V European Conference on Computational Fluid Dynamics
ECCOMAS CFD 2010
J. C. F. Pereira and A. Sequeira (Eds)
Lisbon, Portugal, 14–17 June 2010

DEVELOPMENT OF TWO AND THREE-DIMENSIONAL EULER SOLVERS FOR ADAPTIVELY REFINED CARTESIAN GRIDS WITH MULTIGRID APPLICATIONS

Mehtap Cakmak[*], M. Haluk Aksel[†], and Cuneyt Sert[††]

[*]Georgia Institute of Technology
270 Ferst Drive Atlanta, Georgia 30332 USA
mcakmak@gatech.edu

[†,††]Middle East Technical University
Mechanical Engineering Department METU Ankara, TURKEY
[†]aksel@metu.edu.tr, [††]csert@metu.edu.tr

Key words: Euler Equations, Cartesian Grid, Multigrid, Marching Cubes Algorithm

Abstract. *An automated solver that works on adaptive Cartesian grids is developed to simulate inviscid, compressible flows around simple and complex geometries. In generating the Cartesian grid, marching squares and cubes algorithms are used to form interfaces of cut cells. Geometry-based cell adaptation, which includes box adaptation and cut cell adaptation, is applied in the mesh generation procedure. After obtaining an appropriate initial mesh, flow solution is obtained using either Liou's Advection Upstream Splitting Method (AUSM) or Roe's approximate Riemann solver with a cell-centered finite volume approach. Least squares reconstruction of flow variables within the computational cells is used to determine high gradient regions of flow. Solution based grid adaptation is then applied in order to refine high gradient regions. Multistage time stepping with local time steps and Fully Approximation Storage (FAS) multigrid technique is used in order to increase the convergence rate.*

1 INTRODUCTION

Cartesian grids are a special form of the unstructured grids. They consist of squares in two-dimensions and cubes in three-dimensions that are oriented parallel to the coordinate axes. Since the elements are perfectly aligned with the coordinate axes, there is no need for any complex formulation of velocity vectors for flux calculations in order to get their normal and tangential components. To store connectivity information, it is possible to use quadtree and octree data structures for two- and three-dimensional problems, respectively. An important advantage of using Cartesian grids is easy handling of complex geometries. Also, multigrid scheme can easily be applied to a Cartesian flow solver. Finally, geometric and solution based adaptations can be implemented without any user intervention.

The most important difficulty for creating a Cartesian mesh arises from contact surfaces. Computational (leaf) cells, which have intersections with boundaries, are called cut or split cells. Split cells are irregular cells, which violate the simplicity of implementation of marching squares algorithm. But both cut and split cells play an important role in working with curved geometries. In the very former studies of Cartesian grids, curved boundaries were represented as stair-step boundaries. This method was very simple, however; it produced unacceptable errors. Therefore, it is decided to use cut and split cells as the most accurate method to define the curved boundaries even though sizes of some cut cells are extremely small and as a result, they may put severe restrictions on the convergence rate. In 1993, De Zeeuw [1] solved two-dimensional Euler equations using a Cartesian grid based on quadtree data structure. He implemented a multigrid scheme to increase the convergence rate of the solution. In 1994, Coirier [2] solved two-dimensional Euler and Navier Stokes equations using Cartesian grids. He preferred binary tree data structure and performed solution based grid refinement and coarsening. In 1995, Aftosmis [3] developed techniques for meshing geometries with complex surfaces and the code CART3D solved three-dimensional Euler equations using Cartesian grids accurately. In 2004, Hunt [4] developed a code to solve the three-dimensional Euler equations by using parallel, block adaptive Cartesian grid approach. Data structure and handling the geometry were very similar to studies of Aftosmis. In 2005 and 2008, Bulgok [5] and Siyahhan [6] developed two-dimensional Euler solvers for Cartesian grids. These references are the milestones of the work presented in this paper.

2 QUADTREE DATA STRUCTURE AND TWO-DIMENSIONAL GRID GENERATION

2.1 Quadtree data structure

Dynamic data structures are used in this work to enable the variation of total number of cells during the execution of the program. In addition, quadtree data structure enables to store connectivity information between cells such as parental and neighboring information. They are important for flux calculations, reconstruction and multigrid.

Quadtree data structure can be thought as a family tree which demonstrates the relationships beginning from the oldest individual followed by his children, grandchildren, etc. The oldest individual of the family tree becomes the root of the quadtree [7]. Since each cell in a quadtree has a parent and four children, connectivity information can be extracted using parent-children relations. Figure 1 illustrates a three level quadtree data structure.

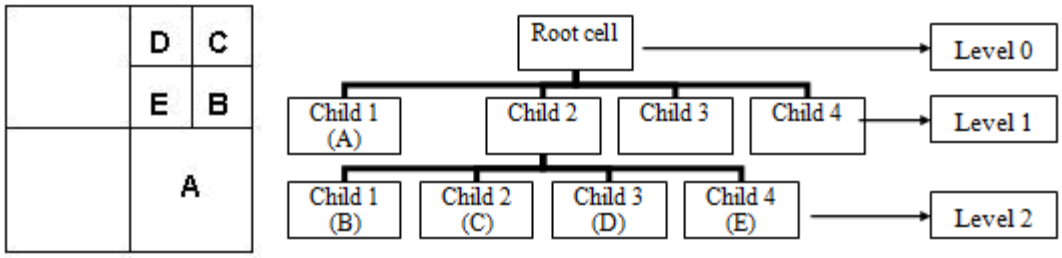


Figure 1: Numeration of cells in a three level quadtree data structure

In the developed code for two-dimensional problems, all cells are identified with their parents, four edge neighbors and four children. There are two types of neighbors for two-dimensional problems. One is edge neighbor which is stored for each cell and the other is vertex neighbor which is determined when it is required [7]. Since vertex neighbors are not used as many times as edge neighbors and also they are easily determined by the edge neighboring information, storage of them would be inefficient in terms of memory usage. Moreover, level of a cell, center and corner coordinates are stored for each cell, although the last two can also be computed when they are required in case of memory shortage.

Cells that do not have any children are called computational cells and there are four types of them. These are inside, outside, cut and split cells. All calculations are performed using computational cells except the inside ones. Additional parameters are stored for these special cells such as conservative variables, square index of a cell, gradient and curl of velocity vector, etc. Finally, additional parameters such as coordinates of centroid and cut locations are stored for only cut and split cells. Detailed information can be found in Cakmak [7].

Grid refinement and coarsening is performed according to the one level rule, which states that the level differences between two edge or vertex neighbors cannot exceed one. This rule enables grid smoothness and facilitates the flux calculation and application of reconstruction schemes. In addition, neighbor cells through the vertices of a cell can easily be determined by means of this restriction.

2.2 Initial grid generation and geometry adaptation

After the specification of input geometry in the form of line segments, first the root cell is created, followed by a uniform grid generation in which the root cell is divided into squares successively until the initial desired resolution is obtained. A sample grid obtained after uniform grid generation is shown in Figure 2.a.

The next step is inside-outside testing of corners of each computational cell, which is necessary for determination of cut cells. In this study, ray-casting method is used for this purpose since it is easily implemented in both two- and three-dimensional problems. Detailed information about ray-casting method can be found in Cakmak [7].

After the testing of all four corners of a cell, cell type has to be determined. Each corner of a computational cell has a parameter denoted by ϕ. This parameter is set as 1 or -1 for outside or inside corners, respectively. If all corners of a cell are outside or inside of the geometry, the cell type is set as outside or inside, respectively. Types of other remaining cells are set as cut cells. However, as demonstrated in Figure 3, exceptional cases should be handled carefully. Although four corners of two sample cells given in this figure are outside, the cells are actually cut by the given geometry. These cells are set as split cells and they require special treatment.

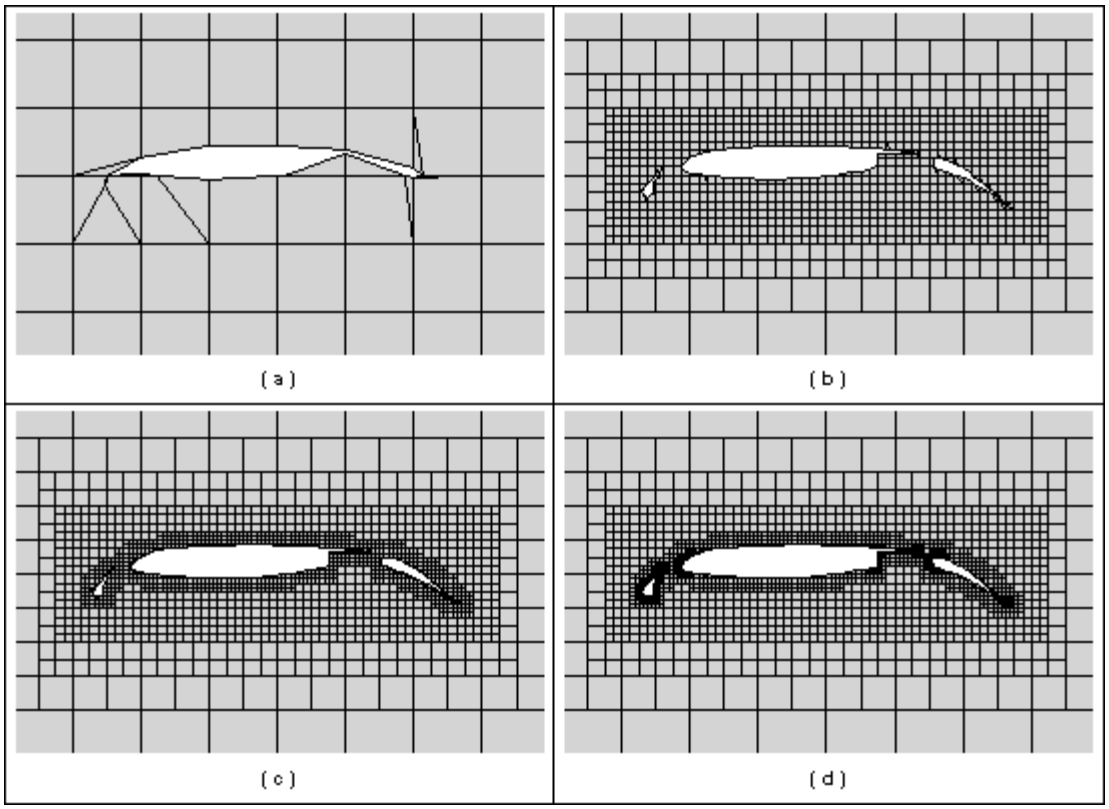


Figure 2: Sample grids for flow around a multi-element airfoil. (a) uniform grid, (b) box adaptation, (c) cut and split cell adaptation, (d) curvature adaptation

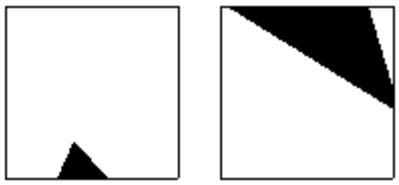


Figure 3: These two split cells are exceptional cases for inside-outside check

After the determination of cell types, it is time to make geometric adaptations around the given geometry to obtain the desired grid resolution around the geometry. Geometric adaptations can be divided into three groups for two-dimensional code: box adaptation, cut and split cell adaptation and curvature adaptation. In box adaptation cells located in an imaginary rectangular box that encloses the input geometry are refined. Size of this box is determined by the user of the program. A sample grid obtained after box adaptation is shown in Figure 2.b.

Cut and split cell adaptation is to refine the cut and split cells and also their neighbors. A sample grid obtained after this adaptation is shown in Figure 2.c. Finally, curvature adaptation is applied for cut and split cells. In curvature adaptation, angle between the normal vectors of neighboring faces that are cut by a solid body is calculated. If the angle is higher than a pre-specified threshold angle, these cells are refined. Detailed information regarding this adaptation can be found in Siyahhan [6]. A sample grid obtained after curvature adaptation is shown in Figure 2.d.

In this work, locations of cut points of cut cells are determined using marching squares and cubes algorithms for two- and three-dimensional problems, respectively. In marching squares algorithm, cut edges are automatically determined by the table given in Figure 4; hence, cut locations are found and stored easily. Marching squares

algorithm starts by indexing each corner and edge of a cell from 0 to 3 as seen in Figure 4. Square indices of corners of a cell are determined by using their ϕ values. These values are summed to calculate a total square index. By using this total square index and the given table, cut edges are determined. For example, total square index of the cell shown in Figure 4 is nine and cut edges are found as zeroth (edge0) and second (edge2) edges. After determining the cut edges, exact points at which these edges are cut are found. These points are labeled as p0 and p1 in Figure 4. By locating these points it is possible to determine the exact shape of the computational cut cell given by region defined by points p0, cr1, cr2, p1.

```
int lineTable[16][4] =
{{-1, -1, -1, -1}, //0
 {0 ,  3, -1, -1}, //1
 {1 ,  0, -1, -1}, //2
 {1 ,  3, -1, -1}, //3
 {2 ,  1, -1, -1}, //4
 {2 ,  1,  0,  3}, //5
 {2 ,  0, -1, -1}, //6
 {2 ,  3, -1, -1}, //7
 {3 ,  2, -1, -1}, //8
 {0 ,  2, -1, -1}, //9
 {3 ,  2,  1,  0}, //10
 {1 ,  2, -1, -1}, //11
 {3 ,  1, -1, -1}, //12
 {0 ,  1, -1, -1}, //13
 {3 ,  0, -1, -1}, //14
 {-1, -1, -1, -1}   };
```

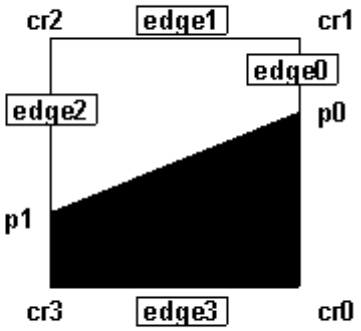


Figure 4: Line table for marching squares algorithm and a sample cut cell

As it is mentioned previously, split cells require special attention. They may contain only a single or multiple fluid regions. Flux calculation of split cells, which have only one fluid region, is the same as cut cells. On the other hand, for split cells that have two or more fluid regions, separate flux calculations are performed for each fluid region [7]. Although split cells make the data structure more complicated, they are required for the implementation of multigrid method. Besides their complexity, split cells increase the computational time and decrease the efficiency of the memory usage. For these reasons, in some previous studies, multiple fluid regions inside split cells are combined into a single fluid region [1, 5].

General method to determine and classify the split cells is to count the number of cut points on the edges of a cell. It is obvious that inside and outside cells should not have any cut points. If they have, they are set as split cells. Moreover, cut cells should have only two cut points. If they have more than two cut points, they are also set as split cells. After identifying split cells they need to be classified. A cell, labeled as inside cell initially, can have two or four cut points on its edges. Another possibility is that a cell that is labeled as outside cell initially can have two or four cut points on its edges. These cells are assigned special total square index values. A final type of split cell forms when a cell initially labeled as a cut cell has three, four or more than four cut points. If these cells have three or four cut points their total square index remains the same. Otherwise, if they have more than four cut points they are assigned a special total square index value. These final types of split cells are rarely encountered and the flux formulations for these cells are really difficult. Hence, in this work they are recursively refined until their children become other defined types. Generally, they disappear after one or two refinement processes.

Classification of split cells is important because their specially assigned total square indices are used in many processes such as determining the sequence of cut points, flux, centroid and area calculations. Classified split cells can be found in Appendix A of reference [7]. Flux calculations of split cells which have two separate fluid regions are performed by considering them as two cut cells.

3 OCTREE DATA STRUCTURE AND THREE-DIMENSIONAL GRID GENERATION

3.1 Octree data structure

Like the quadtree data structure, the octree data structure starts with the root cell at the top followed by its children, grandchildren, etc. Any cell in the data structure uses fifteen pointers to other cells in the mesh. One is for its parent; eight of them are for its children and the rest of them are for its surface neighbors which are east, north, west, south, top and bottom surface neighbors, respectively. Twelve edge and eight corner neighbors are not stored but determined when they are necessary.

In the three-dimensional code, there are three types of computational cells: inside, outside and cut cells. Since the three-dimensional grid generation is more complicated compared to two-dimensional one, split cells are not treated specially. Instead, these irregular cells which are not inside, outside or cut cells are recursively refined until their children become one of these three regular types. Consequently, the data structure is not as complicated as the two-dimensional grid generation code. In two-dimensional grid generation, centroid and area of cut cells are calculated by triangulation of the outside part of cut cells. For three-dimensional case, centroid and volume of cut cells are calculated by dividing the outside part into tetrahedrons.

3.2 Initial grid generation and geometry adaptation

After providing the solid geometry to the code as a triangulated surface, similar to the two-dimensional case uniform refinement of the root cell, box adaptation and cut cell adaptation are used to obtain the final grid that can be used by the flow solver. Sample grids obtained after box and cut cell adaptations are shown in Figure 5.

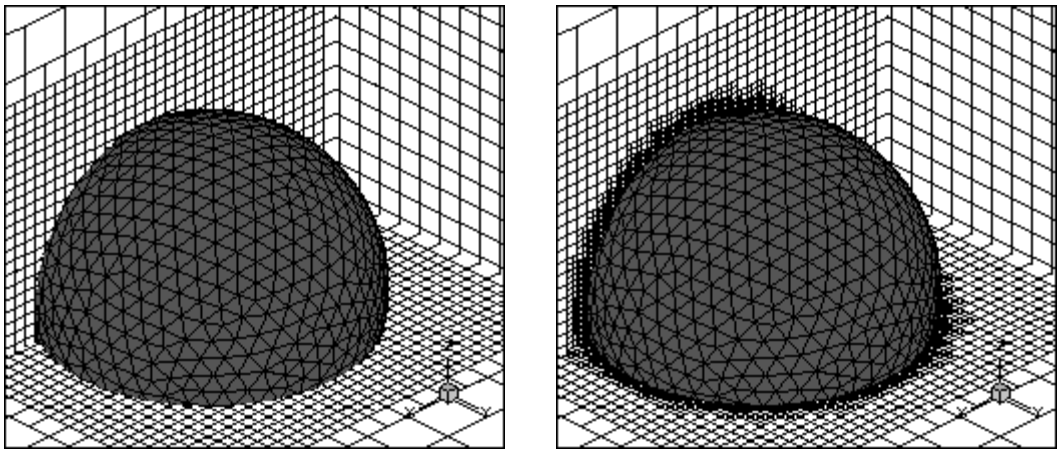


Figure 5: Application of box and cut cell adaptations for the flow around a sphere problem

After geometric adaptations, it is time to calculate the coordinates of cut points and create the cut surfaces of cut cells by using total cube indices and the table defined by marching cubes algorithm. This table can be found in reference [8]. Numbering of edges and corners is presented in Figure 6.a. Cube indices of a cell are calculated by using ϕ

value of each corner. Cube index of an outside corner is zero and cube index of an inside corner is equal to two to the power of corner number defined in Figure 6.a. For example, if the third corner of a cell is an inside cell, its cube index is two to the power three. Total cube index of a cell is calculated by summing up cube indices of all corners. Total cube index of the cell given in Figure 6.b is equal to seven, because its zeroth, first and second corners are inside the geometry. Cut edges of this cell and triangular cut surfaces are found by means of the marching cubes table, which has the following row for a cube index value of seven,

```
{ 2, 8, 3, 2, 10, 8, 10, 9, 8, -1, -1, -1, -1, -1, -1, -1}
  1st Triangle  2nd Triangle  3rd Triangle
```

According to this information, cut edges are the 2nd, 8th, 3rd, 10th and 9th edges. When cut edges are known, cut locations on these edges are found by the line-triangle intersection method given in reference [9]. After finding cut locations, three cut-triangular surfaces which pass through these points can be drawn by following the sequence given in the table. It is important to note that normal vectors of each triangular surface are pointing to the outside part of the cell. This feature of triangles facilitates the flux, volume and centroid calculations.

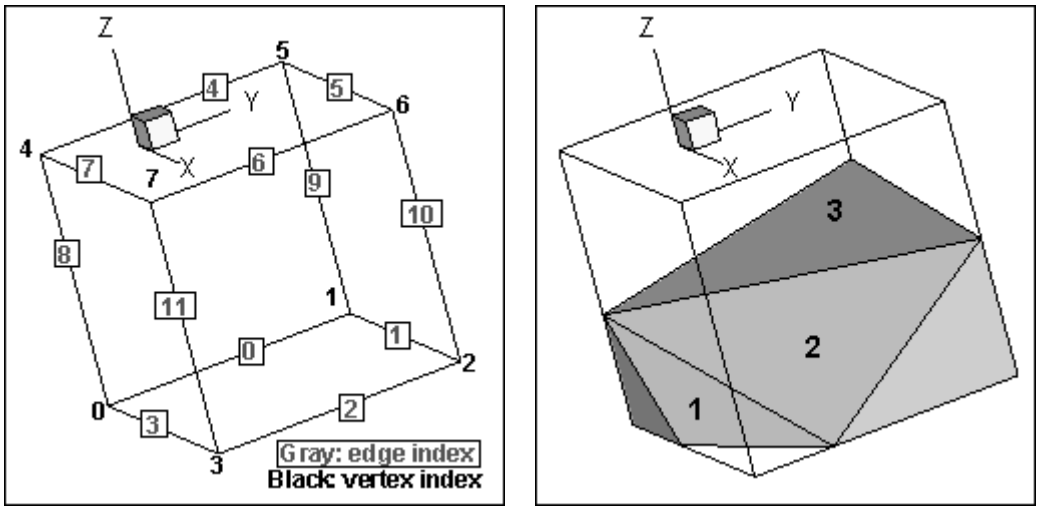


Figure 6: Details of marching cubes algorithm for a 3D case. (a) Numbering of edgesand vertices, (b) Cut surfaces obtained by using the marching cubes table

4 FLOW SOLVER

Finite-volume formulation of the three-dimensional conservative Euler equations is achieved by using a cell-centered approach. Flow variables are stored at the centroids of cells and it is assumed that they do not vary inside of the control volume. As a result, spatial discretized form of three-dimensional Euler equations can be written as

$$\frac{\partial q}{\partial t} = -\frac{1}{\Omega}\sum_{i=1}^{nFaces} \Phi_i A_i = -\frac{Res(q)}{\Omega} \tag{1}$$

where A_i and Ω are the area of the ith surface and the volume of a cell, respectively. Vector of conserved variables q and vector of fluxes passing through the cell faces Φ are defined as follows

$$q = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix} \qquad \Phi = \begin{pmatrix} \rho V_n \\ \rho u V_n + p n_x \\ \rho v V_n + p n_y \\ \rho w V_n + p n_z \\ \rho V_n H \end{pmatrix} \tag{2}$$

where $V_n = u n_x + v n_y + w n_z$ is the velocity along the unit outward normal of a surface of a cell, ρ is the density of the fluid. H is the specific total enthalpy and p is the fluid static pressure.

4.1 Multistage time stepping

In the explicit multistage time stepping method used, discretized Euler equations are solved starting from a known initial solution. In this work, three-stage time stepping as defined is used

$$\begin{aligned} q^0 &= q^n \\ q^m &= q^0 - \upsilon \frac{\alpha_m \Delta t Res(q^{m-1})}{\Omega} \\ q^{m+1} &= q^m \end{aligned} \tag{3}$$

where α_m denotes the stage coefficients and Δt is the time step. CFL numbers (υ) and stage coefficients are used as defined in reference [10].

For steady flows that are solved in the current work, local time stepping can be used. In three-dimensional problems, local time step of each cell is defined as

$$\frac{\Delta t}{\Omega_{cell}} = \frac{1}{\sum_{i=1}^{nFaces} (c_{cell} + |V_n|)_i A_i} \tag{4}$$

where c_{cell} is the local speed of sound which is calculated by using the flow variables stored at the cell centroid.

4.2 Flux computation

Two different methods, which are approximate Riemann solver of Roe [4, 7, 11, and 12] and Liou's Advection Upstream Splitting Method (AUSM) [6, 7, 11, and 13] are used for the calculation of fluxes, Φ_i, through each face.

In AUSM method, Mach number and pressure appearing in the convection flux terms are split. Split Mach number $M_{½}$, and split pressure vector $p_{½}$ are the average of Mach numbers and pressure vectors on the left and right sides of a face. The cell, whose flux values will be calculated, is denoted as left side and the neighboring cell is represented as right side.

$$\Phi(q_L, q_R) = \frac{1}{2}\left[M_{½}\left(F'(q_L) + F'(q_R)\right) - \left| M_{½} \right| \left(F'(q_R) - F'(q_L)\right) \right] + p_{½} \tag{5}$$

where

$$F'(q_L)=\begin{pmatrix}\rho c\\ \rho uc\\ \rho vc\\ \rho wc\\ \rho Hc\end{pmatrix}_L \qquad F'(q_R)=\begin{pmatrix}\rho c\\ \rho uc\\ \rho vc\\ \rho wc\\ \rho Hc\end{pmatrix}_R \tag{6}$$

$$M_{1/2}=\frac{1}{2}\left(M_L^{+}+M_R^{-}\right) \qquad p_{1/2}=\frac{1}{2}\left(p_L^{+}+p_R^{-}\right) \tag{7}$$

$$M_L^{+}=\begin{cases}\frac{1}{4}\left(M_L+1\right)^2 & |M_L|\le 1\\ \frac{1}{2}\left(M_L+|M_L|\right) & |M_L|>1\end{cases} \qquad M_R^{-}=\begin{cases}-\frac{1}{4}\left(M_R-1\right)^2 & |M_R|\le 1\\ \frac{1}{2}\left(M_R-|M_R|\right) & |M_R|>1\end{cases} \tag{8}$$

$$p_L^{+}=p_L M_L^{+}\begin{cases}2-M_L & |M_L|\le 1\\ \frac{1}{M_L} & |M_L|>1\end{cases} \qquad p_R^{-}=p_R M_R^{-}\begin{cases}-2-M_R & |M_R|\le 1\\ \frac{1}{M_R} & |M_R|>1\end{cases} \tag{9}$$

4.3 Initial guess and boundary conditions

Far-field conditions are set as the initial guess for all computation cells and two different boundary conditions are implemented into the code. First one is the solid boundary condition that size, density, pressure, specific enthalpy and tangential components of the velocity vector for the ghost cells are taken the same as the cut cells. On the other hand, normal velocity vector of a ghost cell is taken as the opposite direction of the normal velocity vector of the cut cell. The other boundary condition is far-field condition in which flow variables of ghost cells are equated to those of the far-field.

4.4 Multigrid method

Formulation of multigrid method is different for linear [14, 15, and 16] and nonlinear [1 and 17] problems.

Multigrid method is used to accelerate the convergence rate of iterative solutions. It is based on error smoothing and coarse grid principles, in which high and low frequency errors are eliminated, respectively. Transformation from fine to coarse grid is called restriction and retransformation from coarse to fine grid is called prolongation. Implementation of multigrid is achieved in four steps known as fine grid iteration, restriction, prolongation and correction as explained below.

(i) Fine grid iterations: Initially, a number of iterations are performed on the finest grid which is denoted by *h*-level by using multistage time stepping method. However, in multigrid method, formulation of multistage time stepping scheme is different. An additional parameter called forcing function (FF^h), which is set as zero for all computational cells of the finest grid, is added to the equation and the new multistage time stepping scheme becomes

$$q_m^h=q_0^h-\upsilon\frac{\alpha_m\Delta t}{\Omega}\left[Res\left(q_{m-1}^h\right)+FF^h\right] \tag{10}$$

(ii) Restriction: In this step, grid is coarsened to start eliminating low frequency errors. Transfer of *h*-level grid to 2*h*-level grid is summarized to explain the coarsening algorithm for Cartesian meshes. First of all, parent cells, whose children are all

computational cells, are flagged [18]. If the flagged parent cells do not violate the one level rule when they are coarsened, they are set as new computational cells. If they violate the rule, their flags are removed and they remain as parent cells. When 2*h*-level grid is obtained, initial approximate solutions and forcing functions of new computational cells are required. These are defined as

$$q_0^{2h} = qI_h^{2h} q_m^h = \begin{cases} \dfrac{\sum_{i=1}^{nChildren} \left(q_m^h \Omega\right)_i}{\sum_{i=1}^{nChildren} \left(\Omega\right)_i} & for\ parent\ cells\ in\ h\ level\ grid \\ q_m^h & for\ leaf\ cells\ in\ h\ level\ grid \end{cases} \tag{11}$$

$$FF^{2h} = \begin{cases} \sum_{i=1}^{nChildren} \left(Res\left(q_m^h\right) + FF^h\right)_i - Res\left(q_0^{2h}\right) & for\ parent\ cells\ in\ h\ level\ grid \\ Res\left(q_m^h\right) + FF^h - Res\left(q_0^{2h}\right) & for\ leaf\ cells\ in\ h\ level\ grid \end{cases} \tag{12}$$

After the determination of approximate solutions and forcing functions for computational cells for the coarse level, new approximate solutions are found by using the modified multistage time stepping scheme defined in the previous section.

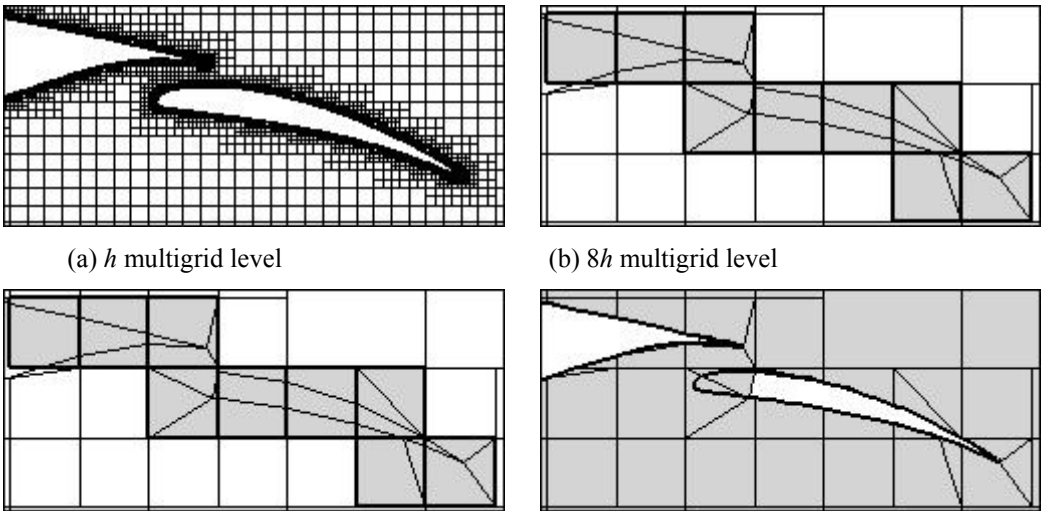
(iii) Prolongation: Approximate solution obtained using coarse grid is interpolated to fine grid which become new initial solution of the fine grid. For the interpolation operation, injection operator is used. The following equation exemplifies this process, which shows the interpolation of approximate solution from 4*h* level grid to 2*h* level grid.

$$q^{2h(new)} = q_m^{2h} + I_{4h}^{2h}\left(q^{4h(new)} - I_{2h}^{4h}\left(q_m^{2h}\right)\right) \tag{13}$$

where injection operator is defined as $I_{4h}^{2h}\left(q^{4h}\right) = q^{4h}$. In this work, the saw-tooth cycle is used and therefore, no iteration is performed in the prolongation step.

(iv) Correction and final iterations: After the improved approximation solutions, $q^{h(new)}$, of the finest mesh are obtained in the prolongation step. These values are substituted into the modified multistage time stepping scheme given previously and a number of iterations are performed.

In previous Cartesian studies, split cells are recursively refined and newly formed cells are treated as an outside, inside or cut cells. However, this procedure harms the effectiveness of the multigrid technique and it is not used in this two-dimensional study. For example, the grid given in Figure 7.a is generated around a multi-element airfoil. A few of split cells remain in the grid after recursive refinements and their sizes are very small. Therefore, their types can be transferred from split to outside, inside or cut cells by modifying the geometry. However, if the grid in Figure 7.a is taken as the first level (*h* level) of multigrid, three level (8*h* level) coarsening results in the grid given in Figure 7.b. As seen number of split cells increase in the coarsened grid and the elimination of these cells by modifying their types causes solution errors. In the extreme case, all the cells around the flap become split cells at 32*h* level grid and modifying the geometry at this level will result in a completely deleted flap. This of course degrades the solution accuracy. Generated grid around the geometry for the coarsest grid when split cells are not eliminated is shown in Figure 7.d and it is clearly seen that only the leading edge part of the flap is eliminated instead of eliminating the whole flap.



(a) *h* multigrid level (b) 8*h* multigrid level

(c) 32*h* multigrid level (d) The coarsest mesh obtained by using split cells

Figure 7: Multigrid levels for Cartesian mesh around multi-element airfoil

4.5 Reconstruction and solution adaptation

Reconstruction is required for determination of cells to be refined or coarsened. Least squares reconstruction is used in this work to calculate gradients of flow variables in a cell and estimate the value of these variables at a certain point inside the cell. In addition, a limiting procedure is used so that flow variables obtained inside a cell do not exceed the limits of variables defined for that cell and the neighboring cells. The derivation and detailed information regarding least squares reconstruction method and limiting can be found in references [7, 19, and 20].

Solution adaptation is to put more grid points in the high gradient regions and remove grid points from the regions where the gradients are low. The criteria used in this work are the gradient and curl of the velocity vectors and the strength of the entropy wave [4]. These criteria for each cell are calculated by

$$\tau_G = \left|\nabla \cdot \vec{V}\right| \Omega^{0.5} \tag{14}$$

$$\tau_C = \left|\nabla \times \vec{V}\right| \Omega^{0.5} \tag{15}$$

$$\tau_{EW} = \left|\nabla p - c^2 \nabla \rho\right| \Omega^{0.5} \tag{16}$$

Then, the standard deviations of these three criteria are found for the whole mesh by the following equation

$$\sigma_\alpha = \sqrt{\frac{\sum_{i=1}^{nCells} (\tau_\alpha)_i^2}{nCells}} \tag{17}$$

After the calculations of standard deviations of three criteria, cells that need to be refined and coarsened can be determined. A cell is selected for refinement if $(\tau_\alpha)_i > \sigma_\alpha$ for any α and selected for coarsening if $(\tau_\alpha)_i < 0.1\sigma_\alpha$ for all three criteria.

5 RESULTS AND DISCUSSIONS

5.1 Transonic flow about RAE 2822 airfoil

The inviscid, steady-state flow is computed around RAE 2822 airfoil at a Mach number of 0.75 and an angle of attack of 3°. Transonic flow is selected to demonstrate that shock location and surface pressure coefficients can be obtained accurately and effectively by using Cartesian mesh. Importance of solution adaptation and multigrid application are depicted by comparing different test cases for this flow. Table 1 presents lift and drag coefficients, and convergence histories of 8 cases, which differ by the number of adaptive refinement cycles used. As seen from the table, accuracy of computed lift and drag coefficients of the test cases are directly proportional to the number of refinement cycles. It is important to note that when the number of refinement cycles is increased too much, very slight improvement of the accuracy is observed for the drag and lift coefficients but the increment of refinement cycle number has a drastic influence on the convergence time.

# of cases	# of adaptive ref. cycles	C_L	C_D	# of cells	Time
Case 1	None	0.745	0.070	2308	36 s
Case 2*	None	0.745	0.070	2308	89 s
Case 3	1	0.857	0.055	4848	52 s
Case 4	2	0.921	0.048	10029	163s
Case 5	3	0.958	0.046	18492	428 s
Case 6	4	0.973	0.045	32268	1477 s
Case 7	5	0.988	0.044	54254	3478 s
Case 8*	5	0.988	0.044	54254	22087 s
Case 9	Reference [21]	1.104	0.045	20480	-

(*Multigrid is not applied to these cases)
Table 1: Comparison of results for transonic flow around RAE 2822

The far-field boundary is approximately located 10 chords ahead the airfoil. For the computed solutions with multigrid application, six levels of grids are used. As seen in Table 1 and Figure 8, the best results are the 6th and 7th cases where numbers of adaptive refinement cycles are four and five, respectively. Mach contours of the 7th case are given in Figure 9. As seen in this figure, Mach number just before the upper shock reaches to 1.5. In addition, the shock and the wake are resolved well. The grid used for 7th case is shown in Figure 10. Finer meshes around the shock are easily seen in this figure. As it is seen in Figure 8, results for the lower surface pressure coefficients are quite successful. However, the upper surface pressure coefficients around the leading edge region for the 7th case are underestimated. The reason of this condition may be the omission of the viscous effects. Finally, the effect of multigrid is seen in Table 1. It is obvious that solution accuracy is not affected by the multigrid application; however, the convergence time is significantly reduced. According to the values given in the table, the convergence time of 1st case with multigrid application is about 40% of the 2nd case, which is the solution without multigrid application. On the other hand, the convergence time of 7th case with multigrid application is about 16% of the 8th case, which is the solution without multigrid application. It is concluded that multigrid application decreases the convergence time effectively when the grid is relatively fine.

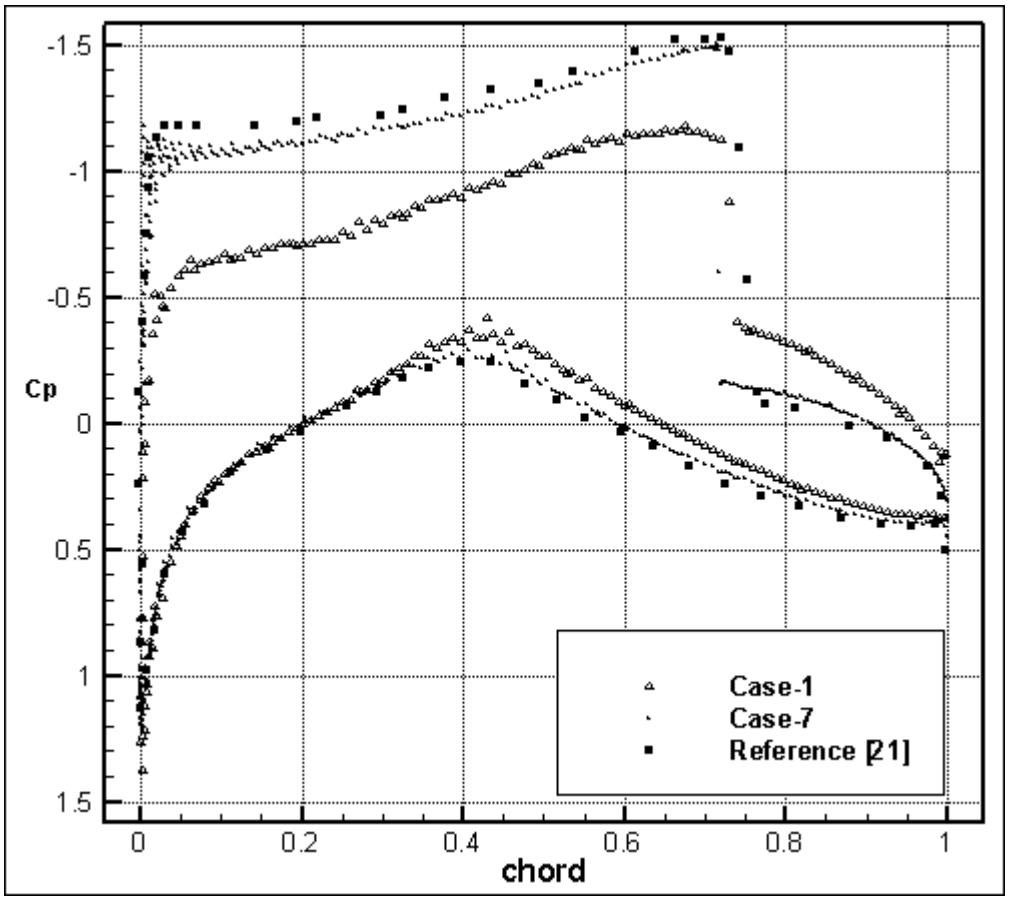


Figure 8: Pressure coefficients on RAE 2822 airfoil at $M_\infty = 0.75$ and $\alpha = 3°$

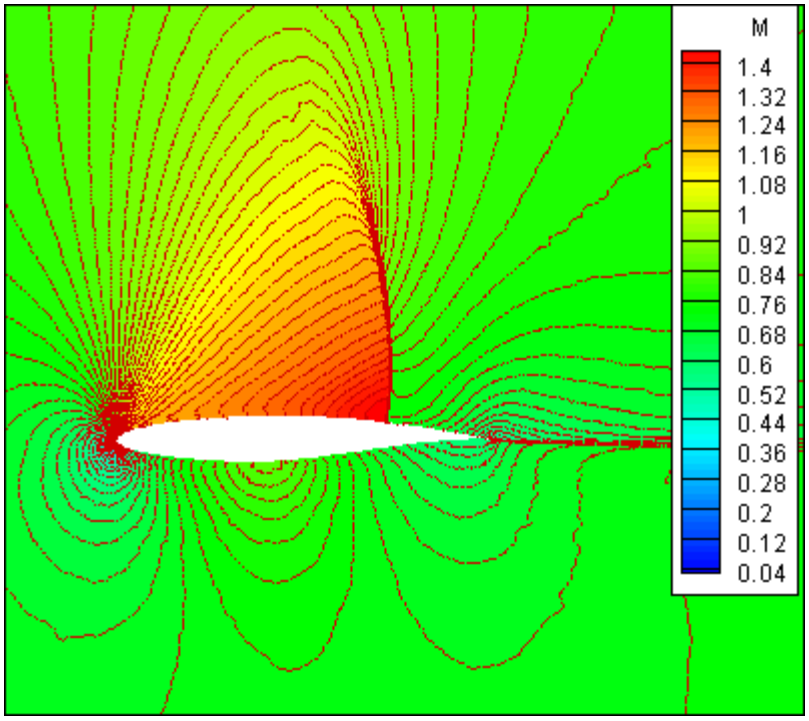


Figure 9: Mach contours of 7th case

Figure 10: Finest grid obtained after the 5th refinement cycle for 7 case

5.2 Subsonic flow about a two-element airfoil

For this case, flow is around NLR7301 airfoil and flap at a Mach number of 0.185 and angle of attack of 6°. The far-field boundary is approximately located 15 chords ahead of the airfoil. The mesh is adaptively refined three times based on the solution and six levels of grids are used for multigrid application. The lift and drag coefficients computed for the flow are C_l=1.49 and C_d=0.148, respectively. The comparison between the calculated and experimental [22] pressure coefficients are given in Figure 11. It is clearly seen from the figure that the peak of the pressure coefficient on the upper surface of the airfoil is not captured correctly. The reason is that the flow regime of this problem is not suitable for the developed code, which is designed to work better for compressible flows. As a result of the underestimation of pressure coefficient, lower lift coefficient is obtained for this case.

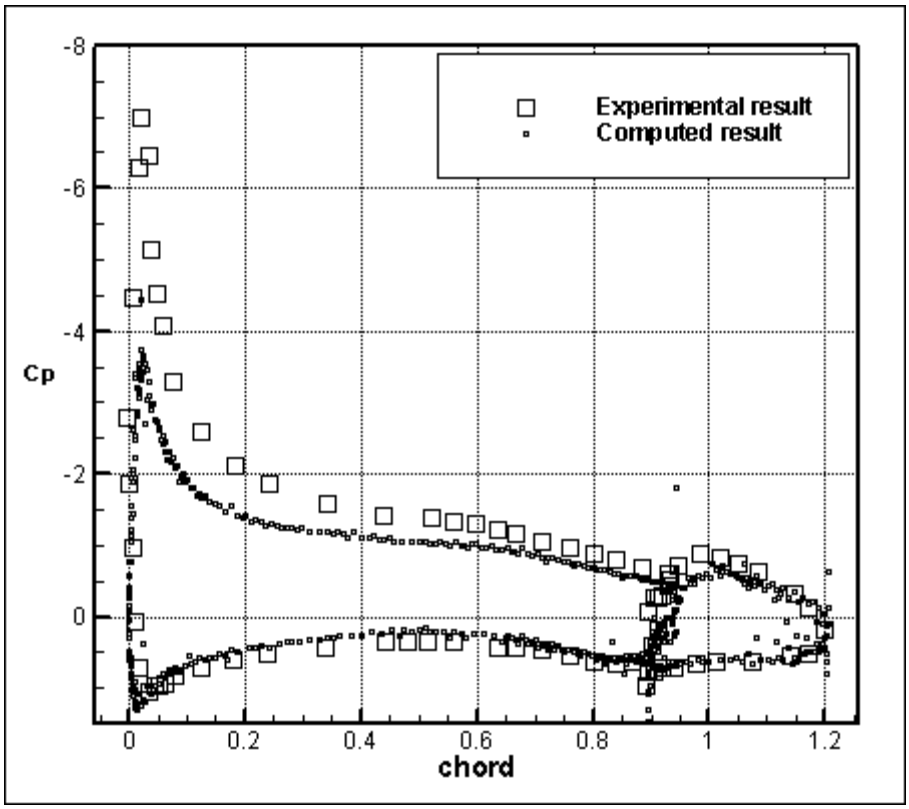


Figure 11: Pressure coefficients on two-element airfoil at $M_\infty = 0.185$ and $\alpha = 6°$

5.3 Three-dimensional transonic flow about a projectile

The inviscid, steady-state flow around a secant-ogive-cylinder-boat tail projectile (SOCBT) with a boat tail angle of 7° is tested at a Mach number of 0.95 and zero angle of attack. The far-field boundary is located 10 times the maximum length of the projectile ahead of the tail. The developed flow solver is iterated until the average density residual reaches 10^{-7}. Only one refinement cycle is applied to the geometric-adapted grid due to the limits of the available computational resources.

Figure 12 shows pressure coefficients compared with experimental results extracted from [23]. Mach contours on the symmetry plane passing through the centerline of the projectile are given in Figure 13.

As it is seen in Figures 12 and 13.b, there are two shock waves, one is at the midpoint of the chord and the other is at the boat tail. The computed and experimental results are in good agreement. Using more adaptive refinements is expected to obtain more accurate solutions.

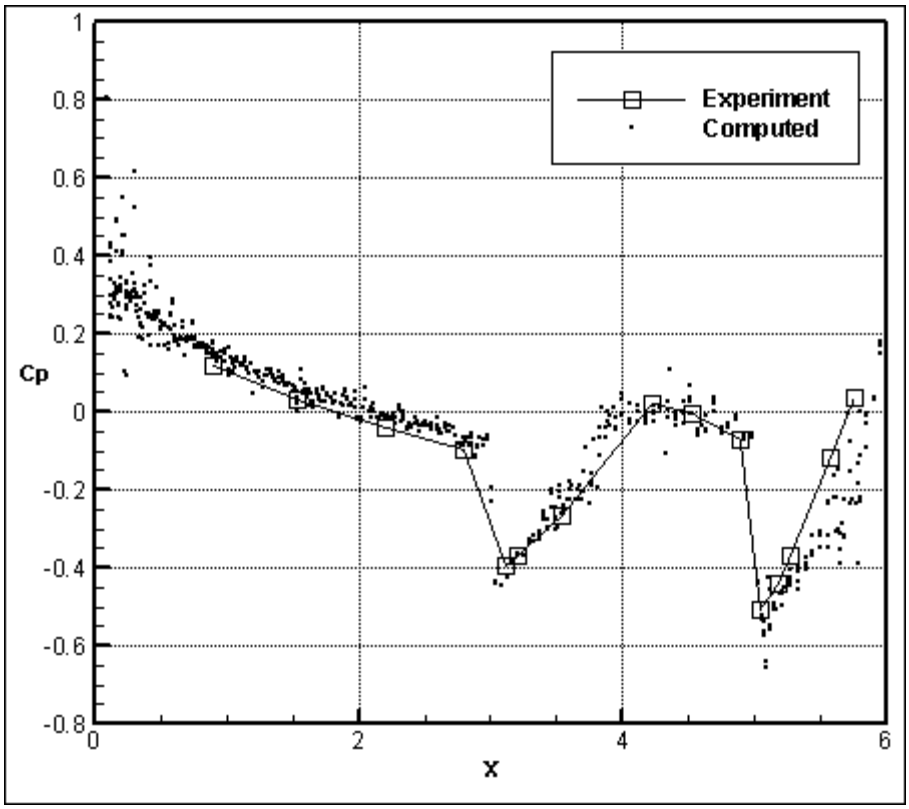
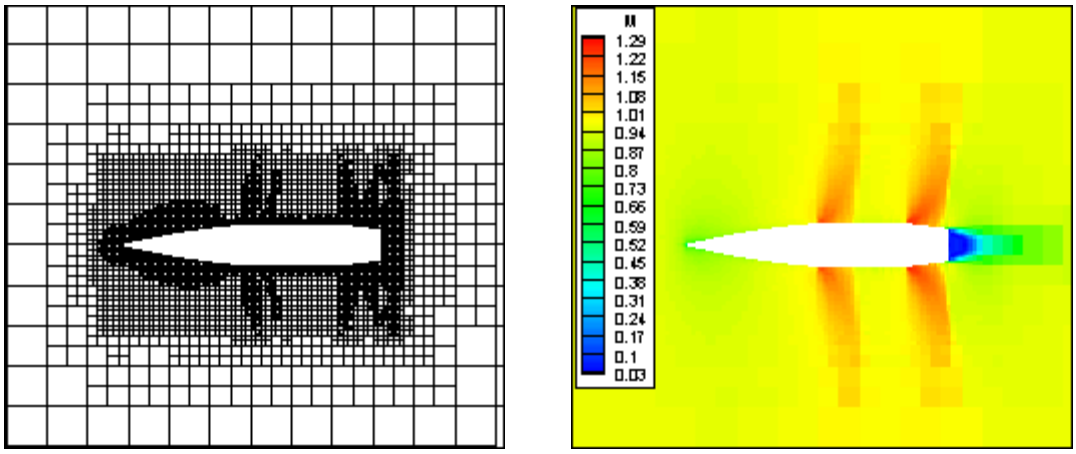


Figure 12: Pressure coefficients on the projectile at $M_\infty = 0.95$, $\alpha = 0°$ and $\beta=7°$



(a) (b)

Figure 13: (a) A slice of the mesh in *xz* plane at *y*=0, (b)Mach contours in *xz* plane at *y*=0

6 CONCLUSION

A solver based on finite volume formulation using adaptive Cartesian grids is developed for the simulation of steady, inviscid and compressible flows. Quadtree and octree data structures are applied successfully and connectivity information is extracted from these trees effectively. Automated grid generation and solution adaptation are easily implemented and their benefits are validated by the results.

The solver is verified with the experimental and previous computational results. The ability of capturing shock waves and wakes, advantages of solution based refinement and multigrid application are presented in this work. It is important to note that domain size selection and determination of refinement cycles affect the ability of capturing shock waves. Therefore, the distance between far field and the geometry is generally at

least 10 times the maximum length of the geometry and the number of adaptive refinement cycles is taken at least three for two-dimensional test cases. Multigrid application is used for almost all solutions and its contribution to the convergence rate can be easily detected as given in Table 1.

Main future work is to enhance the code so that it can perform parallel, block adaptive Cartesian grid solutions of three-dimensional problems. Oscillations of pressure coefficients seen for some cases are due to the existence of very small cut cells. Advanced treatment for small cut cells can be considered to increase the accuracy and run time efficiency of the code.

REFERENCES

[1] D. L. DeZeeuw, A Quad-Tree Based Adaptively-Refined Cartesian-Grid Algorithm for the Solution of The Euler Equations, *PhD Thesis in the University of Michigan* (1993)

[2] W. J. Coirier, An Adaptively Refined, Cartesian, Cell-Based Scheme for the Euler and Navier Stokes Equations, *PhD Thesis in the University of Michigan* (1994)

[3] M. J. Aftosmis, Solution Adaptive Cartesian Grid Methods for Aerodynamic Flows with Complex Geometries, *Von Karman Institute for Fluid Dynamics Lecture Series 28th Computational Fluid Dynamics* (1997)

[4] J. Hunt, An Adaptive 3D Cartesian Approach for the Parallel Computation of Inviscid Flow about Static and Dynamic Configurations, *PhD Thesis in the University of Michigan* (2004)

[5] M. Bulgök, A Quadtree-based Adaptively-refined Cartesian-grid Algorithm for Solution of the Euler Equations, *MS Thesis in the Middle East Technical University* (2005)

[6] B. Siyahhan, A Two Dimensional Euler Flow Solver on Adaptive Cartesian Grids, *MS Thesis in the Middle East Technical University* (2008)

[7] M. Cakmak, Development of Two and Three-dimensional Euler Solvers for Adaptively-refined Cartesian Grids with Multigrid Applications, *MS Thesis in the Middle East Technical University* (2009)

[8] P. Bourke, Polygonising a Scalar Field (1994)

[9] T. Möller and B. Trumbore, Fast, Minimum Storage Ray/Triangle Intersection, *Journal of Graphics, gpu and Game Tools*, **2**, pp. 21-28 (1997)

[10] J. Blazek, Computational Fluid Dynamics: Principles and Applications (2005)

[11] B. Laney Culbert, Computational Gas Dynamics 1998.

[12] C. Hirsch, Numerical Computation of Internal and External Flows Volume 1 & 2 (1990)

[13] M. S. Liou and C. J. Steffen, A New Flux Splitting Scheme, *Journal of Computational Physics*, **107**, pp.23-39 (1993)

[14] U. Trottenberg, C. W. Oosterlee and A. Schüller, Multigrid (2001)

[15] W. L. Briggs and S. F. McCormick, Multigrid Tutorial (2000)

[16] H. K. Versteeg and W. Malalasekera, An introduction to Computational Fluid Dynamics: The Finite Volume Method (2007)

[17] A. Jameson, Solution of the Euler Equations for Two-Dimensional Transonic Flow by a Multigrid Method, *Applied Mathematics and Computation*, **13**, pp.327-355 (1983)

[18] M. J. Aftosmis, M. J. Berger and G. Adomavicius, A Parallel Multilevel Method for Adaptively Refined Cartesian Grids with Embedded Boundaries, *AIAA Paper AIAA 2000-0808 38th Aerospace Sciences Meeting and Exhibit* (2000)

[19] T. J. Barth, and P. O. Frederickson, Higher Order Solution of the Euler Equations, *AIAA Paper AIAA-90-0013* (1990)
[20] T. J. Barth and D. C. Jespersen, The design and Application of Upwind Schemes on Unstructured Meshes, *AIAA Paper AIAA-89-0366* (1989)
[21] AGARD Subcommittee C., Test Cases for Inviscid Flow Field Methods, *AGARD Advisory Report 211* (1986)
[22] B. Van den Berg, and J. H. M . Gooden, Low-speed Pressure and Boundary Layer Measurement Data for the NLR 7301 Airfoil Section with Trailing Edge Flap
[23] Fu Jan-Kaung, and Liang Shen-Min, Drag Reduction for Turbulent Flow over a Projectile: Part I, *Journal of Spacecraft and Rockets*, **31**, pp. 85-92 (1994)