DEVELOPMENT OF CARTESIAN BASED MESH GENERATOR WITH BODY FITTED BOUNDARY LAYERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MERVE ÖZKAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

DECEMBER 2019

Approval of the thesis:

DEVELOPMENT OF CARTESIAN BASED MESH GENERATOR WITH BODY FITTED BOUNDARY LAYERS

submitted by **MERVE ÖZKAN** in partial fulfillment of the requirements for the degree of **Master of Science in Mechanical Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. M. A. Sahir Arıkan
Head of Department, **Mechanical Engineering** _____

Prof. Dr. Mehmet Haluk Aksel
Supervisor, **Mechanical Engineering, METU** _____

Assist. Prof. Dr. Özgür Uğraş Baran
Co-supervisor, **Mechanical Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Cüneyt Sert
Mechanical Engineering, METU _____

Prof. Dr. Mehmet Haluk Aksel
Mechanical Engineering, METU _____

Prof. Dr. Mehmet Metin Yavuz
Mechanical Engineering, METU _____

Prof. Dr. Sinan Eyi
Aerospace Engineering, METU _____

Assist. Prof. Dr. Onur Baş
Mechanical Engineering, TED University _____

Date: 31.12.2019

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Merve Özkan

Signature :

ABSTRACT

DEVELOPMENT OF CARTESIAN BASED MESH GENERATOR WITH BODY FITTED BOUNDARY LAYERS

Özkan, Merve
M.S., Department of Mechanical Engineering
Supervisor: Prof. Dr. Mehmet Haluk Aksel
Co-Supervisor: Assist. Prof. Dr. Özgür Uğraş Baran

December 2019, 118 pages

In this thesis, the development of a Cartesian based mesh generator with body-fitted boundary layer is presented. The base of the developed mesh generator consists of Cartesian mesh. However, the boundary layer handling is a challenge with Cartesian mesh. Therefore, a body-fitted boundary layer is introduced to the mesh generator by putting a customized open source mesh generator into the main Cartesian based mesh generation. The boundary layer mesh generation part comes from the customized SUMO code, which is an open source code for body-fitted boundary layer mesh. By using the customized open source code, the boundary layer mesh is generated as wedge volume elements which inflate from the triangular surface elements of the geometry. After generating the boundary layer, there is gap between the Cartesian mesh and the boundary layer wedge volume elements. Since there must be a transition to triangular surface to square surface, this transition is supplied with pyramid and tetrahedral volume mesh. Pyramid volume elements are generated at the inner boundary of the Cartesian mesh as a part of the code. Tetrahedral mesh is generated by using an open source code TetGen, which uses Delaunay tetrahedralization. The Cartesian

based mesh generator with body-fitted boundary layer is developed in this thesis for less time-consuming and more efficient meshing process.

Keywords: Computational Fluid Dynamics, CFD, Mesh Generation, Finite Volume Method, Cartesian Mesh, Hybrid Mesh, Tetrahedral Mesh

ÖZ

GÖVDE UYUMLU SINIR TABAKAYA SAHİP KARTEZYEN TABANLI ÇÖZÜM AĞI ÜRETİCİSİ GELİŞTİRİLMESİ

Özkan, Merve
Yüksek Lisans, Makina Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Mehmet Haluk Aksel
Ortak Tez Yöneticisi: Dr. Öğr. Üyesi. Özgür Uğraş Baran

Aralık 2019 , 118 sayfa

Bu tezde, gövde uyumlu sınır tabakaya sahip Kartezyen tabanlı bir çözüm ağı üreticisi geliştirilmesi sunulmuştur. Geliştirilen çözüm ağı jeneratörünün tabanını Kartezyen çözüm ağı oluşturmaktadır; fakat Kartezyen çözüm ağı ile düzgün bir sınır tabaka elde etmek çok zordur. Bu nedenle, çözüm ağı jeneratörüne sınır tabakayı çözüm ağı oluşturabilen bir açık kaynak çözüm ağı üreticisi ile sınır tabakası oluşturulmuştur. Sınır tabakası çözüm ağı bir açık kaynak kodu olan SUMO ile oluşturulup, bu açık kaynak kod bu çözüm ağı geliştiricisinde kullanılmak üzere düzenlenmiştir. Düzenlenen gövde uyumlu sınır tabaka çözüm ağı üreticisi ile üçgen prizma hacim elemanları gövdeyi oluşturan üçgen yüzey çözüm ağından yükselerek üçgen prizma şeklinde hacim elemanlarına sahip sınır tabaka çözüm ağını oluşturur. Gövde uyumlu sınır tabakanın oluşturulmasından sonra Kartezyen çözüm ağı ile arasında boşluk olduğu görülür. Kartezyen çözüm ağı elemanları ve sınır tabakası çözüm ağı elemanları arasında üçgen yüzeyden kare yüzeye geçiş bulunduğu için bu boşluk piramit hacim elemanları ve dörtyüzlü çözüm ağı ile doldurulur. Piramit hacim elemanları Kartezyen

çözüm ağının bir parçası olarak geliştirilir. Dörtyüzlü çözüm ağı ise bir açık kaynak kodu olan ve Delaunay tetrahedralizasyonu kullanan TetGen kodu kullanılarak oluşturulur ve diğer çözüm ağları ile birleştirilir. Daha az zaman harcayan ve verimli bir gövde uyumlu sınır tabakaya sahip Kartezyen tabanlı bir çözüm ağı üreticisi bu tez ile geliştirilmiştir.

Anahtar Kelimeler: Hesaplamalı Akışkanlar Dinamiği, HAD, Çözüm Ağı Üretimi, Sonlu Hacim Metodu, Kartezyen Çözüm Ağı, Hibrit Çözüm Ağı, Dörtyüzlü Çözüm Ağı

To my grandmother, mother and sister

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my supervisior, Prof. Dr. M. Haluk Aksel for his excellent supervision, guidance and support.

I would like to expess my gratitude to my co-supervisor Assist. Prof. Dr. Özgür Uğraş Baran for his endless motivation, guidance and support.

I would like to present my thanks to Mehmet Ali Ak for his support and advice.

I would like to offer my most profound gratitude to my deceased grandmother Emine Yılmaz who always encouraged me to pursue my education.

I would like to present my sincerest thanks to my mother Pınar Yılmaz and my sister Tuğçe Hande Özkan for their patience, encouragement, support and unconditional love during this thesis.

I would like to express my special thanks to my colleagues at TÜBİTAK SAGE for their support throughout this thesis.

I would like to express my thanks to my colleague Erdem Dikbaş for his support.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Computational Fluid Dynamics (CFD) is an analysis tool which becomes a very important tool for engineers. Using CFD tool chain Design Engineers evaluate the engineering problems before the prototypes are built. Since computers become more powerful, CFD is employed in design problems and optimizations increasingly. The governing equations are solved efficiently using numerous techniques. With the increasing computational power, the detailed analysis of fluid flow can be conducted by using Computational Fluid Dynamics (CFD).[1] The goal of CFD is to understand the physical fluid flow problems in designs. The physicals of a fluid flow are the interactions of the fluid. [2]

CFD is defined as the"third approach" in the study of fluid dynamics. In twentieth century, fluid mechanics studies are mostly experimental. In 1950, CFD was only existed for two dimensional problems due to the insufficient computer power and algorithms relatively primitive. Then, in 1970s, the three-dimensional CFD is started to be employed in design of fluid machinery, airplanes, wing,etc. The capacity of the computing power is increasing ever since.[1]

The set of algebraic equations are derived from the partial differential equations discretized in time and space. The solutions of the algebraic equitations are the discrete variables over the field. The discretization of the computational field requires identifying the points and computational cells on the boundaries of the geometry.[3]

In most CFD methods, a discretization method is used. Discretization method is an approximation of differential (or integral) equations by a system of algebraic equations on a set of sub domains often as simple topological shapes. Then, these equa-

tions are adapted to be solved by computers. In discretization method, both time and the space can be discretized. The accuracy of the results obtained by discretization methods depends on the quality of the discretization as well as the quality of space sub-divisions. [4]

CFD work chains have three parts: (1) Pre-processing, (2) Solver and (3) Post-processing as shown in the given in Figure 1.1.

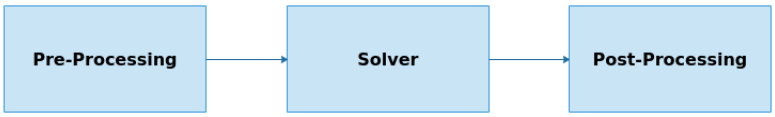


Figure 1.1: Flowchart - Steps of CFD

The computational domain is defined in the first step of the pre-processing part. Then the second step of pre-processing part is the mesh generation. The mesh generation is the sub division of the computational domain with the element size and size-distribution defined by the user. Mesh generation step is the most important part of pre-processing stage. The computational domain is divided into preferably nonoverlappping and well connected sub-domains by the mesh generation process. [5] The accuracy of a solution of a CFD problem is determined by the number and quality of elements of the mesh. In the solver part, the selection of the model for solving the problem (such as inviscid, laminar, turbulence methods, etc.). [6] In solver part, there are three main separate numerical solution techniques. These are finite difference method (FDM), finite element and spectral methods (FEM) and finite volume method[6] The most common technique is the finite volume method which is approximately used 80% of the modern CFD solvers. The other 15% is for the finite element method and 5% is for the other methods.[7] Finally, in post-processing part, the solution obtained from the solver part is precessed.[6]

1.1 Mesh Generation

Mesh generation is a very important process and has basic considerations. The data of discretization elements must be stored and accessed easily and efficiently.

The second is the connectivity of the elements. The communication between elements must be maintained and processed smoothly between the neighbor elements. Thirdly, the quality of the discretization must be seek throughout the computational domain for running numerical algorithm.[8]

Accurate computational solution require high quality meshing. A computational mesh which does not meet the quality metrics leads to the inaccurate results in the solution stage. [1] The basic terminology used in the computational meshes is shown in Figure 1.2. It should be noted that, terminology is based on Finite Volume method. For finite elements methods, cell term is replaced with element, and faces are often not needed.

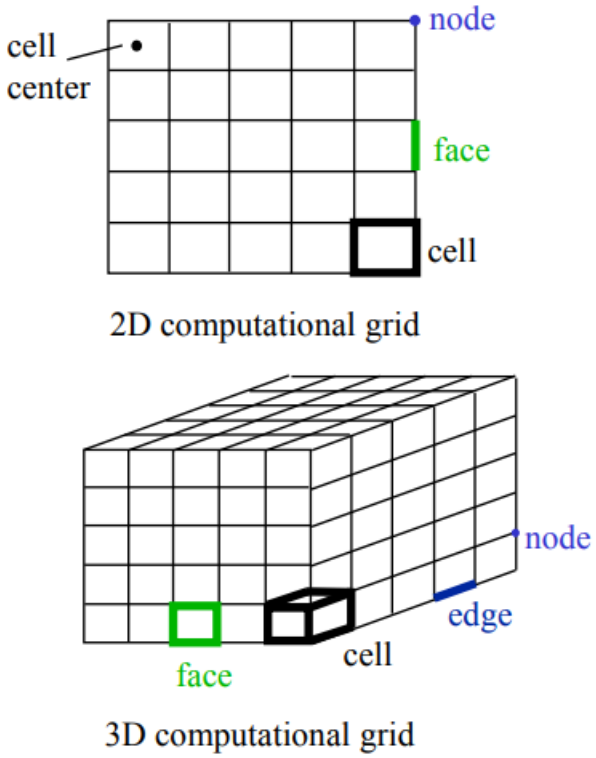


Figure 1.2: Mesh Generation Terminology

The cells are written in files which are transferred to the solver with two different methods: Cell-centered scheme and Vertex-centered scheme. The definitions are shown in Figure 1.3.

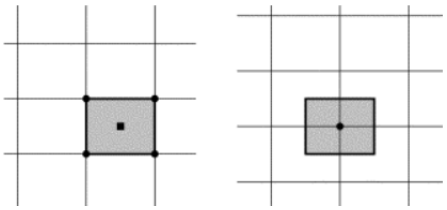


Figure 1.3: Cell-centered and Vertex-centered Schemes of FVM

In cell-centered scheme, the integral form of conservation equations is discretized over the control volume which is same as the cell volume . [4] The calculations of the variables are performed at the centers of the volume elements. [5] The surface integrals are approximated at the control volume face, where the flux is calculated. The calculation of the values at the faces are calculated by interpolation. [9] The flow properties and quantities are stored at the centroids of the mesh cells. Therefore, the volume of each mesh cell is equal to the control volume.

In vertex-centered scheme, the flow properties and quantities are stored at the nodes of the mesh. Therefore, the control volume is a volume constructed at the solution time where the element centers are located at nodes. Besides, the flux calculations are similar with the cell-centered method. [10]

1.1.1 Mesh Quality

Mesh quality has vital effect on the solution accuracy. The quality of the mesh elements should be ensured before starting the solution. The basic and the most common mesh quality metrics are skewness, orthogonality, smoothness and aspect ratio.

Skewness and orthogonal quality shows the deviation of a cell form from the ideal shape for the cell in terms of angles. The ideal shape of the cell is an equilateral cube for hexahedral mesh and equilateral tetrahedron for tetrahedral mesh. Skewness of the cell is calculated by using either two methods: Equilateral Volume deviation or

$$\max\left[\frac{\theta_{max}-\theta_e}{180-\theta_e}, \frac{\theta_e-\theta_{min}}{\theta_e}\right]$$

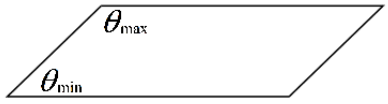


Figure 1.5: Normalized Angle Deviation

Normalized Angle deviation.

The Equilateral Volume Deviation is the ratio of the difference of optimal cell size from current cell size to optimal cell size. This metric can only be calculated for tetrahedral elements. This method is explained in Figure 1.4. The green edged cell is an optimal (equilateral) cell and the blue edged cell is an actual cell. As the deviation increases, the skewness increases. In Normalized Angle Deviation method, skewness is the maximum of the angle ratios of the cell. The skewness calculation and the angle which is used in the calculation are shown in Figure 1.5. The angle(θ) with e subscription is the 60^0 for tetrahedral and 90^0 for hexahedral cells. This method can be applied to all type of cells; tetrahedral, hexahedral, prisms and pyramids.

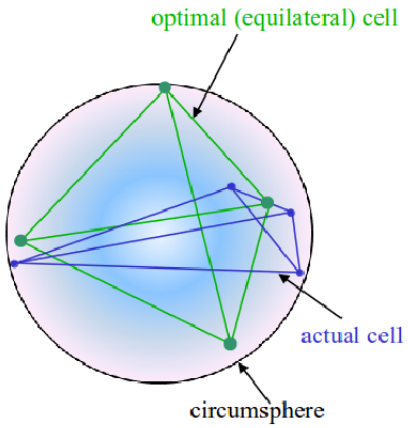
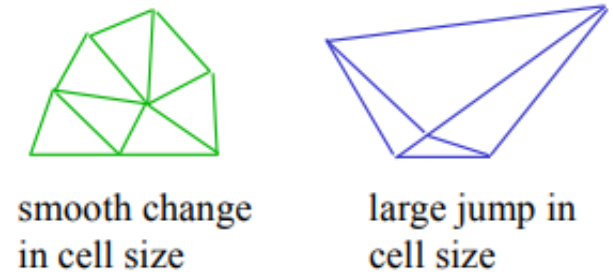


Figure 1.4: Equailateral Volume Deviation

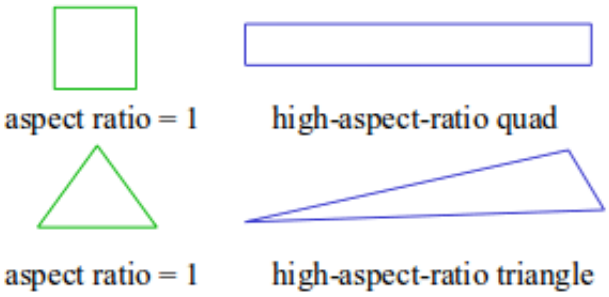
The best skewness quality is achieved when the skewness metric, which is found by using one of the methods, is close or equal to zero. Orthogonal quality of a cell shows how close the angles between faces of the elements, which are adjacent to each other, to the optimal angle. If the adjacent faces of the face of the cell tend to be in the

normal direction of the face of the cell, then it is said to be the orthogonal quality of the mesh is good.

Smoothness is the rate of size change of the adjacent cells. The unsmooth and the smooth transition between elements are shown in Figure 1.6a. The ideal mesh spacing should be under 20 percent, i.e. the adjacent cell size should be 1.2 times of the cell size. [7] The cell size is volume of the cell. Aspect ratio is the ratio of the length of long side to the short side. The ideal ratio is one and can be achieved with square and equilateral triangles in 2D as shown in Figure 1.6b. The high aspect ratio cells are not desirable in the most part of the flow domain. However, they are accepted and even desired in boundary layer mesh where there is a strong flow gradient normal to the wall and less parallel to the wall.



(a) Smoothness



(b) Aspect Ratio

Figure 1.6: Smoothness and Aspect Ratio

1.1.2 Mesh Types

There are three types of computational mesh. The types of meshes are structured mesh, unstructured mesh and hybrid mesh.

1.1.2.1 Structured mesh

In structured mesh, the mesh points (nodes) have i,j,k indexing with integers for locating each node in the computational domain for three dimensions. This allows the cells of the mesh can be identified by the indices. [11] In structured mesh, the mesh consists of quadrilateral cells in 2D, hexahedra volume elements for 3D domains. The indexing of the nodes supply easy identification of the nodes when solving the computational domain with fluid flow governing equations. [5]

In Figure 1.7, the indexing of the nodes is given for both two dimensional and three dimensional mesh.

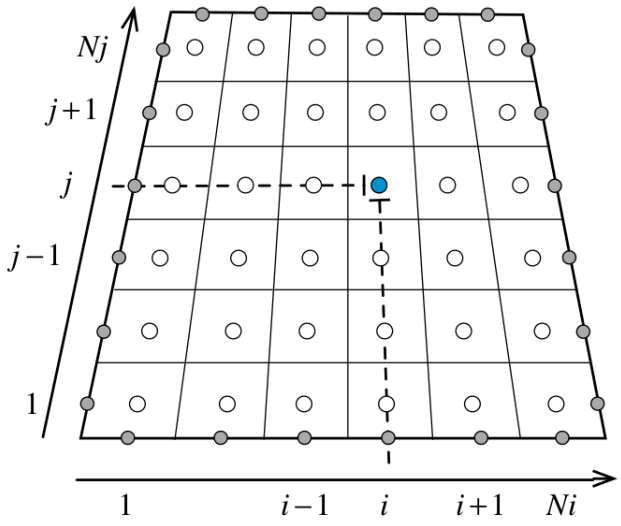


Figure 1.7: Structured mesh

The mesh properties such as orthogonality and skewness can be out of usable bounds on such geometries.[5] On the other hand, structured mesh gives best orthogonality

if successfully generated and also gives the most accurate solution compared to other type of meshes.

1.1.2.2 Unstructured mesh

In unstructured meshes, the neighbor relations of the cells cannot be directly found by the indexing. Mostly, the unstructured mesh consists of triangles in two dimension, tetrahedral volume elements in three dimension. Also, from other volume elements, such as hexadehra and prisms, the unstructured mesh can be generated.[10]. On a broader basis, an unstructured mesh can involve any mixture of non-self-intersecting positive volume cells. The main disadvantage of unstructured meshes is the absence of fixed coordinate lines and nodes. The nodes and the edges can be organized automatically, therefore the mesh is generated without logical representation compared to structured meshes.[6] The memory usage is higher than the structured mesh.[7] An example unstructured mesh is given with Figure 1.8.

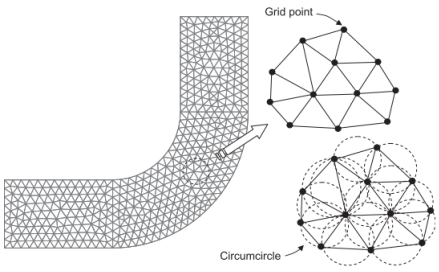


Figure 1.8: Unstructured mesh

For complex geometries, meshing with unstructured mesh is easier compared to structured mesh. However, it is reported that unstructured meshes has less accurate solutions compared to structured meshes at the same computational cell number.[7]

1.1.2.3 Hybrid Mesh

In CFD applications, the complex geometries are often encountered such as aircraft geometries. Unstructured meshes provides great flexibility for body fitting to complex

geometries. However, the unstructured mesh may require special features, especially, in the problems with viscous flows where the boundary layer flow demands special treatment. In viscous flow regions where the flow gradients change rapidly, very thin and preferably stacked parallel cells is needed to solve boundary layer equations which results in high-aspect-ratio cells. With high-aspect-ratio, tetrahedral volume elements causes high truncation errors, due to high skewness of the cells. Therefore, at the viscous flow regions, prismatic (wedge) elements are introduced. An example on using two types of elements on a turbine airfoil cascade is shown in Figure 1.9. In such cases, the hybrid mesh, which consists of several types of volume elements, can reduce the total cell count and improve the solution efficiency. [12]

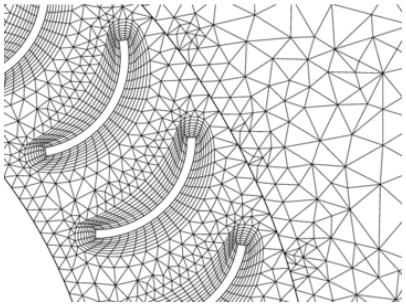


Figure 1.9: Hybrid mesh

1.1.2.4 Cartesian mesh

Cartesian meshes are constructed as subdividing (or refining) quadrilateral or hexahedral cells recursively by dividing into four or eight child cells. This procedure is called quad-tree or octree division strategy. Cartesian mesh is generated encapsulating the whole geometry, and the biggest cell divides into cells until the size of the child cell equal to the criterion. An example to Cartesian mesh is shown in Figure 5.9. One of the advantages of the Cartesian mesh is the quick generation compared to other mesh types. [13] With the utilization Cartesian meshes, the higher order solution schemes are implemented easily and geometric multi-level convergence acceleration is achieved.[14] Moreover, same solution accuracy can be achieved with using less number of cells compared to tetrahedral mesh. Moreover, since the cells are orthogonal, the quality is better than unstructured mesh.

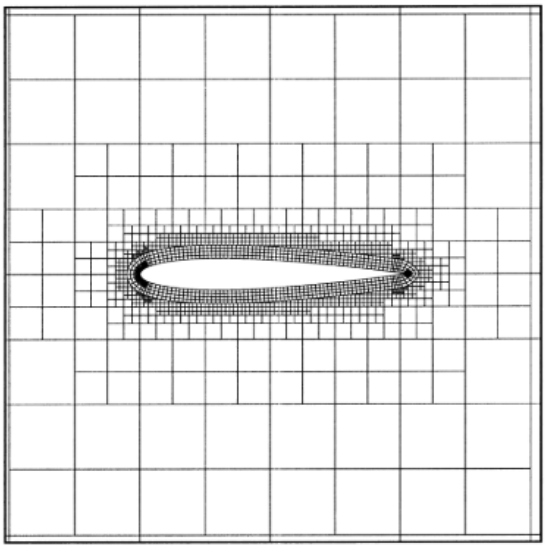


Figure 1.10: Cartesian mesh

On the other hand, it is often not possible to achieve body conforming Cartesian meshes. The complex geometries or the curved surfaces of the geometries are not captured well with Cartesian mesh. There are three solutions to this problem. First one is the interpolation/extrapolation of cell values at the boundary. The second method is the insertion of local geometry or flow aligned meshes, which is used for high speed viscous flows to generate high-aspect-ratio cells at the boundary. The third method is the trimming the surface cells and having numerous of various type cut cells, besides the poorly formed cut cells and extremely high number of cells at the boundary layers where the solution accuracy is needed the most.[14]

1.1.3 Extended Mesh Terminology

There are two mesh terminology: body-fitted mesh and block-structured mesh. Body-fitted meshes are based on mapping of the computational domain. This type of meshing is useful for the curved bodies. Block-structured mesh contains mesh regions which are meshed separately. This type of mesh provides an increase in mesh quality for complex geometries. [6]

1.1.3.1 Body-Fitted Mesh

In most cases for the single block mesh, since the geometry has complexities, the mesh cannot be made body-fitted. Namely, the nodes of the mesh at the boundary of the geometry do not match with the boundary of the geometry. An example can be given to this situation with cut cell approach. Without cut cells, the Cartesian mesh nodes cannot be on the same coordinate with the boundary of a complex geometry. As mentioned before, handling of the cut cells at the boundary of the geometry is time-consuming and needs high amount of computational power. Body-fitted meshes are utilized in the vicinity of the geometry where mostly the viscous region exists. By using body-fitted mesh, the boundary layer region is solved more accurately. The body-fitted meshes can handle curvatures. An example to a body-fitted mesh is shown in Figure 1.11. [15]

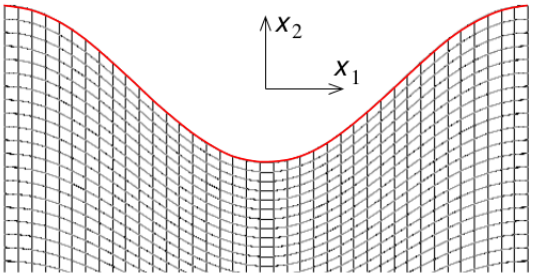


Figure 1.11: Body-fitted mesh

1.1.3.2 Block-Structured Mesh

Block-Structured mesh also has body-fitted feature such as the block boundaries coincide with the boundaries of another block mesh in the domain. An example for the block-structured mesh is shown in Figure 1.12. The mesh quality and the mesh properties can be different for each block of block-structured meshes to satisfy the quality requirements of the boundaries. [16]

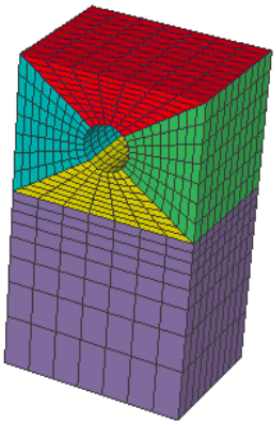


Figure 1.12: Block-Structured Mesh

1.2 Motivation

The mesh generation is the most important and crucial part of solving a flow problem with CFD. Since the quality of the computational mesh is directly effects the solution of the problem, the mesh generation should be handled with caution. For complex geometries, the meshing process becomes cumbersome due to the enormous number of mesh elements in the computational domain necessary to capture the geometric details.

Cartesian mesh can be used to improve computational requirements during the mesh generation, since it can be generated rapidly compared to other type of volume meshes. Cartesian mesh has also other advantages over other types of meshes. One of its advantages is the meshing with less number of volume elements in the computational domain. For instance, the Cartesian mesh requires less volume elements when it is compared to tetrahedral mesh in the same domain for similar solution accuracy. As another advantage, the mesh generation time would be less than the tetrahedral mesh generation, especially in three dimensional domain. High order solvers such as LES solvers often require orthogonal cells that is easily provided by Cartesian mesh.

However, the Cartesian meshes have irregular shaped finite volume elements at the boundaries, the cut cell approach used in at the boundary. At the boundary of the geometry, the mesh generation requires more attention with the irregular shaped vol-

ume elements. The irregular shape elements affect the mesh quality metrics badly. In boundary region, there should be high aspect ratio cells with high orthogonal quality for capturing the large gradient change in the boundary layer region. However, cut-cell approach cause high skewness/low orthogonal quality of the mesh at the boundary layer region. Therefore, this is not an elegant solution at the boundary layers. Cut cell generation algorithms creates cells with irregular shapes. The cutting algorithms add processing time to the mesh generation. Moreover, the volume elements should not be overlapped with each other and should be body-fitted to the watertight geometry.

Therefore, the motivation to this study is to develop Cartesian based mesh generator with body-fitted boundary layers. The main focus is the development of body-conforming high quality meshes for fluid mechanics problems.

- The mesh generator provides high quality mesh and high orthogonality mesh adjacent to solid boundaries.

- Single zone mesh should be generated for maximum compatibility. The core mesh has high quality mesh metric by using the hexahedral mesh.

- The developed mesh generator is focused on external flows, and it can be extended to internal flows too.

- The solution would be more accurate compared to tetrahedral mesh in the flow domain with very high quality Cartesian core mesh.

- The boundary layer consists of wedge volume elements. This type of mesh provides body-conformity to the final mesh. Besides, the high aspect ratio cells provide capturing the high gradients of the flow variables at the boundary layer.

- The connection is made between the Cartesian core mesh and the boundary layer mesh with wedge volume elements by using fast-generated tetrahedral volume elements and pyramids.

1.3 Literature Review

In this thesis, developed mesh generator contains a Cartesian core mesh connected to a body-fitted mesh at the vicinity of the geometry. In literature, Cartesian mesh has been used in studies for several years. In addition, in literature, the transition between the structured or Cartesian mesh to the body-fitted mesh is provided in literature by using unstructured mesh, especially the triangles in two-dimension and tetrahedral elements in three-dimension. The literature review for the mesh generation is given in the chronological order and mostly emphasized on Cartesian based mesh generators and the unstructured transition mesh on the body-fitted boundary layer mesh.

In 1986, Clarke et al. used the Cartesian mesh for solving a finite volume formulation for the Euler equations. The boundary cells are cut to fit to the geometry. The results of the study showed the simplicity of using Cartesian mesh with finite volume formulation. [17]

In 1987, Gaffney et al. used the Cartesian Mesh for modeling Euler equations on airfoils. The cells at the boundary are handled by cutting the boundary cell. [18]

In 1971, Peskin et al. studied both the numerical computation of the heart valve motion and the force exerted by the fluid on, with the immersed boundary method. With the immersed boundary method, at the nodes of the Cartesian mesh domain, the boundary is replaced by a field of body forces. By this way, the moving boundaries can be handled. Since the forces on the boundary are highly sensitive, the numerical instability can be produced by changing in the boundary conditions. However, the instability on the boundary is handled by using implicit method for boundary forces. [19]

In 1995, Steinbrenner and Noack studied on a three dimensional hybrid mesh generation. In the mesh generation, both structured and unstructured mesh types are used. Firstly, the structured mesh is generated around the geometries independently. Then, if the structured mesh of a geometry is overlapping with another structured mesh of a near geometry, the overlapping structured meshes are deleted. After the rejection of the overlapping cells, there will be gaps where the structured mesh has been deleted. Unstructured mesh is generated up to the regions with no cell. Therefore, the mesh

generation results in hybrid mesh. The structured mesh part of the hybrid mesh is generated by using Octree data storage. Since for the generation of structured cells and deletion of the overlapping cells is done by using advancing front technique, it requires significant amount of data to search for the cells. An example of meshing is given in Figure 1.13. [20]

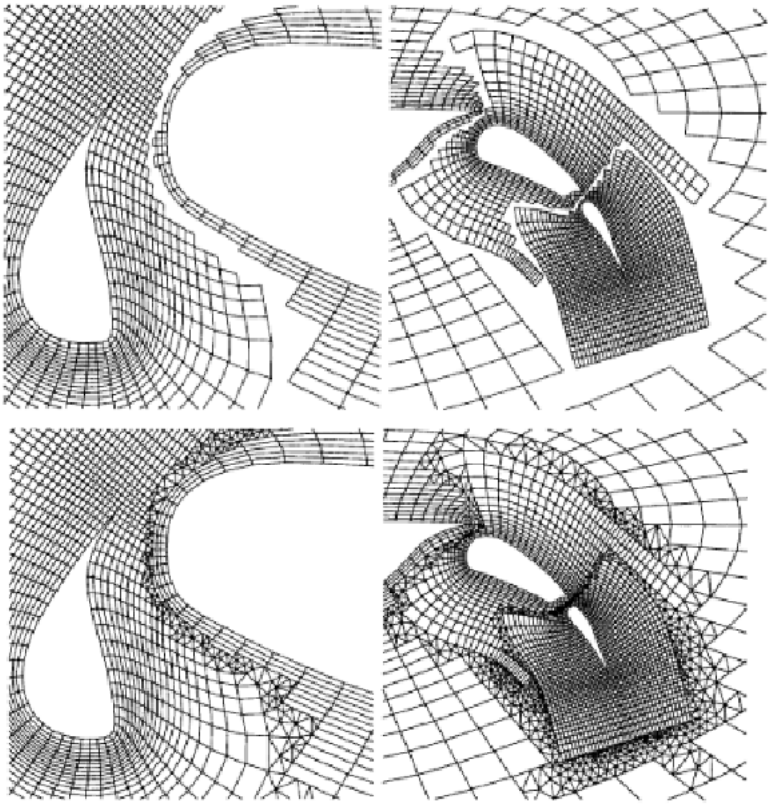


Figure 1.13: Meshing Process

Thanks to the octree algorithm, searching is done fast for the cells and the neighbor cells can be found easily. As a result of this study, it is seen that using structured mesh in the far-field reduces the time required for mesh generation. Also, the number of mesh is reduced by using the structure mesh in far-field. Moreover, it is stated that the hybrid mesh generation time should be improved and the quality of the mesh at the transition parts should be improved by enhancing the face selection criteria for the surface where the transition begins from structured to unstructured mesh. [20]

In 1997, Charlton and Powell studied an octree-based mesh generation method. Hybrid mesh is generated. Cartesian and structured mesh is both utilized for non match-

ing zones. The hybrid mesh used in the study is given in Figure 1.14. [21]

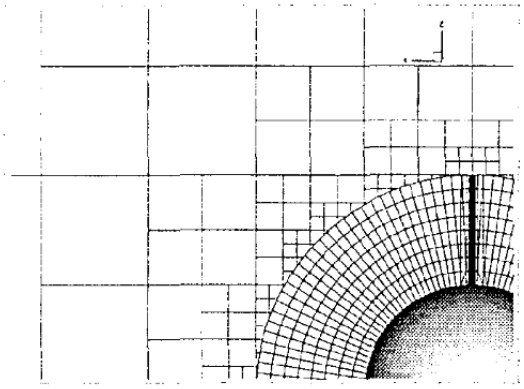


Figure 1.14: Octree-Based Hybrid Mesh

In 1997, Tchon et al produced an unstructured hexahedral mesh generator for viscous flow simulations. The mesh generator is suitable for three dimensional geometries. The first generation consists of only hexahedral mesh, then the boundary, the corners and the small regions are captured by body-fitting methods and by degenerating the hexahedral cells. The approach of the study is to lower the time required for three dimensional meshing process. Also, the methods need maturity in terms of mesh quality. An example to a generated mesh on Peugeot 405 car without wheels is given in Figure 1.15. [22]

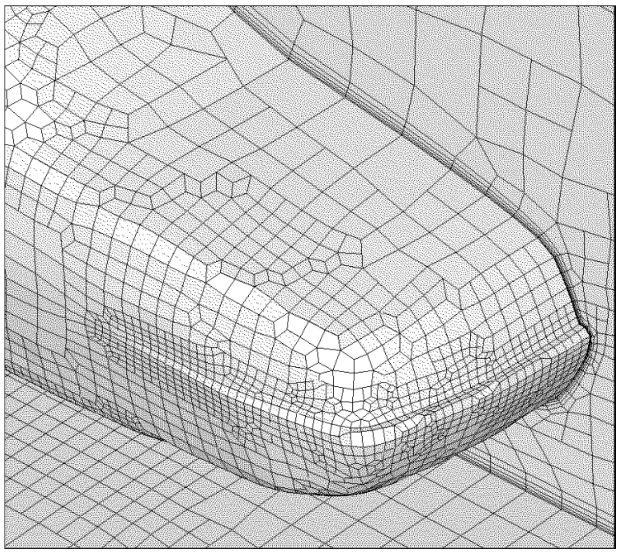


Figure 1.15: Peugeot 405 Car Without Wheels Volume Mesh

In 1998, Wang studied a quadtree-based adaptive Cartesian/Quad mesh to solve the Navier-Stokes equations. First, body-fitted quad mesh are generated in the study. Then, the Cartesian mesh is generated. The smallest cell of the Cartesian mesh is has the same cell size as the cell of quad mesh. Thus, the Cartesian and the quad mesh are overlapped with the outer surface of the quad mesh and the cells are cut to have the final mesh. An example mesh form the study is given in Figure 1.16. [23]

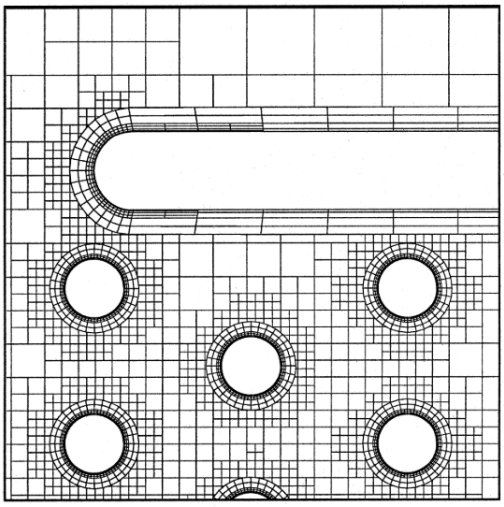


Figure 1.16: Final Cartesian/Quad Mesh

The Quadtree data structure is used when generating the quad in the near boundary. According to author, the refinement and the coarsening of the mesh becomes trivial in this way. It is stated that use of two types of mesh, namely the Cartesian and the quad mesh, increases robustness and successes of the solution. Complex geometries can be meshed successfully with this mesh generator. Also, the storage of data based on the Quadtree data structure stores the mesh and allows mesh adaptation easily. As a result of the study, it is concluded that this study can be restudied in three dimensions and the mesh can be Cartesian/adaptive prism mesh. [23]

In 1999, Tucker and Pan studied the cut cell approach with Cartesian mesh for the incompressible laminar flow. The boundary of the mesh with the geometry is made up with cut cell approach. A mesh is given from a test case from the study in Figure 1.17.

[14]

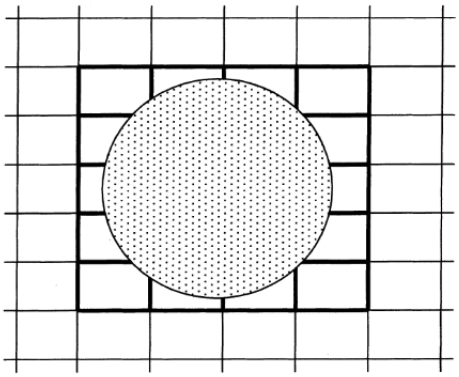


Figure 1.17: Flow Around a Cylinder Test Case Mesh

In 1999, Delaneye et al. studied on a Hybrid-Cartesian mesh generation. The mesh generation process consists of four steps and the process is fully automated without any user involvement. Firstly, Cartesian mesh is generated at the far-field of the geometry. Then, the unstructured body-conforming mesh is generated around the wall of the geometry. The unstructured mesh on the wall of the geometry is generated by using a front marching technique. By this way, the cells at the near wall of the geometry have high orthogonal ratios and high aspect ratios. At this step, there may be regions where the unstructured mesh overlaps with another unstructured mesh layer. The unstructured mesh is discarded and erased on overlapping regions. Thus, the blank regions ,where the mesh is erased, are filled with another Cartesian mesh. The filler Cartesian mesh is different than the Cartesian mesh, which is generated in the first step. At the intersection of the Cartesian mesh and the unstructured wall mesh is provided by cut cell approach. Test cases are the flat plate with a turbulent boundary layer, RAE2822 airfoil AGARD Case 1 and Case 9, and finally a multi-element airfoil. Two cases are tested with flat plate: x-axis aligned case and the 30 degrees rotated case. For the x-axis aligned case, small triangular shaped cells are formed from the Cartesian mesh. However, except those, there is a smooth transition between the Cartesian Mesh and the unstructured mesh at the near wall. In the 30 degrees aligned case, there are irregularly cut cells with different shapes, but they are mostly quadrilateral. Thus, with the rotated case, the effect of the irregular cells are seen. These test cases can be seen in Figure 1.18.

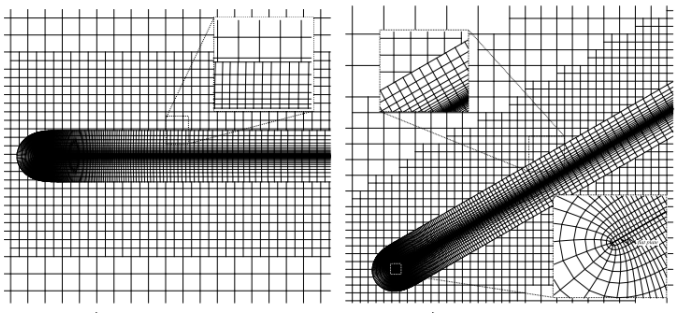


Figure 1.18: Test Cases - Flat Plate and Rotated Plate

The solution of the two cases is compared with the experimental results. It is shown that the solution matches with the experimental results. In addition to the flat plate, RAE2822 airfoil geometries are tested. Two cases of the airfoil are selected as test cases: AGARD Case 1 with Mach number equal to 0.676 and Reynolds number $5.7x10^6$, and AGARD Case 9 with Mach number equal to 0.73 and Reynolds number $6.5x10^6$. AGARD Case 1 geometry is meshed with three different configurations: Body-conforming wall mesh intersects with the coarse Cartesian mesh, Body-conforming wall mesh intersects with the fine Cartesian mesh and Body-conforming wall mesh intersects with the high aspect ratio Cartesian Mesh. These test cases are given in the Figure 1.19.

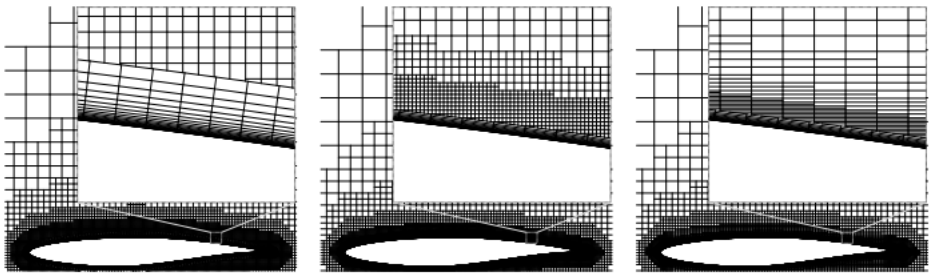


Figure 1.19: Test Cases - AGARD Case 1

It is shown that the first one to be converged is the mesh with second configuration. It is resulted that the irregularities in the mesh cause non-linear system to solve. In AGARD Case 9, a shock, which causes the thickening of the boundary layer, is observed on the upper surface of the geometry. Although the location of the shock is predicted on the upstream of the results obtained from the experimental measurements

with the mesh generated for AGARD Case 9, the lift and drag coefficients agree well with the experimental measurements. It is stated that the difference in the location of the shock is caused by the behavior of the Spalart-Allmaras turbulence model. Finally, the multi-element airfoil shows close results to the experimental results and it is shown that the mesh generation works well for the complex geometries which have concave and convex edges.[24]

In 2003, Zheng and Liou presented a hybrid mesh generator. The name of the mesh generator is called DRAGON that stands for Direct Replacement of Arbitrary mesh Overlapping by Nonstructured mesh. In their method, the structured mesh is generated around the geometry. Then, the body-fitted mesh is generated on the geometry wall. Therefore, there is an overlapping region between them and the overlapped region is shown in Figure **??**. The structured mesh on the overlapped region is deleted. Then, the deleted blank region is filled with unstructured mesh. The filled region is shown in Figure 1.20.

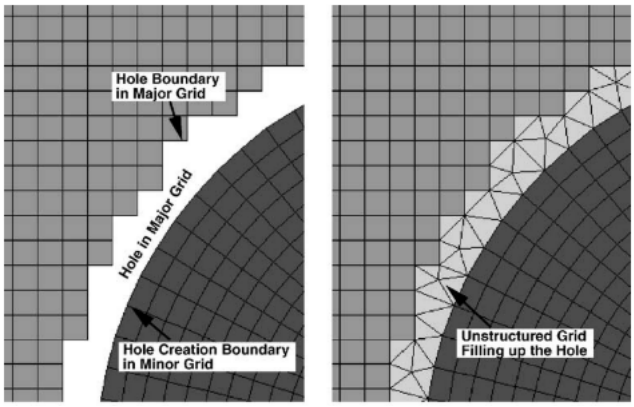


Figure 1.20: Filling with Unstructured Mesh

In 2006, Martineau et al. studied the mesh with Cartesian far-field, unstructured tetrahedral and advancing triangular mesh in the near field region. In the study, the Cartesian mesh far-field is compared to the hexahedral dominant unstructured far-field mesh. The transition to Cartesian and unstructured hexahedral dominant mesh is provided by the cut cell approach of the Cartesian mesh. The solution obtained with CFD, is compared to the experimental results of RAE2822 Case 9. The mesh of the test case RAE2822 Case 9 is given in Figure 1.21.

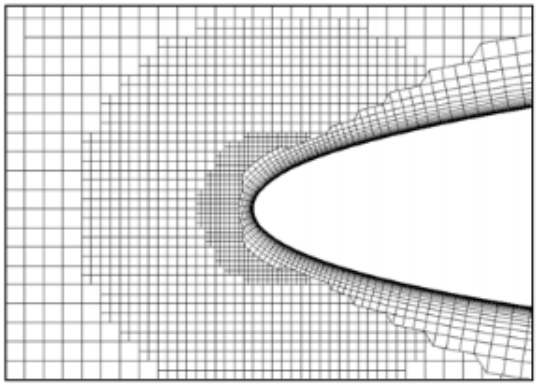


Figure 1.21: Test Case - RAE2822 Case 9

It is concluded that the mesh applications are successful in two-dimensional domain. Moreover, for three-dimensional ONERA M6 geometry, the comparisons came up to be successful and it shows that the mesh applications are effective for three-dimensional and simple geometries with transonic flows. [25]

In 2010, Luo et al. studied on a hybrid mesh generation method on complex geometries in two dimensional domain. Orthogonality and directionality of the structured mesh is aimed to be used in the boundary layer mesh. Also, Cartesian mesh generation method is used to create the flow domain for its simplicity and efficient generation. In addition, a triangle mesh is used between the structured boundary layer mesh and Cartesian mesh, which provides flexibility to the mesh for complex geometries. By this way, a robust and fast generated hybrid mesh generator is created. Firstly, the semi-structured boundary layer grid is created on the wetted surface of the geometry. Then, Cartesian mesh is generated by using Quadtree-based Cartesian Method. The Cartesian cells which are not in the computational domain is removed by an Alternating Digital Tree method. Finally, to fill the empty region between the Cartesian mesh and the boundary layer mesh, triangular mesh is generated. In the study, the cells with very high aspect ratio at the boundary layer are self-divided. This self-division maintains the sizes of the boundary layer cells similar with the Cartesian cells. By this way, the transition can be smooth along the triangular mesh. Hence, the quality of the mesh is improved. This mesh generator is tested for a number of test cases. One of the test cases is shown in Figure 1.9. The study by Luo et al. is conducted for only two

dimensional cases, but it is stated that the study can be extended to three-dimensional cases. [26]

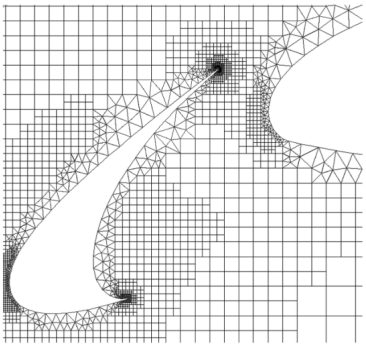


Figure 1.22: Hybrid Mesh Example

In 2010, Wissink et al. studied the aerodynamic performance of hovering rotors by using unstructured-adaptive Cartesian Mesh. The unstructured mesh is used at the near of the geometry, where the study calls near-body. Cartesian mesh is used in the off-body part of the mesh. In this study, off-body term refers to farfield mesh of the computational domain and near-body term refers to the boundary layer mesh of the computational domain. The unstructured mesh in the near-body region is used to capture the viscous effects and the complexity of the geometry. Cartesian Mesh is used to capture the wake in the far-field region. Two different solvers are used for off-body and near-body mesh. The connectivity between off-body and near-body is made with a code PUNDIT which uses overset grid approach. The parts of the mesh is shown in Figure 1.23. [27]

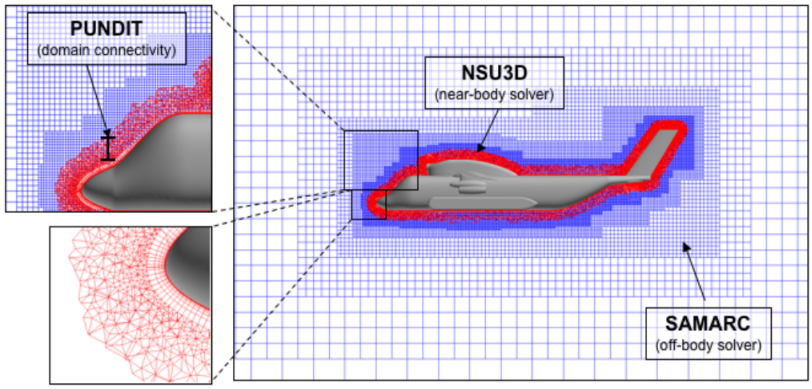


Figure 1.23: Off-body, Near-body and Overset Grid

In addition, adaptive refinement is done to capture the tip vortices on the Cartesian Mesh within the scope of this study. [27]

In 2018, Roget et al. developed mesh generator for boundary layer mesh at the vicinity of the wall of the geometry to capture the viscous layer in Reynolds Averaged Navier Stokes (RANS) with high Reynols Numbers. A new algorithm is developed in the study for the generation for the viscous layer mesh. The generation of the prismatic layer in the viscous layer starts with a point placement. Then, the closest point is placed and smoothing is applied by using the elastic spring analogy. Cartesian mesh is used for the far-field mesh. Generated mesh for the helicopter body is shown in Figure 1.24.

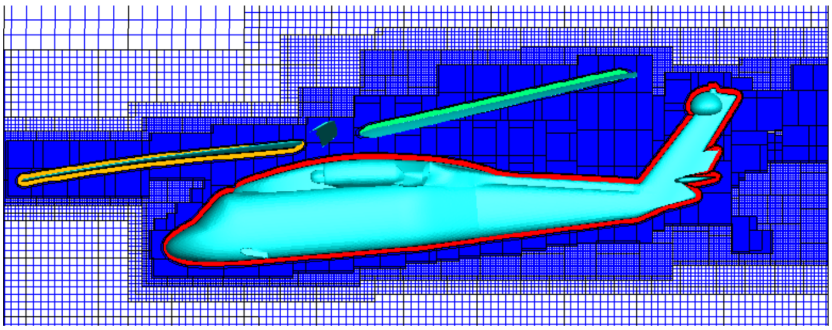


Figure 1.24: Helicopter Body / Cartesian Mesh and Prismatic Layer

The transition between the prismatic mesh and the Cartesian mesh is provided through overlapping mesh. The overlapped region can be seen in Figure 1.25.

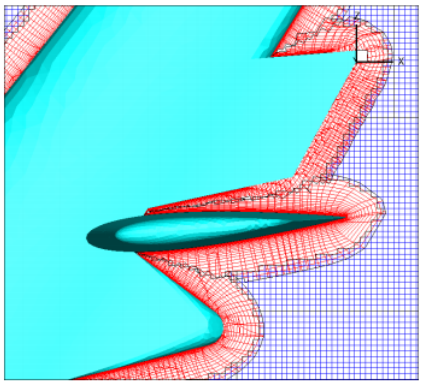


Figure 1.25: Overlapping Cells

The overlapping part of the mesh provides minimum possible overlapping between prismatic volume elements and Cartesian volume elements. The study is conducted for both two and three dimensional cases.[28] In addition, there are solvers with mesh generators, which use Cartesian mesh.

In 1999, Aftosmis et al. created CART3D which is a Cartesian mesh based parallel flow solver. The program starts with only a surface geometry and generates Cartesian based volume mesh with pre-processed geometry. Then, the flow solution is computed by Euler equations and post-processing is done.[29]

In 2003, Allred and Colloway validate the NASCART-GT Flow Solver with data comparison in Georgia Institute of Technology. This computer program was developed by Dr. Stephen Ruffin. NASCART-GT is used for generating Cartesian mesh around the geometry and the calculation is done on intersection points. Then, the flow variables are visualized by using Fieldview, which is a visualization program. [30]

In 2007, OctVCE is created by Tang. OctVCE is a numerical simulation code for shock and blast wave interactions on complex geometries. It uses Cartesian mesh which has octree based adaptive refinement and parallel processing. [31]

In 2009, Hashimoto et al. created HexaGrid which is an automatic mesh generator based on hexahedral mesh. The code handles complex geometries and automatically generates Cartesian mesh around the geometry. Multi-component geometries can be used in STL format as the input to the program. In the program, flow solver is also embedded: TAS-code (Tohoku University Aerodynamic Simulation code) is used.[32]

There are open source mesh generators such as SnappyHexMesh and HexPress. SnappyHexMesh is a mesh generator which is controlled by OpenFOAM libraries. It uses high quality hex-dominant mesh. The mesh generation can be done in parallel. The geometry data is given as input to the mesh generator. The steps of meshing process is given Figure 1.26.

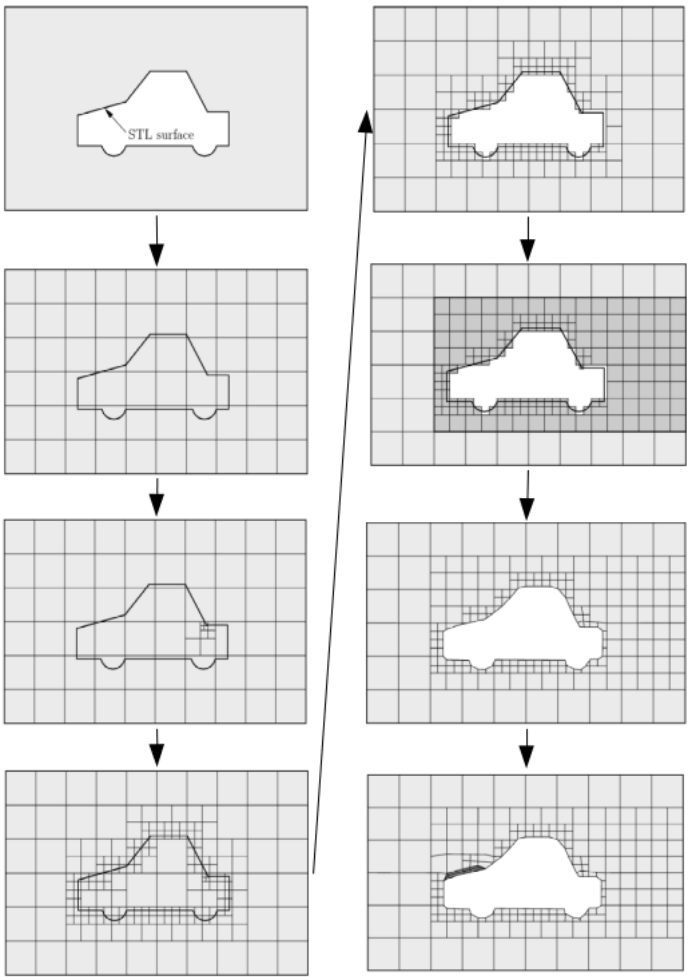


Figure 1.26: SnappyHexMesh Meshing Methodology

In snappyHexMesh methodology, the first step is to create base mesh as seen in Figure 1.26. The geometry file in STL or Nastran type is given to the program and hexahedral base mesh is generated around the geometry. Secondly, surface refinement an volume refinement is done. Then, unused cells are removed from the domain. The mesh is snapped to the geometry by cutting the hexhedral cells. Finally, the snapping layer is pushed away from the geometry and boundary layers are added between the snapping layer and geometry wall. The output of the mesh generator is well compatible with OpenFOAM solvers.

Another mesh generator is HexPress by Numeca. HexPress is a parallel unstructured hex dominant meshing for complex and/or unclean geometries. It has an hole searcher algorithm that fixes and cleans the geometry before starting to mesh. The

mesh generator can create both surface mesh and volume mesh. It generates full hexahedral mesh with hanging nodes, hexahedral dominant mesh with non-hanging nodes or fully tetrahedral mesh. The surface mesh generation is done with fully triangle or quad-dominant mesh. High quality viscous mesh is included in volume mesh capability. An example of mesh which is generated by HexPress is shown in Figure 1.27 as hexahedral mesh with hanging nodes on quad-dominant surface mesh.

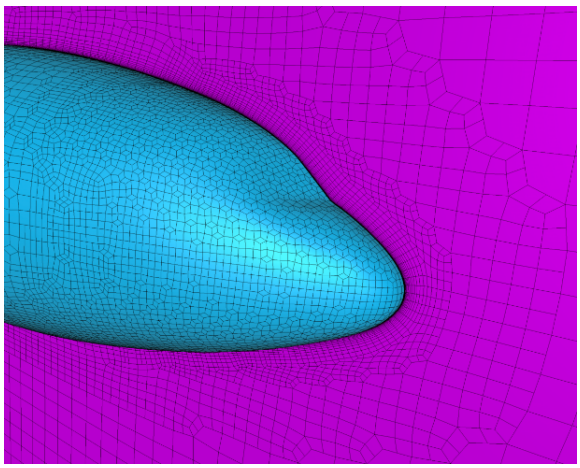


Figure 1.27: HexPress Hexahedral Mesh with hanging Node and Quad-Dominant Surface Mesh on NASA Common Research Model (CRM) airplane

1.4 Aim and Organization of the Thesis

1.4.1 Aim

Development of a body conforming, Cartesian based with high quality boundary layer mesh generator is aimed in this thesis. The developed mesh generator aimed to be fast and robust with smooth transition at the boundary layer cells as well as boundary layer to Cartesian cells.

Alternative to the cut cell approach used in the literature, a Cartesian based mesh is generated with pyramids as transitional elements to the boundary layer mesh. In the developed mesh generator, the volume elements are not cut in the Cartesian mesh to handle the transition between the far-field mesh and the near boundary mesh. Therefore, the mesh quality can be increased by not using cut cells in the computational

domain. Firstly, the Cartesian mesh is generated with Octree algorithm which is an algorithm to generate the hexahedral volume elements quickly. Starting from root cell, which is an encapsulating cube of geometry, the Cartesian mesh is generated by dividing each cell into eight children cells until the desired level of refinement is reached. The cells which are in the out of computational domain is omitted out. Then, the pyramids are created on the geometry side of the boundary of the Cartesian mesh. After generation of the Cartesian mesh with pyramid volume elements, the boundary layer mesh is generated by using a customized open source mesh generator. The boundary layer mesh is made of wedge volume elements having the triangular face. Since both Cartesian and the boundary wedge mesh have triangular faces, the gap between the Cartesian mesh with pyramid volume elements and the boundary layer wedge volume elements is filled with another open source mesh generator.

The aim is to have a mesh generator which produces high quality volume mesh for fluid mechanics problems. The mesh generator produces high quality meshes at the solid boundaries by using wedge eelements at the boundary layer mesh. For maximum compatibility, single zone mesh is used. This mesh generator is aimed to be used for external CFD problems and it can be extended to internal flows too. The primary specialty of the mesh is having Cartesian core mesh for very high quality. The body conformity is provided by boundary layer mesh with wedge volume elements. Finaly, the developed Cartesian based mesh generator with body-fitted boundary layers is aimed to be used in the cell-based SU2 solver.

Since all types of mesh generation are quick generation processes, the overall mesh generator requires little amount of time to generate the Cartesian based mesh around a three dimensional geometry. The flow chart of the developed mesh generator which shows the generation steps of the mesh is given in Figure 1.28.

Figure 1.28: Flow Chart of Developed Mesh Generation

1.4.2 Organization

In Chapter 2, boundary layer wedge mesh generator and the customized open source boundary layer mesh generation code is introduced. Firstly, the open source code Larosterna SUMO is given in detail and the customization of the code for the new mesh generator is presented.

In Chapter 3, Cartesian mesh generation is presented. Firstly, Octree Data Structure is introduced. The generation of pyramid volume elements are given. Finally, in this chapter, the polyMesh structure is introduced and the polyMesh properties and the creation of the polyMesh files are presented.

In Chapter 4, the tetrahedral mesh generator TetGen is introduced. The tetrahedral mesh generator is given in detail.

In Chapter 5, test cases are given for the validation of the developed mesh generator code.

In Chapter 6, the obtained solution results are discussed and the conclusion of the study is presented. Finally, future works are introduced.

CHAPTER 2

BOUNDARY LAYER MESH GENERATION

In aeronautical applications, the turbulence and the thin boundary layer affects the flow properties at the vicinity of the solid wall due to no slip condition. Most of the solutions contains Reynolds-averaged Navier-Stokes equations (RANS), therefore obtaining solutions on where the boundary layer is close to the wall of the geometry, a well-defined and fine finite volume mesh is needed at the wall boundary.

In viscous flows, due to the viscous effects of the wall on the flow, boundary layer is developed on the wall. At the boundary layer, the mesh must be able to resolve the large velocity gradients in the turbulent boundary layer along the normal of the wall. Therefore, there should placed many grid points as much as possible in the boundary layer. Highly stretched meshes are needed to be created in the boundary layer and shear layer region. There are two alternatives for the boundary layer mesh: structured mesh and unstructured mesh. For stretching the meshes, structured mesh is more suitable than stretching tetrahedral volume element for the viscous layer with high Reynolds number flows. The boundary layer mesh should be generated carefully by giving attention to the interior angles of the elements which should not be larger than 90^o. Also, the high aspect ratio cells should be aligned with the wall boundary. [9] In most of the studies, the body-fitted structured mesh is used for the boundary layer and unstructured tetrahedral mesh is used for the entire flow domain except the boundary layer. The boundary layer mesh is consist of high quality elements and the large flow domain is filled quickly by tetrahedral volume mesh.

Turbulence models have different ways of modeling the boundary layer. According to turbulence model, the resolution of boundary layer mesh can change. The quantity for this resolution is y+ value. This value defines the dimensionless height of the

first layer of the boundary layer grid point from the wall. In turbulent boundary layer region, there are three sub-layers in boundary layer: Viscous sub-layer, log-law sub-layer and outer layer. y+ is less than 5 for viscous sub-layer where the thickness of sub-layer is very thin; shear stress is approximately constant and equal to the wall shear stress. y+ value is between 30 and 500 for a region where the turbulent and viscous effects are both important and effective. In boundary layer region, turbulence is maintained by shear in the flow. The magnitude of the shear is very large near the wall. Also, the velocity and its fluctuations are high. The distribution of velocity is anisotropic. The shear layer is very thin and the flow characteristics are changing very rapidly in the direction perpendicular to the wall boundary. The boundary layer mesh must have very thin layers of elements to capture the gradients of flow variables in the very thin boundary layer at the vicinity of the wall. Therefore, the boundary layer mesh must be generated such that y+ value is achieved for the turbulence model.[6]

The quality of the mesh in the boundary layer mesh is very important to achieve the flow variables accurately. Also, the mesh generation process should be automated. A robust strategy is needed to take care of the deformations, such as warped elements and negative volume elements, of mesh due to the geometric complications. The purpose of the creating hybrid prismatic mesh is to increase the mesh resolution at the boundary layer mesh and capture the viscous effects stemmed from no-slip boundary condition. Therefore, large flow gradients are observed at the normal direction of the wall that requires high cell density in the normal direction, but not necessarily high cell density along the direction parallel to wall.

The boundary layer in the vicinity of the surface mesh of the geometry is generated using the quadrilateral prismatic cells called wedges. The wedges are generated by a library of the open-source hybrid mesh generator by Larosterna SUMO. The motivation for creating the hybrid mesh generator by the developers is the need for the boundary layer mesh where the boundary layer effects is liked to be seen. The most important aspects of generating the boundary layer wedge mesh is the flexibility of the mesh generation with increasing of geometry complexity. The generation of boundary layer wedge mesh becomes a tedious task. The generated mesh is aimed to used for the cell-centered solvers. The main property of the hybrid mesh generator is being fast and robust even with the large mesh sizes. [33]

The generation of the wedge volume mesh consists of four parts. The first part is reading the surface mesh of the geometry. The second part involves the generation of farfield surface mesh and the envelope mesh. Envelope prismatic layer is the surface mesh where the last layer of the wedge elements are located. Third step is the generation of boundary layer wedge elements by inflating from triangular surface mesh. Final step is the generation of tetrahedral mesh in between the farfield and envelope layer surface meshes.

The mesh generator requires input files which involves a triangular surface mesh with CGNS format [34] for the compatibility and a configuration file. The boundary layer mesh parameters are given in the configuration file. The input parameters of mesh in the file is given in Table 2.1.

Table 2.1: Hybrid Mesh Generator Configuration File Inputs with Example Values

Configuration File Property	**Example Value**	**Units**
Number of Layers	10	-
Initial Height of the First Layer	0.1	m
Maximum Growth Ratio	1.2	-
Maximum Layer Thickness	100	m

The outputs of the boundary layer mesh generator are two files: boundaries.smesh and hybrid.su2 files. They are transfered to the Cartesian Mesh Generator. The boundaries.smesh file consists of the envelope layer surface mesh nodes and faces. The hybrid.su2 file consists of the nodes and cells of the wedge elements. The faces of wedge volume elements and the connectivity of wedge cells are generated using the hybrid.su2 file. The faces and the owners/neighbors of the faces are processed by using the hybrid.su2 file with another C++ program: WedgeHandler. This program generates two files: wedgeFaces and wedgeNodes text files. The wedgeFaces file consists of faces with owner, neighbor and the marker of the face. The wedge cells are numbered starting from the first layer which is close to the geometry surface. Therefore, there is an ID for every wedge cell to write them for owner and neighbor. Finally, wedgeFaces and wedgeNodes are transfered to a program which makes the final polyMesh.

2.1 Boundary Layer Mesh Generation Process

2.1.1 Surface Mesh

The surface mesh is created by any compatible surface mesh generation tool. The format of the surface mesh must be a CGNS file and surface normal orientation should be arranged as into the geomerty.

2.1.2 Envelope Prismatic Layer

After surface mesh is read by the program, the envelope prismatic layer is created. It is another triangular surface mesh which is created inflating the surface mesh. The wedge volume elements will be generated later between the envelope prismatic layer and the surface mesh. The envelope prismatic layer is generated with the same topology as the surface mesh. It is generated at a distance from the surface mesh. The distance is given from the configuration file and it is the boundary layer mesh total thickness. This thickness is calculated by using number of layers, initial height of the boundary layers and the growing ratio of the cells. For example, if the initial height of the first layer is 0.1 mm, and the growth ratio is 1.2, then the second layer height will be 0.12 mm. Therefore, if there is 10 layers, the total height of the boundary layer will be 2.60 mm. However, at some parts of the complex geometries, the height of the envelope prismatic layer is reduced for not clustering of the boundary layer. Hence, the construction of the prismatic envelope layer is not an simple extrusion process. The warped wedge elements are corrected while inflating the surface mesh by untangling the surface normals with local untangling and warp reduction algorithms.

Since the geometry can be complex, the generation of the prismatic layer is done step by step along local normals of the elements, not by simply extruding the surface mesh. The steps of the envelope prismatic layer generation is given in Table 2.2 and the steps are explained below. .

Table 2.2: Envelope Prismatic Layer Generation Steps

1	feature extraction and surface node classification
2	Selective Smoothing of Growth Directions and Layer Height
3	local untangling and warp reduction
4	global collision avoidance of opposing layers

2.1.2.1 Feature Extraction and Surface Node Classification

The nodes are detected and arranged according to their locations on the surface mesh. The angles between the neighbor triangles of the surface mesh is calculated. Then, the nodes of the surface mesh assigned with flags according to their positions and the angles of the surface normals in the mesh. A node can have more than one flag depending on the shape of the geometry. For example, a node can be on a topological edge associated to both concave and convex surfaces of the geometry. Therefore, this node is flagged as a saddle node. As an example, F-16XL geometry is shown with Figure 2.1. In this figure, the edges which are shown in black are the regions where the nodes are flagged with multiple flags, these black ones are different form the other nodes. These nodes are on both two surfaces which makes them to be on a concave or convex edge unlike the nodes on a single surface.

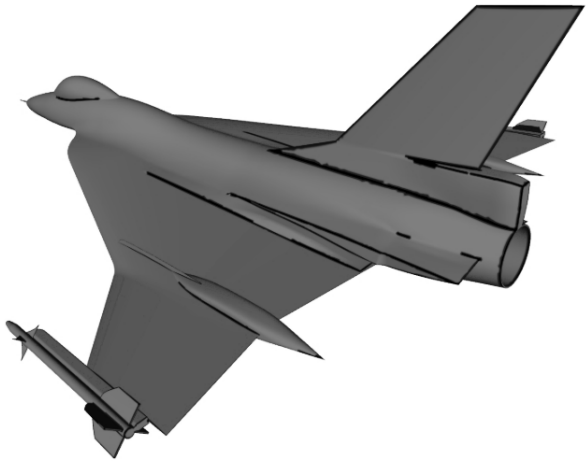


Figure 2.1: Geometric Feature Detection

One of the local criteria is the angle between the normal vectors of the triangles with common edge between them. Also, concave or convex surfaces are detected by flagging the nodes. The flagging of nodes process only includes the evaluating the local geometry and not includes the evaluating of boundaries.

2.1.2.2 Selective Smoothing of Growth Directions and Layer Height

The envelope prismatic layer is generated using the vertex-based surface normal vectors. These normal vectors are obtained from angle-weighted average of the normal vectors of adjacent triangles at each vertex. The local layer thickness is initialized by first computing the mean length of the incident edges at a vertex. Then, the initial height of the local element of the layer is computed by using Equation 2.1 where r_j, which is the local height expansion ratio of adjacent prisms and cannot exceed the value given by the user in the configuration file as maximum layer thickness (r_m), is computed by Equation 2.2. $h_0$is taken from the configuration file as initial height value of the first layer.

$$h_j = h_0 \frac{1 - r_j^n}{1 - r_j} \tag{2.1}$$

$$r_j = min[(\frac{l_j}{h_0})^{\frac{1}{n-1}}, r_m] \tag{2.2}$$

If the generated envelope surface is not smooth to generate the prismatic volume elements, the boundary layer wedge mesh cannot be created properly. An example can be seen in Figure 2.2 is given. Unsmoothed envelope mesh is shown on the left and smoothed envelope mesh is shown on the right. [35, 33]

Initial envelope surface mesh has irregular shapes due to the computations of the normal vectors. Therefore, smoothing operation is applied to the surface mesh. Three smoothing passes are applied. The first pass is the application of the modified Laplacian smoothing factor to the height field by using small Jacobi iterations. Also, the height modifications of first pass uses the vertex flags which was generated in the previous step. Second pass is applied to the growth directions. This pass also uses

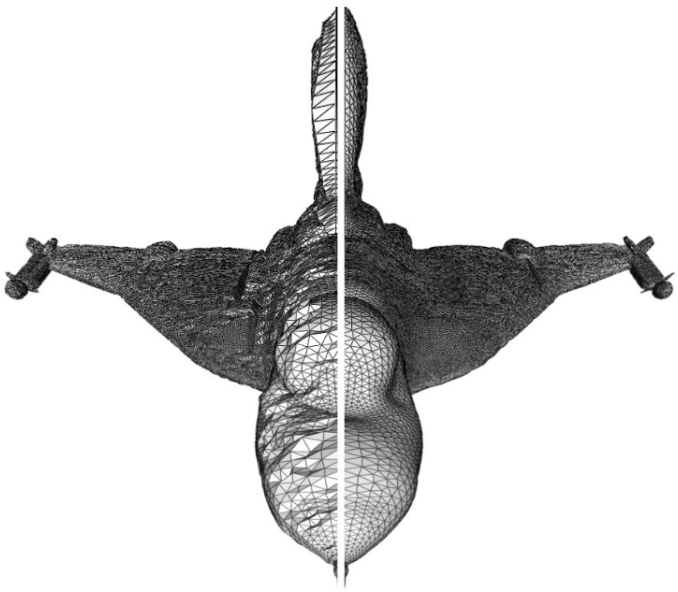


Figure 2.2: Envelope Smoothing Example

flag-dependent modifications. The third pass involves the smoothing of the edges. As a result of the smoothing, resulted envelope mesh for the F16 example is shown in the right half of Figure 2.2. As it can be observed from the figure, the quality of the wedge volume elements are good between the wall surface and the envelope prismatic envelope mesh. [35, 33]

2.1.2.3 Local Untangling and Warp Reduction

Entangled wedge elements in the boundary layer wedge mesh are observed sometimes due to the intersections of the normals of the surface elements while inflating from the surface. Two step untangling algorithm starting from local untangling of the normal heights is applied.

First phase of the untangling is the limiting the layer height. The local height of the envelope layer cannot exceed h_j value computed, for each edge e, by Equation 2.3 when $0 < \gamma < \pi$ and $\gamma = \pi - \beta_1 - \beta_2$.

$$h_j < |e| \frac{sin\beta_j}{sin\gamma} \tag{2.3}$$

The angles between the edge direction and the vertex normal vectors in the endpoints of the edge, which are $\beta_1$and β_2 respectively, are computed for each edge in wall surface mesh.

The initial envelope layer my contain warped elements and these elements can be inverted. Hence, this causes negative volume element at the boundary layer mesh. There warped elements should be avoided. An example to the warped elements are shown in on the leftmost side Figure 2.3. Although the envelope layer triangle normals should be towards outer, on warped elements, they point towards wall surface. [35, 33]

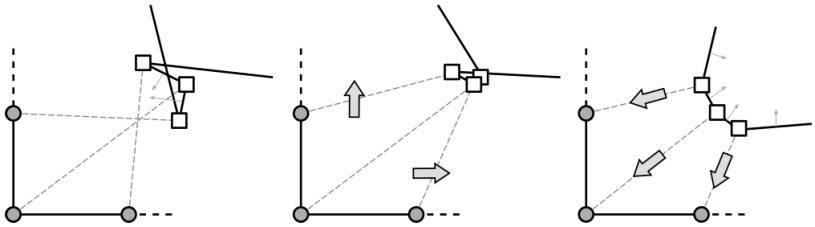


Figure 2.3: Untangling of Elements

As it can be seen from the center figure of the Figure 2.3, the variation between envelope direction vectors are not enough to solve the warping problem. So, the height of the envelope layer is reduced to a maximum possible value to overcome the warped elements. The limit of this height value is zero and also yields and acceptable wedge element. The reduction of the envelope layer height is shown with the rightmost figure in Figure 2.3. [35, 33]

2.1.2.4 Global Collision Avoidance of Opposing Layers

The height of the envelope layer is reduced for the potential overlapping envelope layers. The height of the envelope layer is changed and decided by local geometric considerations. The boundary layers are thinned between the encroaching bodies to generated tetrahedral volume mesh between them. Therefore, in some complex geometries, the boundary layer can be thin in one part and can be thick in another part of the geometry. An example is given with Figure 2.4. [35, 33]

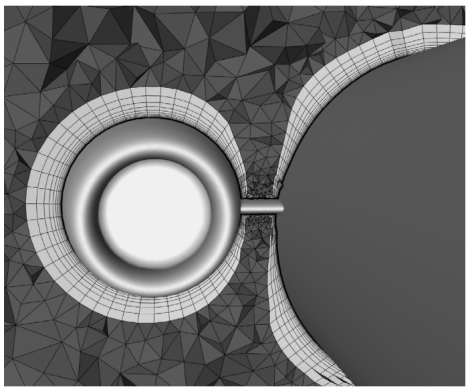


Figure 2.4: Example - Fuselage and Engine Nacelle

The detection of possible collision of boundary layer mesh on a geometry is provided by an efficient search tree data structure: using a balanced binary tree bounding volume hierarchy. This data structure is very fast for detecting the points, where the envelope layer collides. Then, it reduces the local height of the envelope layer. [35, 33]

2.2 Customizations on Hybrid Mesh Generator

At the end of the program, the wedge volume elements, which are created between wall and envelope surface meshes, is obtained. The first layer height is given with the configuration file. After generating the first layer, the other layers are created by applying the growth ratio which is given in the configuration file. The generation of the boundary layer mesh is very fast, since the mesh is structured in the normal direction of the wall surface.

There are some customizations on the hybrid mesh generator code. The tetrahedral mesh generator part is suppressed in the code to extract only the wedge mesh generator part of the code. Also, for the Cartesian mesh generator code, the envelope prismatic surface mesh is required to generate the Cartesian mesh and the pyramid volume elements. One of the output of hybrid mesh generator is “boundaries.smesh” file. This file contains three surface meshes. One is the far-field mesh of the geometry. The far-field would be the boundary of the tetrahedral mesh if hybrid mesh generator code had created the tetrahedral mesh. Since tetrahedral mesh is not generated in the

code, the far-field surface is suppressed to be written in the boundaries file. Therefore, the other two surface meshes, which are envelope prismatic layer and the wall surface mesh of the geometry, is written to the file. The wall surface mesh of the geometry is also supressed to be written to the boundaries file. As a result, the only surface in boundaries file becomes the envelope prismatic surface mesh. Therefore, the "boundaries.smesh" file is directly taken by the Cartesian mesh generator as the main input file.

An example of envelope layer on surface wall mesh is given in Figure 2.5. The mesh with yellow surface is the envelope layer mesh and the mesh with blue surface is the wall mesh. Also, the boundary layer which is generated between the envelope layer and the wall mesh and the boundary layer mesh is shown in Figure 5.7.

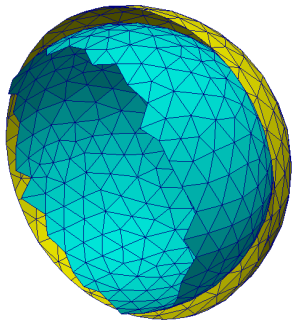


Figure 2.5: Envelope Layer with Surface Wall Mesh

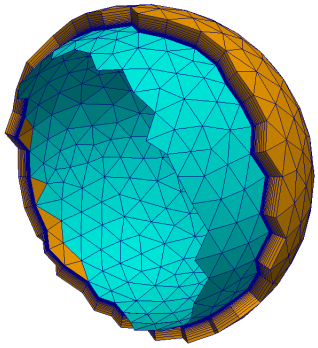


Figure 2.6: Boundary Layer Mesh with Surface Wall Mesh

CHAPTER 3

CARTESIAN MESH GENERATION

The Cartesian Mesh is generated outside of the boundary layer zone of the computational domain. As given in the Motivation part in Chapter 1, the core section is aimed to be generated with the Cartesian Mesh which consists of high quality hexahedral cells. Octree Data Algorithm is used for generation of cells for the Cartesian Mesh generation. Transitional elements from Cartesian mesh to boundary layer mesh is provided by pyramid volume elements at the Cartesian side and tetrahedral meshes at the boundary layer side. Pyramid volume elements are generated within the Cartesian Mesh generation code since they are generated from the faces of Cartesian mesh cells.

3.1 Octree Data Structure

In Cartesian mesh generation code, the computational domain is divided into Cartesian cells with multiple refinement levels and sizes. Octree data structures and algorithm is employed for the generation of Cartesian cells. Octree algorithm is suitable for three dimensional meshes, while the quadtree data structure is utilized for two dimensional domain. [36]

Octree data structure is a hierarchic data structure that spans in various computer science applications. Octree is a tree data structure where each internal node has exactly three children. When applied to computer graphics and mesh generation, the main trunk of tree represents the biggest cell of the mesh that is to be divided into eight children and this first cell is called "`root cell`". The root cell is a rectangular prism which encapsulates the entire geometry or in mesh generation terms, computational domain. The other cells are generated by the successive subdivision of the

root cell into eight equal octant. The division continues until a size criteria is satisfied. Using this procedure, the number of necessary division at a location known and hence, the octree hierarchy is generated. [37]

An example of the hierarchy is shown in Figure 3.1.

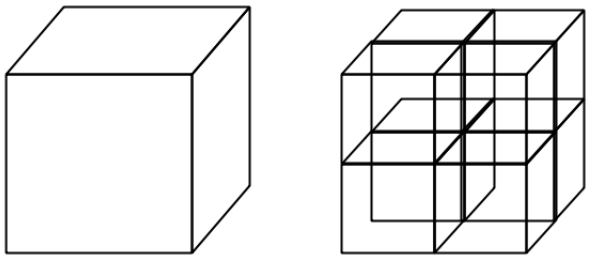


Figure 3.1: Quadtree and Octree Data Structure

The size of the smallest cell is decided by a size criteria. Refinement level is a value which shows the number of dividing process to reach the size of the smallest cell. Therefore, when the root cell is divided, the size of the smallest cell is identical to the size of the boundary layer cell. By changing the size criteria, the refinement value and the root cell size, the size of the whole domain and the cell sizes can be changed. The refinement level of a cell can be found from Equation 3.1 and can be seen that it is dependent on the criteria and the root cell edge length. Also, after every refinement of the cells, the size of edge length is cut in half. If cells are divided by n, the edge length is decreased as 2^n. If the desired local length criteria is known, the refinement level can be found by 3.1.

$$Refinement\ Level = \log_2 \left(\frac{root\ cell\ edge\ length}{local\ length\ criteria} \right) \tag{3.1}$$

The location of cells can be found easily. Since every cell has its own parent cell, the particular cell can be found traversing starting from the root cell. Every cell has a parent cell except the root cell. By the children-parent relation, a cell can be reached from the root cell. Besides the refinement level, identification numbers (ID) are given to the each children cell, during the construction of children cells following the sub-

division request. Unique combination of Cell IDs is given for all three directions for three dimensional domain. Parent-children relation as well as coordinates of the cell center can accessed following the ID and refinement level of the cell. The ID is assigned by binary designation. An example of the binary identification number is given in Figure 3.2.

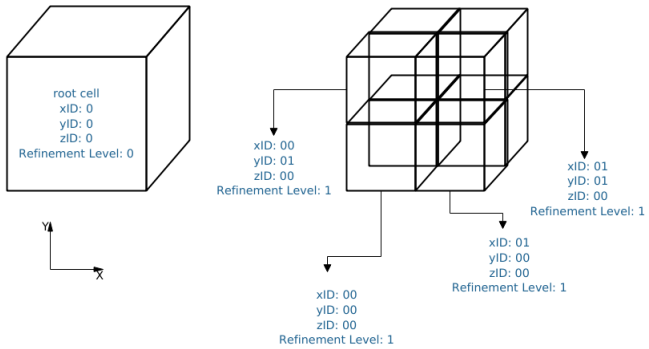


Figure 3.2: Binary Identification Numbering

The ID of the root cell is 0 in each dimension and the refinement level is also 0. When the root cell is divided into children cells, four cells at the lower part at a given direction are assigned with the ID as 0 at the given direction, next four cells at the higher upper part at the same direction is assigned with the ID as 1 of that direction. In binary expressions, the ID of the lower (or left, back) cell is 0 and the ID of the upper (right, forward) cell is 1. The number of digits used for cell is determined by refinement levels of the cells . When the refinement level is increased by one, the numbers in the binary expression is shifted to left. For example, if the second cell, which has ID as x-ID equals to 1, is divided into its children cells, the first cell of the children cells has the ID as 1 and the refinement level is 2. For the next cell to this cell, x-ID is 2 and refinement level equals to 2.

3.2 Cartesian Mesh Generator Module

Cartesian Mesh Generator code is responsible of generation of the Cartesian mesh and the pyramids volume elements. Cartesian mesh generator is constructed around

C++ classes representing Cartesian cells, faces, nodes and pyramids. The list of the base classes are given in Table 3.1.

Table 3.1: Classes of Cartesian Mesh Generator Code

Class Name	**Property**
HexCell	Generates hexahedra volume elements
HexNode	Generates nodes
HexFace	Generates faces of hexahedra volume elements
HexPyramid	Generates pyramids
HexApex	Generates apex nodes
SurfaceMesh	Reads envelope mesh and configuration file
PolyMeshWriter	Writes polyMesh files

The Cartesian Mesh generation process starts with reading of envelope mesh and the configuration file which is used as the boundary of the hexahedral cell generation domain. The hexahedral cells are generated and then marked whether they are removed or kept as computational cells. After generation of cells, the faces and the nodes of the hexahedral cells are also generated. Finally, the pyramids are generated, then all of the faces and the nodes are exported as polyhedral mesh format. The detailed explanations of the generations of faces, nodes and pyramids are given in the following sections.

3.2.1 Surface Mesh Import

The open source Larosterna SUMO mesh generator is used and it is customized into a boundary layer wedge mesh generator. From the boundary layer wedge mesh generator, a boundary surface mesh is extracted. The boundary surface mesh is the surface triangles of the last layer of the boundary layer wedge mesh. This surface mesh is imported with a input file to the Cartesian Mesh Generator. The surface file format is TetGen based ASCII .smesh.

The .smesh file type is made for defining Piecewise Linear Complex (PLC) surface mesh. [38] Surface mesh data is processed and utilized for the future use in Cartesian

mesh. The surface mesh file consists of mesh coordinates and the triangulation. The triangulation is stored in the correct normal orientation. The surface mesh is used in Cartesian mesh and will be used in tetrahedral meshing.

After reading the surface mesh and saving the nodes and elements to arrays, the configuration file is read. The configuration file stores the following information.

- Root Cell Edge Length
- Root Cell Center Coordinates
- Seed point coordinates

Root cell edge length is the half of the edge length of the root cell. The coordinates of the root cell center is given by root cell center coordinates in the configuration file. Also, the seed cell coordinates in three dimensions are given in the configuration file. The explanation of seed cell will be given in later parts when the computational Cartesian cells are marked. The centers of the triangular faces of the envelope mesh are calculated and stored. Next, the criteria for number of division criteria for the cells at the vicinity of the triangles are calculated based on edge lengths of the triangular surface elements of envelope mesh. For this purpose, the function "`calculateCellEdgeLength()`" is used. The selection of the number of division steps is based on the criteria of the smallest Cartesian cell's edge size which should be similar in size of the triangular surface edge length of the envelope mesh element. [39] The functions for reading of the surface mesh and configuration file, calculating the cell center and criteria of distance for the refinement is implemented in the "`SurfaceMesh`" class.

3.2.2 Generation of Hexahedral Cells

In the generation of hexahedral cells, the aim is to create by successive subdivision at each envelope layer surface mesh triangle element. The x, y and z coordinate ID of the hexahedral cell which intersects the envelope surface layer mesh is aimed to be found. Also, the found hexahedral cell should be in similar size with the triangle of the surface mesh. The hexahedral cells surrounds the center coordinate of the

triangular cell of the surface mesh. The created hexahedral cells which surrounds the triangle cell centers of the surface mesh is marked as CUTCELL type hexahedral cell. According to the coordinates of the elements center coordinates of the envelope surface mesh, the hexahedral cells are created. The equations for obtain the ID of hexahedral cells are given with Equations 3.2-3.4. The root edge is the half of the root cell edge length in Equations 3.2-3.4.

$$cutcellID_x = \frac{[ElementCenter_x - (rootCenter_x - rootEdge)]}{\frac{2*rootEdge}{2^{Refinement\,Level}}} \tag{3.2}$$

$$cutcellID_y = \frac{[ElementCenter_y - (rootCenter_y - rootEdge)]}{\frac{2*rootEdge}{2^{Refinement\,Level}}} \tag{3.3}$$

$$cutcellID_z = \frac{[ElementCenter_z - (rootCenter_z - rootEdge)]}{\frac{2*rootEdge}{2^{Refinement\,Level}}} \tag{3.4}$$

After obtaining the center coordinates of the triangular surface elements and distance criteria, CUTCELL ID in x,y and z directions are calculated using the Equations 3.2 - 3.4. The size has been read from the configuration file and has been saved to a value. The name "initialOctree" is given to the root cell.

3.2.2.1 CUTCELL Marking

CUTCELL type cells are the cells which are at the boundary of the surface mesh. The hexahedral cells, which surround the cell center of the triangles of the surface mesh, are created and marked as CUTCELL. Then, diffusion is done with these cells. The neighbor cells of the initially marked cells are also marked as CUTCELL. The repeat number of the diffusion can be given to the code. Hence, for sphere surface wall mesh, the CUTCELL type cells are created and marked as in Figure 3.3.

The CUTCELL cells do not only consist of the boundary intersecting cells. CUTCELL type cells are the cells which separates the inner and outer cells of the computational domain. CUTCELL type cells are removed later and they do not exist in the computational domain. These cells are generated with a given criteria, such that, the size of

the cells are similar to the smallest edge length of the triangular surface elements at their closest vicinity. The cells other than CUTCELL type cells are marked as NONE type cells. The other cells with marked as NONE type are shown in Figure 3.4

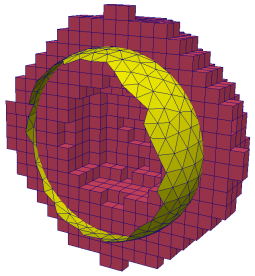
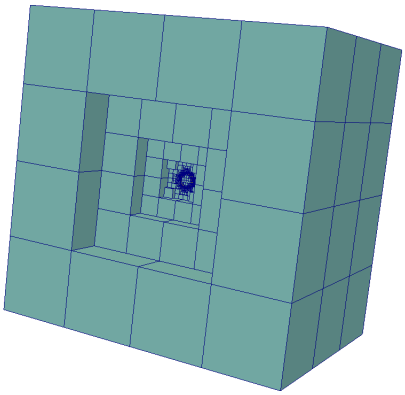
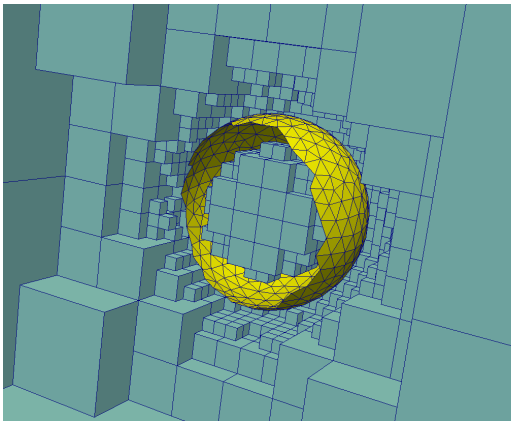


Figure 3.3: Initial Octree CUTCELL Type Cells and Surface Mesh



(a) Initial Octree NONE Cells



(b) Initial Octree NONE Cells - Close Up

Figure 3.4: Initial Octree NONE Cells and Surface Mesh

After marking the cells at the boundary of the envelope surface mesh as CUTCELL, a new root cell with HexCell class type is generated. This new Root cell is created a new copy of the initial octree mesh with improved smoothness between NONE and CUTCELL cells and called Diffusion Octree. The size of the NONE cells are changing so rapidly that the smoothness quality of the mesh is not achieved. Therefore, the aim of copying the initial octree is to improve the smoothness of the NONE cells. Hence, the initial octree is copied as diffusion octree. Starting from the finest refinement level, it is ensured that each refined cell has at least two cells with same refinement level before the neighbor of one highest level. This diffusion process is a recursive process. The operations are repeated successively for larger cells. The Cartesian mesh after the application of the refinement diffusion operation is shown in Figure 3.5. In Figure 3.5, the blue cells are the NONE cells. From Figure 3.4, it can be seen that refinement level of the cells do not allow smoothness. However, as can be seen from Figure 3.5, there are two cells with same refinement level between the cells with different refinement level. This provides smoothness throughout the Cartesian mesh. The pink cells are CUTCELL cells. The yellow mesh is the envelope layer mesh.

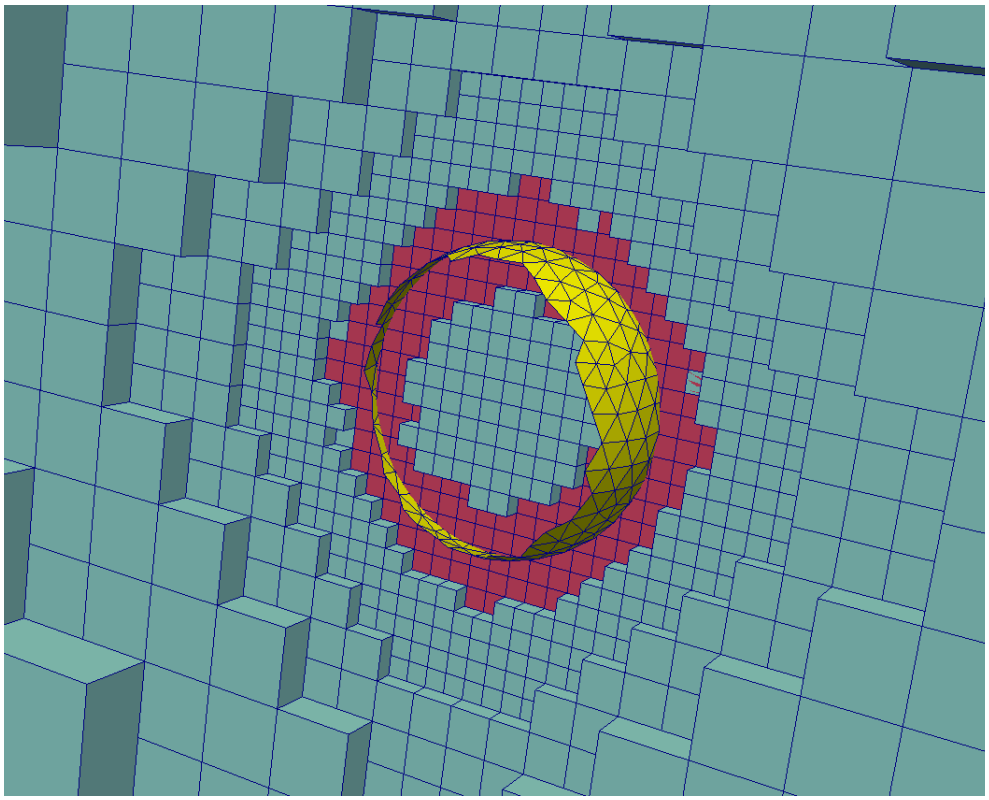


Figure 3.5: Diffusion Octree with NONE and CUTCELL Cells and Surface Mesh

3.2.2.2 Marking the in-Domain Cells

At this point Diffusion Octree is created, and this octree will be the main tree to work on. As the last part of the generation of hexahedral cells, after marking `CUTCELL` type cells, in-domain cells, namely `INCELL` type cells are marked. These in-Domain cells will be used as computational cells. `INCELL` cells are marked by staring from a seed cell. Seed cell location is defined in the configuration file and it is read at the beginning of the code. The seed cell x-ID, y-ID and z-ID can be found from the coordinates by using 3.2,3.3 and 3.4. The ElementCenter in Equation 3.2 to 3.4 is the coordinates of the seed cell for seed cell ID calculation. An example to the seed cell is shown in Figure 3.6. The red cell indicates the location of the seed cell.

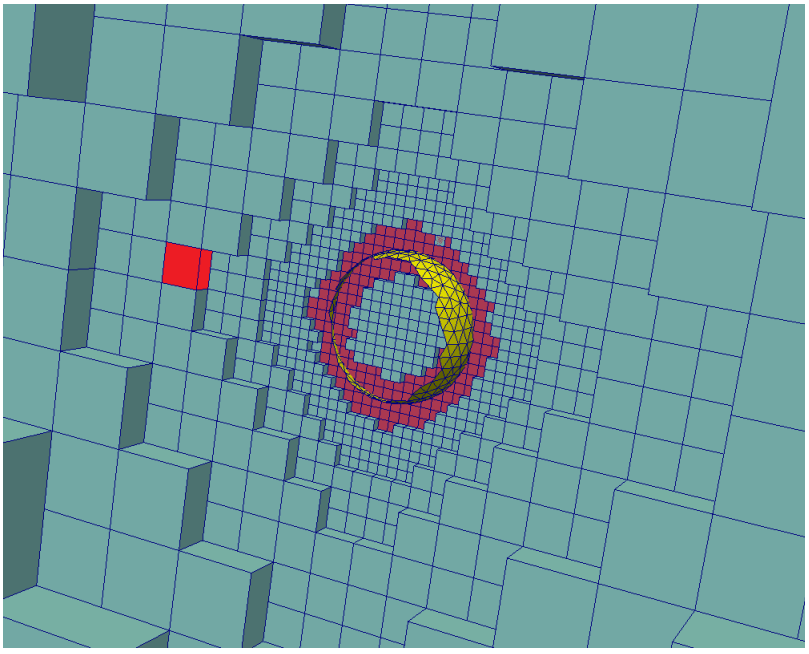


Figure 3.6: Seed Cell Location

Then, the cell, which shares the same x-ID, y-ID and z-ID with the seed cell, is founded recursively staring from the root cell. The cells, which make up the computational domain, are marked as INCELL by using the function "`markIncells()`". Now, cells neighboring the first `INCELL` is marked as `INCELL` too. If the neighbor is not `CUTCELL` or if there is neighbor cells. This procedure is repeated recursively. `CUTCELL` cells become a barrier between `INCELL` and `NONE` cells. This explains

the necessity of special diffusion process in CUTCELL cells. Otherwise INCELL type cells can leak to NONE type cells. In "markIncells()" function, the seed cell is marked as INCELL and calls another function "diffuseIncell()". This function starts to the right of seed cell by adding 1 to the x-ID of the seed cell. If the x-ID plus one is less than the maximum possible x-ID which a cell can have with the refinement level of seed cell, the cell is the finest cell, means has no children cells, with the given x-ID in the refinement level. Also, the case of which the x-ID is less than zero is checked as same as for the x-ID plus one. The marking is also repeated in both y and z directions in same manner. Then, the code continues as checking if the cell has child. If the cells does not have children cells, the refinement level is increased by one and x-ID, y-ID and z-ID of the cell is shifted bitwise to left. By this way, if there are cells which are children cells of the neighbor cell, they can be marked as well. The marking of the children cells of the neighbor cell is given in Figure 3.7

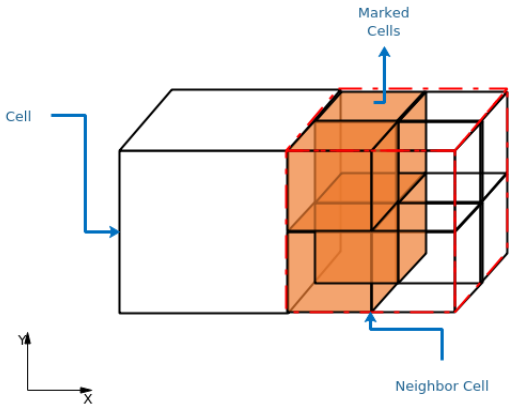


Figure 3.7: Marking of Children Cells of Neighbor Cell

If there is not children cells and the neighbor cell has the refinement level same as the seed cell, from seed cell, "markAndDiffuseIncell()" is called. This function checks if the neighbor cell is cell with the mark NONE. If the cell is NONE, the cell status is changed to INCELL. Then, "diffuseIncell()" function is run again from the root cell. In addition to x direction, INCELL marking is done in the y and z directions. By this way, the complete computational domain is marked as INCELL from the outer boundary to inner boundary. The outer boundary is the root cell and the

inner boundary is the CUTCELL which are generated at the first step and copied to the Diffusion Octree. For example, the INCELL are shown in Figure 3.8 and Figure 3.9 with green. The pink cells are CUTCELL and the yellow mesh is the triangular surface mesh.

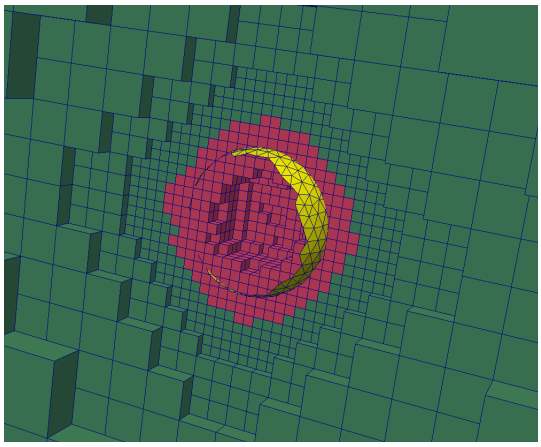


Figure 3.8: INCELL and CUTCELL and Envelope Layer Mesh

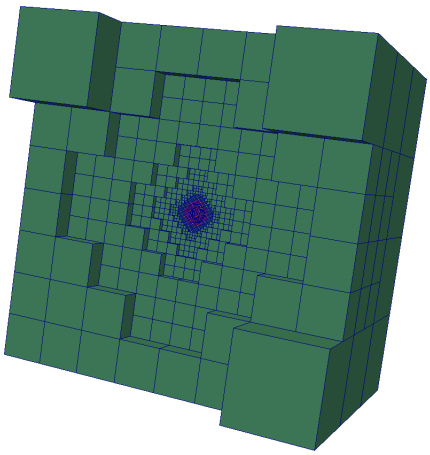


Figure 3.9: INCELL and CUTCELL Type Cells and Envelope Layer Mesh from Far Angle

After finishing the core Cartesian mesh, next step is to generate the pyramid cells at the CUTCELL faces. Each pyramid cell will have its square base at the outer surface of the INCELL and apex at the center of the CUTCELL cells. Therefore, CUTCELL cells that are neighbor to an INCELL should be found and marked. Also, INCELL cells with a neighbor to a CUTCELL should be found to ensure the connec-

tivity. The marking of the cells are continue with marking "INCELL-INNER" and "CUTCELL_OUTER" typed cells. The function starts again from the root cell and marks recursively. For all the cells with no children cell are traversed. Then, the cell is checked whether it is an INCELL or INCELL-INNER cell. If this is the case, all neighbors are checked. If the neighbor is CUTCELL, the neighbor cell is marked as CUTCELL_OUTER. If the neighbor cell has children cells, the children cells of the neighbor cell is marked as CUTCELL_OUTER. The INCELL is also marked as INCELL-INNER.

3.2.2.3 Setting the Cell ID

The ID to the cells is given other than x-ID, y-ID and z-ID. A class variable is defined as "cellID". Setting cell ID is to assign a unique number to the cells. This is an important setting for the file export. Only INCELL and INCELL-INNER will be exported to the mesh file. Therefore, ID is assigned only the cell with INCELL and INCELL-INNER markings and has no children.

3.2.3 Connectivity of Hexahedral Cells

3.2.3.1 Face Generation

The mesh is Cartesian mesh which is a non-conformal mesh. The finite element type of conformal mesh formats are not suitable. Instead, the face-based data format is used to export for the Cartesian mesh. Hence, the mesh can be imported to the solvers such as OpenFOAM and FLUENT. OpenFOAM mesh type is selected because OpenFOAM is an open source solver which uses polyMesh format. In polyMesh format, the computational cells are shown with nodes and faces. Therefore, the faces are generated from the hexahedral cells.

Firstly, the cells are generated and marked in the previous steps of the code. The generation of faces are handled with a new class called HexFace class. HexFace is derived from the HexCell class. It should be noted that, an algorithmic connection exist between faces and cells. Also each division operation divides faces, too.

The faces are generated using the function "`createFace()`". The face creation function is a recursive function. Starting from root face, the faces are created. Therefore, the root face is defined. Unlike in the creation of cells, there are three different face tree in the code. One root face tree is for the faces with normals in x direction, the second face tree is for the faces with normals in y direction and another root face for the faces with normals in z direction: "rootFaceX", "rootFaceY" and "rootFaceZ", respectively.

The faces are created by using "`createCellWithID()`" function as in the case as cells called from `HexCell` class. In every direction, there are two faces. Therefore, from a root face, two children cells are obtained. One of the faces coordinate IDs are equal to the parent face and the ID of the other face is one more of the parent face ID in each direction. The ID of the faces are given as Table 3.2.

Table 3.2: Face IDs

	HEX_CELL	x-ID	y-ID	z-ID	Refinement Level
rootFaceX	HEX_FACE	x-ID	y-ID	z-ID	Refinement Level + 1
rootFaceX	HEX_FACE	x-ID + 1	y-ID	z-ID	Refinement Level + 1
rootFaceY	HEX_FACE	x-ID	y-ID	z-ID	Refinement Level + 1
rootFaceY	HEX_FACE	x-ID	y-ID + 1	z-ID	Refinement Level + 1
rootFaceZ	HEX_FACE	x-ID	y-ID	z-ID	Refinement Level + 1
rootFaceZ	HEX_FACE	x-ID	y-ID	z-ID + 1	Refinement Level +1

In addition, as can be seen from Table 3.2, the refinement level of the faces are increased by one as in the node creation. Since the x-ID ,y-ID and z-ID of the faces are related to the x-ID, y-ID and z-ID of the cell, if a cell is known, then the faces

of the cell can be found. Also, if x-ID, y-ID, z-ID and refinement level of a face is known, then the cell which it belongs to, can be known. The connectivity between the cells, node and the faces are required for the owner and neighbor cell search while the generation of polyMesh files.

Also, the ID location of the faces are given for x direction in Figure 3.10. The generation of faces in y and z direction is same as the x direction.

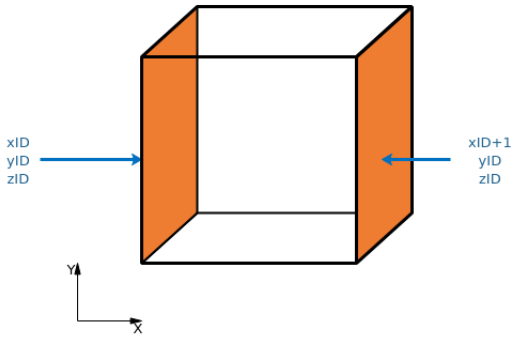


Figure 3.10: Face IDs

3.2.3.2 Face Marking

For faces, face status are defined. When a face is generated, the default value of the status of the face is defined as `NONE`. The faces must be marked to indicate for the purpose of that face in the mesh. Types of face status are `INCELL-FACE`, `FARFIELD-FACE` and `PYRAMID-BASE-FACE`. Types are defined in the `HexCell` class with enumeration technique. Face cell markings are conducted x, y and z directions separately, since respective trees are separate.The following sections should be read as such.

`INCELL-FACE` Marking First of all, all of the faces marked with `INCELL-FACE` status with a recursive function. The function calls root cell and refines the root cell until a cell with no children is found. If such cell is found, it is checked whether the cell is a cell with cell status `INCELL` or `INCELL-INNER`. If the cell is either one of them, both right and left faces of the cell is marked as `INCELL-FACE`. The faces

are reached from the cell's x-ID, y-ID, z-ID and refinement level. As seen from Table 3.2, there is relation with the cell IDs and refinement level with their counterparts of faces of those cells. x-ID, y-ID and z-ID of left face of a cell is equal to the x-ID, y-ID and z-ID of the cell, respectively. However, the refinement level of the face is one more of the refinement level of the cell. This marking procedure is applied for all three face trees.

`FARFIELD-FACE` Marking After all the faces are marked as `INCELL-FACE`, the faces with `FARFIELD-FACE` status is marked. The `FARFIELD-FACE` faces are the faces which are at the farfield boundary of the mesh. The boundary conditions are given to these cells to be used in the solver part. The marking function for the `FARFIELD-FACE` faces are done by a recursive function. The difference from the function of the `INCELL-FACE` marking is that this function starts from the root face. Therefore, there are three functions to mark far-field faces in x, y and z directions. The functions visit every face with no child face. If the face has not a child and the status is `INCELL-FACE` and for the function marking in x-direction, x-ID of the face is zero or x-ID is equal to maximum possible id at that refinement level ($2^n - 1$), the neighbor cell of the face is marked. x-ID equals to zero means the face is on the first face that can be obtained in that refinement level. The face is described by x-ID, y-ID, z-ID and refinement level value. The procedure for marking the far-field faces are repeated for the faces which have normals in the y direction and z direction. By this way, the faces in all directions are marked.

`PYRAMID-BASE-FACE` Marking The pyramids is build on the faces between the cells with status of `CUTCELL-OUTER` and `INCELL-INNER`. The base of the pyramid volume elements are marked in the function "`markPyramidBaseFaceX()`" for the faces with face normals in x direction. For y and z directions, same functions which are named for y and z directions are used. All faces are marked as `INCELL-FACE` at the beginning. The faces which make the base of the pyramids are marked as `PYRAMID-BASE-FACE`. Pyramid base faces are important for creating pyramid elements. They are between `INCELL-OUTER` and `CUTCELL-INNER` faces and marked recursively. The same check is done for the positive x direction, the

neighbor cell with x-ID, y-ID, z-ID and refinement level - 1 of the face is checked. If the neighbor cell exists and the status of the neighbor cell is `CUTCELL-OUTER`, the neighbor cell is marked as `PYRAMID-BASE-FACE` and the counting of the pyramid base face is increased once. For other directions, the same procedure applies.

3.2.3.3 Node Generation

Nodes are organized in an octree data structure as in case of cells and faces. The class containing the nodes is `HexNode` and it is inherited from `HexCell` class. The nodes are created as reversing from root cell. The creation of nodes starts with the function "`createNode()`" from the root node is the root of the node tree. The node creation function is a recursive function. The nodes are created with the function "`createCellWithID()`" as explained in Figure 3.11. The nodes have x-ID, y-ID and z-ID values as in the Figure 3.11.

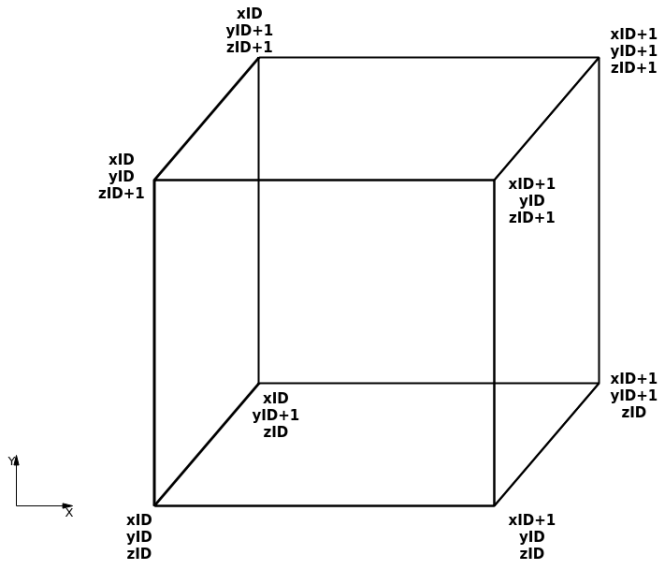


Figure 3.11: Node IDs

The root node is the node at the left corner of the root cell and has the same x-ID, y-ID, z-ID and refinement level. When the root node is refined once, eight children nodes are generated as shown in 3.11. Therefore, the refinement level of the nodes are one level more than the refinement level of the cells. By this connection of node

and cell, when the cell is known the desired nodes of the cell can be known.

3.2.3.4 Node Marking

For nodes, the node status are defined. When a node is generated, the default value of the status of the node is defined as NONE. The nodes must be marked to indicate for the purpose of that node in the mesh. The types of the node status are `INCELL-NODE`, `PYRAMID-BASE-NODE` and `TRIANGLE-FACE-NODE`.

`INCELL-NODE` Marking

Nodes of the `INCELL-FACE` and `FARFIELD-FACE` are marked as `INCELL-NODE`. All of the faces are searched and the faces with type of `INCELL-FACE`

and `FARFIELD-FACE` are found. From the relation between the face and the nodes, the four node of these type of faces are marked as `INCELL-NODE`.

`PYRAMID-BASE-NODE` Marking

The nodes of `PYRAMID-BASE-FACE` are marked as `PYRAMID-BASE-NODE`. All of the faces are searched and the faces with type of `PYRAMID-BASE-FACE` are found. From the relation between the face and the nodes, the four node of these type of faces are marked as `PYRAMID-BASE-NODE`.

`TRIANGLE-FACE-NODE` Marking The nodes of the triangle surfaces of the pyramid volume elements are marked as `TRIANGLE-FACE-NODE`. Firstly, the cells are checked if it is a `CUTCELL-OUTER` type cell. Then, the pyramid elements are generated by using the relation between the `CUTCELL-OUTER` cell and the pyramid element which is generated in the cell. While exporting the triangular faces, each triangular face contains two nodes and one edge. It is checked whether there is a pyramid which shares the same edge. If that is the case, the nodes of the edge is marked as `TRIANGLE-FACE-NODE`.

3.2.3.5 Node ID Setting

The procedure for setting node ID is same as the cell ID setting. The ID setting is in the order of marking. Firstly, the `INCELL-NODE` type nodes are given ID, then `PYRAMID-BASE-NODE` type nodes are given ID.

Finally, the `TRIANGLE-FACE-NODE` type nodes are given ID. The nodes are numbered with an recursive function which visits all the nodes in the tree of nodes with same type. As done in the cell ID setting, a node is numbered only once. A get&set function is used for giving number to the unnumbered nodes. All of the nodes in the mesh is numbered.

3.2.4 Generation of Pyramid Volume Elements

The pyramid cells are generated as same as the face generation. The function for creating the pyramids is named as "`createPyramids()`" which is located in `HexCell` class. The function is a recursive function. The pyramids are generated from three different root pyramid for every three direction; x, y and z directions since each pyramid is generated based on a particular `PYRAMID-BASE-FACE`. The root pyramids are named as rootPyramidX, rootPyramidY and rootPyramidZ. The pyramid ID is given to the pyramids is shown in Figure 3.12.

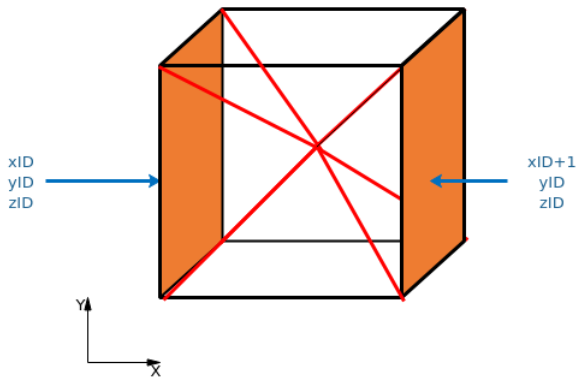


Figure 3.12: Pyramid IDs

The relation between the pyramid ID and the ID of the cell where the pyramid is located (an `CUTCELL-OUTER` cell) also given in Table 3.3for x direction, y direction and z direction.

Table 3.3: Pyramid IDs

	HEX_CELL	x-ID	y-ID	z-ID	Refinement Level
rootPyramidX	HEX_ PYRAMID	x-ID	y-ID	z-ID	Refinement Level + 1
rootPyramidX	HEX_ PYRAMID	x-ID + 1	y-ID	z-ID	Refinement Level + 1
rootPyramidY	HEX_ PYRAMID	x-ID	y-ID	z-ID	Refinement Level + 1
rootPyramidY	HEX_ PYRAMID	x-ID	y-ID + 1	z-ID	Refinement Level + 1
rootPyramidZ	HEX_ PYRAMID	x-ID	y-ID	z-ID	Refinement Level + 1
rootPyramidZ	HEX_ PYRAMID	x-ID	y-ID	z-ID + 1	Refinement Level +1

As seen from Table 3.3, the refinement level of the pyramids are one more than the cell in which the pyramid located. This relation between the pyramid and cell is important for the neighbor relations while writing output files. Moreover, it is seen that for a cell, the pyramid and the base face of the pyramid have same x-ID, y-ID, z-ID and refinement level, but the elements are reached from different root elements. Therefore, although the pyramid and the face have same ID values and the refinement level, the generated element is different in type of the element.

3.2.4.1 Pyramid ID Setting

Pyramid ID are set to each pyramid element. As for the cell ID setting, the pyramids are numbered to for the ease of reaching to the particular pyramid element while

writing the owner and neighbor files for the mesh output file. The ID of the pyramids starts from the ID number where the cell ID of hexahedral cells finished. Therefore, the numbering is continuation of the cell ID of hexahedral cells.

3.2.4.2 Pyramid Apex Generation

The apex coordinates of the pyramids are generated. The x-ID, y-ID, z-ID and the refinement level of the apex is same as the x-ID, y-ID, z-ID and the refinement level for the cell, since the apex is the center of the cell. Although x-ID, y-ID, z-ID and the refinement level of the apex is same as the cells', the root element is different. Therefore, there are not same element generation with the same x-ID, y-ID, z-ID and the refinement level.

3.2.4.3 Apex ID Setting

Apex of the pyramids are the center points of the cells where the pyramid is located. The status of the cells which host pyramids is `CUTCELL-OUTER`. Therefore, all of the cells are visited by the recursive function "`setApexID()`". If the status of the cell is `CUTCELL-OUTER`, the apex with the same x-ID, y-ID, z-ID and refinement level as the cell, is given ID. As had been the case for the nodes, cells and pyramid ID setting, apex numbers stars from the last node ID of the Cartesian cells.

3.2.5 Generation of PolyMesh Files for Cartesian and Pyramid Volume Elements

The output format of the Cartesian mesh generator is OpenFOAM polyMesh format. The information of the format of polyMesh files are given in Appendix A. The points file involves nodes of the hexahedral and pyramid volume elements. The faces file contains faces in the order of the face status. The first group is the faces of Cartesian mesh with `INCELL-FACE` status. The second group is the faces with `FARFIELD-FACE` and they are boundary cells. The third group is the faces with `PYRAMID-BASE-FACE`. Fourth and final group of faces is the triangular faces of

the pyramid elements. In the code, while writing the faces, the owner and neighbor cells are found in the same function and written in the owner and neighbor files. Lastly, the boundary file is generated. For the Cartesian and pyramid volume mesh, there are only two boundaries: the far-field boundary and the inner boundary where the tetrahedral elements will be generated. Therefore, in boundary file, there are two patches with the names as far-field and wall, respectively.

3.2.5.1 Coordinates of the Nodes at the Points File

The points file is created using the nodes of the mesh. The coordinates of the nodes are calculated firstly. The function for coordinate calculation is "`calculateNode Coords()`" which is a recursive function to visit all the nodes in mesh, starting from the root node to the nodes with no children node. In the function, an inner or boundary node without a child is visited, the coordinates of the this node calculated according to the Equations 3.5-3.7.

$$x\,coordinate = xID * (\frac{Root\,cell\,edge\,length}{2^{Refinement\,Level-1}}) + (Root\,cell\,center_x - \frac{Root\,cell\,edge\,length}{2}) \tag{3.5}$$

$$y\,coordinate = yID * (\frac{Root\,cell\,edge\,length}{2^{Refinement\,Level-1}}) + (Root\,cell\,center_y - \frac{Root\,cell\,edge\,length}{2}) \tag{3.6}$$

$$z\,coordinate = zID * (\frac{Root\,cell\,edge\,length}{2^{Refinement\,Level-1}}) + (Root\,cell\,center_z - \frac{Root\,cell\,edge\,length}{2}) \tag{3.7}$$

The calculated coordinates are written in an array. After generating the coordinate array, the coordinates are written to the file which is named "points" according to the points file format. While generating the points file, the first group of point coordinates

belongs to the node coordinates of the hexahedral cells. The second group is the apex point coordinates. The coordinates of the apex points are calculated. All of apex elements with an ID is visited to write them to an array. The coordinates of the apex points are calculated using Equations 3.8-3.10.

$$x\,coordinate = (xID + 0.5) * (\frac{Root\,cell\,edge\,length}{2^{Refinement\,Level-1}}) + (Root\,cell\,center_x - \frac{Root\,cell\,edge\,length}{2}) \tag{3.8}$$

$$y\,coordinate = (yID + 0.5) * (\frac{Root\,cell\,edge\,length}{2^{Refinement\,Level-1}}) + (Root\,cell\,center_y - \frac{Root\,cell\,edge\,length}{2}) \tag{3.9}$$

$$z\,coordinate = (zID + 0.5) * (\frac{Root\,cell\,edge\,length}{2^{Refinement\,Level-1}}) + (Root\,cell\,center_z - \frac{Root\,cell\,edge\,length}{2}) \tag{3.10}$$

The coordinates are written in the columns of array for x, y and z coordinates respectively. The coordinates of the apex points are written after the node coordinates in the points file.

3.2.5.2 face, owner and neighbor File

The faces file contains face vertexes which make up the faces. Firstly, `INCELL-FACE` faces are written in the file. Later, `FARFIELD-FACE`, `PYRAMID-BASE-FACE` and triangle faces of the pyramids are written.

`INCELL-FACE` faces First group of the faces file is written x-y and z-directions successively. These functions are recursive functions and visit all of the faces starting from the root face. The function "`writeIncellFacesX()`" is explained below,

since the other functions are the same function with a difference of root face. The first function starts search for the faces from rootFaceX, the other functions starts from rootFaceY and rootFaceZ, respectively. In the writer function, starting from root face, all the faces are visited. When a face with `INCELL-FACE` status is found, the owner cell and the neighbor cell is accessed with the known relations of face and cell. For rootFaceX, the relations of the owner cell and neighbor cell is given in Table 3.4.

Table 3.4: Face, Owner Cell and Neighbor Cell Relations

face	x-ID	y-ID	z-ID	Refinement Level
owner cell	x-ID	y-ID	z-ID	Refinement Level - 1
neighbor cell	x-ID - 1	y-ID	z-ID	Refinement Level - 1

As seen from Table 3.4, x-ID, y-ID and z-ID of the face and the owner cell is same, but the refinement level of the cell is one less of the face's. The neighbor cell has one less of the x-ID of the owner cell. The schematic of the the face, owner cell and neighbor cell relation is shown in Figure 3.4.

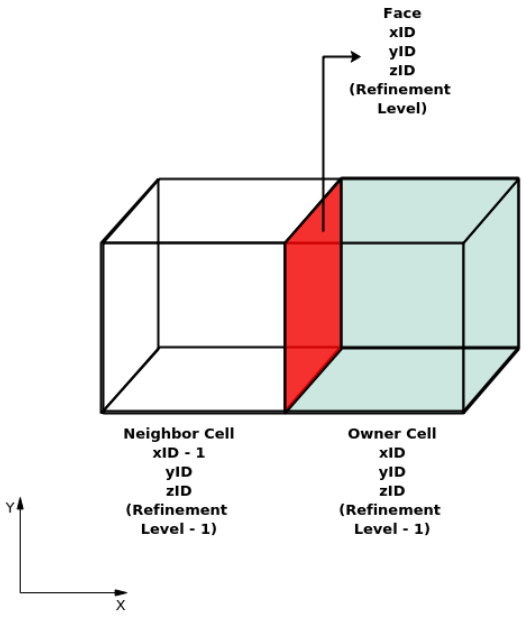


Figure 3.13: Face, Owner Cell and Neighbor Cell Relations

For the faces in x direction, if neighbor cell ID is larger than owner cell ID, the writing order of the nodes to the faces file for the face is given in Table 3.5. If neighbor cell ID is less than the owner cell ID, the writing order of the nodes to the faces file for the face is given in Table 3.5 and Table 3.6 for the faces in x direction. Tables for the faces in y and z direction is given in Appendix B.

Table 3.5: Writing Order of Nodes for Neighbor Cell ID > Owner Cell ID (X Direction)

rootFaceX	face	x-ID	y-ID	z-ID	Refinement Level
rootNode	node	x-ID	y-ID	z-ID	Refinement Level
rootNode	node	x-ID	y-ID	z-ID + 1	Refinement Level
rootNode	node	x-ID	y-ID + 1	z-ID + 1	Refinement Level
rootNode	node	x-ID	y-ID + 1	z-ID	Refinement Level

Table 3.6: Writing Order of Nodes for Owner Cell ID > Neighbor Cell ID (X Direction)

rootFaceX	face	x-ID	y-ID	z-ID	Refinement Level
rootNode	node	x-ID	y-ID	z-ID	Refinement Level
rootNode	node	x-ID	y-ID	z-ID + 1	Refinement Level
rootNode	node	x-ID	y-ID + 1	z-ID + 1	Refinement Level
rootNode	node	x-ID	y-ID + 1	z-ID	Refinement Level

After writing the faces with their nodes in the faces file, owner cell IDs to the owner file and neighbor cell ID to the neighbor file the counting of the face, owner and neighbor is increased once. By this way, when the functions complete, the number of faces are known besides the number of owner and neighbor inputs in the files.

`FARFIELD-FACE` faces The second group of the faces are the `FARFIELD-FACE`. The `FARFIELD-FACE` is written with three different functions starting from root face at x, y and z directions separately. The functions start from root face and search for the face with status of `FARFIELD-FACE`, recursively. If the face has not children

faces and the status of the face is FARFIELD−FACE, the neighbor cell of the face is found from the relation between face and the cell. The relation between the face and the cell is given in Table 3.4. The cell in the negative x direction of the face has x-ID -1 according to the face is neighbor cell. If the neighbor cell does not exist in the given position, the cell in the positive x direction is saved as the owner cell of the face and the cell ID is written in owner file. Since the neighbor cell does not exist, the face is at the boundary of the computational domain, "-1" is written in neighbor file for that face. The face is written in according to the order given in Table 3.7.

The cell in the positive x direction of the face has x-ID according to the face is neighbor cell. If the neighbor cell does not exist in the given position, the cell in the negative x direction is saved as the owner cell of the face and the cell ID is written in owner file. "-1" is written in neighbor file for that face. The face is written in according to the order given in Table 3.8.

Table 3.7: Writing Order of Nodes (+x direction)

rootFaceX	face	x-ID	y-ID	z-ID	Refinement Level
rootNode	node	x-ID	y-ID + 1	z-ID	Refinement Level
rootNode	node	x-ID	y-ID + 1	z-ID	Refinement Level
rootNode	node	x-ID	y-ID	z-ID + 1	Refinement Level
rootNode	node	x-ID	y-ID	z-ID + 1	Refinement Level

Table 3.8: Writing Order of Nodes (-x direction)

rootFaceX	face	x-ID	y-ID	z-ID	Refinement Level
rootNode	node	x-ID	y-ID + 1	z-ID + 1	Refinement Level
rootNode	node	x-ID	y-ID + 1	z-ID	Refinement Level
rootNode	node	x-ID	y-ID	z-ID	Refinement Level
rootNode	node	x-ID	y-ID	z-ID + 1	Refinement Level

For the faces in y and z directions, the procedure for searching neighbor cell and the owner cell is same. The writing order of the nodes to the faces file is given in Tables

in Appendix C. After writing the faces with their nodes in the faces file, owner cell IDs to the owner file and neighbor cell ID to the neighbor file the counting of the face, owner and neighbor is increased once. By this way, when the functions complete, the number of faces are known besides the number of owner and neighbor inputs in the files.

`PYRAMID-BASE-FACE` faces The third group of faces to write into faces file is the faces with `PYRAMID-BASE-FACE` status. Faces are written in x, y and z directions successivelywith recursive functions that visits all faces. If the faces has no children cells and the face status is `PYRAMID-BASE-FACE`, it can be concluded that there is a pyramid on that face and the pyramid is defined with x-ID, y-ID, z-ID and refinement level of the face. The relation between the face and the pyramid on the face is given with Table 3.9.

Table 3.9: Relation Between Face and Pyramid on the Face

face	rootFace	x-ID	y-ID	z-ID	Refinement Level
pyramid	rootPyramid	x-ID	y-ID	z-ID	Refinement Level

Since the ID of the pyramids are given after the hexahedral cell ID, the ID of an pyramid is always bigger than the ID of a hexahedral cell. Therefore, the owner cell is always the pyramid cell. After getting the pyramid cell, the ID of the pyramid cell is known and written in owner file for that face and the counting of the owner element is increased once. Then, the neighbor cell is get by the relations given in Table 3.10.

Table 3.10: Relations Between Face and Neighbor Cells

face	rootFaceX	x-ID	y-ID	z-ID	Refinement Level
positive direction neighbor cell	rootCell	x-ID	y-ID	z-ID	Refinement Level - 1
negative direction neighbor cell	rootCell	x-ID - 1	y-ID	z-ID	Refinement Level - 1
face	rootFaceY	x-ID	y-ID	z-ID	Refinement Level
positive direction neighbor cell	rootCell	x-ID	y-ID	z-ID	Refinement Level - 1
negative direction neighbor cell	rootCell	x-ID	y-ID - 1	z-ID	Refinement Level - 1
face	rootFaceZ	x-ID	y-ID	z-ID	Refinement Level
positive direction neighbor cell	rootCell	x-ID	y-ID	z-ID	Refinement Level - 1
negative direction neighbor cell	rootCell	x-ID	y-ID	z-ID - 1	Refinement Level - 1

If the neighbor cell in positive or negative direction exists and the status of the cell is `INCELL-INNER` ,the neighbor cell ID is the cell ID of the neighbor cell. The neighbor cell ID is written to the neighbor file and the count for the neighbor is increased once. Since the normal of the face must be towards to the pyramid cell, the face nodes are written to the faces file in an order. The order is given in Table 3.11 and 3.12 for

x direction. For other directions, the tables are given in Appendix D.

Table 3.11: Relations Between Face and Nodes (+x direction)

face	rootFaceX	x-ID	y-ID	z-ID	Refinement Level
node	rootNode	x-ID + 1	y-ID	z-ID	Refinement Level
node	rootNode	x-ID + 1	y-ID	z-ID + 1	Refinement Level
node	rootNode	x-ID + 1	y-ID + 1	z-ID + 1	Refinement Level
node	rootNode	x-ID + 1	y-ID + 1	z-ID	Refinement Level

Table 3.12: Relations Between Face and Nodes (-x direction)

face	rootFaceX	x-ID	y-ID	z-ID	Refinement Level
node	rootNode	x-ID	y-ID + 1	z-ID	Refinement Level
node	rootNode	x-ID	y-ID + 1	z-ID	Refinement Level
node	rootNode	x-ID	y-ID	z-ID + 1	Refinement Level
node	rootNode	x-ID	y-ID	z-ID + 1	Refinement Level

Triangle Faces of Pyramids Triangle faces of the pyramids are written with the function "`writeTriangleFace()`" which is also a recursive function to visit all of the faces in the mesh. The cells are checked if the cell has children cells. If a cell does not have children cells and the status of the cell is `CUTCELL-OUTER`, the possible pyramid locations are found and defined. The possible pyramid relations to the cell is given in Table 3.13 in all directions.

Table 3.13: Relations of Cell and Pyramids

cell	rootCell	x-ID	y-ID	z-ID	Refinement Level
pyramid 1	rootPyramidX	x-ID	y-ID	z-ID	Refinement Level - 1
pyramid 2	rootPyramidX	x-ID + 1	y-ID	z-ID	Refinement Level - 1
pyramid 3	rootPyramidY	x-ID	y-ID	z-ID	Refinement Level - 1
pyramid 4	rootPyramidY	x-ID	y-ID + 1	z-ID	Refinement Level - 1
pyramid 5	rootPyramidZ	x-ID	y-ID	z-ID	Refinement Level - 1
pyramid 6	rootPyramidZ	x-ID	y-ID	z-ID + 1	Refinement Level - 1

Each triangle face is related to an edge of the cell which hosts the pyramids. Each edge is checked if there is pyramid located in the normal direction of the triangle face. If there are pyramids both sides of the triangle face, it is not written in the faces file. If there is only one pyramid in either direction of the triangular face, the face is written in the faces file with the normal direction towards to the tetrahedral cell. Since tetrahedral cells get ID later than the pyramids cells, the ID of the tetrahedral cells are bigger than the pyramid cells. The check order for the edges are given with Figure 3.14.

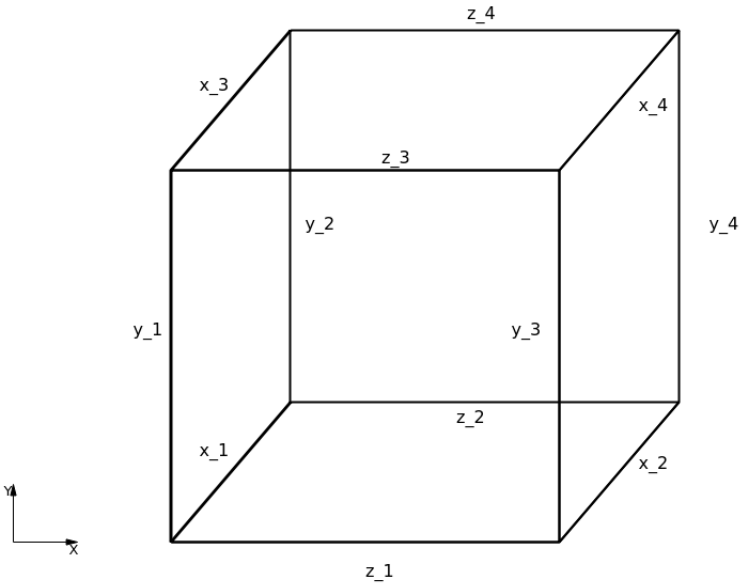


Figure 3.14: Edges of the Cell for Triangular Faces

Firstly, the x_1 edge is checked. According to Table 3.13, pyramid 1 and pyramid 5 is checked. If pyramid 1 exists and status of pyramid 1 is `PYRAMIDCELL`, and if pyramid 5 exists and the status is `PYRAMIDCELL`, the triangular face is not written in the file. However, If pyramid 1 exists and status of pyramid 1 is `PYRAMIDCELL`, and if pyramid 5 does not exist, the triangular face is written in faces file. The node order is given in Appendix E. The owner cell ID comes from the ID of pyramid 1. For the other edges and triangular faces the node order is given in Appendix D.

3.2.5.3 boundary File

For boundary file, the number of boundary faces is required. Since there is two boundary patches in the mesh, there are two patches written as far-field and wall in the boundary file. Moreover, the start number of the boundary faces are required in the file. The faces are written with the order, the first group is the faces with status as `INCELL-FACE`, the second group is the faces with status as `FARFIELD-FACE`s, the third group is the faces with status as `PYRAMID-BASE-FACE` and the final group is the triangle faces of the pyramid volume elements. The start number of the far-field boundary faces can be known.

CHAPTER 4

TETRAHEDRAL MESH GENERATION

4.1 Tetrahedral Mesh Generator

After generating the Cartesian mesh, pyramid volume elements and boundary layer wedge mesh, there is a gap between the pyramid volume elements and the boundary layer mesh. The Cartesian mesh has pyramid elements at the inner boundary and pyramid volume elements have triangle faces as shown in Figure 4.1. It is guaranteed in the Cartesian mesh generation that, no quad meshes are at the interface between Cartesian Mesh and tetrahedral mesh. The pyramid volume elements provides triangle face transition to the tetrahedral mesh. The pyramid volume elements are generated on a single refinement of Cartesian mesh , i.e. the size of the Cartesian volume elements are of same size on the inner boundary of the Cartesian mesh. The multi-level Cartesian mesh can be implied to the Cartesian mesh generation. Therefore, the pyramid volume elements with different sizes can be generated.

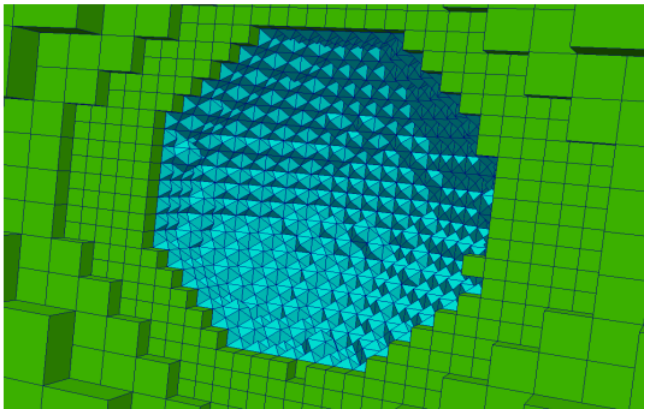


Figure 4.1: Cartesian Mesh with Pyramid Volume Elements

Also, in Figure 4.2, the boundary layer mesh is shown with Cartesian mesh and pyramid volume elements. Notice that, The boundary layer mesh is unusually thick. This allows the easy visualization of the boundary layer mesh.

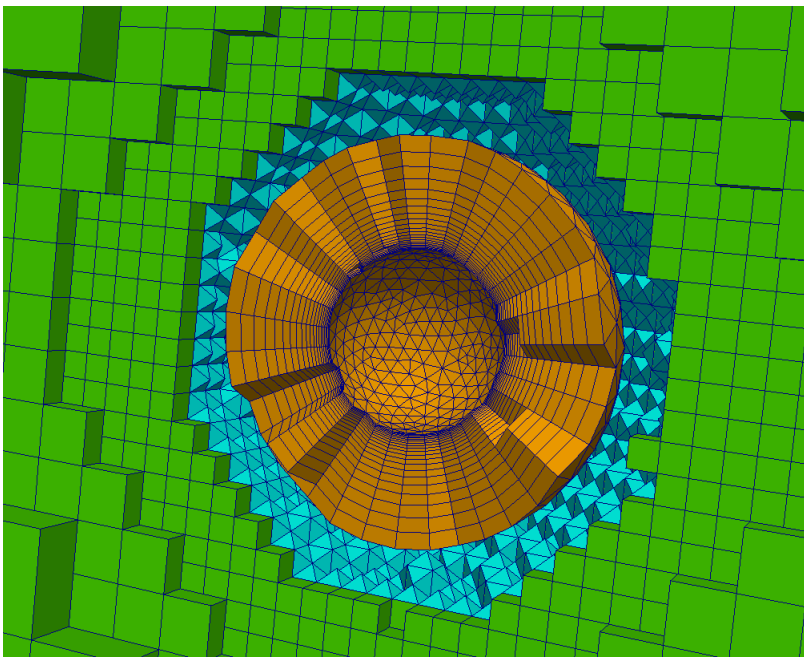


Figure 4.2: Tetrahedral Volume Mesh Gap

As it seen in the Figure 4.2, there exists a gap between the the pyramids extension of the Cartesian mesh and boundary layer mesh. This void is to be filled with Tetrahedral volume mesh by connecting the triangular surface elements of the pyramid volume mesh and the envelope triangles of the boundary layer mesh. Tetrahedral volume elements are generated using the open source mesh generator TetGen. Generated tetrahedral mesh is shown in Figure 5.10.

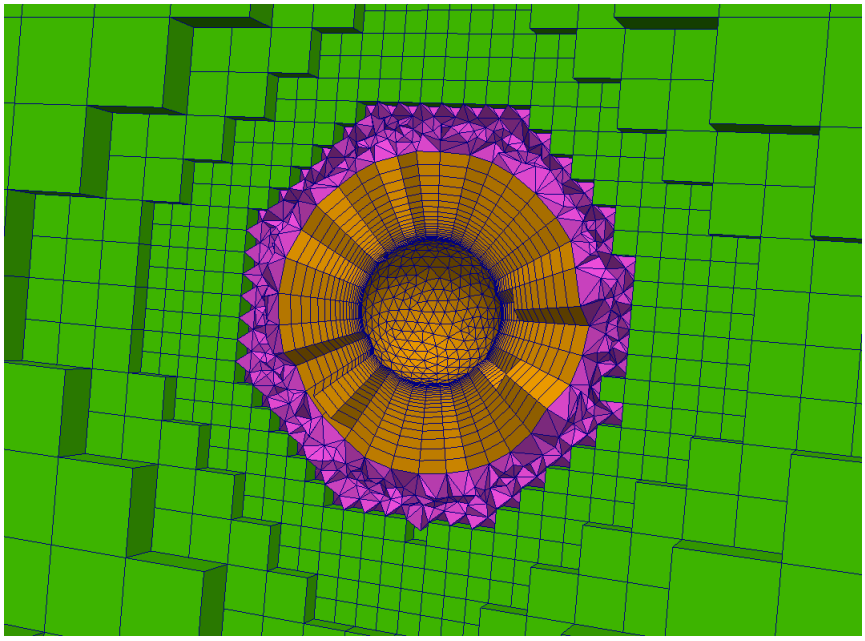


Figure 4.3: Tetrahedral Mesh

The open source mesh generator TetGen is an open source tool that generates tetrahedral mesh in three dimensional domain. [38] The tetrahedralization is applied by using an algorithm called Dealunay tetrahedralization. TetGen can generate three type of tetrahedral mesh depending on the surface geometry introduced to the program. The types are constrained tetrahedralization, conforming tetrahedralization and quality mesh. The code of TetGen is written in C++. The code can be compiled in another program or it can be compiled into executable program. [38]

4.1.1 Delaunay Triangulation

Tetrahedral mesh generator is based on an extended verion of Delaunay Triangulation.Delaunay inroduced a criteria for triangulation in 1934 and his method is a very useful foundation to triangulate a set of vertices.[38] Delaunay triangulation constructs triangles using a set of points in Euclidean plane. Let there is a set of points in given space. Assume that the number of distinct points is more than three, and the points are not collinear. From this set of points many sets of triangles can be formed.A triangle set conforms the of Delaunay triangulation criteriaif the circumcircle of the

triangle does not contain any point of another triangle.

A Voronoi diagram is complementarywith Delaunay trianglation. Voronoi diagram is constructed such that points in each voronoi cell is closest the the associated vertex. Vertices can be thought as the nuclei of the Voronoi cell. A Delaunay triangulartion is the dual graph of Voronoi graph. Edges of the triangulation expand normally from the nuclei and form Voronoi tessellation on the entire plane. The dual graph is constructed such that The circumcenters of Delaunay triangles are the nuclei of the Voronoi diagram. The Voronoi and Delaunay triangulation is shown in Figure 4.4. In this figure, the dashed lines show Delaunay Triangulation and the solid lines show Voronoi Diagram.[40]

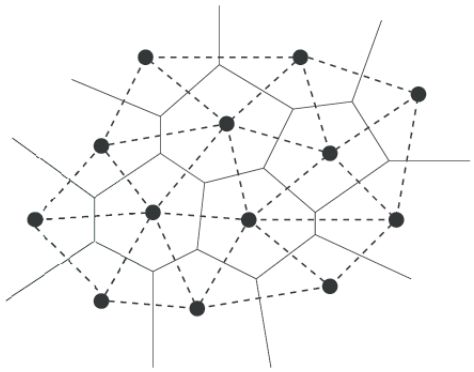


Figure 4.4: Voronoi Diagram and Delaunay Triangulation

4.2 Implementation of the TetGen into Mesh Generation

An input file is provided automatically to TetGen created in the Cartesian Mesh Generator. This input file consists of the triangular faces of envelope layer of boundary layer mesh and the pyramids. These faces make up the boundaries of tetrahedral mesh.

The TetGen is used as stand-alone program and takes mesh input from Cartesian mesh generator and gives tetrahedral mesh output files. TetGen is compiled by using command switches. The command switches, which are used, are given in Appendix F.

The outputs of TetGen are consist of two files that contains node coordinates (.node file) and face vertexes (.face file). Firstly, a face information file is created by using a TetGen switch command. In this face file, only the boundary faces of the tetrahedral mesh is written. Hence, the boundary faces are transfered to the file merger program with a known order. Then, the inner tetrahedral faces file is generated. The node coordinates file is also created along with the file of the tetrahedral connectivity. The nodes are ordered according to their status of being the boundary node or inner node. Finally, all tetrahedral mesh related files are transfered through TetGenHandler program written in C++ and TetNodes and TetFaces text files are generated for the final mesh generator code.

4.3 Exporting to Final Mesh

All of different mesh zones should be put together to form the final mesh. The final mesh is created with a code which combines the output of the the boundary layer mesh generator, Cartesian Mesh generator and tetrahedral mesh generator. The final mesh is written with polyMesh file format. The detailed information about polyMesh files is given Appendix A.

The points and faces files with owner and neighbor relations are given to the final mesh generator code as input. The code combines all the points, faces, owner and neighbor files into final polyMesh files.

The final mesh is generated by using polyMesh files. The solutions are obtained by using Fluent. Therefore, polyMesh format is converted to .msh file format by using the “foamMeshToFluent” command of OpenFoam.

CHAPTER 5

RESULTS AND VALIDATION OF MESH GENERATOR

In this chapter, three validation cases are presented including a sphere test case, a missile test case aimed to study vortex interactions and basic finner test case, which is a standard test case in missile aerodynamics. All three test cases are external flow test cases and air is the working fluid.

5.1 Test Geometries

The first test case of the mesh generator is a sphere. The mesh generated for the sphere geometry which is shown in Figure 5.1. Test geometry has 6" radius. Number of surface triangles are relatively low with 6676 faces.

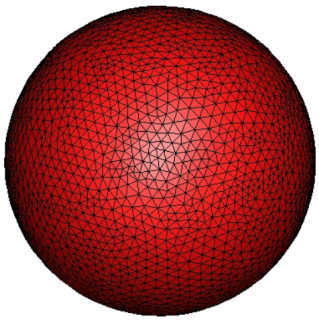


Figure 5.1: Surface Mesh - Test Case 1

The second test case is a missile geometry which is used in the Missile Facet of NATO STO AVT 316 (Vortex Interaction Effects Relevant to Military Air Vehicle Performance). [41] The geometry and the surface mesh is shown in Figure 5.2. The

surface geometry consist of 208684 faces. Number of division at the fin leading and trailing edges is kept as 2 triangle. The test case aimed for vortex interaction between the strake and the fins. Therefore the number of division at the strake, wind tips are rather small for this test case.

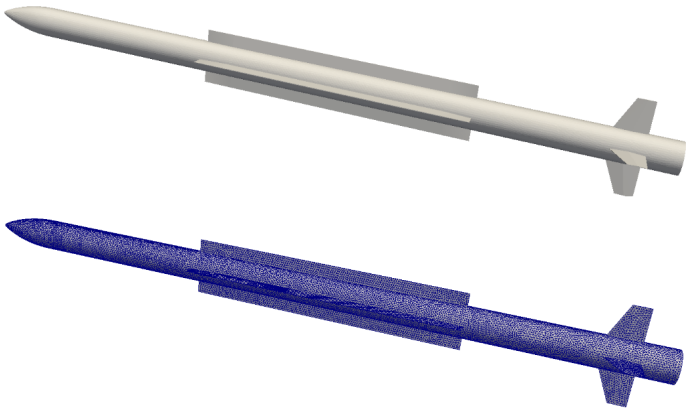


Figure 5.2: Geometry and Surface Mesh - Test Case 2

The third test geometry is Basic Finner geometry. Basic finner geometry is a well known test case, which is initially generated for determination of dynamic stability and damping coefficients. Number of experimental data is available in the literature. In this study only static coefficients are used as validation. The surface mesh of the geometry is given in Figure 5.3. [42] The surface mesh contains 130898 number of cells.

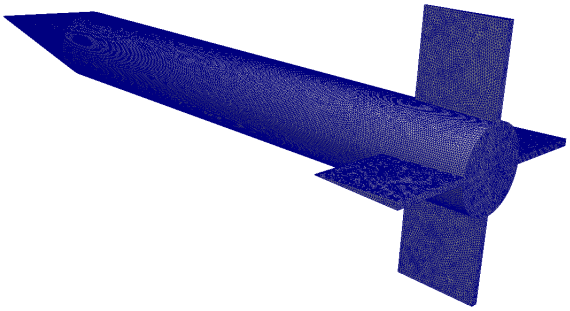


Figure 5.3: Surface Mesh of Basic Finner

5.2 Mesh Metrics

The meshes of the test cases are given in this part. The mesh quality of the test cases are evaluated by using `checkMesh` tool of the OpenFOAM. Skewness is the measure of the distance between the intersection of the line connecting two cell centers with their common face and the center of that face. The skewness is considered as better when it is closer to zero. [43]The allowable maximum skewness in OpenFOAM skewness calculation methodology is reported as 20. [44] Non-orthogonality is the angle made by the vector between the two adjacent cell centers across the common face and the face normal. For a high quality mesh, this value should be as small as possible. [45]

5.3 Mesh Generation Results

Mesh generation procedure is completed for the given meshes and mesh quality and meshing time is reported. The required time for generation of all of the meshes are less than total time of 2 minutes.

5.3.1 Test Case 1

For the sphere test case, the boundary layer initial height is given as $2x10^{-5}\ m$ and the number of layers is 20. The boundary layer mesh is given in Figure 5.4, enclosing Cartesian mesh of Test Case 1 is shown in Figure 5.5. The final mesh is shown in Figure 5.6.

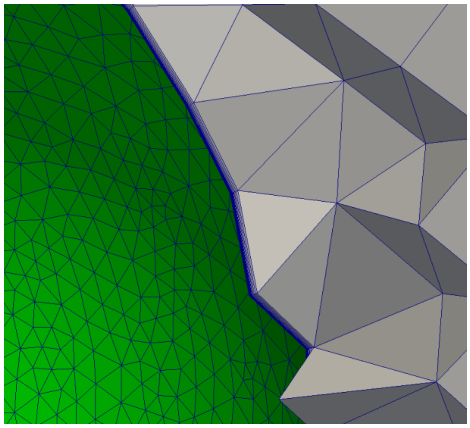


Figure 5.4: Boundary Layer - Test Case 1

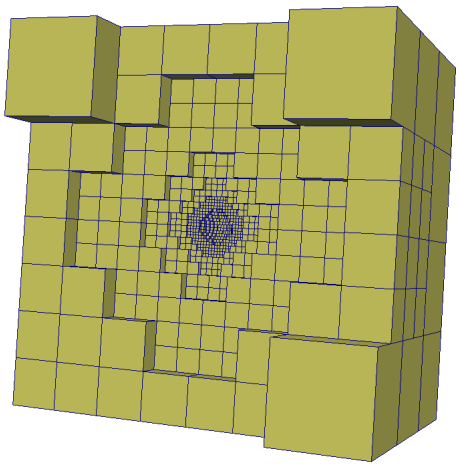


Figure 5.5: Cartesian Mesh - Test Case 1

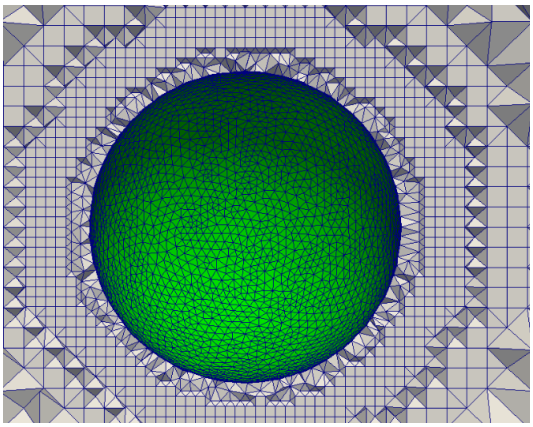


Figure 5.6: Final Mesh - Test Case 1

The mesh statistics taken from "checkMesh" utility of OpenFoam is shown in Table 5.1. As seen from here all mesh metrics are very good. The minimum cell volume and aspect ratio is dictated by the first layer thickness and most modern CFD solvers can easily handle them.

Table 5.1: Mesh Properties from OpenFoam checkMesh - Test Case 1

Points	18926
Faces	100056
Cells	41928
Maximum Skewness	1.40892
Minimum Cell Volume	$1.23x10^{-7}m^2$
Maximum Aspect Ratio	2606.08
Maximum Non-Orthogonality	89.0298^o

5.3.2 Test Case 2

The boundary layer mesh is generated with $4x10^{-5}\,m$ initial height at the first layer and 20 layers. The boundary layer is shown in Figure 5.7 with a cut through the geometry. The geometry involve difficulties for the boundary layer mesh generator, especially around fin tips and fin-body connections. It is expected that at the corners of the fins and the wings, the boundary layer mesh requires boundary layer thickness handling as explained in Chapter 2. The close up figure of the boundary layer is given in Figure 5.8. As seen from 5.8a, the boundary layer is thinned at the corner which is the intersection of the body and the fin. Therefore, this case is a good case for seeing difficulties of meshing with corners.

The Cartesian mesh of the geometry is given in Figure 5.9.

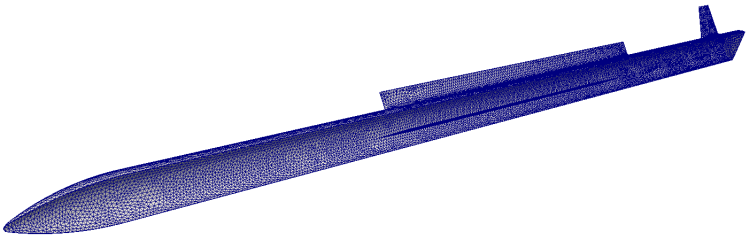
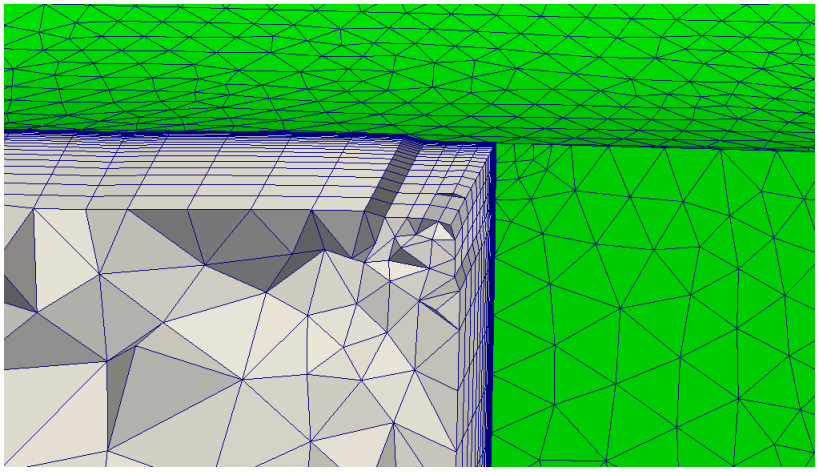
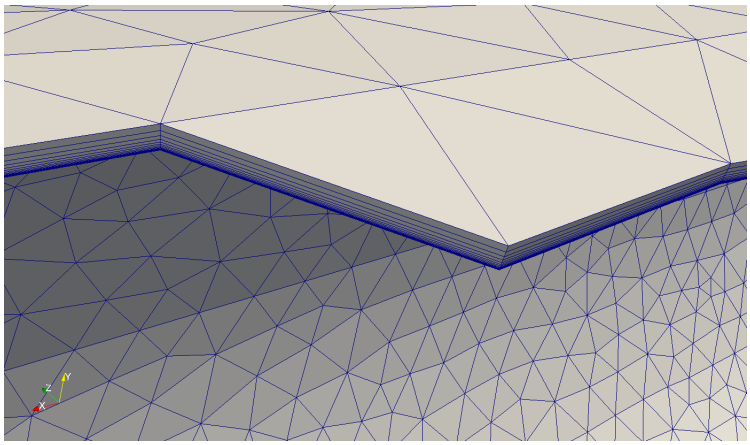


Figure 5.7: Boundary Layer Mesh - Test Case 2

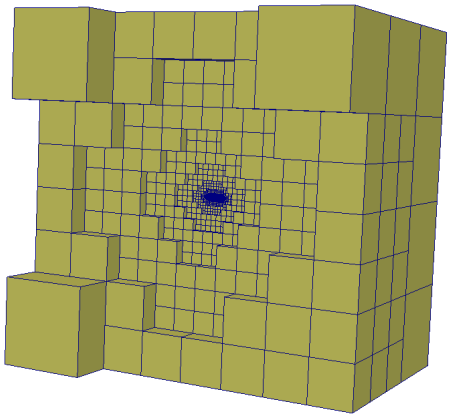


(a) Boundary Layer at Corner

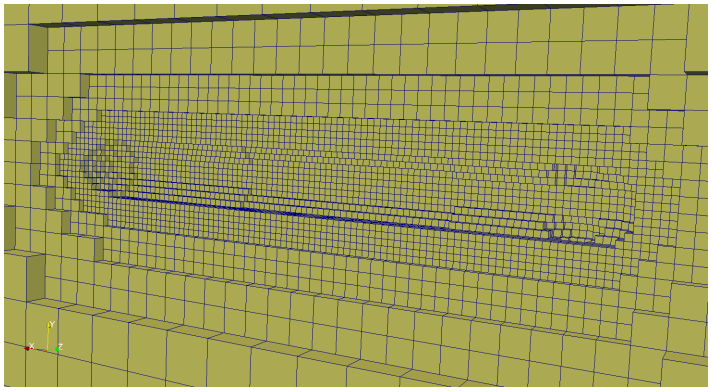


(b) Boundary Layer at Body

Figure 5.8: Close Figure of Boundary Layer - Test Case 2



(a) Cartesian Mesh - Test Case 2



(b) Close up of Cartesian Mesh - Test Case 2

Figure 5.9: Cartesian Mesh - Test Case 2

The tetrahedral mesh is given in Figure 5.10 separately. The tetrahedral mesh is shown with a cut through the rocket body.

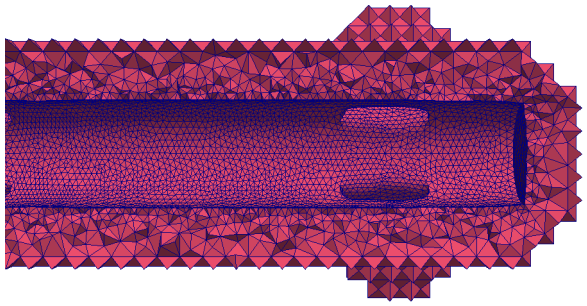


Figure 5.10: Tetrahedral Mesh - Test Case 2

The final mesh which has all type of meshes is shown in Figure 5.11. The cut is taken to see the meshes thorough the geometry body.

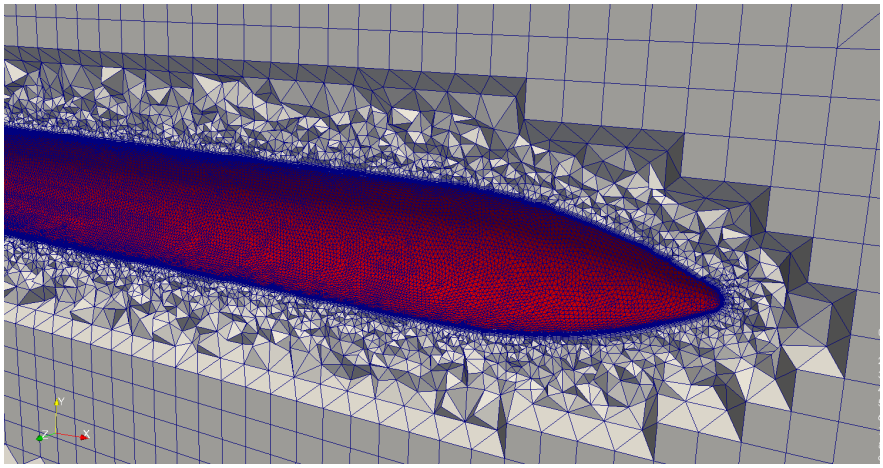


Figure 5.11: Final Mesh - Test Case 2

The mesh statistics taken from "checkMesh" utility of OpenFoam is shown in Table 5.2. Again, despite the diffcult-to-mesh zones at especially fin tips and fin to body connection zones, mesh metrics are acceptable. It should be noted that, this is a very common situation for all boundary layer generators. Mesh generators provide skew meshes where the unit normals changes abruptly. The developed algorithm provides a decent solution at those locations.

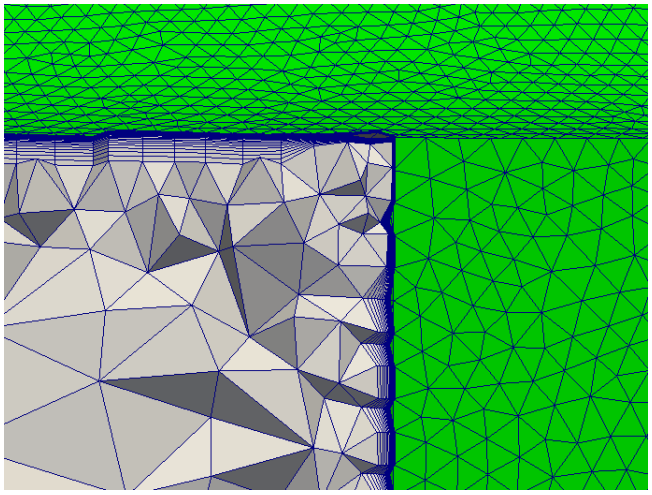
Table 5.2: Mesh Properties from OpenFoam checkMesh - Test Case 2

Points	2443229
Faces	13985157
Cells	5877446
Maximum Skewness	2.97672
Minimum Cell Volume	$4.85x10^{-12}m^2$
Maximum Aspect Ratio	93.8459
Maximum Non-orthogonality	78.064^o

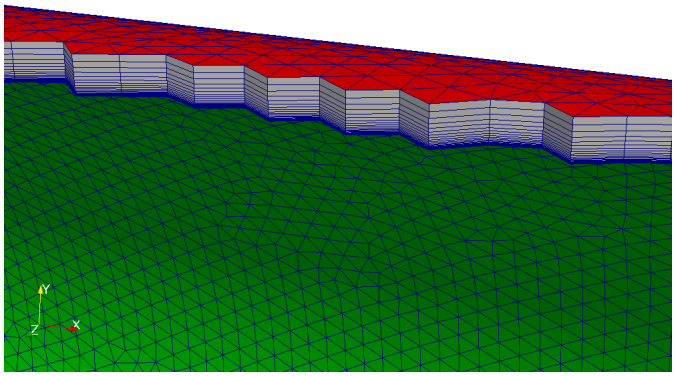
5.3.3 Test Case 3

5.3.3.1 Mesh with Developed Mesh Generator

The boundary layer mesh with 20 layers and initial height of $1x10^{-4}$ m is given in Figure 5.12. As in the Test Case 2, the corners where the wings and body coincide, the boundary layer is thinned not to have negative volume elements.



(a) Boundary Layer Mesh at Corner



(b) Boundary Layer Mesh at Body

Figure 5.12: Boundary Layer Mesh of Basic Finner Geometry - Test Case 3

The Cartesian mesh is of the 3D volume mesh generated by the developed mesh

generator is shown in Figure 5.13 with a cut along the geometry. The close figure of the Cartesian mesh at the vicinity of the boundary layer is shown in Figure 5.14.

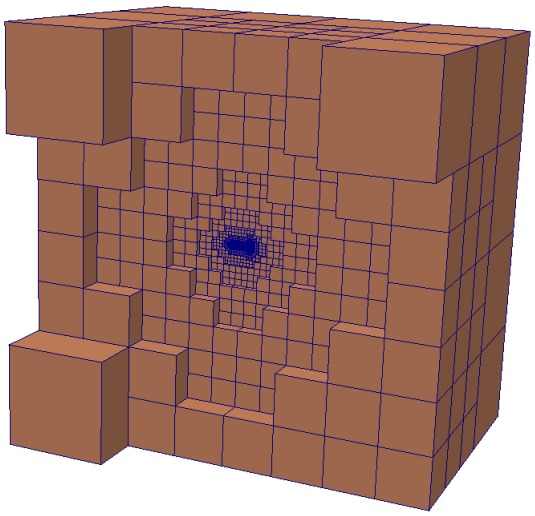


Figure 5.13: Cartesian Mesh of Basic Finner Geometry - Test Case 3

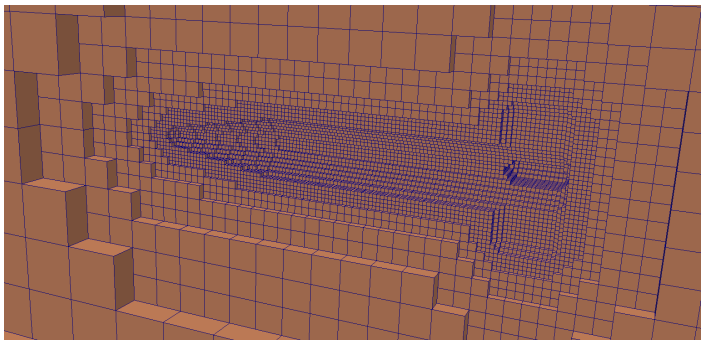


Figure 5.14: Close up Figure of Cartesian Mesh - Test Case 3

The tetrahedral mesh is given in Figure 5.15 separately. The tetrahedral mesh is shown with a cut through the geometry.

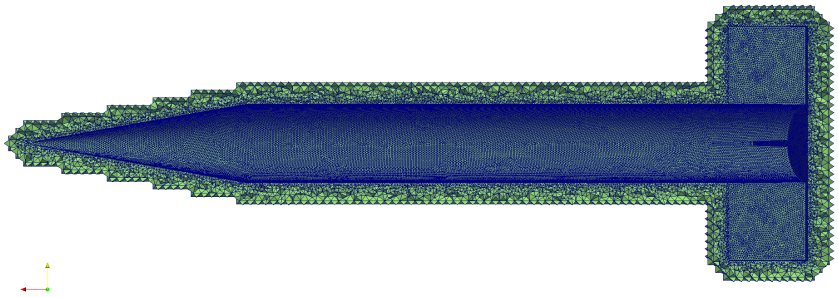


Figure 5.15: Tetrahedral Mesh of Basic Finner - Test Case 3

The final mesh which has all type of meshes is shown in Figure 5.16. The cut is taken to see the meshes thorough the geometry body.

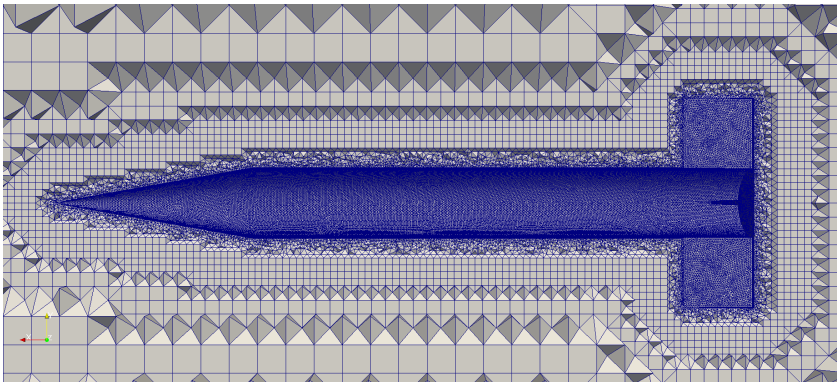


Figure 5.16: Final Mesh of Basic Finner Geometry - Test Case 3

The mesh statistics taken from “checkMesh” utility of OpenFoam is shown in Table 5.3. Again, despite the diffcult-to-mesh zones at especially fin tips and fin to body connection zones, mesh metrics are acceptable. It should be noted that, this is a very common situation for all boundary layer generators. Mesh generators provide skew meshes where the unit normals changes abruptly. The developed algorithm provides a decent solution at those locations.

Table 5.3: Mesh Properties from OpenFoam checkMesh - Test Case 3

Points	1568213
Faces	8386828
Cells	3443324
Maximum Skewness	5.07291
Minimum Cell Volume	$6.57x10^{-11}m^2$
Maximum Aspect Ratio	213.871
Maximum Non-orthogonality	77.6124^o

5.3.3.2 Mesh with ANSYS Mesher

The same geometry is meshed with using ANSYS Mesher in order to compare the meshing and solution quality of the developed mesh generator. ANSYS Mesher mesh is tetrahedral mesh with same size of boundary as in the case of the mesh generated with the developed mesh generator. The farfield of final ANSYS Mesher mesh is shown in Figure 5.17 and the boundary layer is given in Figure 5.18. The close up figure of the ANSYS Mesher is given in Figure 5.19.

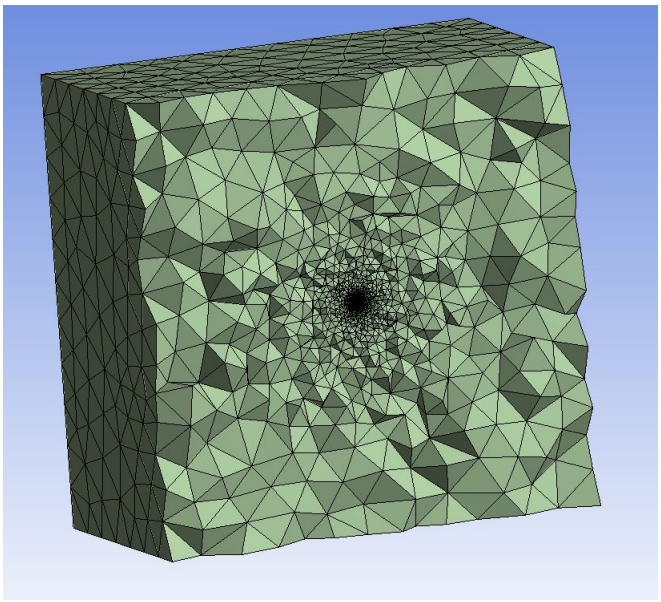


Figure 5.17: Farfield of Final Mesh with ANSYS Mesher - Test Case 3

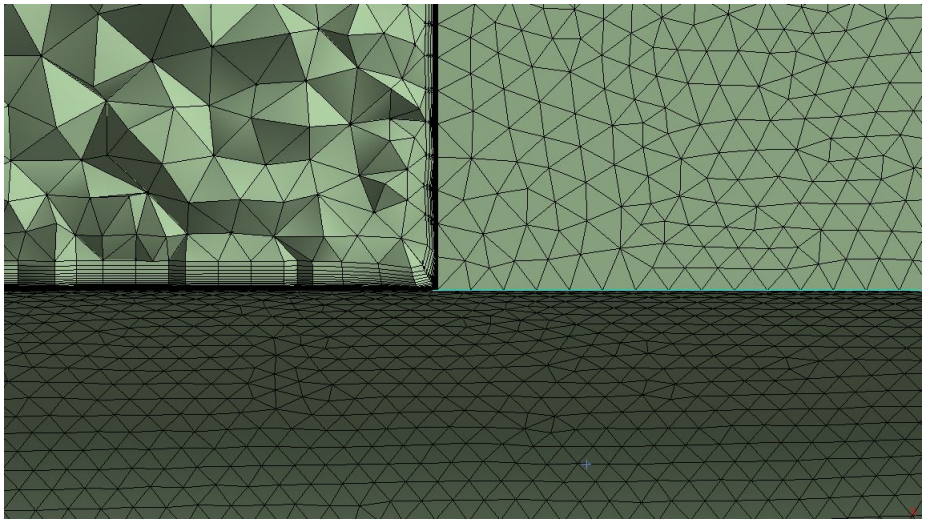


Figure 5.18: Boundary Layer of Final Mesh with ANSYS Mesher - Test Case 3

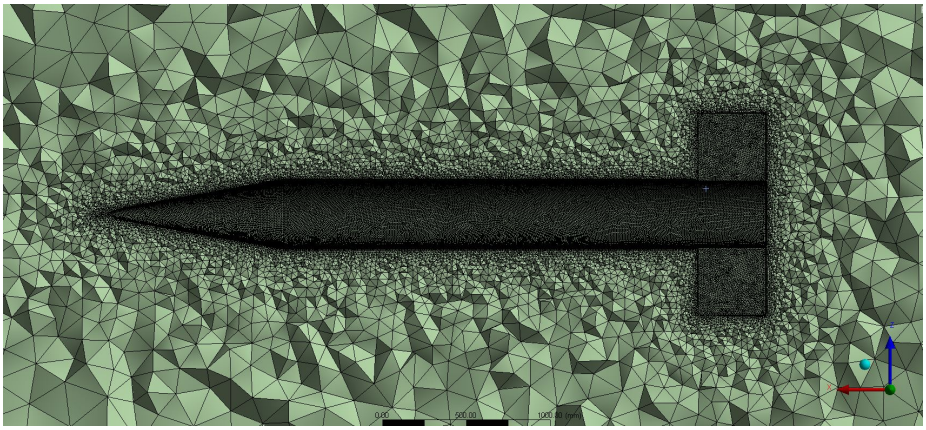


Figure 5.19: Close Up Figure of Final Mesh with ANSYS Mesher - Test Case 3

Mesh file format is exported to be read by OpenFoam. The mesh statistics taken from "checkMesh" utility of OpenFoam for both ANSYS Mesher and Developed Mesh is shown in Table 5.4 for comparison.

Table 5.4: Mesh Properties from OpenFoam checkMesh for ANSYS Mesher Mesh-Test Case 3

	ANSYS Mesher Mesh	Developed Mesh
Points	1732950	1568213
Facets	10543896	8386828
Cells	4530372	3443324
Maximum Skewness	7.63629	5.07291
Minimum Cell Volume	$4.63x10^{-11}$	$6.57x10^{-11}m^2$
Maximum Aspect Ratio	505.85	213.871
Maximum Non-orthogonality	89.2^o	77.6124^o

In Table 5.3, the number of point, face and cell of Developed Mesh are less than ANSYS Mesher Mesh for the same volume of computational domain. It is seen that the maximum skewness and non-orthogonality is less for Developed Mesh. In total, it is easily seen that the Developed Mesh is better than ANSYS Mesher Mesh in mesh metrics. Moreover, Test Case 3 is meshed using ANSYS Mesher in 7 minutes for the same length of farfield, same surface mesh of the geometry and with same computer, whereas the Developed Mesh is generated in 2 minutes. Therefore, the required time for generating the mesh with the developed mesh generator is less than ANSYS Mesher, which is a well-known mesh generator for CFD simulations.

5.4 Simulation Results

All simulations are conducted with FLUENT. The simulation results of Test Cases are presented in this part.

5.4.1 Test Case 1

The turbulence model for the solution is Standard k-epsilon model. The fluid is chosen as air. The boundary conditions are selected as pressure far field and viscous wall.

The pressure-far-field boundary condition is given as static pressure of $101325Pa$ and static temperature of $300K$. The Reynolds number is $4.7x10^6$. The wall boundary condition is no-slip condition viscous wall model. The solution method is selected as Pressure-Velocity Coupling Scheme is coupled and the discretization is set to second order. Initialization is done by hybrid initialization. The solution is obtained for steady-state. Solutions are reported at figrures 5.20 to 5.22.

The static pressure field near the sphere is given in Figure 5.20. Similarly velocity is presented as velocity vector plot from normal z direction is given in Figure 5.21. The y+ contour plot of Test Case 1 is given in Figure 5.22 which shows as y+ values are acceptable for turbulence model.

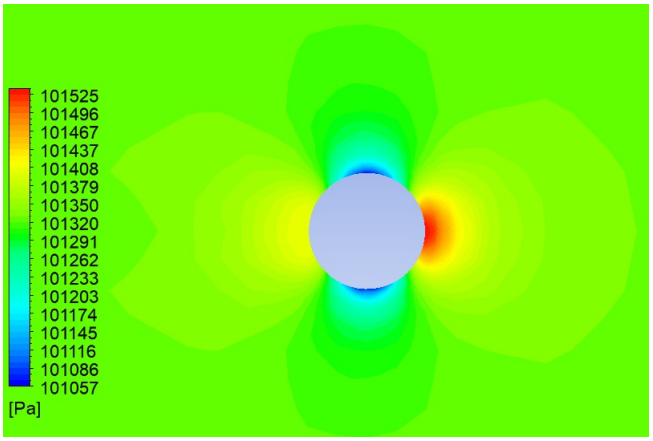


Figure 5.20: Static Pressure Contour - Test Case 1

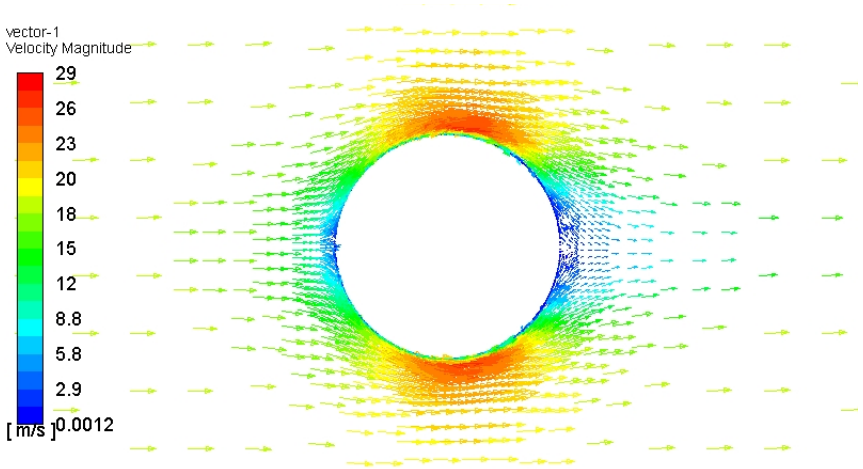


Figure 5.21: Velocity Vector Plot (Normal z Direction) - Test Case 1

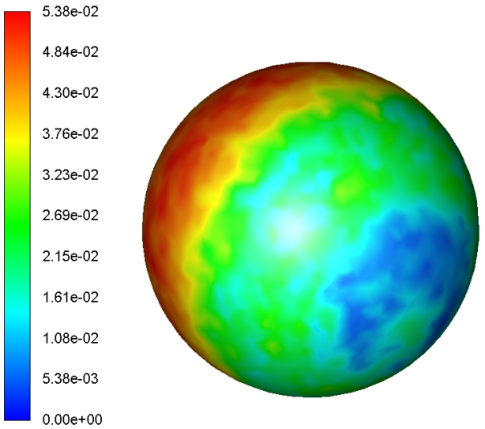


Figure 5.22: y+ Contour Plot of Test Case 1

The pressure coefficient which is calculated with 5.1 on the wall of the sphere with respect to position in terms of degrees is given in Figure 5.23. It is seen that the pressure decreased at the back of the sphere due wakes.

$$Pressure\,Coefficient = \frac{(P - P_{inf})}{(\frac{1}{2}\rho_{inf}V_{inf}^2)} \tag{5.1}$$

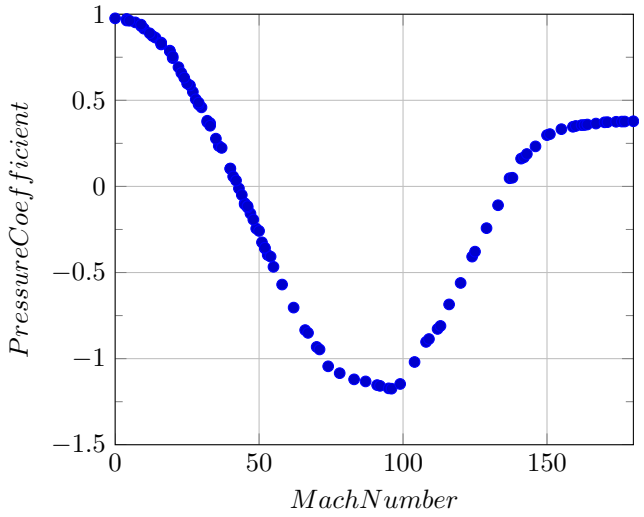


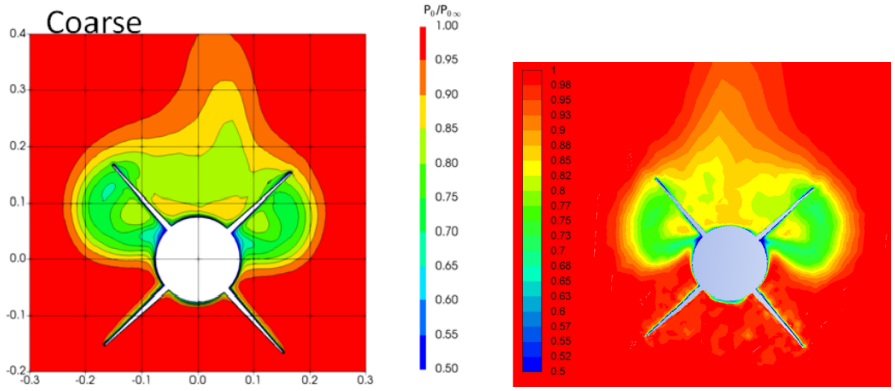
Figure 5.23: Pressure vs Position Plot - Test Case 1

5.4.2 Test Case 2

The turbulence model for the solution is Standard k-epsilon model. The fluid is chosen as air. The boundary conditions are selected as pressure far field and viscous wall. The pressure-far-field boundary condition is given as static pressure of $101325Pa$ and static temperature of $288.15K$. The Reynolds number is $4.89x10^6$. The wall boundary condition is no-slip condition viscous wall model. The solution method is selected as Pressure-Velocity Coupling Scheme is coupled and the discretization is set to second order. The simulation is initialized by using standard initialization from farfield boundary conditions. The solution is obtained for steady-state.

This study is a work in progress in many research group, where some results are published. The aim of the study is to evaluate CFD capabilities for missiles with severe vortex interactions and highly compressible flow. It should be noted that the comparison plots obtained from the NATO study are run with advanced CFD models and very high (more than 100M) cell counts. Again,despite the low cell-count the solver captured most of the vortex structure.

The total pressure ratio contour plot is given from NATO study and with developed mesh in Figure 5.24. [41]



(a) Total Pressure Ratio From NATO Study

(b) Total Pressure Ratio From Developed Mesh

Figure 5.24: Total Pressure Ratio From NATO Study

As can be seen from Figure 5.24, the solutions are similar to each other.

The y+ on wall contour plot is given in Figure 5.25.

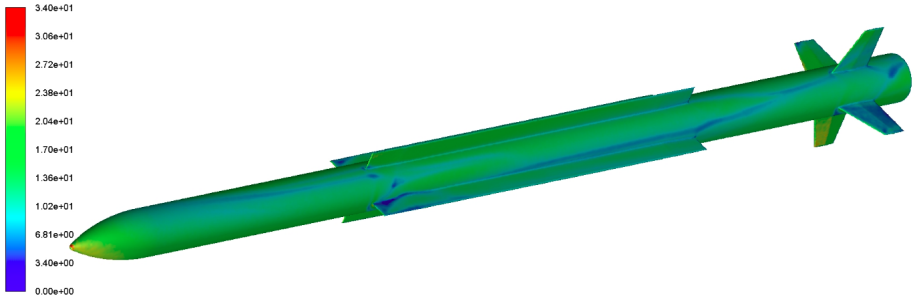


Figure 5.25: y+ on Wall Contour Plot - Test Case 2

5.4.3 Test Case 3

The turbulence model for the solution is Standard k-epsilon model. The fluid is chosen as air. The boundary conditions are selected as pressure far field and viscous wall. The pressure-far-field boundary condition is given as static pressure of $101325Pa$ and static temperature of $288.15K$. The Reynolds number is $4.89x10^6$. The wall boundary condition is no-slip condition viscous wall model. The solution method is selected as Pressure-Velocity Coupling Scheme is coupled and the discretization is set to second order. The simulation is initialized by using standard initialization from farfield boundary conditions. The solution is obtained for steady-state.

The aerodynamic experiments are conducted with this geometry by Dupuis.[46][47] All the experiments are done with the angle of attack is 5^o. Therefore, the simulations are done on the experimental points. The simulations are done with both Developed Mesh and ANSYS Mesher Mesh. The wind tunnel test result of pitch moment coefficients according to Mach number are given in Figure 5.26 for both meshes. As can be seen from Figure 5.26, CFD solution of the developed mesh and the experimental data is very similar to each other at different Mach numbers.

The plot of axial force coefficient with respect to Mach number is given in Figure 5.27 for both meshes. As can be seen, the results are similar to test results at different Mach numbers.

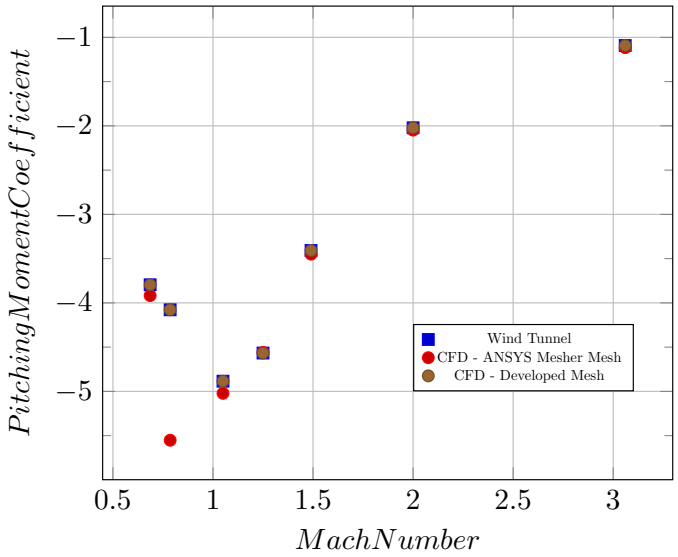


Figure 5.26: Comparison Graph of Experimental and Simulation Results

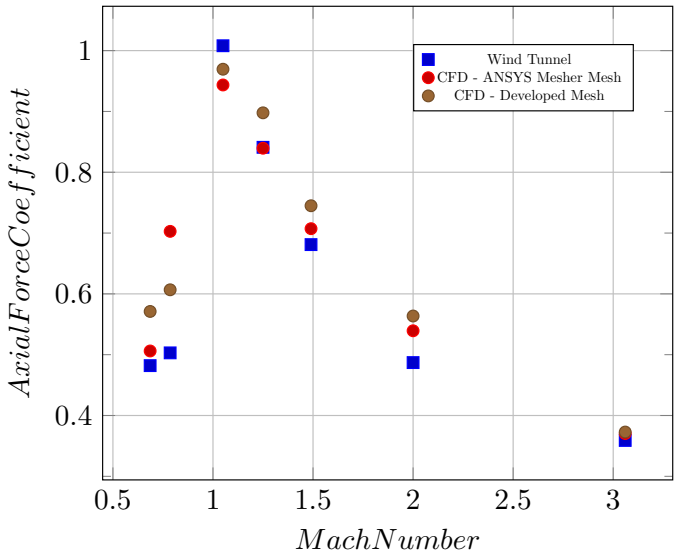
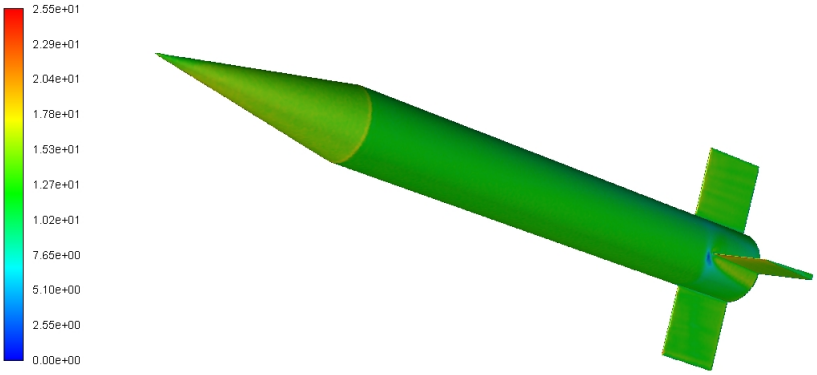
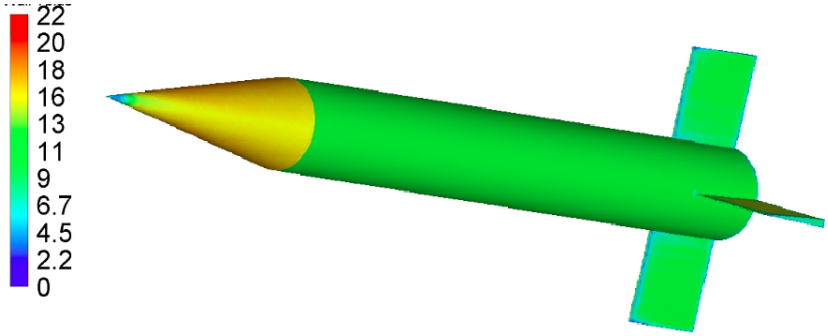


Figure 5.27: Drag Coefficient Plot for Freestream Mach 2.0

The y+ on wall boundary plot for developed mesh and ANSYS Mesher mesh is given in Figure 5.28.



(a) y+ Plot for Developed Mesh



(b) y+ Plot for ANSYS Mesher Mesh

Figure 5.28: y+ Plot on Wall Boundary - Test Case 3

CHAPTER 6

CONCLUSION AND FUTURE WORK

Computational Fluid Dynamics is an analysis tool which is very important for engineers for modeling fluid mechanics problems. Mesh Generation is one the most important step of a CFD problem. The mesh with good quality provides accurate results and rapid convergence. With the aim of getting a high quality mesh in mind, a Cartesian based mesh generator with body-fitted boundary layers is developed and presented in this thesis. The mesh generated with the developed grid generator consists of Cartesian, pyramid, tetrahedral cells as well as wedge boundary layer mesh. Cartesian mesh makes up the core and farfield mesh. A new Cartesian mesh generator is developed for this purpose. As presented in the literature review, the Cartesian mesh is connected to a boundary layer grid to be utilized the viscous flow simulations. For this purpose, first a surface mesh should be provided by the user. Then, the boundary layer mesh is generated by inflating wedge volume elements on the geometries from this initial triangular surface mesh. A size-fitting Cartesian mesh is the generated around this boundary layer mesh zone. It is discussed that, Cartesian-only mesh cannot be used for this purpose since the cells at the vicinity of the boundary layer mesh should be cut into arbitrary shape polyhedra in order to fit fit to the body geometry. As a result, development Cartesian based mesh generator with body-fitted boundary layers which are connected with a buffer mesh consist of tetrahedrons and pyramids is the main drive of this thesis.

Boundary layer is generated by using an open source mesh generator Larosterna SUMO. The open source mesh generator provides boundary layer wedge mesh along with tetrahedral mesh in the farfield. The boundary layer mesh generator is extracted from the overall code of Larosterna SUMO. The required libraries of is taken and

compiled to have the stand-alone boundary layer mesh generator for this study. Customizations are applied on the extracted libraries. The output of the code is adjusted for utilization together with Cartesian mesh generator. By using boundary layer mesh generator, at the vicinity of the geometry, the cells have high orthogonality which is a required feature for RANS models. Cells at the Boundary layer has high aspect ratio cell at the aligned parallel with the flow direction and normal to the velocity gradient.

After boundary layer mesh generation, the Cartesian mesh is generated. Cartesian mesh cells has the highest quality since they consist of only cube volume elements with 90^0 angles between faces, therefore the all quality metrics are the best in a Cartesian mesh. In addition, the motivation for this thesis study is to overcome the difficulties of meshing in three dimensional domain, especially for large geometries. Cartesian mesh is generated by using Octree Data Structure. This data structure is also utilized in the neighbor search algorithms. The searching with the Octree data structure is very fast and easy compared to the other type of search algorithms. In addition, Cartesian mesh has less number of cells compared to other types of meshes in the same computational domain. For example, in same computational domain, number of tetrahedral mesh required for the same accuracy is significantly more than that of Hexahedral\Cartesian mesh, according to literature. Besides, the smoothness is enhanced between the successive cells of the same refinement level by cell refinement diffusion algorithms implemented in Cartesian mesh generator. In order to ensure conformal connectivity between Cartesian and boundary layer mesh a pyramid mesh connected to Cartesian mesh is generated within Cartesian mesh generator code. They are created by using the outermost elements of Cartesian cells that are facing to outer layer of Boundary layer mesh faces. The pyramids in this mesh generator have equilateral triangles as their faces. Pyramid cells have both triangular and square faces which allows transition from square to triangle face. Therefore, the pyramid volume elements are generated along with Cartesian mesh. Tetrahedral mesh is generated by using the open source mesh TetGen. Therefore, the quality of the pyramid volume elements are also very high. The rest of the domain is filled with tetrahedrons using another open source mesh generator TETGEN. It is reported that, overall mesh quality provided by the Cartesian mesh generator is very high. For example, in Chapter 5, the mesh properties of the Developed Mesh and ANSYS Mesher Mesh properties are

compared. It is seen that for the same computational domain size, the cell count of the mesh decreases by using in-house developed mesh generator. Also, the skewness and non-orthogonality values of the developed mesh are better compared to ANSYS Mesher mesh.

This property of Cartesian mesh makes it more attractive for using in a mesh generation. Since, there is less number of cells and Octree algorithm is very efficient in time, the generation of Cartesian mesh is very fast. For example, about a million cell Cartesian mesh is generated approximately in 20 seconds. As given in Chapter 5, the meshing time for the Test Case 3 is only 2 minutes, whereas, the meshing time with ANSYS Mesher for the same surface mesh is 7 minutes.

As a result, a Cartesian based mesh generator with body-fitted boundary layer developed with this thesis stud provides high quality meshes for fluid flow problems with three dimensional large domains. The mesh generated by this developed mesh generator merged into a single zone mesh. The generation is fast, the mesh is body-fitted to the geometry by the boundary layer mesh. The developed mesh generator is focused on external flows, and can be extended to internal flows too. Finally, meshes created by the developed mesh generator is tested in CFD problems, where solutions are obtained by using FLUENT are also given.

The mesh generator developed in this thesis can be extended in a couple of basic areas. The current mesh generator works for the geometries with triangular surface meshes only. One of the possible extensions for the mesh generator is to make the mesh generator compatible with quad or mixed initial surface mesh. Another study can be conducted is to extend the mesh generator for internal flows where current code can only handle external flow domains. Last but not least, the mesh generator code can be upgraded to make solution adaptation.

REFERENCES

[1] J. D. J. Anderson, "Computational Fluid Dynamics: The Basics with Applications," 1995.

[2] H. Lomax, T. H. Pulliam, and D. W. Zingg, "Fundamentals of Computational Fluid Dynamics," p. 249, 1999.

[3] J. F. Thompson, Z. Warsi, and C. W. Mastin, *Numerical Grid Generation Foundations and Applications*. 1985.

[4] J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*. Springer, 3rd ed., 2012.

[5] J. Tu, G.-H. Yeoh, and C. Liu, "CFD Mesh Generation: A Practical Guideline," *Computational Fluid Dynamics*, pp. 125–154, 2018.

[6] H. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics*, vol. M. 2007.

[7] B. André, "Lecture 5 -Solution Methods.".

[8] J. F. THOMPSON, "Grid generation techniques in computational fluid dynamics," *AIAA Journal*, vol. 22, no. 11, pp. 1505–1523, 2008.

[9] Andre Bakker, "Meshing, Applied Computational Fluid Dynamics," *PPT: bakker*, no. 2002, pp. 1–35, 2006.

[10] J. Blazek, *Computational fluid dynamics:Principles and Applications*. 2001.

[11] F. Moukalled, L. Mangani, and M. Darwish, *The Finite Volume Method in Computational Fluid Dynamics - An Advanced Introduction with OpenFOAM and Matlab*, vol. 113. 2016.

[12] D. Sharov and K. Nakahashi, "Hybrid prismatic/tetrahedral grid generation for viscous flow applications," *AIAA Journal*, vol. 36, no. 2, pp. 157–162, 1998.

[13] M. Koike, T. Kojima, D. Sasaki, T. Misaka, K. Shimoyama, S. Obayashi, K. Hirakawa, and N. Tani, "Numerical Simulation of Cascade Flows Using Block-Structured Cartesian Mesh," no. January, pp. 1–19, 2017.

[14] P. G. Tucker and Z. Pan, "A Cartesian cut cell method for incompressible viscous flow," *Applied Mathematical Modelling*, vol. 24, no. 8-9, pp. 591–606, 2000.

[15] T. J. Craft and G. B. Building, "Body-Fitted Grids Non-Orthogonal Grids," 2010.

[16] P. Block, "Block structure creation creation of the auxiliary drawing which serves as basic for the block structure meshing of individual numerical blocks under conservation of boundary conditions at the shared interfaces between neighbor blocks . Automatic block structure creation," pp. 1–6, 2019.

[17] D. K. CLARKE, H. A. HASSAN, and M. D. SALAS, "Euler calculations for multielement airfoils using Cartesian grids," *AIAA Journal*, vol. 24, no. 3, pp. 353–358, 1986.

[18] R. GAFFNEY, JR. and H. HASSAN, "Euler calculations for wings using Cartesian grids," *AIAA Journal*, 1987.

[19] C. S. Peskin, "Flow patterns around heart valves: A numerical method," *Journal of Computational Physics*, vol. 10, no. 2, pp. 252–271, 1972.

[20] J. P. Steinbrenner and R. W. Noack, "Three-dimensional hybrid grid generation using advancing front techniques," pp. 333–358, 1995.

[21] E. F. Charlton, K. G. Powell, and K. G. Powell, "An octree solution to conservation laws over arbitrary regions (OSCAR)," no. January, 1997.

[22] K.-F. Tchon, C. Hirsch, R. Schneiders, K.-F. Tchon, C. Hirsch, and R. Schneiders, "Octree-based hexahedral mesh generation for viscous flow simulations," pp. 1–9, 1997.

[23] Z. J. Wang, "A quadtree-based adaptive Cartesian/Quad grid flow solver for Navier-Stokes equations," *Computers and Fluids*, vol. 27, no. 4, pp. 529–549, 1998.

[24] M. Delanaye, M. Aftosmis, M. Berger, Y. Liu, and T. Pulliman, "Automatic hybrid-Cartesian grid generation for high-Reynolds number flows around complex geometries," 1999.

[25] D. Martineau, S. Stokes, S. Munday, A. Jackson, B. Gribben, and N. Verhoeven, "Anisotropic Hybrid Mesh Generation for Industrial RANS Applications," no. January, pp. 1–13, 2006.

[26] H. Luo, S. Spiegel, and R. Löhner, "Hybrid grid generation method for complex geometries," *AIAA Journal*, vol. 48, no. 11, pp. 2639–2647, 2010.

[27] A. Wissink, M. Potsdam, V. Sankaran, J. Sitaraman, Z. Yang, and D. Mavriplis, "A Coupled Unstructured-Adaptive Cartesian CFD Approach for Hover Prediction," *American Helicopter Society 66th Annual Forum*, 2010.

[28] B. Roget, J. Sitaraman, V. Lakshminarayan, and A. Wissink, "Prismatic Mesh Generation Using Minimum Distance Fields," pp. 1–27, 2018.

[29] "Cart3D Documentation,"

[30] J. Allred, D. Calloway, and S. Ruffin, "Validation of the NASCART-GT Flow Solver via Data Comparison," no. July, 2003.

[31] J. Tang, "User guide for shock and blast simulation with the," 2007.

[32] A. Hashimoto, K. Murakami, T. Aoyama, and P. R. Lahur, "Lift and drag prediction using automatic hexahedra grid generation method," *47th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2009.

[33] M. Tomac and D. Eller, "Towards automated hybrid-prismatic mesh generation," *Procedia Engineering*, vol. 82, pp. 377–389, 2014.

[34] AIAA, "R-101A - AIAA recommended practice for the CFD General Notation System - Standard interface data structures," p. 156, 2005.

[35] D. Eller and M. Tomac, "Implementation and evaluation of automated tetrahedral-prismatic mesh generation software," *CAD Computer Aided Design*, vol. 72, pp. 118–129, 2016.

[36] H. Samet, "An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures," *Theoretical Foundations of Computer Graphics and CAD*, pp. 51–68, 2012.

[37] S. Popinet, "Gerris : a tree-based adaptive solver for the incompressible Euler equations in complex geometries," *Journal of Computational Physics*, vol. 190, no. 2, pp. 572–600, 2017.

[38] H. Si, "Tetgen-Manual.Pdf," tech. rep., 2006.

[39] L. Maréchal, "Advances in octree-based all-hexahedral mesh generation: Handling sharp features," *Proceedings of the 18th International Meshing Roundtable, IMR 2009*, pp. 65–84, 2009.

[40] D. T. Lee and B. J. Schachter, "Two algorithms for constructing a Delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[41] N. Taylor, M. Uk, G. Mcgowan, M. Anderson, C. Schnepf, K. Richter, M. Tormalm, and M. Uk, "The Prediction of Vortex Interactions on a Generic Missile Configuration Using CFD : Current Status of Activity in NATO AVT-316," pp. 1–18, 2018.

[42] G. S. Makeich and I. A. Kryukov, "Aerodynamics and Flight Dynamics Simulation of Basic Finner Supersonic Flight in Aeroballistic Experiment," *Journal of Physics: Conference Series*, vol. 1009, no. 1, 2018.

[43] "CheckMesh," 2020.

[44] C. Support, "Mesh quality check."

[45] U. Guide, "Keywords," 2020.

[46] A. Dupuis and W. Hathaway, "Aeroballistic range tests of the basic finner reference projectile at supersonic velocities," 1997.

[47] E. DIKBAS, "Design of a Grid Fin Aerodynamic Control Device for Transonic Flight Regime," no. June, p. 104, 2015.

Appendix A

POLYMESH FILE FORMAT

A.1 PolyMesh Properties

A polyMesh consists of five different files: points, faces, owner, neighbor and boundary files. With a polyMesh format, the nodes and the faces are introduced to the solver along with the owner, neighbor and boundary files. The reason behind introducing the owner and neighbor is to calculate the computational volume which occupied by the cell. every face has its owner cell and neighbor cell. The faces with the same owner cell forms the volumetric cell where the computational equations would be solved.

Also, one of the rule of thumb is that the owner of a face must have higher identification number than the neighbor of the face. In addition, as for the check of the files, the number of the faces must be equal to the number of owner count and neighbor count in the owner and neighbor files respectively. The properties of these files are explained.

A.1.1 points File

The points file contains the list of coordinates of the nodes. The first line is the number of nodes in the mesh. Then, between parenthesizes, the coordinates of the nodes are written in the order of x coordinate, y coordinate and z coordinate.

The format of the points file given in the Figure A.1.

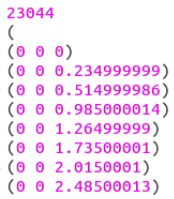


Figure A.1: points File format

A.1.2 faces File

The faces file contains the face elements. The face elements are defined with nodes from the points file. The first line is the number of face elements and the next line starts with the number of vertex of the face element. For example, for square shaped faces with four nodes, there are four vertices.

The format is given in Figure A.2.

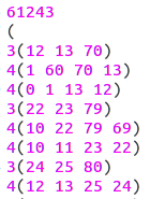


Figure A.2: faces File Format

A.1.3 owners File

The owner file is the file contains the owners of the faces. The first line is the number of owners in the file. After the first line, the first owner value is the owner cell of the first face in the faces file. Then, the second value in the owner file is for the owner cell of the second face in the faces file.

The format is given in Figure A.1.

A.1.4 neighbor File

The neighbor file consists of the neighbors of the faces. The first line is the declaration of the neighbor number. The first line in the list of the neighbors is the neighbor cell of the first face. The second and the other elements in the file goes as like the previous. The format of neighbor file is shown in Figure A.1.

```
61243
(
2
18757
17
7
18765
27
725
28
29
18767
```

Table A.1: neighbor and owner File Format

A.1.5 boundary File

The boundary file consists of the boundary faces of the mesh. There is a patch part for every boundary type of the mesh. The format of boundary file is given in Figure A.3.

```
(
    SYMP3
    {
        type            patch;
        startFace       53708;
        nFaces          3191;
    }

    INLE1
    {
        type            patch;
        startFace       56899;
        nFaces          45;
    }
```

Figure A.3: boundary File Format

Appendix B

NODE FACE CONNECTIVITY FOR INCELL-FACES

Table B.1: Writing Order of Nodes for Neighbor Cell ID > Owner Cell ID (Y Direction)

rootFaceY	face	xID	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID	yID	zID + 1	Refinement Level
rootNode	node	xID	yID	zID + 1	Refinement Level

Table B.2: Writing Order of Nodes for Owner Cell ID > Neighbor Cell ID (Y Direction)

rootFaceY	face	xID	yID	zID	Refinement Level
rootNode	node	xID	yID	zID + 1	Refinement Level
rootNode	node	xID	yID	zID + 1	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level

Table B.3: Writing Order of Nodes for Neighbor Cell ID > Owner Cell ID (Z Direction)

rootFaceZ	face	xID	yID	zID	Refinement Level
rootNode	node	xID	yID + 1	zID	Refinement Level
rootNode	node	xID	yID + 1	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level

Table B.4: Writing Order of Nodes for Owner Cell ID > Neighbor Cell ID (Z Direction)

rootFaceZ	face	xID	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID	yID + 1	zID	Refinement Level
rootNode	node	xID	yID + 1	zID	Refinement Level

Appendix C

NODE FACE CONNECTIVITY FOR FARFIELD-FACES

Table C.1: Writing Order of Nodes (+y direction)

rootFaceY	face	xID	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID	yID	zID + 1	Refinement Level
rootNode	node	xID	yID	zID + 1	Refinement Level

Table C.2: Writing Order of Nodes (-y direction)

rootFaceY	face	xID	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID + 1	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID	yID	zID	Refinement Level
rootNode	node	xID	yID	zID + 1	Refinement Level

Table C.3: Writing Order of Nodes (+z direction)

rootFaceZ	face	xID	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID	yID + 1	zID	Refinement Level
rootNode	node	xID	yID + 1	zID	Refinement Level

Table C.4: Writing Order of Nodes (-z direction)

rootFaceZ	face	xID	yID	zID	Refinement Level
rootNode	node	xID + 1	yID + 1	zID	Refinement Level
rootNode	node	xID + 1	yID	zID	Refinement Level
rootNode	node	xID	yID	zID	Refinement Level
rootNode	node	xID	yID + 1	zID	Refinement Level

Appendix D

NODE FACE CONNECTIVITY FOR PYRAMID-BASE-FACES

Table D.1: Relations Between Face and Nodes (+y direction)

face	rootFaceY	xID	yID	zID	Refinement Level
node	rootNode	xID + 1	yID + 1	zID	Refinement Level
node	rootNode	xID + 1	yID + 1	zID + 1	Refinement Level
node	rootNode	xID	yID + 1	zID + 1	Refinement Level
node	rootNode	xID	yID + 1	zID	Refinement Level

Table D.2: Relations Between Face and Nodes (-y direction)

face	rootFaceY	xID	yID	zID	Refinement Level
node	rootNode	xID	yID	zID	Refinement Level
node	rootNode	xID	yID	zID + 1	Refinement Level
node	rootNode	xID + 1	yID	zID + 1	Refinement Level
node	rootNode	xID + 1	yID	zID	Refinement Level

Table D.3: Relations Between Face and Nodes (+z direction)

face	rootFaceZ	xID	yID	zID	Refinement Level
node	rootNode	xID	yID	zID + 1	Refinement Level
node	rootNode	xID	yID + 1	zID + 1	Refinement Level
node	rootNode	xID + 1	yID + 1	zID + 1	Refinement Level
node	rootNode	xID + 1	yID	zID + 1	Refinement Level

Table D.4: Relations Between Face and Nodes (-z direction)

face	rootFaceZ	xID	yID	zID	Refinement Level
node	rootNode	xID	yID	zID	Refinement Level
node	rootNode	xID + 1	yID	zID	Refinement Level
node	rootNode	xID + 1	yID + 1	zID	Refinement Level
node	rootNode	xID	yID + 1	zID	Refinement Level

Appendix E

CARTESIAN ELEMENT CONNECTIVITY

Table E.1: Edge/Triangle Face Node Order

		cell	**rootCell**	**xID**	**yID**	**zID**	**Refinement Level**
x_1 edge	Pyramid 1 exists / Pyramid 5 not exist	node	rootNode	xID	yID	zID	Refinement Level + 1
		node	rootNode	xID	yID + 1	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
	Pyramid 1 not exist / Pyramid 5 exists	node	rootNode	xID	yID	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID	yID + 1	zID	Refinement Level + 1
x_2 edge	Pyramid 2 exists / Pyramid 5 not exist	node	rootNode	xID + 1	yID	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID + 1	yID + 1	zID	Refinement Level + 1
	Pyramid 2 not exist / Pyramid 5 exits	node	rootNode	xID + 1	yID	zID	Refinement Level + 1
		node	rootNode	xID + 1	yID + 1	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
x_3 edge	Pyramid 1 exists / Pyramid 6 not exist	node	rootNode	xID	yID	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID	yID + 1	zID + 1	Refinement Level
	Pyramid 1 not exist / Pyramid 6 exits	node	rootNode	xID	yID	zID + 1	Refinement Level + 1
		node	rootNode	xID	yID + 1	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
x_4 edge	Pyramid 2 exists / Pyramid 6 not exist	node	rootNode	xID + 1	yID	zID + 1	Refinement Level + 1
		node	rootNode	xID + 1	yID + 1	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
	Pyramid 2 not exist / Pyramid 6 exits	node	rootNode	xID + 1	yID	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID + 1	yID + 1	zID + 1	Refinement Level + 1

Table E.1: Edge/Triangle Face Node Order

		cell	**rootCell**	**xID**	**yID**	**zID**	**Refinement Level**
y_1 edge	Pyramid 1 exists / Pyramid 3 not exist	node	rootNode	xID	yID	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID	yID	zID + 1	Refinement Level + 1
	Pyramid 1 not exist / Pyramid 3 exits	node	rootNode	xID	yID	zID	Refinement Level + 1
		node	rootNode	xID	yID	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
y_2 edge	Pyramid 1 exists / Pyramid 4 not exist	node	rootNode	xID	yID + 1	zID	Refinement Level + 1
		node	rootNode	xID	yID + 1	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
	Pyramid 1 not exist / Pyramid 4 exits	node	rootNode	xID	yID + 1	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID	yID + 1	zID + 1	Refinement Level + 1
y_3 edge	Pyramid 2 exists / Pyramid 3 not exist	node	rootNode	xID + 1	yID	zID	Refinement Level + 1
		node	rootNode	xID + 1	yID	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
	Pyramid 2 not exist / Pyramid 3 exits	node	rootNode	xID + 1	yID	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID + 1	yID	zID + 1	Refinement Level + 1
y_4 edge	Pyramid 2 exists / Pyramid 4 not exist	node	rootNode	xID + 1	yID + 1	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID + 1	yID + 1	zID + 1	Refinement Level + 1
	Pyramid 2 not exist / Pyramid 4 exits	node	rootNode	xID + 1	yID + 1	zID	Refinement Level + 1
		node	rootNode	xID + 1	yID + 1	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level

Table E.1: Edge/Triangle Face Node Order

		cell	**rootCell**	**xID**	**yID**	**zID**	**Refinement Level**
z_1 edge	Pyramid 3 exists / Pyramid 5 not exist	node	rootNode	xID	yID	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID + 1	yID	zID	Refinement Level + 1
	Pyramid 3 not exist / Pyramid 5 exits	node	rootNode	xID	yID	zID	Refinement Level + 1
		node	rootNode	xID + 1	yID	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
z_2 edge	Pyramid 4 exists / Pyramid 5 not exist	node	rootNode	xID + 1	yID + 1	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID	yID + 1	zID	Refinement Level + 1
	Pyramid 4 not exist / Pyramid 5 exits	node	rootNode	xID + 1	yID + 1	zID	Refinement Level + 1
		node	rootNode	xID	yID + 1	zID	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
z_3 edge	Pyramid 3 exists / Pyramid 6 not exist	node	rootNode	xID	yID	zID + 1	Refinement Level + 1
		node	rootNode	xID + 1	yID	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
	Pyramid 3 not exist / Pyramid 6 exits	node	rootNode	xID	yID	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID + 1	yID	zID + 1	Refinement Level + 1
z_4 edge	Pyramid 4 exists / Pyramid 6 not exist	node	rootNode	xID + 1	yID + 1	zID + 1	Refinement Level + 1
		node	rootNode	xID	yID + 1	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
	Pyramid 4 not exist / Pyramid 6 exits	node	rootNode	xID + 1	yID + 1	zID + 1	Refinement Level + 1
		apex	rootApex	xID	yID	zID	Refinement Level
		node	rootNode	xID	yID + 1	zID + 1	Refinement Level + 1

Appendix F

COMMAND SWITCHES OF TETGEN

For the compilation of TetGen, the required files are

- tetgen.h : TetGen library header file
- tetgen.cxx : TetGen library source code written in C++
- predicates.cxx Geometric predicates source code written in C++

The program can be worked on stand-alone or can be embedded in another program. If the program is used as a stand-alone program, command switches can be given to the compiler via command line along with an input file. Command switches are used for the controlling the tetrahedralization behavior and the output files. The command line syntax for the TetGen is given as

tetgen [-pq__a__AriYMS__T__dzjo_fengGOJBNEFICQVvh] input_file

The underlined blankets are for the numbers which may introduced for some switches. For the tetrahedral mesh generation in this study, the command used is "-pqaYzkffenni" and "-pqaYzgkenni" in this study. The most important command switch is the -p switch. This switch provides the reading of piecewise linear complex(PLC) in the .smesh, then, starts the constraint Delaunay tetrahedralization (CDT).

With switch command -q responsible for quality mesh generation. TetGen guarantees the shape and sizes which is suitable for finite volume methods. The definition of shape and size given by TetGen is that all of the elements would have certain amount of quality and the number of elements is minimum. The default of the program for quality mesh is that minimum radius to edge ratio of tetrahedral element would be

2.0. If the ratio given is less than 1.0, TetGen may not terminate. TetGen suggests 1.414 for the radius to edge ratio for successful termination of the program. However, for having smooth transition between the tetrahedral cells, 1.1 is used as the ratio.

-a is for the restriction of on the maximum volume which can a tetrahedral element can have.

-Y is the command for suppression of the Steiner points on the boundary faces. This command is used when it is desired not to change or split faces of the tetrahedral due to fitting into another mesh like in this study. However, unless using -Y switch twice (-YY), the interior Steiner points will be created. Also, when -q switch is used along with -Y switch, the quality of the mesh may be poor.

-k command switch is used for creating a .vtk file for Paraview visualization.

-f and -n command switches are for the outputs of .faces and .neigh files.

Finally, -i command switch is used for shows the list of additional points.