AIAA AVIATION Forum
5-9 June 2017, Denver, Colorado
23rd AIAA Computational Fluid Dynamics Conference

An Efficient Edge Based Data Structure Implementation for a Vertex Based Finite Volume Method

Semih Akkurt * and Mehmet Sahin †

Faculty of Aeronautics and Astronautics, Istanbul Technical University, Maslak, Istanbul, 34469, Turkey

An efficient edge based data structure has been proposed for the implementation of vertex based finite volume formulation. In the present approach, the quad-edge and halfedge data structures are redesigned and simplified in order to fit requirements of the cell-vertex centered finite volume methods. The present data structure is not limited with triangles, arbitrary polygons are also supported in the mesh without putting any additional effort. An explicit second-order compressible Euler solver based on the present data structure has been implemented in order to demonstrate its efficiency compared to the classical edge-based implementation for a vertex based finite volume formulation. Nevertheless, a fully implicit version of the present numerical algorithm has also been implemented based on PETSc library in order to improve the robustness of the algorithm. As a benchmark problem, the calculations over an RAE2822 airfoil and a circular cylinder are performed in order to demonstrate its efficiency.

I. Introduction

Cache efficiency plays an important role for speeding up the numerical CFD algorithms. However, achieving high cache efficiency is not so straightforward. First of all, the algorithm and the data structure should be compatible with each other. After examining the numerical procedure, a proper algorithm and a data structure can be designed together considering the key points related to cache efficiency. In this paper, a second order vertex based finite volume solver is developed in order to consider these points. First of all, cache conscious design of the algorithm and the underlying data structure will be investigated. A vertex based finite volume solver consist of four main parts that consume most of the computational time; geometric construction of the dual volume, Taylor series expansion for least square reconstruction in order to obtain gradients at vertices, flux integration, and the solution update stage. Construction of the dual volume around a vertex and solution reconstruction by means of Taylor series expansion of required order relies solely on neighbouring vertex data. Hence, neighbouring vertex information is a common requirement. However, flux integration using Roe's scheme[11] requires only left and right vertex data including their gradients. Besides, dual volume area, variable at the vertex, and the residual of the flux integration is used for the solution update stage. Beginning at this point, considerations related to cache efficiency should taken into account. Time required for a single mathematical operation is far less than a read from RAM. Therefore, computer actually reads some chunks of memory rather than a single entry, assuming the data that is spatially close in the memory is used successively in the algorithm. This chunk of data is copied to a much faster memory called CPU cache. Cache access time is much faster than RAM access time therefore an algorithm that is designed taking cache efficiency into account is much more efficient and faster. Each time a read from RAM is made, the CPU is actually stalled and does nothing until the data is available. In order to get rid of the waste of CPU time spent on waiting data from RAM, variables that are being used successively in the algorithm should be continuous in the memory as much as possible.

The aim in a finite volume solver is to integrate fluxes around a finite volume and update variables for each vertex for the next iteration. Therefore, all the data that is being used in the flux calculation should

*Grad. student, Istanbul Technical University, Faculty of Aeronautics and Astronautics, AIAA Student Member.
†Assoc. Prof., Istanbul Technical University, Faculty of Aeronautics and Astronautics, AIAA Senior Member.

be clustered in the memory as much as possible in order to have a cache efficient algorithm. Traditional way to store mesh data is based on representing polygons in the domain by means of their vertex indices. In case of an n-gone, n vertices are referenced for each cell in counter-clockwise manner with integer numbers that refer to the index of the corresponding vertex in the global vertex array. Therefore, there are two separate arrays in this representation; global vertex array and vertex cell connectivity array. Furthermore, for some applications, mesh generators and cell centered finite volume solvers mainly, neighbouring cells also play an important role for the algorithm and therefore stored in a separate array.This method provides a straightforward approach for the storage of mesh data. The structure explained above is being used by many solvers and mesh generators for their input/output files. However, especially for mesh processing, array index based mesh structure have some serious drawbacks. Mesh generation is an additive process and at each step new cells are formed by adding vertices and sometimes vertices are removed causing some cells to collapse. Vertex and cell addition cause vertex and vertex cell connectivity arrays to grow, which requires allocation and deallocation of those arrays and it makes whole process considerably slow. Vertex removal, cause a shrink in the vertex array which leads to a change in vertex index references of the vertex cell connectivity array also.

In order to overcome these difficulties, pointer based approaches are being used.[6] In pointer based approach, a user defined type is created for the base element. Vertex and/or neighbouring references of the base elements are defined using pointers instead of array indices. Thanks to this approach, elements are completely independent from each other. Therefore, addition or removal of elements is just a straightforward process. Whenever a new element is added or destroyed, it is allocated or deallocated in the memory and required changes in the proximity of the new element is made locally, in contrast to global array deallocation and allocation in array index based structure. On the other hand, pointer based approach doubles the required memory directly, however in return gives a chance to improve spatial locality of edges outgoing from same node by allocating them as a group for each node. Furthermore, regardless from the array index based or pointer based mesh data, data structure build around cells naturally matches up with cell centered finite volume solvers' needs.[9] However, this paper deals with a cell-vertex centered finite volume formulation and the standard mesh data structure based on cells is not efficient for this case. Edge based data structures are known to be extremely efficient for cell-vertex centered finite volume methods.[7,8] Marching over edges in the domain prevents repeated calculation of the properties of the same vertex dual volumes over and over again.

Finally, the decision of data structure for the cell-vertex centered solver is chosen as edge based structure with pointer connectivities. As mentioned before, edge based structure provides most efficient basis for cell-vertex centered finite volume methods. And pointer based connectivity approach makes it possible to allocate elements independent from each other. As a result of independently allocated elements, there is no need to deallocate and allocate arrays, or change all indices due to an element removal. Using pointer connectivities and independently allocated elements, studies that involves vertex addition or removal such as adaptive refinement and re-meshing are easily supported. In case of array index based connectivities, it is not so straightforward to manipulate the mesh. Therefore, for adaptive refinement and re-meshing studies, data structures based on array indices are actually an obstacle and this is a serious drawback for many solvers that are using array index based mesh data structure. In order to overcome these difficulties from the beginning, pointer based connectivities are chosen.

Quad-edge data structure[1] and Halfedge data structure[2] are examined, redesigned, and simplified in order to fit requirements of the cell-vertex centered finite volume methods. Since dual mesh is not required for this application, original quad-edge data structure simplified to keep primal mesh only. The design purpose of the halfedge data structure is mesh processing, therefore it also contains unnecessary features in terms of finite volume applications and those are eliminated. To sum up the structure; each geometrical edge in the mesh is consist of two half edges pointing opposite directions to each other. Each half edge has a pointer to its pointing node which is named as 'destination', a pointer to its 'symmetric' edge which points opposite direction of itself on the same geometrical edge, and a pointer to 'next' half edge with respect to its origin point.Elements in the edge element data type are chosen conformable to the cell-vertex based finite volume method algorithm. Schematic explaining the data structure is given in Figure 1. As it is seen from the figure, data structure is not limited with triangles, arbitrary polygons are also supported in the mesh without putting any additional effort. There are three essential information is present in each half edge and it is enough to represent all the mesh data without any additional information. Two additional floating data are also present in each halfedge which will be explained in following sections. It is also possible to get all the other neighbouring elements which are shown as faded in the figure using only `sym` and `Onext`

pointers repeatedly. Additionally, a list of one half edge from all edges is created for the solver. Using the given relations, the solver can loop over all edges for calculating flux, and can loop over all nodes and get neighbouring nodes in order to calculate gradient required for second order accuracy.

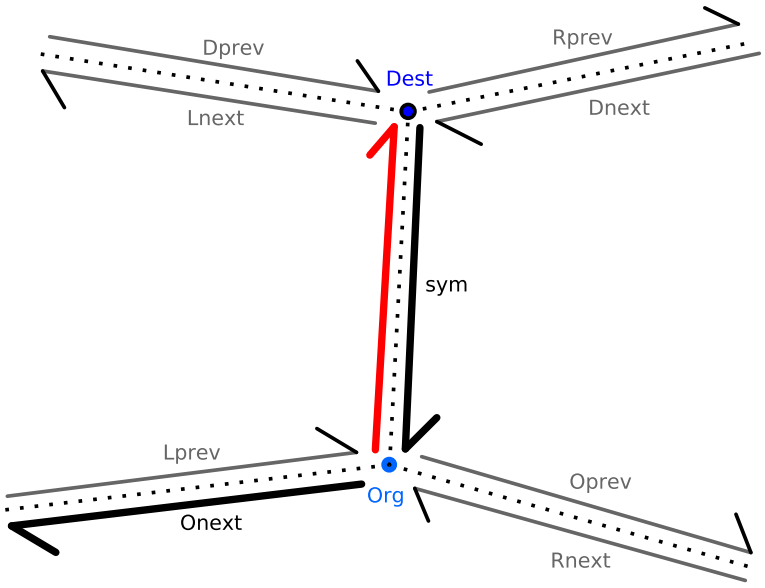

Figure 1. Quad-edge data structure showing origin and destination for an edge (red arrow), its symmetric edge and Onext edge (black arrow).

II. Euler Equations

The integral form of compressible time-dependent Euler equations can be written in the Cartesian coordinate system.

$$\int_{\Omega} \frac{\partial \mathbf{Q}}{\partial t}\, \mathrm{d}A + \oint_{\partial\Omega} \mathbf{F_i} \cdot \mathbf{n}\, \mathrm{d}l = 0 \tag{1}$$

where $\mathbf{n}$ is the unit normal vector. The state vector $\mathbf{Q}$ and inviscid flux vector $\mathbf{F_i}$ are given as;

$$\mathbf{Q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix} \tag{2}$$

$$\mathbf{F}_i = f_i\mathbf{i} + g_i\mathbf{j} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (E+p)u \end{bmatrix} \mathbf{i} + \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ (E+p)v \end{bmatrix} \mathbf{j} \tag{3}$$

The governing equations are closed with the equation of state for a perfect gas

$$p = (\gamma - 1)[E - \rho(u^2 + v^2)/2] \tag{4}$$

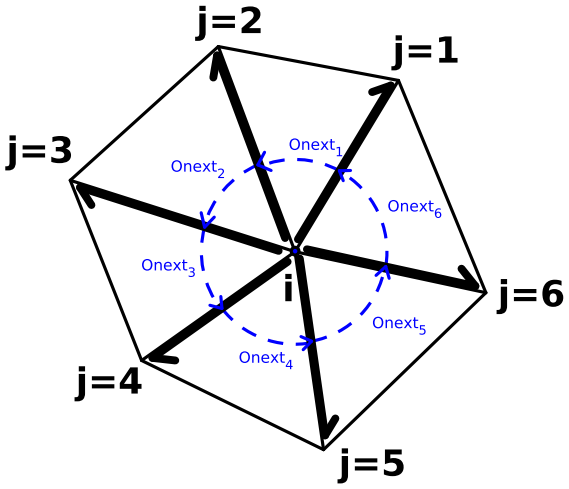

Figure 2. Derivation of neighboring node numbers using half edge data structure.

A. Discretization of the Convective Term

The inviscid fluxes at the mid-point of edges are evaluated using Roe's approximate Riemann solver.[11]

$$\mathbf{\Phi} = \frac{1}{2}\big[\mathbf{F}(\mathbf{Q}_L) + \mathbf{F}(\mathbf{Q}_R)\big] \cdot \mathbf{n} - \frac{1}{2}|\mathbf{A}(\hat{\mathbf{Q}}; \mathbf{n})|(\mathbf{Q}_R - \mathbf{Q}_L) \tag{5}$$

where $\mathbf{F}(\mathbf{Q}_L)$ and $\mathbf{F}(\mathbf{Q}_L)$ are the evaluated inviscid fluxes from the left and right side of an edge. For a second order accurate discretization, the gradient values are required for a Taylor series expansion. The gradient values are obtained initially using unweighted least square approximation.[10] For the evaluation of gradient terms, a Taylor series expansion has to be written from vertex i to the surrounding vertex points j as shown in Figure 2.

$$\begin{bmatrix} \mathbf{x}_1 - \mathbf{x}_i \\ \mathbf{x}_2 - \mathbf{x}_i \\ \mathbf{x}_3 - \mathbf{x}_i \\ \mathbf{x}_4 - \mathbf{x}_i \\ \mathbf{x}_5 - \mathbf{x}_i \\ \mathbf{x}_6 - \mathbf{x}_i \end{bmatrix} \begin{bmatrix} \frac{\partial q}{\partial x} \\ \frac{\partial q}{\partial y} \end{bmatrix} = \mathbf{A}_i \mathbf{x} = \mathbf{b}_i = \begin{bmatrix} \mathbf{q}_1 - \mathbf{q}_i \\ \mathbf{q}_2 - \mathbf{q}_i \\ \mathbf{q}_3 - \mathbf{q}_i \\ \mathbf{q}_4 - \mathbf{q}_i \\ \mathbf{q}_5 - \mathbf{q}_i \\ \mathbf{q}_6 - \mathbf{q}_i \end{bmatrix} \tag{6}$$

Therefore, the solution for calculating gradient terms can be written[10]

$$\frac{\partial q}{\partial x}|_i = \sum_{j=1}^{6} W_j^x (q_i - q_j) \tag{7}$$

$$\frac{\partial q}{\partial y}|_i = \sum_{j=1}^{6} W_j^y (q_i - q_j) \tag{8}$$

However, the extraction of surrounding vertices is a rather challenging task in an unstructured mesh data. The current data structure is optimum to determine the surrounding vertices for a given vertex. Because each node in the data structure contains an arbitrary pointer to one of its outgoing half edges. Using the given outgoing half edge of the node, neighbouring nodes of the present node are easily reachable. In order to further improve the performance of the algorithm, W^x and W^y values are calculated once in the beginning and stored. Since those values are only depend on geometry, they stay constant as long as mesh stays same. Then, at the beginning of each iteration, gradient is calculated only with multiplication and sum over neighbouring edges.The present data structure is particularly important for the high order finite volume implementation where the larger neighboring data have to be obtained. In the current implementation, the

curvature effects on solid surface has been considered using cubic splines, which also helps to reduce the entropy production for the present second order implementation. Even though the third order Taylor series have been implemented, the high order finite volume formulation implementation is not straight forward and requires more complex interpolations as in the k-exact approach.[15]

B. Implicit Formulation

For the temporal discretization, the first order backward Euler method is used.

$$\frac{M}{\Delta t}(\mathbf{Q}^{n+1} - \mathbf{Q}^{n}) = -\mathbf{R}^{n+1}$$

where M is the mass matrix. The residual at time level $n+1$ can be approximated as

$$\mathbf{R^{n+1}} \approx \mathbf{R^{n}} + \frac{\partial \mathbf{R}}{\partial \mathbf{Q}}(\mathbf{Q}^{n+1} - \mathbf{Q}^{n})$$

The Jacobian matrix is $\mathbf{J} = \frac{\partial \mathbf{R}}{\partial \mathbf{Q}}$ and the new equation becomes

$$\left(\frac{M}{\Delta t} + \mathbf{J}\right)\Delta \mathbf{Q}^{n+1} = \mathbf{R}^{n}$$

where $\Delta \mathbf{Q}^{n+1} = \mathbf{Q}^{n+1} - \mathbf{Q}^{n}$. Although the numerical method is currently second-order in space, the Jacobian matrix is approximated using the first-order interpolation in order to reduce the matrix bandwidth. The implicit implementation of the current algorithm is based on the PETSc software package developed at the Argonne National Laboratories in order to improve the parallel performance.[12] A one-level restricted additive Schwarz preconditioner with a block-incomplete factorization within each partitioned sub-domain is utilized for the resulting algebraic system. The level of ILU(k) preconditioner is set to 5 and the amount of overlap for the restricted additive Schwarz preconditioner is fixed to 1. The computational domain is decomposed into a set of partitions using the METIS library.[13] In order to handle the nonlinear nature of the algebraic equations, several Newton's sub-iterations are performed for each time step until a satisfactory convergence criteria is reached.

III. Numerical Results

In this section two-dimensional steady numerical simulations are carried out in order asses the accuracy and the performance of the present edge based finite volume algorithm.

A. RAE2822 Airfoil

The initial numerical results is used to test the spatial convergence of the present finite volume method. For this purpose, the steady flow around the RAE2822 airfoil is solved on meshes M1-M3 with a constant time step of 10. The computational meshes are created using an in-house mesh generation library.[5] Meshing process starts with initial triangulation of all the boundary points in the computational domain. Following the super triangle which contains all the points, new grid points are inserted into the triangulation one by one by using Bowyer-Watson algorithm and the Delaunay triangulation of the points are obtained. The mesh generation library also include Holmes-Snyder,[3] slightly modified versions of Rebay's first and second,[4] and a specifically designed algorithm for the library. All algorithms start with sorting triangles in the domain in terms of a quality ratio that is determined by the algorithm, and proceeds by adding points with a point placement strategy chosen in the algorithm. According to Shewchuk,[6] Triangular Data Structure (TDS) is double times faster compared to edge-based data structures for many different meshing algorithms including the one that used in the in-house mesh generation library. Therefore, triangular data structure is used in the mesh generation library. Conversion from both data structures into each other and various mesh processing subroutines are already included in the library. Although main mesh generation subroutines are written for TDS instead of edge-based data structure, subroutines required for directly editing the edge-based structure are included for direct usage within the solver without converting data structures one to another. Prior to final meshing process, all auxiliary triangles formed due to the temporary points of the super triangle and the triangles appearing in the inner parts of the inner boundaries are removed. Besides, boundary layer

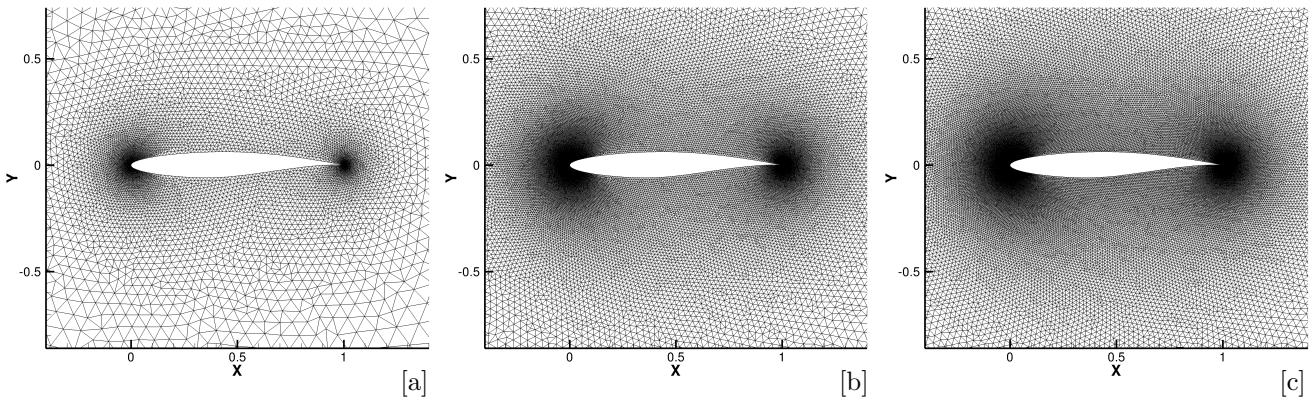

Figure 3. Unstructured computational coarse mesh M1, medium mesh M2 [b] and fine mesh M3 [c] around a RAE2822 airfoil.

mesh generation option is also available within the library. Due to its highly regular nature, Rebay's second method is employed as shown in Figure 3. The details of the computational meshes are provided in Table 1

Table 1. Computational meshes used for simulation of static bubble problem.

Mesh	Min Area	Number of nodes	Number of elements
M1	4.000E-007	7695	15141
M2	3.144E-007	15361	30417
M3	7.613E-008	31811	63228

The mesh convergence of the computed Mach number contours is provided in Figure 4-[a] and the result on mesh M2 is compared with the result obtained from the Stanford SU2 solver[14] in Figure 4-[b]. The free-stream Mach number is set to 0.3 with an angle of attack 0 degree. The physical boundary condition on the solid wall is set to zero convected flux condition. The numerical results indicate that the results on meshes

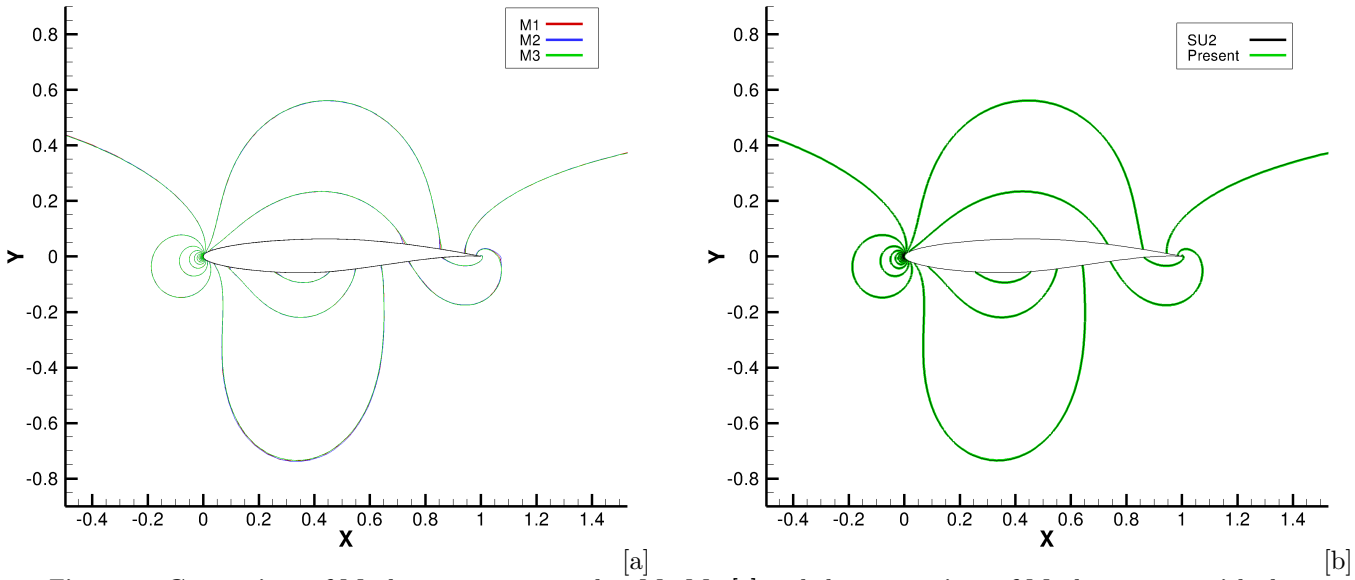

Figure 4. Comparison of Mach contours on meshes M1-M3 [a] and the comparison of Mach contours with the result of SU2 code on mesh M3 [b] at $M_\infty = 0.3$ and $\alpha = 0°$.

M2 and M3 are very similar to each other. The comparison of the numerical results with the results of the Stanford SU2 code on mesh M2 indicates that the results are indistinguishable from one another. The wake due to the entropy layer behind the airfoil is more apparent on coarse mesh M1, where the regions around the leading and trailing edge stagnation points are not sufficiently resolved. The convergence of the computed pressure coefficients over the airfoil surface is also provided in Figure 5 and the results are also compared with the pressure coefficient obtained from the SU2 code on mesh M2. The both solutions are very similar with each other. In the present simulations, the curvature effects are considered using a cubic spline fit on the airfoil surface. In order to show the effect of the curvature on entropy production, the comparison of the entropy contours are provided in Figure 6 for the curved boundary and the linear line segments. The curved boundary case indicates a slight decrease in the maximum value of the entropy production. The computed pressure and temperature contours are provided in Figure 7 on mesh M3.

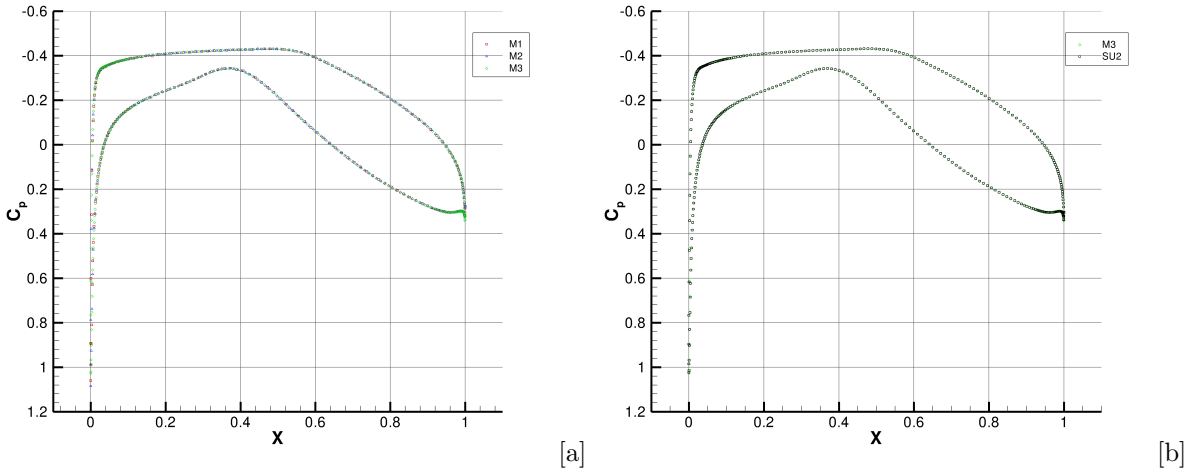

Figure 5. Mesh convergence of the pressure coefficients with mesh refinement [a] and the comparison with the result of SU2 code [b] at $M_\infty = 0.3$ and $\alpha = 0°$.

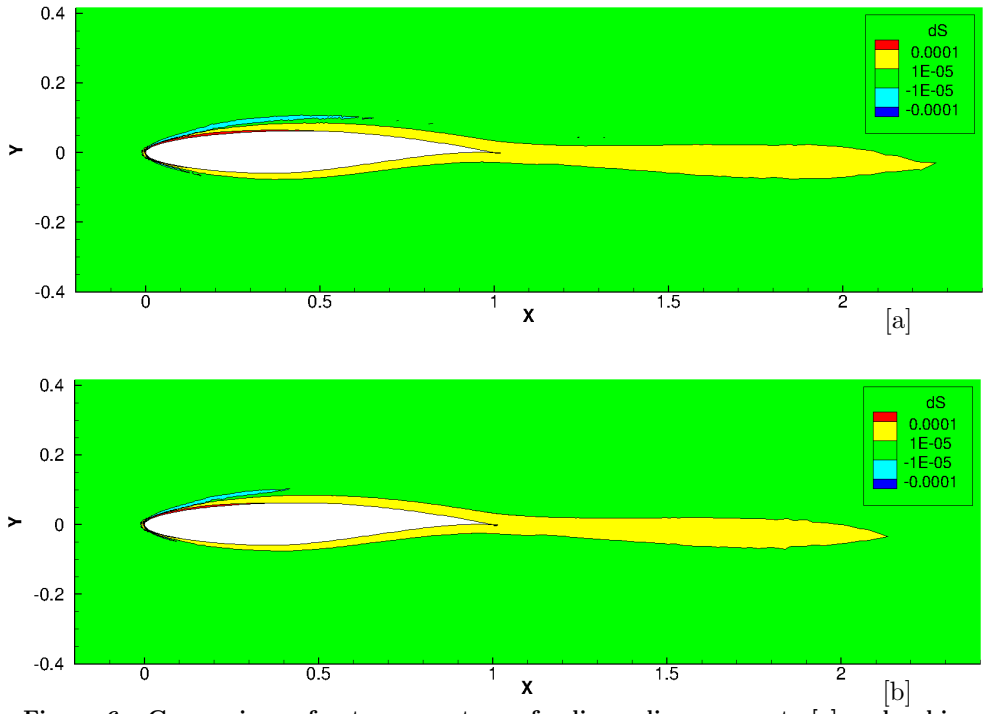

Figure 6. Comparison of entropy contours for linear line segments [a] and cubic spline fit [b] on the airfoil surface at $M_\infty = 0.32$ and $\alpha = 0°$ on mesh M3.

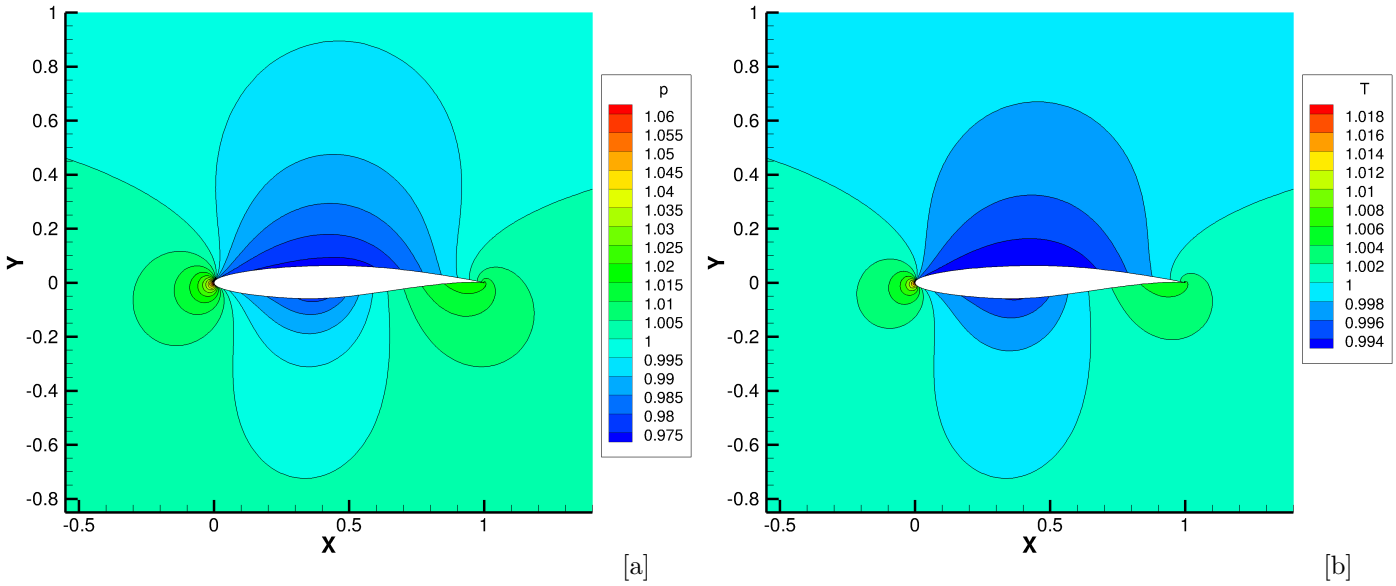

Figure 7. Computed pressure [a] and temperature [b] contours on mesh M3 at $M_\infty = 0.3$ and $\alpha = 0°$.

B. Circular Cylinder

The second set of numerical results correspond to the inviscid subsonic flow around a circular cylinder. The free stream Mach number is set to $M_\infty = 0.3$. The computational mesh is shown in Figure 8 and it consists of 6047 nodes and 11826 elements, leading to 24188 DOF. The computed Mach number and pressure contours are provided in Figure 9. The solution is found out be symmetry according to $x-$axis. However, there is some asymmetry according to $y-$axis due to artificial entropy production. The use of curved surfaces and

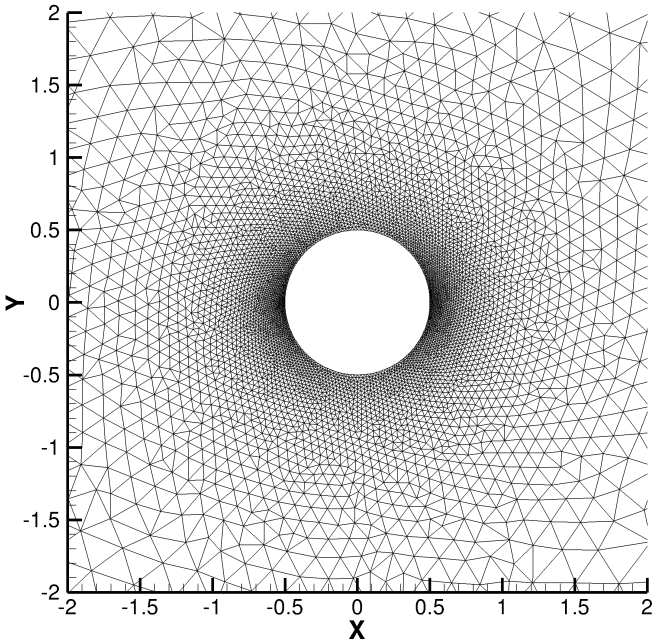

Figure 8. Unstructured computational mesh around a cylinder with 6047 nodes and 11826 elements.

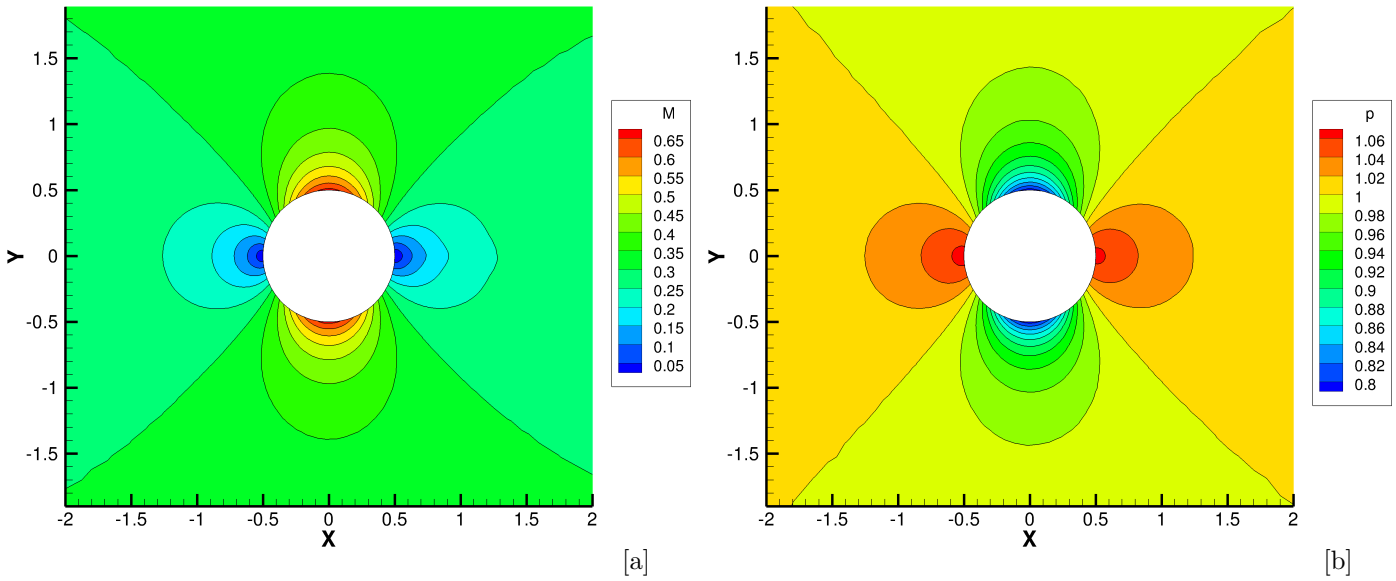

Figure 9. Computed Mach number [a] and pressure [b] contours around a cylinder at $M_\infty = 0.32$ and $\alpha = 0°$.

line segments are compared for the artificial entropy production in Figure 10. The cylinder surface with line segment indicates some increase in the entropy production behind the cylinder. The computed Mach contours are compared with the result of SU2 obtained on the same mesh in Figure 11. The results are indistinguishable from one another.

In order to assess the performance of the present half edge data structure, the present finite volume implementation is compared the Stanford SU2 code. For this purpose, the CPU time for 10,000 time integration steps are compared with each other using the second order Euler explicit approach. The present implementation requires 17.70 seconds, meanwhile SU2 algorithm requires 88.82 seconds for the same number of time

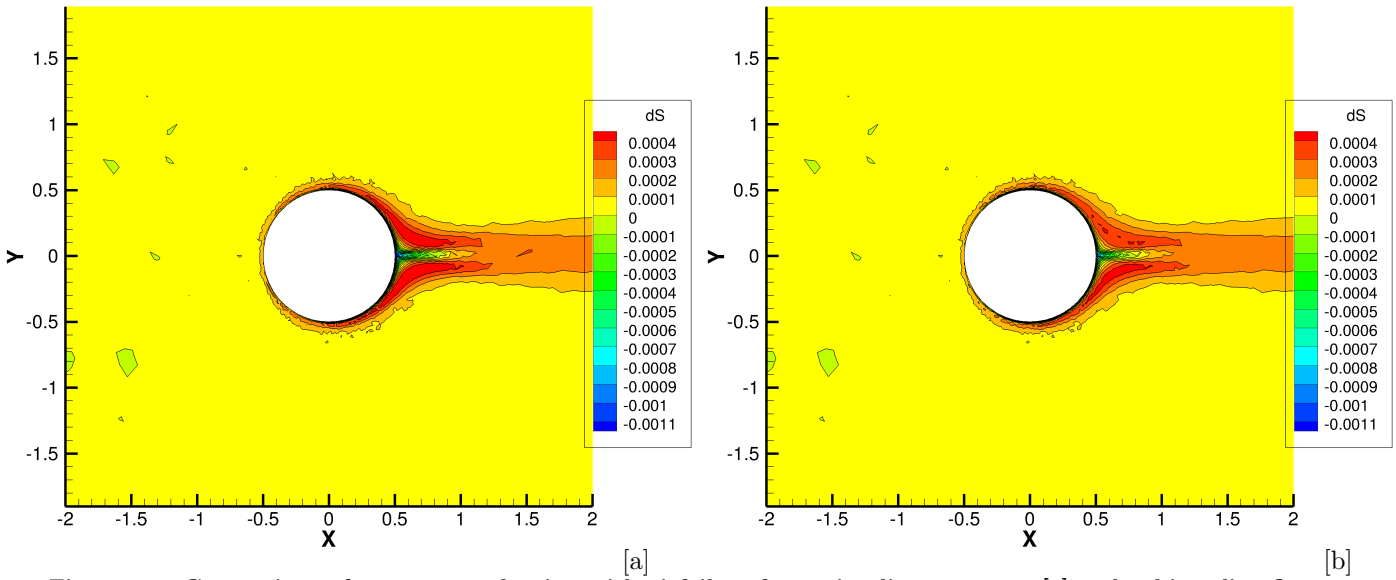

Figure 10. Comparison of entropy production with airfoil surface using line segments [a] and cubic spline fit [b] at $M_\infty = 0.3$ and $\alpha = 0°$.

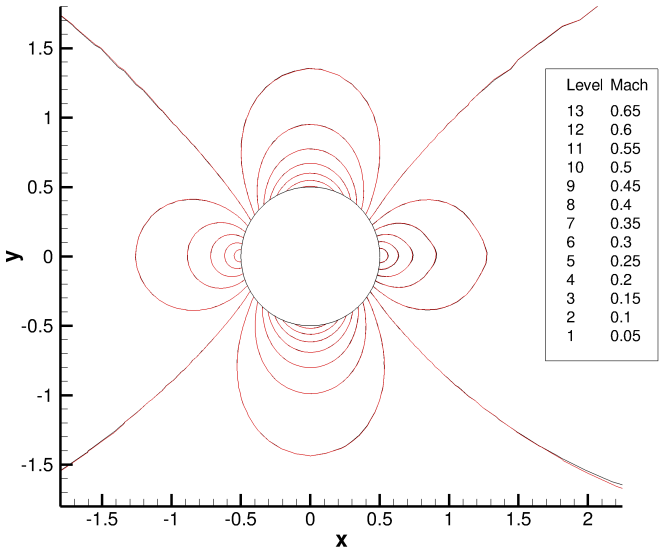

Figure 11. Comparison of computed Mach contours with SU2 results at $M_\infty = 0.3$ and $\alpha = 0°$.

steps. However, we should mention that the SU2 solver spent some time to determine local time step based on CFL number. However, the present simulations use a constant time step within the whole computation domain. The current solver uses approximately 62.3 percent to evaluate Roe's fluxes, 10.9 percent to evaluate gradients for the second order Taylor series, 9.29 percent to update new state variables and the remaining 17.51 percent is to construct dual volume and perform numerical integration. Therefore, the present halfedge data structure is particularly efficient for the edge-based finite volume implementation.

IV. Conclusion

An efficient halfedge data structure has been implemented for an edge based finite volume formulation in order to demonstrate its efficiency compared to the classical edge-based implementation for a vertex based finite volume formulation. In the present approach, the quad-edge and halfedge data structures are redesigned and simplified in order to fit requirements of the cell-vertex centered finite volume methods. The present data structure is not limited with triangles, arbitrary polygons are also supported in the mesh without putting any additional effort. The method inherently allows vertex insertion and deletion, which is particularly suitable for adaptive mesh refinement or local remeshing. Together with its data structure, the numerical algorithm is designed to be cache conscious by increasing the spatial locality of the data that is being used all together for a single computation. The numerical algorithm is validated for the subsonic inviscid flows around a RAE2822 airfoil and a circular cylinder. The efficiency of the numerical algorithm has been tested by comparing the CPU times for the Stanford SU2 code and significant increase in computational performance is achieved. In the future, we will try to extend the current finite volume algorithm to a higher order.

Acknowledgments

The authors also gratefully acknowledge the use of the computing resources provided by the National Center for High Performance Computing of Turkey (UYBHM) under grant number 10752009, and the computing facilities at TUBITAK ULAKBIM, High Performance and Grid Computing Center.

References

[1]Guibas, L. and Stolfi, J., "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams", *ACM Transactions on Graphics, 4*(2), 1985, 75-123.

[2]Botsch, M., Steinberg S., Bischoff S., Kobbelt L., "OpenMesh - a generic and efficient polygon mesh data structure", 1st OpenSG Symposium, 2002.

[3]Holmes, D.G. and Snyder, D.D., "The generation of unstructured meshes using Delaunay triangulation", *Numerical Grid Generation in Computational Fluid Mechanics88,* Pineridge Press, Swansea, United Kingdom, 1988.

[4]Rebay, S., "Efficient unstructured mesh generation by means of delaunay triangulation and bowyer-watson algorithm." *Journal of Computational Physics, 106*(1), 1993, pp. 125-138.

[5]Akkurt, S., *A Delaunay based algorithm for unstructured mesh generation.* Istanbul Technical University, 2016.

[6]Shewchuk, J.R., "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator", volume1148 of *Lecture Notes in Computer Science, Springer-Verlag*, from the First ACM Workshop on Applied Computational Geometry, 1996 pp. 203-222.

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

[8]Wright, Jeffrey A., and Richard W. Smith., "An Edge-Based Method for the Incompressible NavierStokes Equations on Polygonal Meshes." *Journal of Computational Physics*, vol. 169, no. 1, 2001., pp. 24-43

[9]Mavriplis, D. J., "Unstructured-Mesh Discretizations and Solvers for Computational Aerodynamics." *AIAA Journal*, vol. 46, no. 6, 2008., pp. 1281-1298.

[10]Anderson, W. K., and Daryl L. Bonhaus., "An Implicit Upwind Algorithm for Computing Turbulent Flows on Unstructured Grids." *Computers and Fluids*, vol. 23, no. 1, 1994, pp. 1-21

[11]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes." *Journal of Computational Physics*, vol. 43, no. 2, 1981, pp. 357-372

[12]S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith and H. Zhang, PETSc Users Manual. ANL-95/11, Mathematic and Computer Science Division, Argonne National Laboratory, (2004). http://www-unix.mcs.anl.gov/petsc/petsc-2/

[13]G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 20(1), (1998), 359-392

[14]T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso, SU2: An open- source suite for multiphysics simulation and design, *AIAA Journal*, 54(3):828-846, 2016. doi: 10.2514/1.J053813.

[15]Ollivier-Gooch, C., Nejat, A., and Michalack, K., On Obtaining High- Order Finite Volume Solutions to the Euler Equations on Unstructured Meshes, 18th AIAA CFD Conference, AIAA Paper 2007-4464, 2007.