}A_{A,B,C}. \tag{7}$$

Therefore, the removal of edge A, C has to be accounted for by:

- adding $\frac{2}{12}A_{A,B,C}$ to node B (node opposite to edge being modified);
- subtracting $\frac{1}{12}A_{A,B,C}$ from nodes A, C, respectively (nodes of edge being modified).

2.3. Change indicator

The next question to be answered is when to switch the weights. The simplest form to determine an obtuse angle is via the coefficients obtained for the Laplacian operator on an edge. The edge coefficients are given by

$$L^{ij} = \int_{\Omega_{\mathrm{el}}} \nabla N^i \cdot \nabla N^j \, \mathrm{d}\Omega. \tag{8}$$

This coefficient will be negative for angles smaller than 90° and positive for angles larger than 90°. It therefore seems natural to invoke the redistribution of weights whenever $L^{ij} \geqslant 0$.

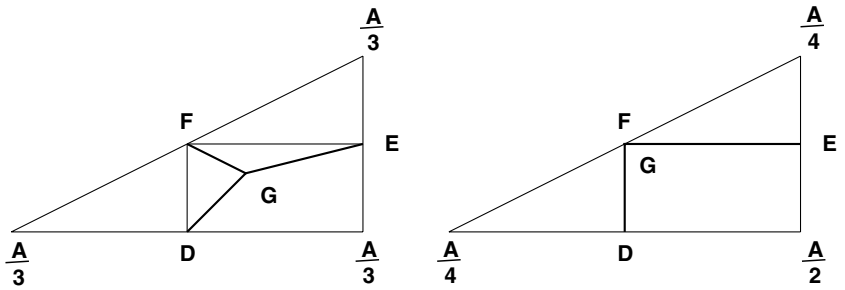

Fig. 2. Triangular element: mass/area associated to nodes.

2.4. Rules for triangles

From the preceding arguments, we can summarize the following weight switching rules for triangles:

(a) Build element coefficients and nodal masses as before (linear finite elements, median dual finite volume).
(b) DO: For each edge ij of the element:
IF: Laplacian coefficient $L^{ij} \geqslant 0$:
- Add advection coefficients C_k^{ij} to the other 2 edges;
- Set advection coefficients $C_k^{ij} = 0$;
- Add $A/6$ to node opposite to edge ij;
- Subtract $A/12$ from nodes i, j;

ENDIF
ENDDO

3. Redistribution of weights: tetrahedra

3.1. Advective coefficients

Consider the tetrahedra shown in Fig. 3. Unlike the 2-D case, the redistribution of advective coefficients can involve a number of other edges.

Assume, without loss of generality, that the advective coefficients of edge 1 with nodes A, B have been marked for removal. One could either add these coefficients by treating the adjacent faces as triangles. This implies adding a fraction ξ of the coefficients to edges 2, 3, and the remaining fraction $1 - \xi$ to edges 4, 5. On the other hand, one of the edges in each of these pairs may have been marked for removal as well. In this case, one has to take a more complex path, adding the coefficients to edges 2, 6, 4 (in case either of edges 3, 5 have been marked for removal) or to edges 3, 6, 5 (in case either of edges 2, 4 have been marked for removal). It is important to note that a closed path through edges between the end-nodes has to be followed when adding the coefficients of the edge being removed to other edges. Otherwise, the balance of fluxes surrounding a node will not be maintained.

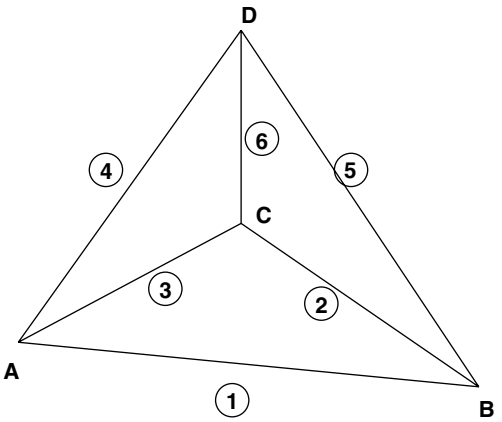

Fig. 3. Tetrahedral element.

3.2. Mass-matrix (volumes)

As in the 2-D case, a change in the finite volume surrounding a node must also be accounted for in the mass (or volume) associated to the node. From Fig. 4, one can see that the change is not negligible. Denoting by V the volume of the element, the removal of edge A, B has to be accounted for by:

- Subtracting $V/16$ from nodes A, B respectively (nodes of edge being modified).
- For option 1 (no addition to edge 6):
  - adding $\xi V/8$ to node C;
  - adding $((1-\xi)V)/8$ to node D.
- For option 2 (addition to edge 6):
  - adding $V/16$ to node C;
  - adding $V/16$ to node D.

3.3. Change indicator

As in the 2-D case, the simplest form to determine an obtuse angle is via the coefficients obtained for the Laplacian operator on an edge. This coefficient will be negative for angles smaller than 90° and positive for angles larger than 90°. It therefore seems natural to invoke the redistribution of weights whenever $L^{ij} \geqslant 0$.

3.4. Rules for tetrahedra

From the preceding arguments, we can summarize the following weight switching rules for tetrahedra:

(a) Build element coefficients and nodal masses as before (linear finite elements, median dual finite volume);
(b) DO: For each edge ij of the element:
IF: Laplacian coefficient $L^{ij} \geqslant 0$;
- Obtain Laplacian coefficients for edges adjacent to end-nodes i, j;
- IF: all these Laplacian coefficients are negative:
  - Determine redistribution weight ξ;
  - Add ξC_k^{ij} to first contiguous edge pair;
  - Add $(1-\xi)C_k^{ij}$ second contiguous edge pair;
  - Add $\xi V/8$ to common node of 1st edge pair;
  - Add $((1-\xi)V)/8$ to common node of 2nd edge pair;
- ELSE:
  - Determine continuous path with negative Laplacians;
  - Add C_k^{ij} to edges comprising the path;
  - Add $V/16$ to intermediate nodes of the path;
- ENDIF
- IF: coefficients were added:
  - Set advection coefficients $C_k^{ij} = 0$;
  - Subtract $V/16$ from nodes i, j;
- ENDIF

ENDIF
ENDDO

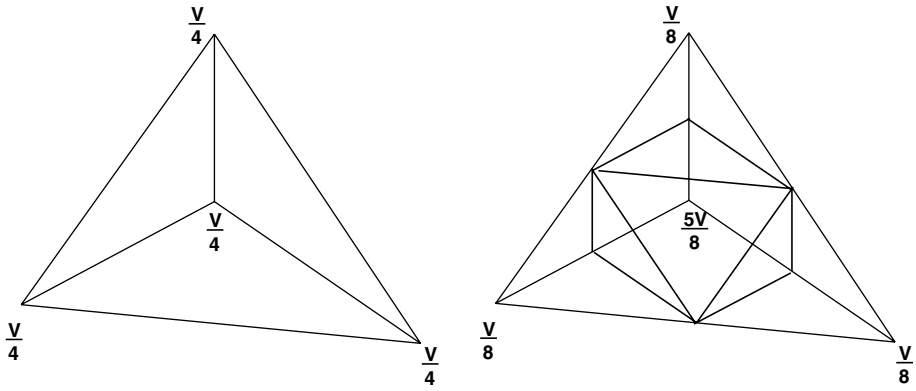

Fig. 4. Tetrahedral element: mass/volume associated to nodes.

4. The general case

Consider a mesh of cubes split into 5 tetrahedra per cube, as shown in Fig. 5(a).

The circumcenters of the outer tetrahedra lie at the center of the cube, i.e., *outside* the tetrahedra. Moreover, the Laplacian coefficients of the large tetrahedron at the center do not vanish. This implies that for subdivisions of this kind the rules derived above will not lead to a substantial reduction of edges. This has led to the search for a more general redistribution procedure. After several tries, the following triangle rule, summarized in Fig. 6, was found to work well:

(a) Build element coefficients and nodal masses as before (linear finite elements, median dual finite volume);
(b) `DO`: For each edge *ij* of the element:
`IF`: A rectangle of neighbouring edges lying in a plane can be constructed:
– Add half of the advection coefficients C_k^{ij} to the other 4 edges;
– Set advection coefficients $C_k^{ij} = 0$;
`ENDIF`
`ENDDO`

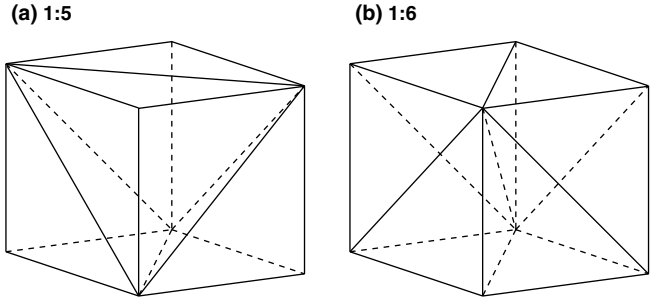

Fig. 5. Subdivision of cube into tetrahedra.

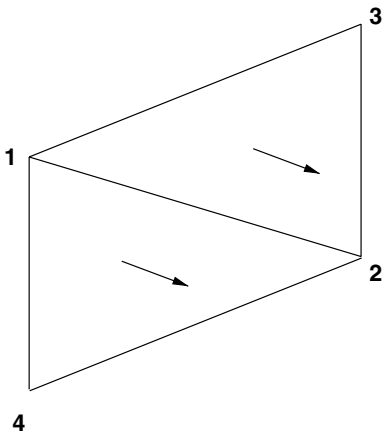

Fig. 6. General rectangle configuration.

The lumped mass matrix is obtained by evaluating the gradient of a known function (e.g. $u = x + y + z$). Given the known gradient values, the best lumped mass coefficient can be derived.

5. Generation of adaptive Cartesian cores

The generation of Cartesian cores for general domains is shown in Fig. 7.

In a first step the surface of the computational domain is discretized with triangles of a size as prescribed by the user. In the present case, this is accomplished through a combination of background grids, sources and element size linked to CAD entities [17]. In a second step a Cartesian mesh that has the element size of the largest element desired in the volume is superimposed on the volume. This mesh is then adaptively refined locally so as to obtain the element size distribution prescribed by the user. Note that adaptive refinement only works well with the subdivision of cubes into 6 tetrahedra (Fig. 5(b)). As points are only added along existing edges, mesh refinement does not add any new points inside the cubes subdivided into 5 tetrahedra (Fig. 5(a)). Once the adaptive Cartesian mesh is obtained, the elements that are outside the domain to be gridded are removed. This yields an additional list of faces, which, together with the surface discretization, form the initial front of the as yet ungridded portion of the domain. In the present case an advancing front technique [16] is used to mesh this portion of the domain. The capability to mesh Cartesian cores only requires a small change within an advancing front technique. The Cartesian core portion is basically an independent module whose main role is to supply an additional list of triangles for the initial front. Meshing the Cartesian core is extremely fast, so that overall CPU requirements for large grids decrease considerably. A preliminary version of the present procedure (without the adaptive refinement of the Cartesian core) was proposed in [6]. Other techniques that generate unstructured grids with Cartesian cores are all those that use point distributions from regular Cartesian grids [3,19] or Octrees [25,9].

6. Examples

The selective edge removal option was implemented in FEFLO, a general purpose edge-based finite element flow solver using linear elements [18], and was tested on a variety of examples, of which four are included here. We remark from the outset that the main aim of the comparison is the quality of

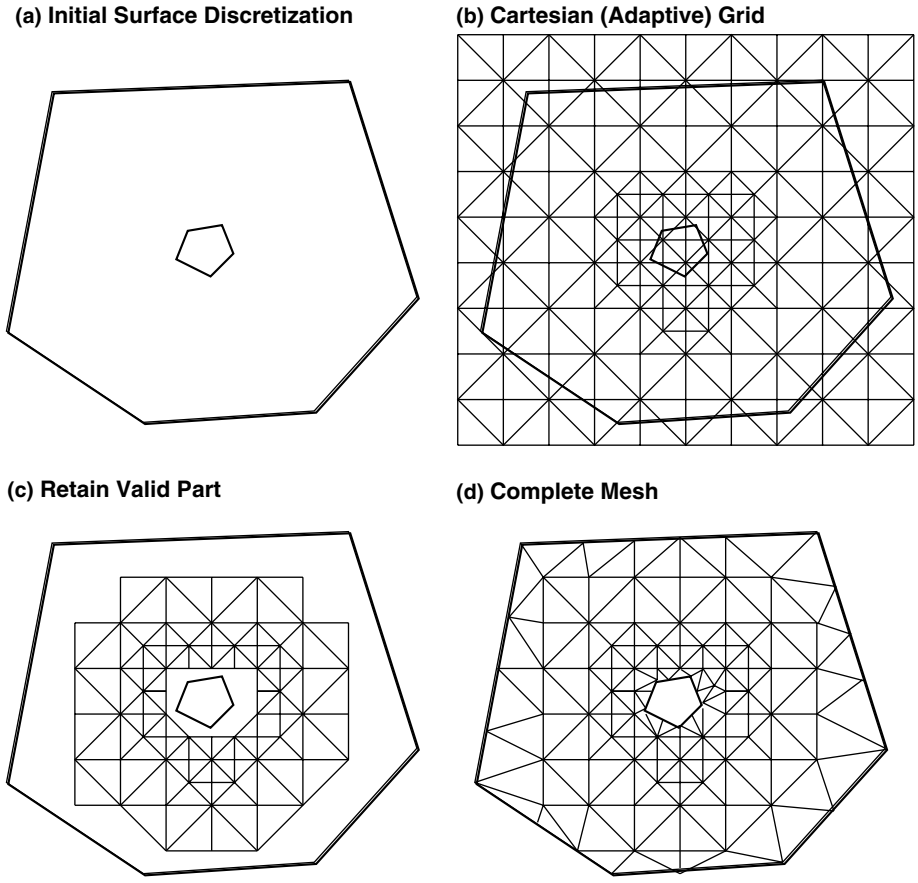

Fig. 7. Mesh generation with adaptive Cartesian core.

the results as well as the relative speed of typical unstructured grids vs. those with Cartesian cores. The first case is a steady inviscid compressible flow case, the second a transient inviscid compressible flow case, and the third a steady inviscid incompressible flow case. For each of these cases the relative CPU requirements of point and edge-loops is different, so that the merits of grids with Cartesian cores and selective edge removal can be seen over a spectrum of applications. These runs were performed on a PC with an Intel-P4 processor running at 2.1 GHz with 1 Gbyte of RAM using the Intel Fortran compiler under Red Hat Linux OS.

For each case, three different grids were used:

- a typical unstructured grid (Grid 1);
- an unstructured grid with Cartesian core obtained by subdivision of cubes into 5 tetrahedra (Grid 2);
- an unstructured grid with Cartesian core obtained by subdivision of cubes into 6 tetrahedra (Grid 3).

The main statistics of these grids were recorded and are listed in Tables 1–3. We denote by `nelem`, `necrt`, `nedge`, `nacte` the nr. of elements, the nr. of elements in the Cartesian core, the nr. of edges and the nr. of active edges, respectively. The CPU requirements are in seconds. The grids with Cartesian

Table 1
Grid statistics for wedge

Grid	nelem	necrt	nedge	nacte	CPU
1	958,951	0	1,148,313	1,148,313	2685
2U	600,821	460,015	743,795	743,795	1831
2C	600,821	460,015	743,795	460,716	1331
3U	694,285	552,018	837,405	837,405	1960
3C	694,285	552,018	837,405	462,327	1360

Table 2
Grid statistics for sod tube

Grid	nelem	necrt	nedge	nacte	ntime	CPU
1	173,140	0	217,142	217,142	937	340
2U	117,314	62,720	154,015	154,015	777	213
2C	117,314	62,720	154,015	113,183	775	170
3U	130,228	75,264	166,927	166,927	751	216
3C	130,228	75,264	166,927	113,551	749	165

Table 3
Grid statistics for bump in channel

Grid	nelem	necrt	nedge	nacte	ntime	CPU
1	819,792	0	984,994	984,994	150	371
2U	503,268	391,990	627,240	627,240	150	230
2C	503,268	391,990	627,240	385,003	150	199
3U	582,455	469,992	706,470	706,470	150	256
3C	582,455	469,992	706,470	385,908	150	203

cores were run with the usual solver (all edges active), as well as with deactivation of edges switched on. We are aware that the definition of element size can be a contentious issue. We have taken the traditional notion, i.e., an element of size h is given by:

- a tetrahedral element whose edges are of length h;
- a hexahedral element whose edges are of length h.

It is clear that for a specified element size h, a portion of space will contain more edges and points if filled by a mesh of tetrahedra than hexahedra. This fact should be taken into consideration when comparing the results obtained. The same applies to the allowable timestep of explicit timemarching schemes: for a specified element size h the allowable timestep of unstructured grids is smaller than that of an equivalent Cartesian grid. Whether the increased number of edges (and smaller allowable timestep) of tetrahedral meshes translates into higher accuracy is also a matter of debate which we do not feel qualified to answer.

The fourth case considers the space shuttle ascend configuration, and is typical of production runs with complex geometries. This larger case was run on an SGI O3900 using 16 processors in shared memory mode. It was included to show the versatility of the techniques proposed.

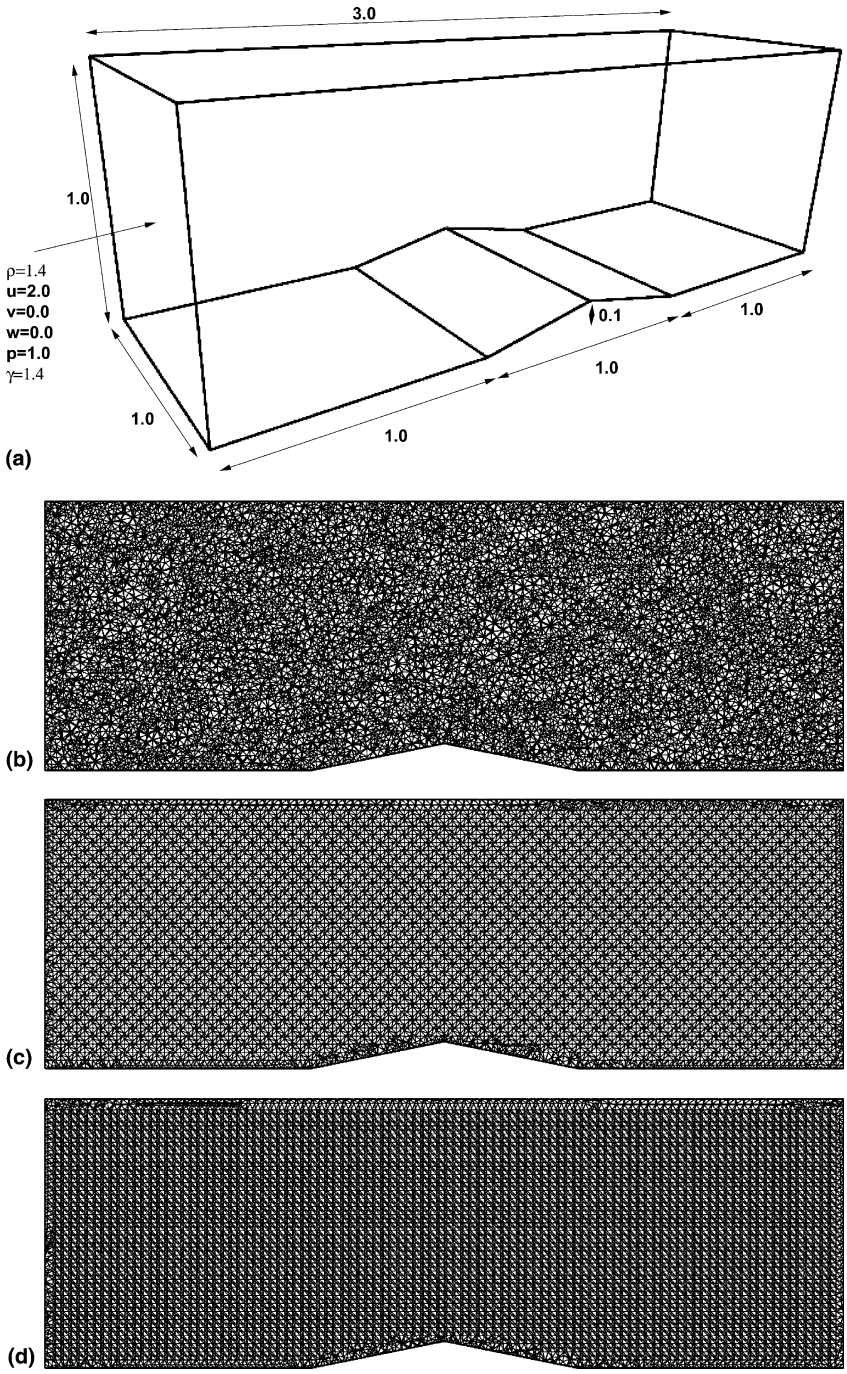

Fig. 8. (a) Problem definition for wedge. (b)–(d) Triangulation of cut plane $z = 0.0$. (e) Mach-Number contours on surface (Cases 2U, 2C). (f)–(j) Mach-Number in cut plane $z = 0.0$.

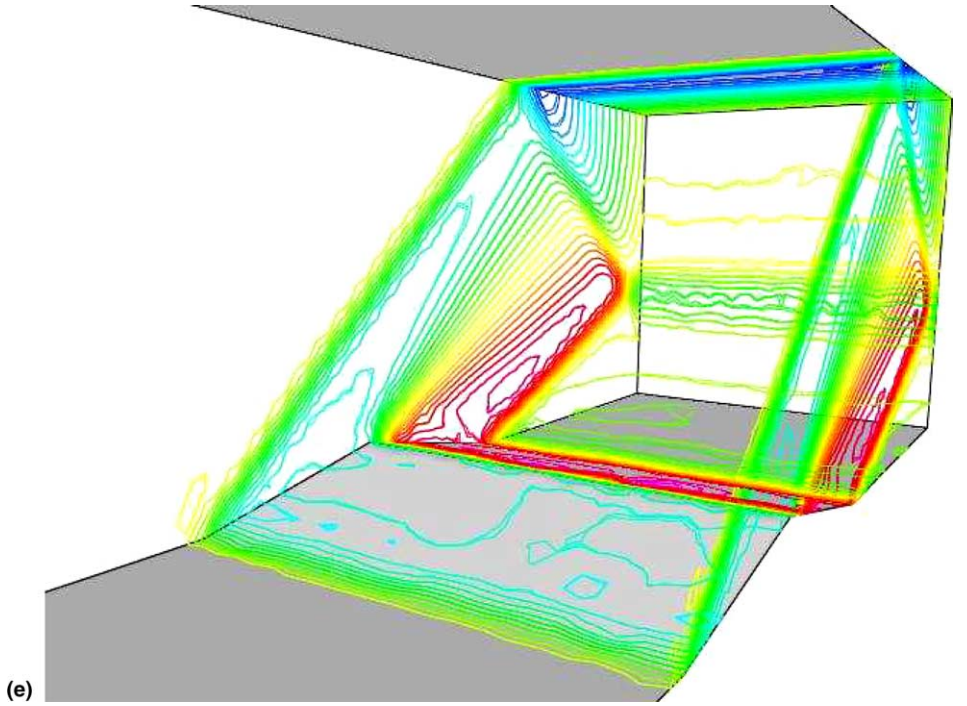

Fig. 8 (*continued*)

6.1. Flow past supersonic wedge

The first case considered is the supersonic flow past a wedge in a channel. The overall dimensions and boundary conditions are shown in Fig. 8(a). The incoming Mach-number is $Ma_\infty = 2.0$. The mesh is of uniform size, as can be seen from Fig. 8(b)–(d). This allows the use of cube subdivisions into 5 and 6 tetrahedra. The Euler equations were solved using the approximate Riemann solver of Roe [23], gradient reconstruction and van Albada limiting [8] on the conserved variables. The solution was advanced in time using a 3-stage Runge–Kutta scheme with a Courant-nr. of $C = 1.0$ for 600 steps. Local timesteps and residual smoothing were used to accelerate the convergence to steady state for this problem. Grid statistics and timings have been summarized in Table 1. As one can see, the timings are proportional to the number of active edges. This is to be expected, as most of the CPU-intensive operations (limiting, approximate Riemann solver) are carried out in edge-loops. In fact, this effect could have been accentuated by shifting to a more costly Riemann solver and/or limiting on characteristic variables.

Fig. 8(e)–(j) compare the Mach-number contours obtained for the different grids and solver combinations. The immediate conclusion is that the contours look very similar, and it becomes difficult to discern a clear 'best solution'. This would make the case for grids with Cartesian cores and deactivation, as for these the CPU requirements are considerably lower.

6.2. Sod shock tube problem

This well known testcase [26] was chosen to see if the transition from the outer unstructured grid to the inner Cartesian core would present problems. The domain was taken to be: $100 \times 5 \times 5$, and the initial conditions are the usual ones: $\rho = 1.0$, $p = 1.0$ for $x < 50$ and $\rho = 0.1$, $p = 0.1$ for $x \geqslant 50$. The solver used was

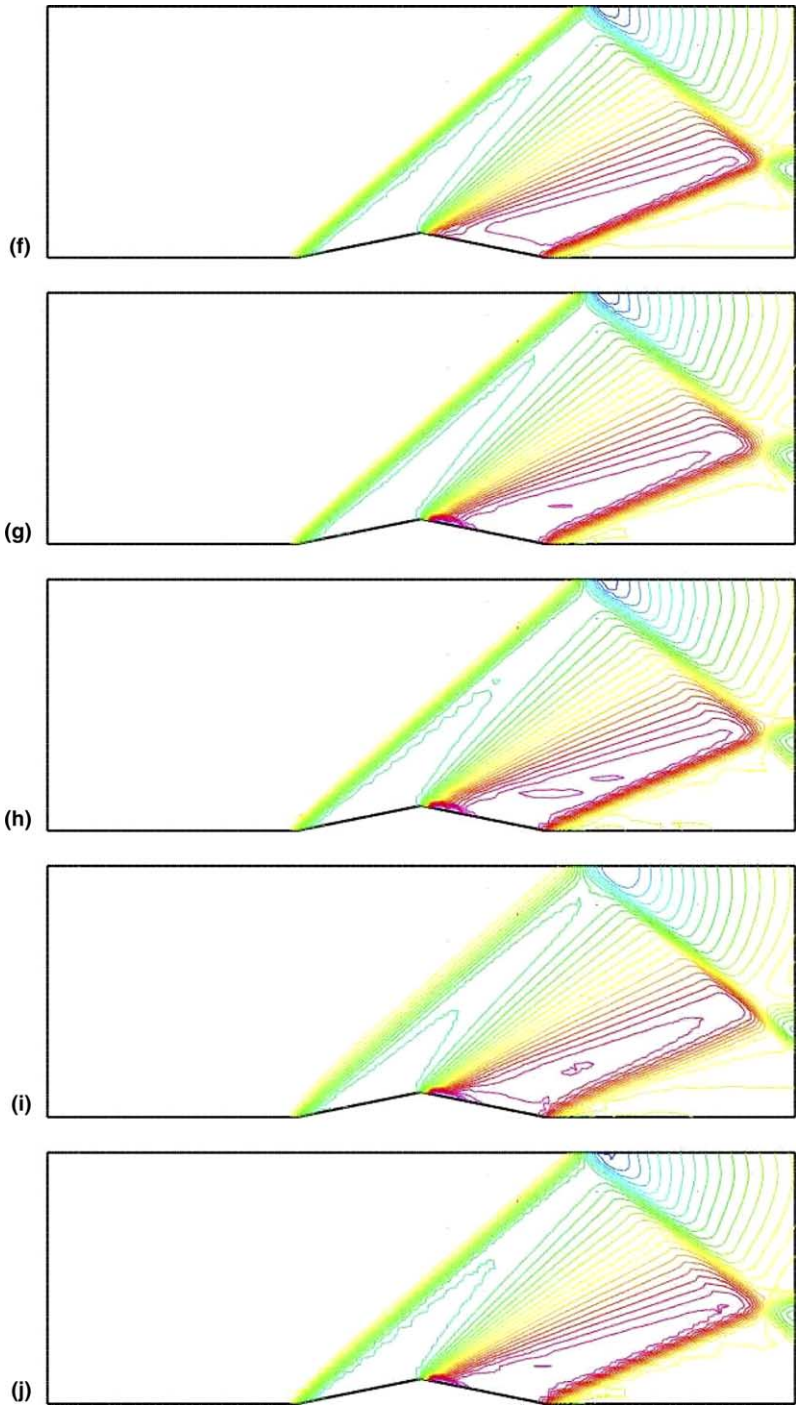

Fig. 8 (*continued*)

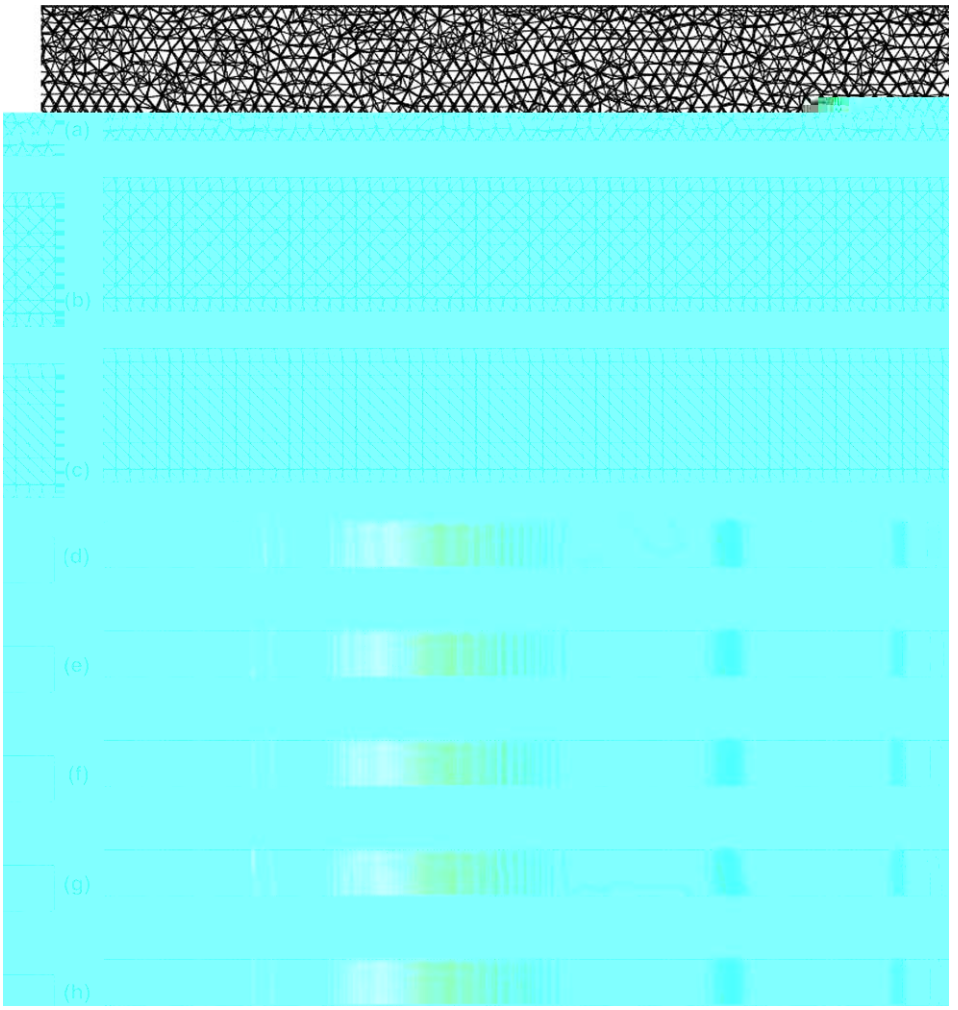

Fig. 9. (a)–(c) Triangulation of cut plane $z = 0.0$. (d)–(h) Density in cut plane $z = 0.0$ at time $t = 20.0$.

the same as in the previous example. The solution was run to a time of $t = 20.0$. As before, three grids were generated and run, and their statistics are listed in Table 2. Note that the number of timesteps required for the grids with Cartesian cores is noticeable smaller than for grid 1. We have observed this behaviour (i.e., larger allowable timesteps for grids with Cartesian cores) for several other problems as well. Fig. 9(a)–(c) shows the triangulations obtained by cutting the mesh in the plane $z = 0.0$. The different grid types can be discerned clearly. Fig. 9(d)–(h) shows 100 density contours in the plane $z = 0.0$ at time $t = 20.0$.

The differences, though discernable, are small, except for the case 3U, which shows a marked bias for the contact discontinuity. It is remarkable that the case 3C, i.e., with the diagonal edges switched off, does not show this bias.

6.3. Bump in channel

This third testcase considers incompressible flow. The overall dimensions and boundary conditions are shown in Fig. 10(a). The incompressible Euler equations were solved using a projection scheme [18]. The

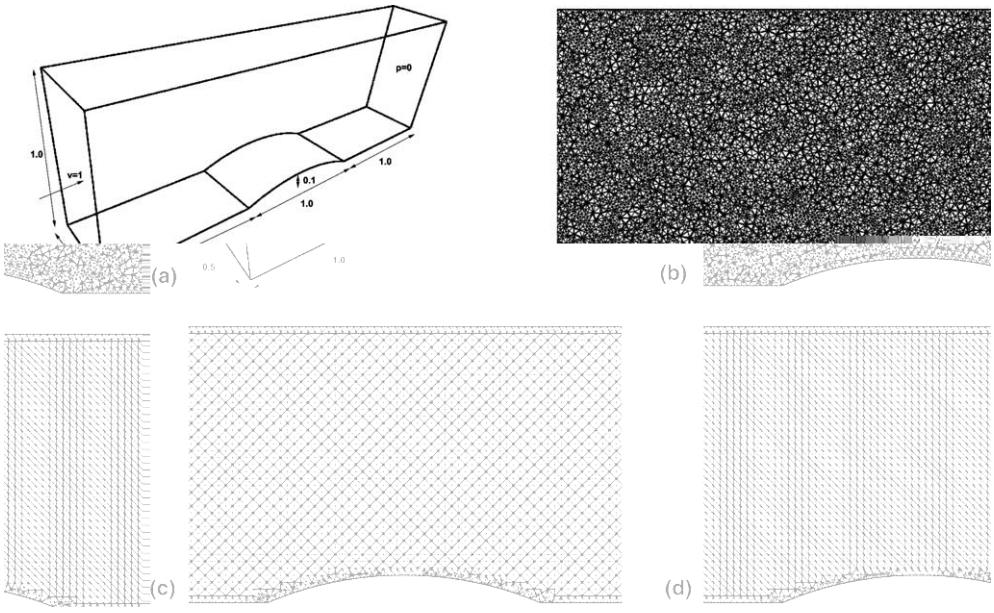

Fig. 10. (a) Problem definition for bump; (b)–(d) triangulation of cut plane $z = 0.0$; (e)–(i) pressure in cut plane $z = 0.0$.

advective fluxes, which are integrated explicitly, are built using upwind bias, gradient reconstruction and van Albada limiting [8]. A divergence-free flow is enforced by solving a pressure-Poisson equation at each timestep. Again, three grids were generated and run, and their statistics are listed in Table 3. The timings in this case are not as favourable as those for the compressible solver. This is expected, as a large percentage of the overall CPU time is spent in the solution of the Poisson problem, where the work per edge is relatively low. This results in a higher percentage of CPU work in point-loops. Given that the number of points stays constant whether the diagonal edges of Cartesian cores are active or deactive, the gains from deactivation are more modest.

Of course, it may be argued that for the Poisson problem very efficient multigrid solvers are available. But that would imply two separate solvers, something impractical for large-scale codes. Fig. 10(b)–(d) shows the triangulations obtained by cutting the mesh in the plane $z = 0.0$. The different grid types can be discerned clearly. Fig. 10(e)–(i) shows 50 pressure contours in the plane $z = 0.0$. As before, the immediate conclusion is that the contours look very similar, making it difficult to discern a clear 'best solution'. This again makes a strong case for grids with Cartesian cores and deactivation, as for these the CPU requirements are reduced considerably.

6.4. Shuttle ascend configuration

This fourth case considers the space shuttle ascend configuration, and is typical of production runs with complex geometries. The overall geometry is shown in Fig. 11(a). Fig. 11(b)–(e) shows the surface discretization used, as well as the triangulations of two planar cuts along the shuttle and a zoom in the region below the wing. One can clearly see the adaptively refined Cartesian cores of the flow domain, as well as the transition to an unstructured grid near the surface. The mesh statistics have been compiled in Table 4. Note that the reduction in active edges is still considerable for this complex geometry case. The Euler equations were solved using the approximate Riemann solver of Roe [23], gra-

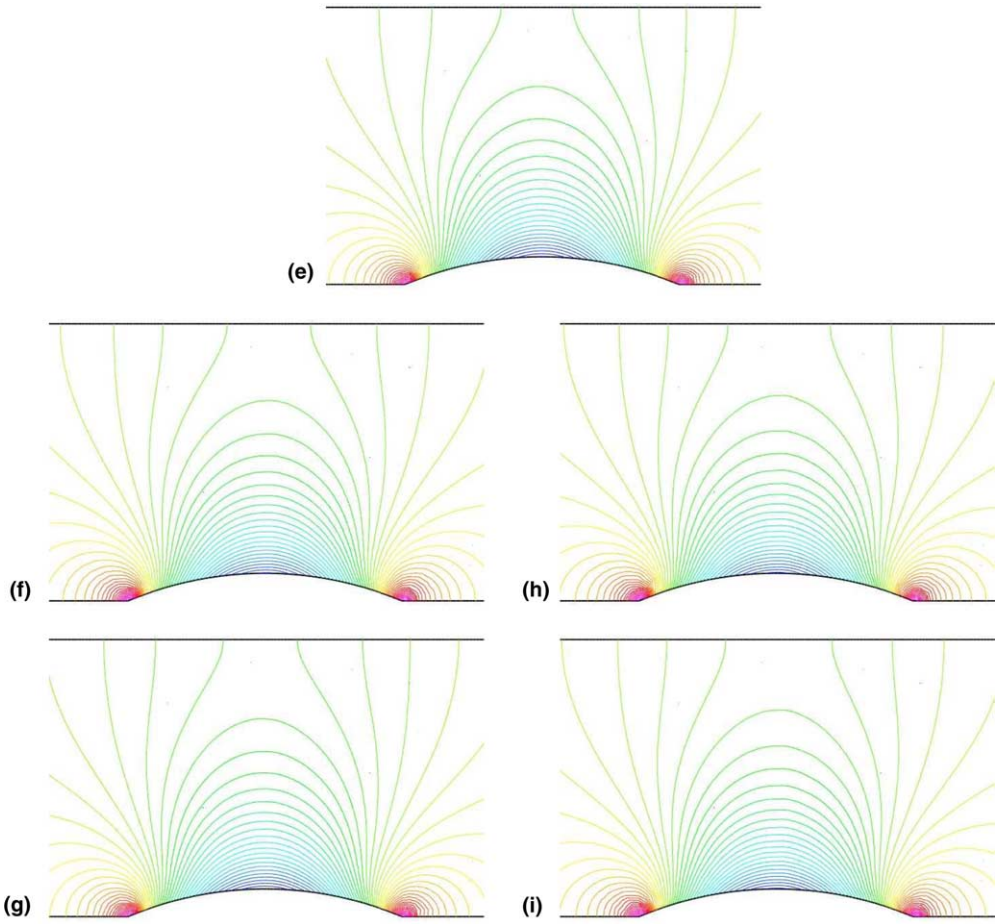

Fig. 10 (*continued*)

dient reconstruction and van Albada limiting [8]. The solution was advanced in time using a 3-stage Runge–Kutta scheme with a Courant-nr. of $C = 1.0$ for 600 steps. Local timesteps and residual smoothing were used to accelerate the convergence to steady state for this problem. This larger case was run on an SGI O3900 using 16 processors in shared memory mode. The surface pressure obtained for an incoming Mach-number of $Ma_\infty = 2.0$ and angle of attack $\alpha = 0°$ is shown in Fig. 11(f).

7. Conclusions and outlook

Several rules for redistributing geometric edge-coefficients obtained for grids of linear elements derived from the subdivision of rectangles, cubes or prisms were presented. By redistributing the geometric edge-coefficients, no work is carried out on approximately half of all the edges of such grids. The redistribution rule for triangles obtained from rectangles was generalized to arbitrary situations in 3-D, and implemented in a typical 3-D edge-based flow solver. The results indicate that without degradation of accuracy, CPU requirements can be cut by half for grids with a large percentage of elements in the Cartesian core. For

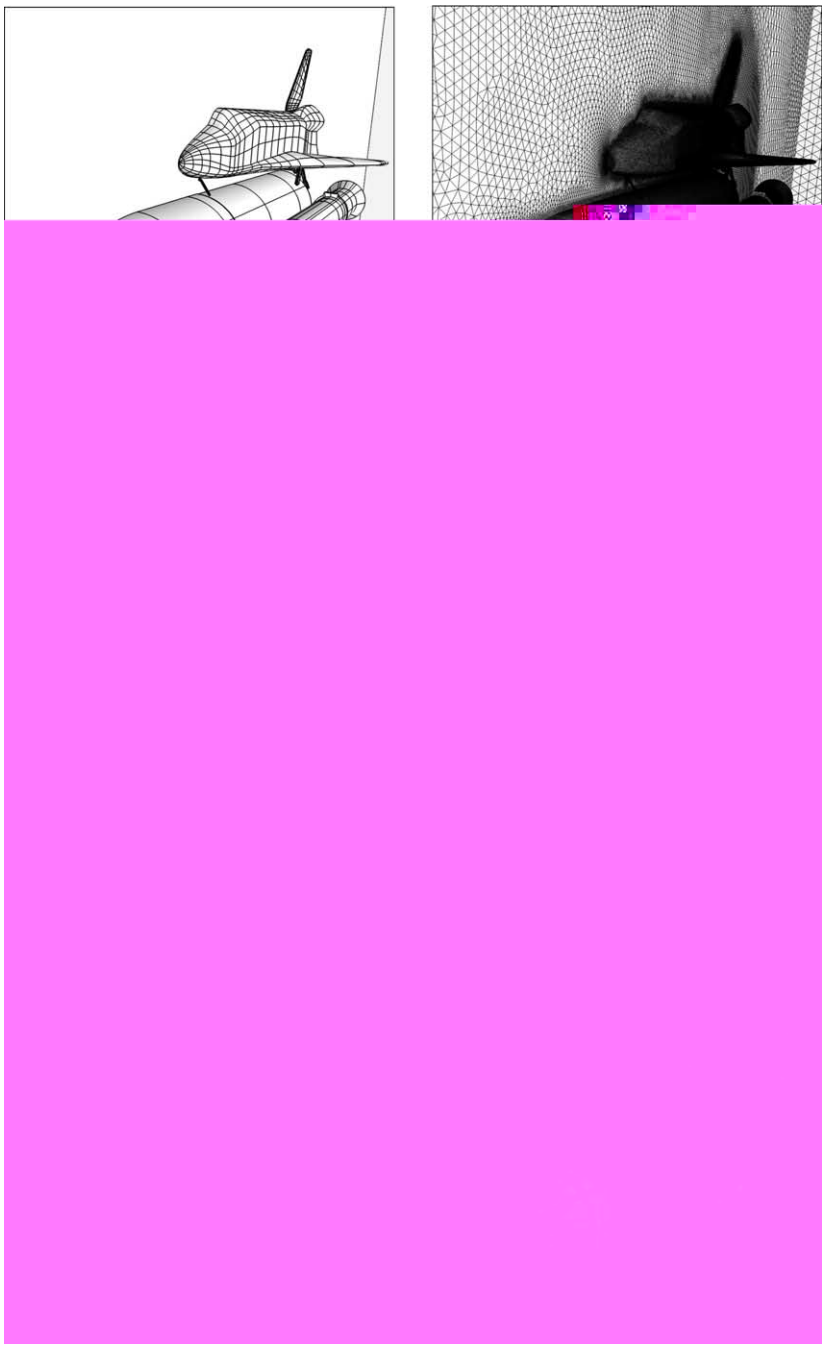

Fig. 11. (a) Shuttle ascend configuration; (b) surface discretization; (c) triangulation of planar cut 1; (d) triangulation of planar cut 2; (e) detail under wing; (f) surface pressure.

large-scale grids, this percentage can easily exceed 90%. This allows a seamless integration of unstructured grids near boundaries with efficient Cartesian grids in the core regions of the domain. A factor of two may not seem much in light of the fact that the number of points and edges in any 3-D mesh can be doubled by

Table 4
Grid statistics for shuttle ascend configuration

Grid	nelem	necrt	nedge	nacte	CPU
U	14,253,494	0	17,049,048	17,049,048	5640
C	12,439,995	5,357,014	14,920,442	11,266,645	4310

reducing the mesh size by a mere 25%, but if it can be accomplished without major changes to a code it may be worth the effort.

Having established the viability of the proposed procedure, there is ample room for improvement. For example, grouping edges according to coordinate directions would eliminate redundant operations during gradient calculations by 60%.

Future work will consider further optimization and generalization of all procedures outlined in the paper.

Acknowledgments

This work was partially supported by DTRA. Dr. Darren Rice and Dr. Young Sohn served as technical monitors.

References

[1] M. Aftosmis, D. Gaitone, T.S. Taivares, On the accuracy, stability and monotonicity of various reconstruction algorithms for unstructured meshes, AIAA-94-0415, 1994.
[2] M.J. Aftosmis, M.J. Berger, G. Adomavicius, A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries, AIAA-00-0808, 2000.
[3] T.J. Baker, Three-dimensional mesh generation by triangulation of arbitrary point sets, AIAA-CP-87-1124, in: Eighth CFD Conference, Hawaii, 1987.
[4] D.K. Clarke, H.A. Hassan, M.D. Salas, Euler calculations for multielement airfoils using Cartesian grids, AIAA-85-0291, 1985.
[5] A. Dadone, B. Grossman, An immersed boundary methodology for inviscid flows on Cartesian grids, AIAA-02-1059, 2002.
[6] E. Darve, R. Lohner, Advanced structured–unstructured solver for electromagnetic scattering from multimaterial objects, AIAA-97-0863, 1997.
[7] A. Haselbacher, J.J. McGuirk, G.J. Page, Finite-volume discretization aspects for viscous flows for mixed unstructured meshes, AIAA J. 37 (2) (1999) 177–184.
[8] C. Hirsch, Numerical Computation of Internal and External Flow, Wiley, New York, 1991.
[9] Y. Kallinderis, S. Ward, Prismatic grid generation with an efficient algebraic method for aircraft configurations, AIAA-92-2721, 1992.
[10] D.J. Kirshman, F. Liu, Cartesian grid solution of the Euler equations using a gridless boundary condition treatment, AIAA-03-3974, 2003.
[11] A.M. Landsberg, J.P. Boris, The virtual cell embedding method: a simple approach for gridding complex geometries, AIAA-97-1982, 1997.
[12] R.J. LeVeque, D. Calhoun, Cartesian grid methods for fluid flow in complex geometries, in: L.J. Fauci, S. Gueron (Eds.), Computational Modeling in Biological Fluid Dynamics, IMA Volumes in Mathematics and its Applications, vol. 124, Springer-Verlag, Berlin, 2001, pp. 117–143.
[13] H. Luo, J.D. Baum, R. Löhner, Edge-based finite element scheme for the Euler equations, AIAA J. 32 (6) (1994) 1183–1190.
[14] H. Luo, D. Sharov, J.D. Baum, R. Löhner, On the computation of compressible turbulent flows on unstructured grids, Int. J. CFD 14 (2001) 253–270.
[15] R. Löhner, Matching semi-structured and unstructured grids for Navier–Stokes calculations, AIAA-93-3348-CP, 1993.
[16] R. Löhner, Extensions and improvements of the advancing front grid generation technique, Comm. Num. Meth. Eng. 12 (1996) 683–702.

[17] R. Löhner, Applied CFD techniques, Wiley, New York, 2001.
[18] R. Löhner, Chi Yang, J.R. Cebral, O. Soto, F. Camelli, J. Waltz, Improving the speed and accuracy of projection-type incompressible flow solvers, AIAA-03-3991-CP, 2003.
[19] J.E. Melton, M.J. Berger, M.J. Aftosmis, 3-D applications of a Cartesian grid Euler method, AIAA-93-0853-CP, 1993.
[20] R.B. Pember, J.B. Bell, P. Colella, W.Y. Crutchfield, M.L. Welcome, An adaptive Cartesian grid method for unsteady compressible flow in irregular regions, J. Comput. Phys. 120 (1995) 278.
[21] C. Pflaum, Semi-unstructured grids, Computing 67 (2) (2001) 141–166.
[22] S. Pirzadeh, Viscous unstructured three-dimensional grids by the advancing-layers method, AIAA-94-0417, 1994.
[23] P.L. Roe, Approximate Riemann solvers, parameter vectors and difference schemes, J. Comput. Phys. 43 (1981) 357–372.
[24] D. Sharov, H. Luo, J.D. Baum, R. Löhner, Unstructured Navier–Stokes grid generation at corners and ridges, Int. J. Num. Meth. Fluid 43 (2003) 717–728.
[25] M.S. Shepard, M.K. Georges, Automatic three-dimensional mesh generation by the finite octree technique, Int. J. Num. Meth. Eng. 32 (1991) 709–749.
[26] G. Sod, A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, J. Comput. Phys. 27 (1978) 1–31.
[27] C. Viozat, C. Held, K. Mer, A. Dervieux, On vertex-centered unstructured finite-volume methods for stretched anisotropic triangulations, INRIA Rep. de Recherche 3464 (1998).