7. ANKARA INTERNATIONAL AEROSPACE CONFERENCE
11-13 September 2013 - METU, Ankara TURKEY

AIAC-2013-027

A QUAD-TREE BASED AUTOMATIC ADAPTIVE CARTESIAN GRID GENERATOR WITH APPLICATIONS ON MULTI-ELEMENT AIRFOILS

Emre Kara* and Ahmet İhsan Kutlar†
University of Gaziantep
Gaziantep, Turkey

Mehmet Haluk Aksel‡
Middle East Technical University
Ankara, Turkey

ABSTRACT

Cartesian methods are a special branch of unstructured techniques under the general grid generation methodology. In this study, it is aimed to improve automatic grid generation using Cartesian grid generation technology. Cartesian grids are generated by using a cell-based data structure. Quad-tree data structure is constructed in order to connect the Cartesian cells to each other. This dynamic data structure stores connectivity information for each cell and can create or destroy cells repeatedly, anywhere in the flow domain as the programmer wills. The solution algorithm is implemented in object-oriented FORTRAN programming language and the code employs Cartesian grid technique to model complex geometries. The grid typically begins with a single root cell, and grows by a recursive subdivision of each cell into its four children which is done in coarsening part of the FORTRAN 90 subroutines. Any kind of adaptation is very easy to accomplish in Cartesian grid generation. Four multi-element airfoil test cases; a two-element airfoil (NLR-7301 airfoil/flap geometry), a three-element airfoil (30P-30AG), a four-element airfoil (BOEING model TR-1332) and a five-element airfoil, are presented at the end of the paper to clarify the use of some special Cartesian algorithms, namely inside/outside test, cut/split cell determination and curvature adaptation algorithm.

INTRODUCTION

In computational fluid dynamics (CFD), a proper grid is to be obtained on the physical domain of the problem. Basically, there are three main grid generation techniques employed: structured, unstructured and Cartesian methods. Cartesian methods are simply a special class of unstructured methods. They were first proposed at the beginning of the 1970's as an alternative to structured and ordinary unstructured methods [Peskin, 1972] and they have recently become one of the popular and widely used methods in CFD [Bai, Li, Zou and Wang, 2007; Berger and Aftosmis, 2012; Ji, Lien and Yee, 2010; Kupiainen and Sjögreen, 2009; Liu, Zhao, Hu, Goman and Li, 2013; Sang and Yu, 2011]. The aim was to enhance automatic grid generation and facilitate solution adaptation.

The governing equations are discretized on a Cartesian grid which does not conform to the immersed boundaries. This immensely simplifies grid generation task and also retains the relative simplicity of the governing equations in Cartesian coordinates. In addition, this method also has a weighty advantage over the conventional body-fitted approach in simulating flows such as with moving boundaries, complicated shapes, or involving topological changes.

Structured and ordinary unstructured methods rely on user interference to some extent. Cartesian methods, on the other hand, permit automatic grid generation. Therefore, in order to handle problems and reduce the user intervention through the grid generation process, both the grid generation and adaptation processes are automated. In this respect, what remains to the user as a task is the suitable definition of the problem.

METHOD

The code utilizes Cartesian grid techniques to model complex geometries, regardless of the body shape, and regardless of number of bodies. An important advantage of Cartesian methods is that any kind of adaptation can easily be implemented. Hence, relatively high accuracy levels can be achieved with a relatively low number of cells. This saves computational time and memory allocation required significantly.

* Res. Ast. and Ph.D. Candidate in Mechanical Engineering Department, Email: emrekara@gantep.edu.tr
† Dr. In Mechanical Engineering Department, Email: aikutlar@gantep.edu.tr
‡ Prof. In Mechanical Engineering Department, Email: aksel@metu.edu.tr

Cartesian grid generation

In the literature, there are various methods used for fluid flow problems to identify connectivity information such as two dimensional arrays, linked lists, binary trees, quad-tree data structures [Çakmak, 2009]. In this study, the Cartesian grid is obtained by using the quad-tree approach in two dimensions. In the quad-tree approach, a square containing the whole domain is divided into its four quadrants to form the grid in a tree structure.

The multiplication of the maximum length by the user-defined size factor defines the whole domain size. This domain size is actually the length of each edge of the root cell. A uniform grid for the two dimensional Cartesian geometry is obtained by dividing squares successively starting from this root cell to the smallest "child" cells so that these child cells have an appropriate cell size near the solid body. This gives a multi-grid system with fine cells near the solid body, where higher resolution is needed for high gradients [Çakmak, 2009]. Each child is geometrically contained within the boundaries of the parent cell, and is located logically below the parent cell in the tree by one-level rule. Arbitrary subdivisions of the cells are premitted during the process, only requiring that the newly created cells be non-overlapping polygons that fill the space occupied by the mother cell. To take advantage of the smooth grid that can be achieved, the root cell is taken to be a square Cartesian cell, and cell division is obtained by isotropically splitting each cell into four, equal area children. The illustration in Figure 1 defines a simplified four-level quad-tree class. Since N-sided cut cells are obtained by "cutting" them out of their background Cartesian cell, they are always logically locatable in the tree [Samet, 1988].

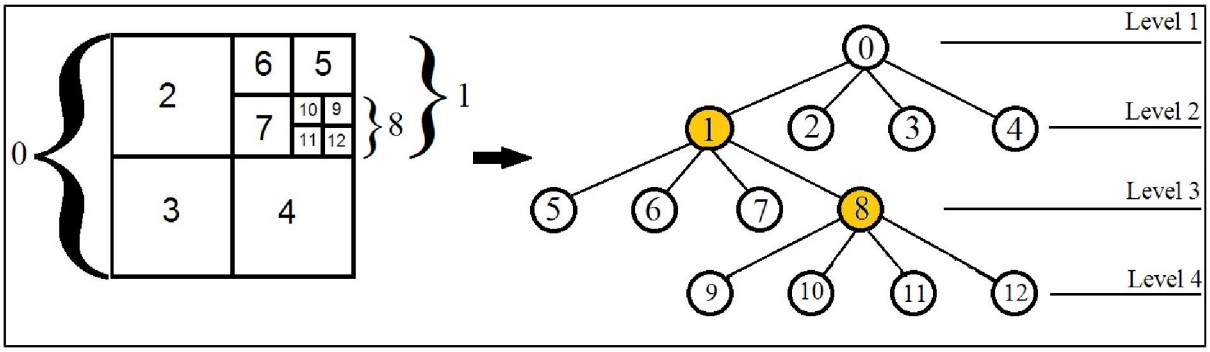

Figure 1: An Illustration of quad-tree data structure

Each cell in the Cartesian grid is classified as; inside the body, cut or split by the body and outside of the body, or in the outer boundary based on the information found at the corners of that cell. With this approach, very small cells can appear in the grid. The only limit placed on the cell size is that a node that is within a very small (machine-zero level) tolerance of a body is considered on that body, which in effect 'pulls' the body out to the node. The remaining cell areas vary as much as six orders of magnitude from one cell to its immediate neighbor. These neighbors are necessary because they balance the size of the cells to give accurate results, especially near the solid boundary.

As a first step, an initial uniform grid is formed in the solution domain which will be refined in the later stages to form the grid on which the solution is to be performed. Generation of the initial grid begins with the creation of the 'root' cell. Its size is determined from the size of the flow field and by what the user determines to be the coarsest acceptable grid for that flow field. Then, cells without children, initially just the root cell, are refined until the grid reaches the coarsest acceptable grid. This grid serves as the initial grid for cases in which no body is cut out of the grid. The procedure for computing the intersections of the body with the grid depends upon how the body has been defined. The next step after the initial grid generation is the box adaptation. An imaginary rectangular box is generated around the input geometry and finer grids are flagged for refinement near the input body. Thirdly, the corners of sufficiently small cells are tested whether they are inside the boundary of the given geometry or not. This is called inside-outside testing. This step is obligatory for Cartesian grids because the cells that are cut by the given geometry are determined by this test. The last step in the grid generation process is the curvature adaptation. The purpose of the curvature adaptation is to ensure that regions of a body that have high curvatures are resolved enough to be represented accurately [Siyahhan, 2008]. In a previous study [Kara, Kutlar and Aksel, 2013], two distinct test cases are presented to clarify the use of the methodology in detail. Sample grids around an airfoil obtained to clarify these four steps is shown in Figure 2. Detailed and close-up view of the curvature-adapted airfoil (Figure 2-d) can be seen in Figure 10.

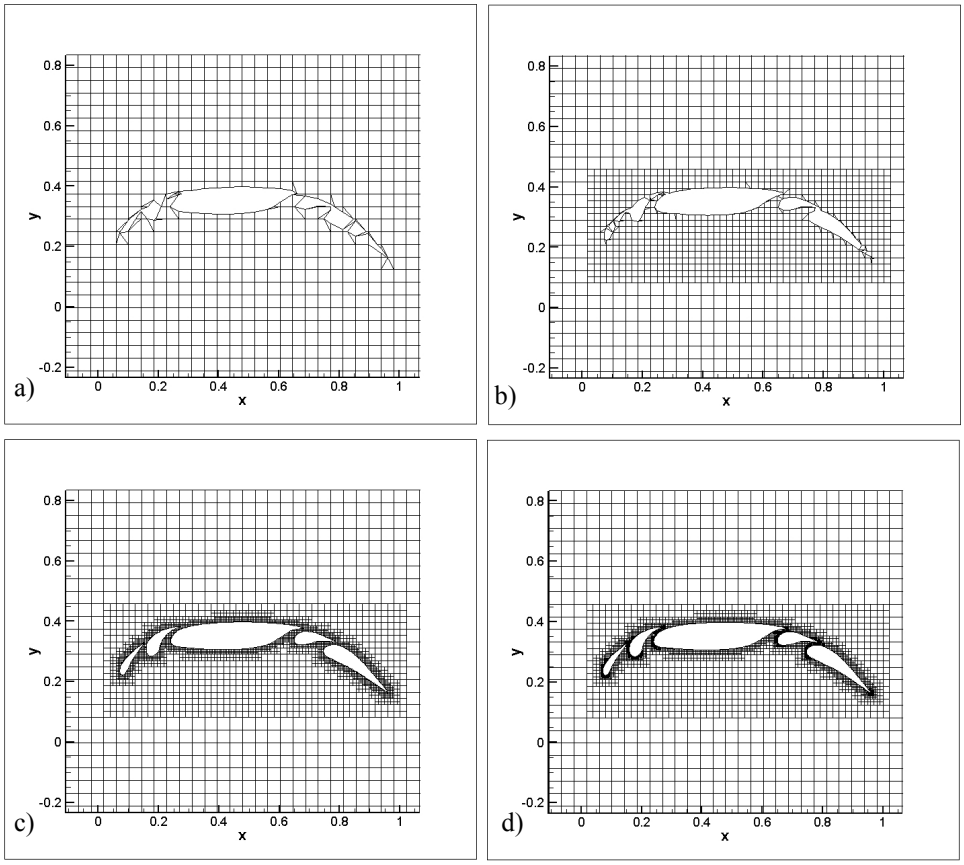

Figure 2: Sample grids around a five-element airfoil. (a) initial uniform grid, (b) box adaptation, (c) cut/split cell adaptation, (d) curvature adaptation

A flowchart implementation in Figure 3 shows a combination of recursively defined algorithms for creating the final grid using the object-oriented [Akın, 2003] FORTRAN programming language. The marching-squares algorithm, the bridge between the adaptations and the initial grid generation, will be explained in detail.

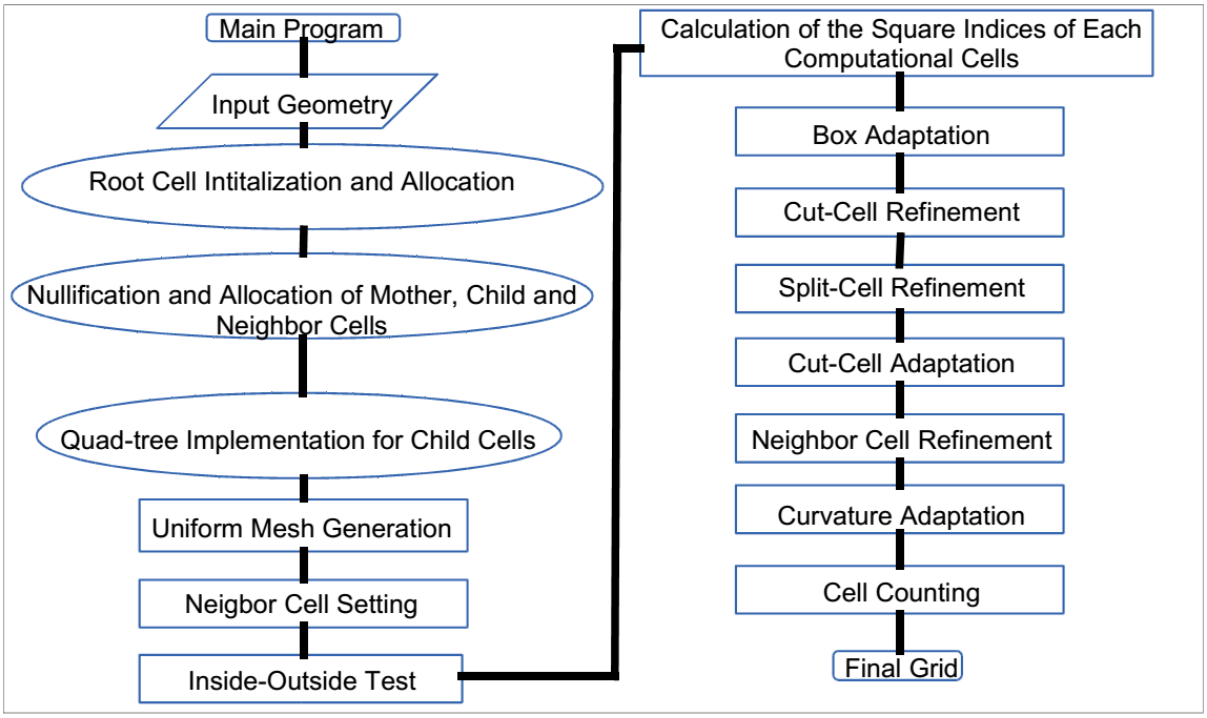

Figure 3: Flowchart diagram of the algorithms written in FORTRAN language

Marching-squares algorithm

Next to the solid boundary, cut and split cell locations are determined by marching-squares method [Çakmak, 2010]. These locations are selected automatically from the table given in Figure 4. The algorithm starts with the calculation of the square indices of each computational cells. The values are summed to calculate a total square index, then the cut edges are determined. An example for numbering of cut points of a computational cell is shown in Figure 4, the total index of the cell is 11 and the cut edges are the first and the second edges. The exact cut points are flagged as p0 and p1. These two points are necessary to determine the exact shape of the computational cut cell by the region. Also, the pseudocode for the determination of split cells is given in Figure 5.

```
integer, dimension(16,6) :: lineTable

lineTable(1,:) = (/-1, -1, -1, -1, -1, -1/) !0
lineTable(2,:) = (/0 ,  3, -1, -1, -1, -1/) !1
lineTable(3,:) = (/1 ,  0, -1, -1, -1, -1/) !2
lineTable(4,:) = (/1 ,  3, -1, -1, -1, -1/) !3
lineTable(5,:) = (/2 ,  1, -1, -1, -1, -1/) !4
lineTable(6,:) = (/2 ,  1,  0,  3, -1, -1/) !5
lineTable(7,:) = (/2 ,  0, -1, -1, -1, -1/) !6
lineTable(8,:) = (/2 ,  3, -1, -1, -1, -1/) !7
lineTable(9,:) = (/3 ,  2, -1, -1, -1, -1/) !8
lineTable(10,:) = (/0 ,  2, -1, -1, -1, -1/) !9
lineTable(11,:) = (/3 ,  2,  1,  0, -1, -1/) !10
lineTable(12,:) = (/1 ,  2, -1, -1, -1, -1/) !11
lineTable(13,:) = (/3 ,  1, -1, -1, -1, -1/) !12
lineTable(14,:) = (/0 ,  1, -1, -1, -1, -1/) !13
lineTable(15,:) = (/3 ,  0, -1, -1, -1, -1/) !14
lineTable(16,:) = (/-1, -1, -1, -1, -1, -1/)
```

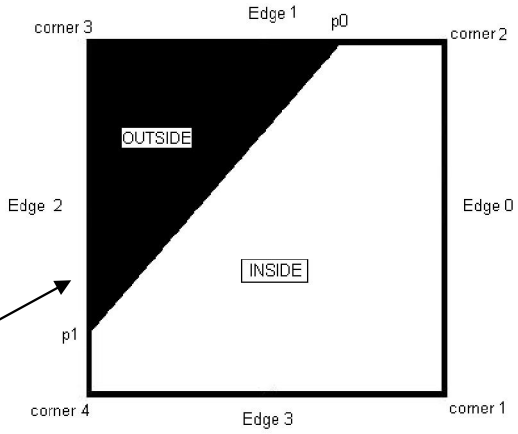

Figure 4: Line table of the marching square method written in FORTRAN and a sample cut cell

```
RECURSIVE SUBROUTINE SQUAREINDEX(CELL)

    IF (CELL IS NOT COMPUTATIONAL) THEN
        CALL SQUAREINDEX(EACH CHILDREN OF THE CELL)
    END IF

ALLOCATE MEMORY FOR THE SQUARE INDEXES
INITIALIZE THE REFINING INDICATORS AND SQUARE INDEXES TO ZERO

IF (CELL IS OUTSIDE) THEN
        SQUARE INDEX OF THE CELL IS ZERO
ELSE IF (CELL IS INSIDE) THEN
        SQUARE INDEX OF THE CELL IS 15
ELSE IF (CELL IS CUTCELL) THEN
        ASSIGN THE SQUARE INDEX WITH RESPECT TO CORNER COUNTS
END IF

ASSIGN INTERSECTION NUMBERS OF THE CORNERS OF THE CELL

..... SOME CELL OPERATIONS .....

WRITE CORNER LOCATIONS OF THE CELL WITH RESPECT TO ITS TYPE
DETERMINE SPLIT CELLS

END SUBROUTINE  SQUAREINDEX
```

Figure 5: Pseudocode for the determination of cut and split cells

RESULTS AND DISCUSSION

Quad-Tree Data Structure Verification Results

The efficiency of quad-tree data structure is verified by comparing with other techniques to treat data structure [Sul and Kwon, 2012]. Test grid is generated by initial cell refinement. Level 3 grid has 256 nodes and 64 cells. Level 5 grid has 4096 nodes and 1024 cells. Level 8 grid has 262144 nodes and 65536 cells. The grid levels are shown in Figure 6. Data size and generating time of each level and each data structure are tabulated in Table 1. Quad-tree data structure is about five times more efficient in time at level 5 grid and ten times more efficient in time at level 8 grid with respect to linked list data structure.

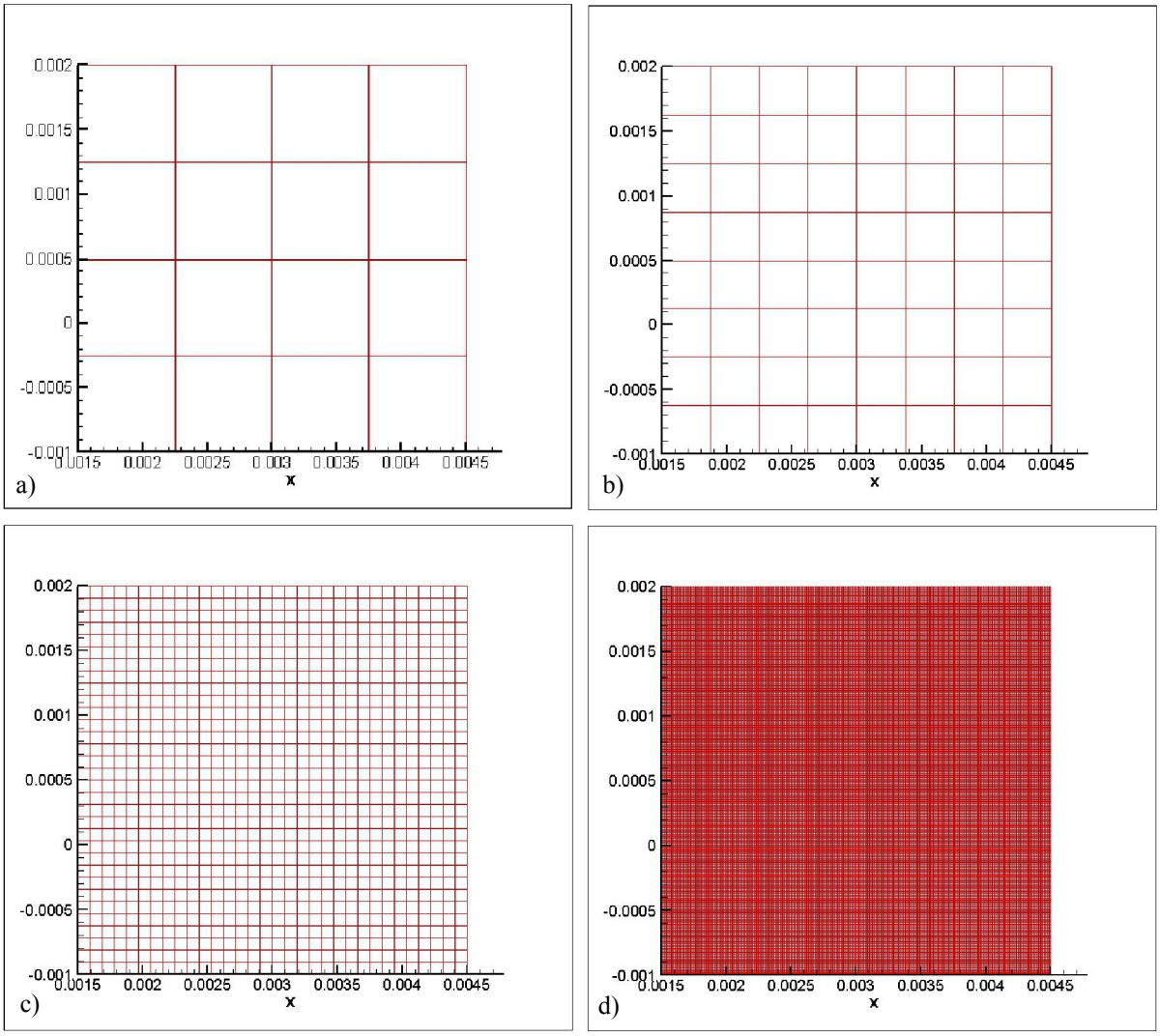

Figure 6: Test Grid; (a) initial grid, (b) level 3, (c) level 5, (d) level 8

	Level	Time	Time Compared to Level 3	Data Size
Two Dimensional Array	3	0.015 s	-	420 kb
Binary Tree	3	1.033 s	-	400 kb
Linked list	3	0.016 s	-	308 kb
Quad-Tree	3	0.010 s	-	11 kb
Two Dimensional Array	5	2.703 s	180.2	772 kb
Binary Tree	5	-	-	-
Linked list	5	1.459 s	91.19	532 kb
Quad-Tree	5	0.270 s	20.7	169 kb
Two Dimensional Array	8	28m30.54 s	114036	26 Mb
Binary Tree	8	-	-	-
Linked list	8	3m10.28 s	11892.5	10 Mb
Quad-Tree	8	11.04 s	1104	11 Mb

Table 1: Comparisons of the performance of quad-tree data structure with [Sul and Kwon, 2012]

Example Grids

Four cases are exemplified to demonstrate the effectiveness of the adaptive grid generation scheme on multi-element airfoils. In all these cases, six successive divisions are applied to generate uniform mesh with one cut-cell adaptation cycle and three curvature adaptation cycles; both can be increased for denser grids. The critical grids on highly skewed, curved and splitted elements are shown in a close-up view (Figure 7-10).

Case 1:

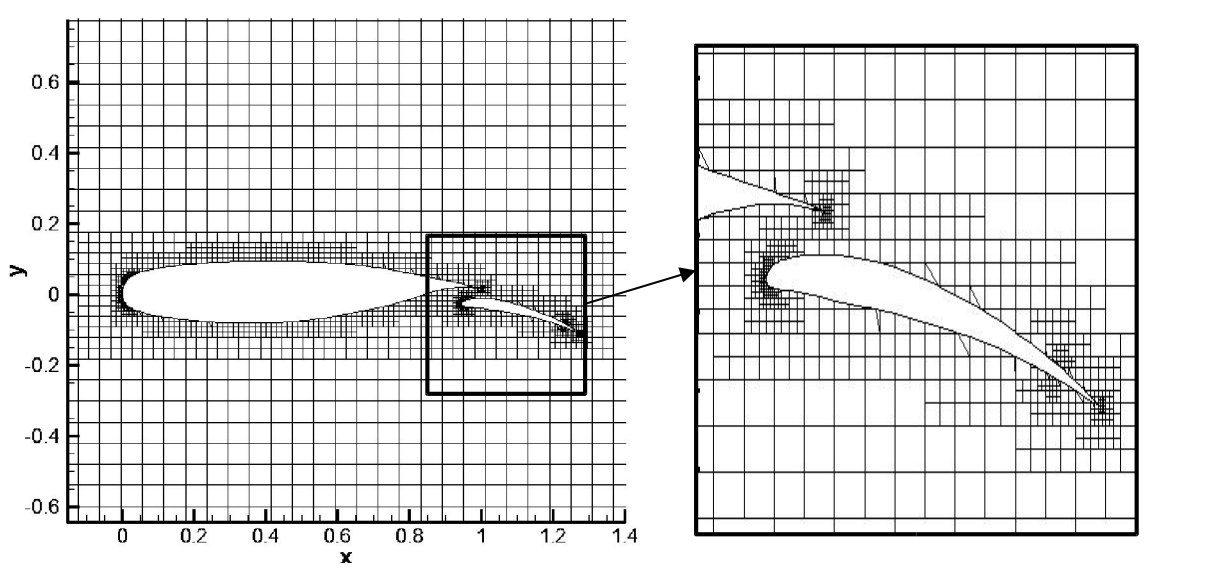

Figure 7: Adaptive Cartesian grid around
NLR-7301 airfoil/flap geometry [Arlinger and Larsson, 1997] and a close-up view

Case 2:

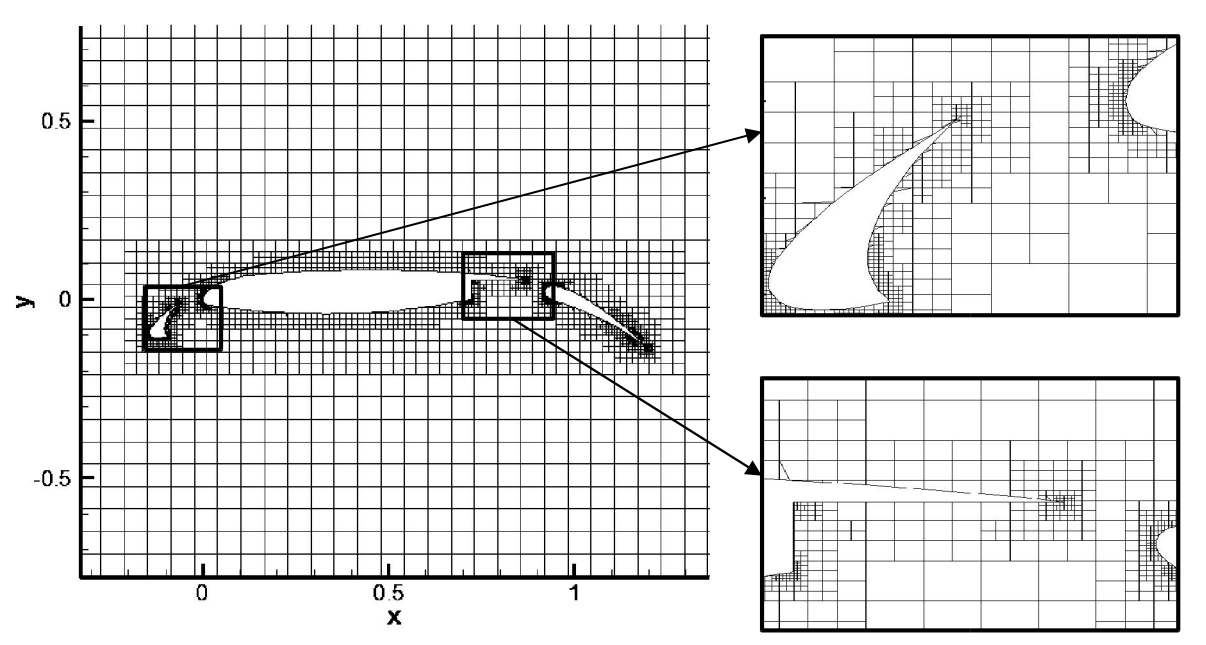

Figure 8: Adaptive Cartesian grid around
30P-30AG airfoil [Anderson and Bonhaus, 1995] and close-up views

Case 3:

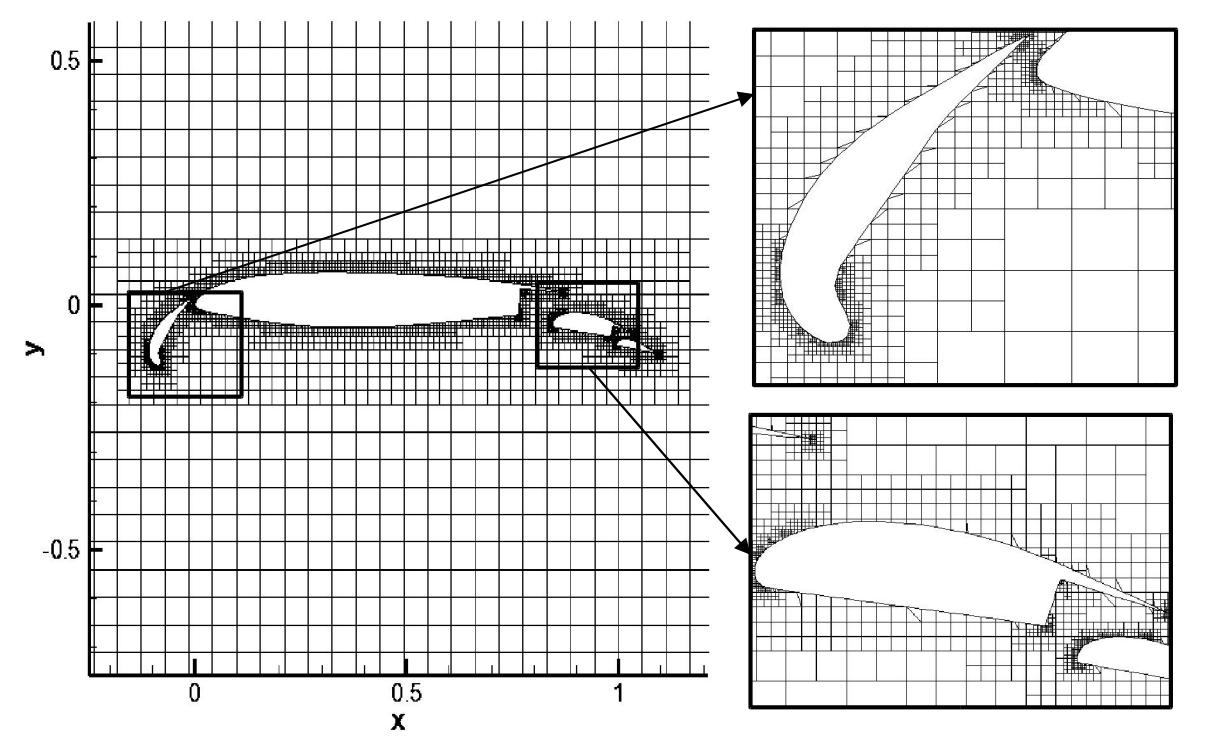

Figure 9: Adaptive Cartesian grid around
BOEING Model TR-1332 airfoil [Brune, 1994] and close-up views

Case 4:

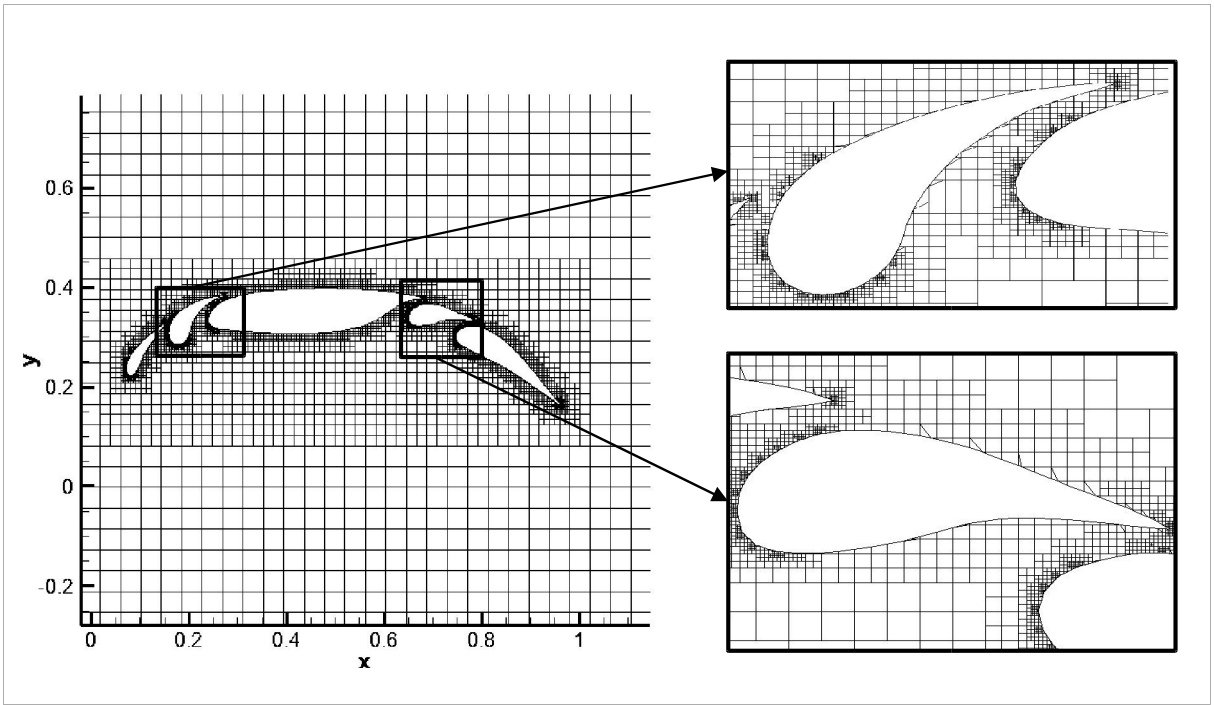

Figure 10: Adaptive Cartesian grid around a five-element airfoil [Petrov, 1980] and close-up views

CONCLUSION

In the present study, it is aimed to enhance automatic grid generation using Cartesian Methods. Quad-tree data structure is selected in order to maintain faster computation. An adaptive refinement/coarsening scheme for generating the final grid is prepared by using some special Cartesian algorithms, namely inside/outside test, cut/split cell determination and curvature adaptation algorithm and resultantly, a "hands-off" grid generator is implemented in FORTRAN programming language. At the end of the study, these adaptation techniques are applied on four different multi-element airfoil cases. Grid generation improvements near the highly skewed and highly curved elements are obtained. The developed grid generator is currently being extended to apply on three dimensional cases.

References

Akın, E. (2003) *Object-Oriented Programming via Fortran 90/95*, Cambridge Univ. Press, Mar 2003.

Anderson, W.K. and Bonhaus D.L. (1995) *Navier-Stokes Computations and Experimental Comparisons for Multielement Airfoil Configurations*, Journal of Aircraft, Vol. 32-6, p:1246-1253, Nov 1995.

Arlinger, B.G. and Larsson T. (1997) *NLR 7301 Two Element Airfoil at High Lift*, Notes on Numerical Fluid Mechanics, Vol.58, p: 375-396, 1997.

Bai J.S. , Li P., Zou L.Y. and Wang T. (2007) *A Quadtree Adaptive Level Set Method for Capturing Interfacial Instability on Cartesian Grid*, Engineering Applications on Computational Fluid Mechanics, Vol. 1-4 p: 263–372, Jun 2007.

Berger, M. and Aftosmis, M.J. (2012) *Progress Towards a Cartesian Cut-Cell Method for Viscous Compressible Flow*, 50th AIAA Aerospace Sciences Meeting, Nashville, TN, USA, Jan 2012.

Brune, G.W. (1994) *Two-dimensional High-lift Airfoil Data for CFD Code Validation*, No: A-13 A Selection of Experimental Test Cases for the Validation of CFD Codes, AGARD AR-303, 1994.

Çakmak, M. (2009) *Development of a Multigrid Accelerated Euler Solver on Adaptively Refined Two- and Three-Dimensional Cartesian Grids*, PhD Thesis in the Middle East Technical University, Jul 2009.

Çakmak, M., Aksel, M.H and Sert C. (2010) *Development of Two and Three-Dimensional Euler Solvers for Adaptively Refined Cartesian Grids with Multigrid Applications*, V. European Conference on Computational Fluid Dynamics ECCOMAS CFD, Lisbon, Portugal, Jun 2010.

Ji, H., Lien, F.S. and Yee, E. (2010) *Numerical Simulation of Detonation Using an Adaptive Cartesian Cut-cell Method Combined with a Cell-merging Technique*, *Computers &* Fluids, Vol. 39-6, p: 1041-1057, Jun 2010.

Kara, E., Kutlar A.İ., Aksel, M.H. (2013) *Quad-Tree Based Geometric Adapted Cartesian Grid Generation,* Proceedings of the 8th International Conference on Continuum Mechanics (CM '13), p:15-19, Rhodes Island, Greece, Jul 2013.

Kupiainen, M. and Sjögreen, B. (2009) A Cartesian Embedded Boundary Method for the Compressible Navier-Stokes Equations, Journal of Scientific Computing, Vol. 41-1, p: 94-117, Oct 2009.

Liu, J., Zhao, N., Hu, O., Goman, M., and Li, X.K. (2013) *A New İmmersed Boundary Method for Compressible Navier–Stokes Equations*, International Journal of Computational Fluid Dynamics, Vol. 27-3, p: 151-163, Jun 2013.

Petrov, A.V. (1980) *Some Features of Flow Past Slotted Wings (O Nekotorykh Osobennostyakh Obtekaniya Razreznykh Kryl'ev),* Royal Aircraft Establishment Library Translation No:2050, Defense Technical Information Center , Farnborough, England, Nov 1980.

Peskin, C.S. (1972) *Flow Patterns Around Heart Valves: A Numerical Method*, Journal of Computational Physics, Vol. 10-2, p: 252-271, Oct 1972.

Samet, H. (1988) An *Overview of Quadtrees, Octrees and Related Hierarchical Data Structures*, Theoretical Foundations of Computer Graphics and CAD, Vol. 40, p: 51-69, Jun 1988.

Sang, W., and Yu, J. (2011) *Numerically Analyzing More Efficiently High-Lift Aerodynamics of Wing/Body Model with Omni-Tree Cartesian Grids*, Aerospace Science and Technology, Vol. 15-5, p: 375-380, Jul 2011.

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

Sul, A. and Kwon, J.H. (2012) *Quad Tree Grid Generation Using the Concept of Linked Lists*, ICWES 15: The 15th International Conference for Women Engineers and Scientists, Adelaide, Australia, Jul 2011.