Technical University of Crete
School of Production Engineering and Management

Software development for the generation of two-dimensional hybrid unstructured grids

by

Alexandros I. Eskandar

Diploma Thesis

Supervisor

Dr. Ioannis K. Nikolos, Professor

Chania, July 2019

Technical University of Crete
School of Production Engineering and Management

Software development for the generation of two-dimensional hybrid unstructured grids

by

Alexandros I. Eskandar

Approved by:

Dr. Ioannis K. Nikolos
Professor
Technical University of Crete,
School of Production Engineering
& Management.

Dr. Anargiros I. Delis
Associate Professor
Technical University of Crete,
School of Production Engineering
& Management.

Dr. Georgios Arampatzis
Assistant Professor
Technical University of Crete,
School of Production Engineering
& Management.

Chania, July 2019

Preface

The purpose of this thesis is the development of a methodology and the corresponding software for the construction of 2D unstructured grids for CFD (Computational Fluid Dynamics) analyses. The geometry to be examined and the boundaries of the domain are imported in 2D coordinates (x, y) form, through a ".txt" file, or from an "obj" file in parametric form, which is the only input given by the user. The direction of the nodes must follow counter clockwise sense for external boundary nodes and clockwise sense for internal boundaries. Specifically, the software developed enables the creation of purely triangular grids for the simulation of inviscid flows, as well as hybrid unstructured grids, consisting of triangular and quadrilateral elements, in order to accurately solve the boundary layer that develops in the area near the solid walls of the computational domain when simulating viscous flows. For the construction of the triangular grid, the known "Delaundo" software, developed by J.D. Muller is used, while for the construction of the viscous layers around the solid walls, an algebraic methodology is used that has been developed in the context of this thesis. Finally, a specially designed algorithm has also been implemented to suitably combine the triangular and quadrilateral sections into a single grid. All data is stored in properly designed structures. The software has been implemented entirely in C++ programming language, and is also enriched with a flexible graphical interface (GUI), for easy user interaction, created with the Qt 5 graphics platform, as well as with a viewer for visualizing computational grids, based on OpenGL. At present, the software provides the ability to create hybrid grids for both simple and more complex computational domains, as presented in the validation chapter. The algebraic mesh algorithm that has been created, follows a logical series of steps-calculations.

Acknowledgements

Foremost, I would like to thank my advisor, Professor Ioannis K. Nikolos, who showed trust in my face and believed in my capabilities. His deep knowledge of the subject and his continuous encouragement for improvement led me to complete this thesis.

Also, I would like to express my sincerely gratitude to my labmate and good friend PhD candidate Stavros N. Leloudas, for his constant support and patience during the period of our collaboration. He was supporting my work with his targeted observations and his talent to break down complexity to elegant simplicity. He is a true perfectionist and a great researcher.

Last but not least, none of these would have been possibly done without the love, support and the great sense of humor of my beloved family and closest friends, which have been the main reason I never lost my will and strength all these years. This thesis is dedicated to them.

Alexandros I. Eskantar

Chania, July 2019

Table of Contents

Table of Figures

Chapter 1: Basic Concepts

In modern practice, in the study of mechanical components and not only, problems of fluid mechanics often appear, required to be solved. Studying and solving those using analytical methods is not a simple procedure, as the differential equations governing such problems are non-linear with high complexity. As experiments proved to be particularly time-consuming and in most cases of high cost, the development of a new branch of fluids engineering was required: Computational Fluid Dynamics (CFD). CFD examines methods of developing and solving discrete models for the partial differential equations (PDEs) that govern fluid mechanics problems, adopting methods from computational physics and numerical analysis. The equations describing the inviscid flow are the Euler equations, while the viscus flow of Newtonian fluids are the Navier-Stokes equations [1]. Those equations cannot be solved analytically, except for a few special cases, so it becomes necessary the use of numerical approximations.

With CFD a numerical approximation for the problem at hand is calculated. In order to receive the solution, it is necessary to use a discretization method, which essentially approximates the system of partial differential equations with one system of linear algebraic equations, which eventually can be solved using a computer. Approximations are applied to small space sectors, (and short time steps for transient problems), thus the solution is ultimately attributed to distinct points of space (and time).

One of the crucial factors for a numerical solution is the discretization of space in small sectors. These small (non-overlapping) sectors are called “cells” and all the cells of the divided domain consist the computational grid, while the production of such a grid is called “mesh generation”. Consequently, the way the grid is constructed is particularly important to effectively solve a problem. A variety of grids can be used in natural problems, such as “structured” or “unstructured” grids, while there is no limitation on the number of cells. Two-dimensional grids usually contain triangular (unstructured grids) or/and quadrilateral (structured or unstructured grids) shaped cells. Furthermore, the combination of triangular and quadrilateral cells (in 2D domains) produces the unstructured hybrid. Accordingly, based on the needs of the flow simulation problem, different mesh generation methods can be applied.

This diploma thesis aims in the development of a mesh generation software that will discretize 2D flow domains with hybrid unstructured grids. Industry’s standard programming language for such software is C++. C++ is deployed for tasks that demand high performance, such as video editing and transcoding, high-end computer-aided design or engineering (CAD, CAE), image processing, games,

telecommunications, and business. For the graphical user interface (GUI) QT 5 and OpenGL tools have been used, as they have been implemented in world-leading commercial software, demonstrating the high performance of this combination.

1.1 2D Unstructured Triangular Grids

2-dimensional (2D) unstructured grids have clear advantages over structured ones. In particular, unstructured grids can discretize any domain with triangular elements (Fig. 1) without any problem; on the contrary it is very difficult and time-consuming to build structured grids around complex geometries [1].

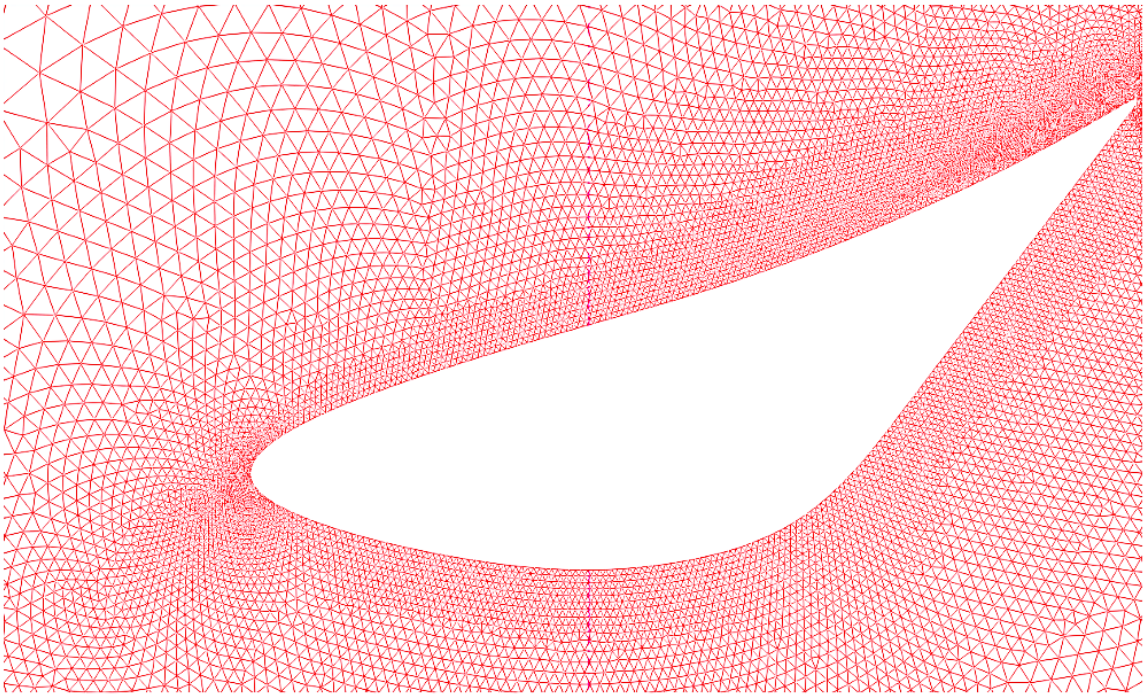


Figure 1: 2D unstructured grid around an axisymmetric diffuser.

Another important advantage of the unstructured grid is the ability of local refinement, in areas where heavily changing phenomena (such as boundary layers or shock waves) appear. The aforementioned local adaptation of the unstructured grid can be implemented during the solution of a problem, if transient phenomena appear, greatly increasing the accuracy of the simulation results. In contrast, in the case of structured grids the refinement should be performed across the grid so it can maintain the character of the structured grid [2]. The most important disadvantage of unstructured grids is the difficulty in handling them. Since there is no sense of direction against the length of the grid lines, it results an irregular connection of the cells. Consequently, lack of structure requires the creation of appropriate data structures, where the topological information of the elements will be stored.

Creation of such structures but also their handling is quite time consuming, while at the same time it has high memory requirements, especially for 3D cases.

1.2 2D Structured Quadrilateral Grids

The application of a structured grid in any aspect of grid generation has certain advantages and disadvantages. The advantage of such a grid is that the points of an elemental cell can be easily addressed by a double of indices (i, j) in 2D. The connectivity is straightforward, because cells adjacent to a given elemental face are identified by the indices and the cell edges form continuous mesh lines that begin and end on opposite elemental faces, as illustrated in Fig. 2.

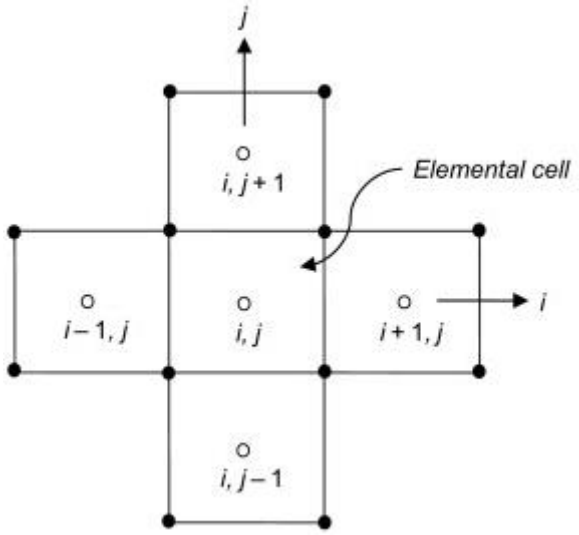


Figure 2: Structured quadrilateral grid cells numbering [3].

In two dimensions, the central cell is connected by four neighboring cells. It also allows easy data management, and connectivity occurs in a regular fashion, which makes programming and data handling fast and easy [3]. Furthermore, structured grids provide high degree of quality and control. This is arguably an area where structured grids will always be supreme. Unstructured algorithms are highly automated, and as a result, engineers have to sacrifice control. With structured grids, a higher degree of control means almost the perfect match with the necessary grid. Structured meshing typically allows the user to have a better control of interior node locations and cell sizes.

Structured grids are usually aligned in the flow direction, producing more accurate results and a better convergence in CFD solvers. This alignment in a structured grid is achieved almost implicitly, because grid lines and flow follow the contours of the boundaries; such an alignment is impossible for an unstructured grid. Application of boundary conditions and turbulence models work better, when there is a well-defined direction normal to a wall boundary, or a jet [4].

The disadvantage of adopting such a grid, particularly for more complex geometries, is the time consumption and the increase of grid nonorthogonality or skewness that can cause unphysical solutions, due to the transformation of the governing equations [3].

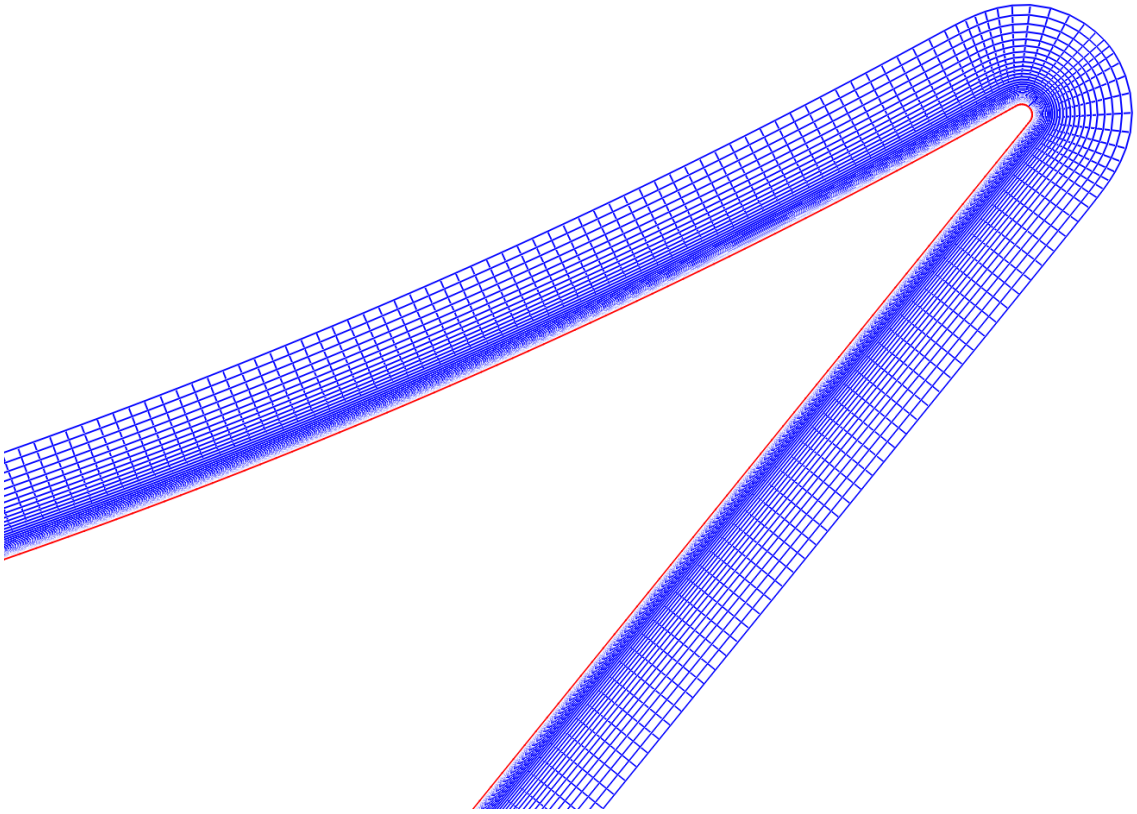


Figure 3: 2D structured grid close to the solid boundary of an axisymmetric diffuser.

1.3 Hybrid Mesh Generation

For successful flow simulation, grid generation and numerical simulation methods need to be closely coupled [5]. This diploma thesis focuses on an adaptive algebraic method for the construction of hybrid 2D grids, composed of quadrilateral and triangular elements. Hybrid grid generation methods have attained a respectable status for discretizing arbitrarily complex computational flow domains. The quadrilaterals cover the region close to the body surface and triangles discretize the remaining domain (Fig. 4).

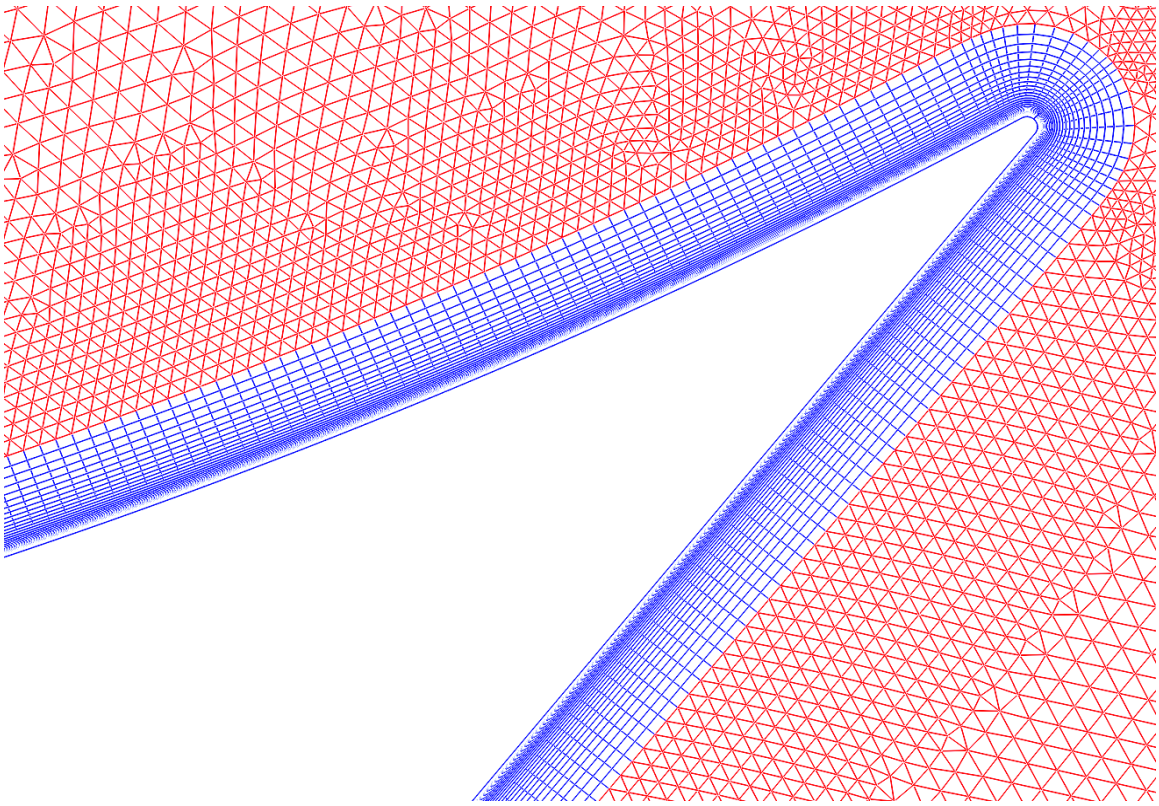


Figure 4: A 2D hybrid grid around an axisymmetric diffuser.

Different element types allow the exploitation of the advantages of both structured and unstructured grid generation techniques. As described in the previous paragraphs, the main advantage of unstructured grids is the high automation level of the meshing procedure, and the grid adjustment almost in any geometry. On the other hand, the cost of such fast method is the lower quality of the numerical simulation, as irregular grid lines lead in a series of inaccurate flow results. In contrast structured grids provide higher cells quality and normal grid lines on the viscous boundaries, leading to simulation results of higher quality and accuracy. However, high quality leads on a time consuming meshing procedure and (usually) impossible mesh adaptation in complex geometries. Thus, the engineers through the use of hybrid grids adopted a golden ratio in between mesh quality and time consuming mesh generation.

1.4 Data Structures and C++

The use of dynamic data structures cannot be avoided, as the grid information as well as information about the proper operation of the graphical environment is constantly changing. Thus, the first step for the development of a functional meshing software is the organization of the required data structures, to ensure adequate and quick information exchange during the mesh generation procedure. At the same time, the graphical environment should be adjusted accordingly to the user's

selections, as different functions can be activated throughout the mesh generation procedure. Thanks to those data structures, all desired software functions are achieved. C++ programming language is the industry standard language for decent graphical environments and high speed data processing. In addition, its flexibility as well as its widespread use of various tools (such as OpenGL and Qt) renders it ideal for creating such software, where the speed and high-quality graphics are indispensable.

1.5 About OpenGL

OpenGL is the leading environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms [6]. OpenGL promotes innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. OpenGL was used in this work as it provides useful functions for creating a graphical environment, as well as rich material and information easy to find on the web. It is also effective, as it can handle large amounts of data without delay [6]. The developed in this work T2GR environment uses 2D graphics. OpenGL was used to visualize boundaries, edges, nodes, and the constructed computational grid.

1.5.1 Cartesian Coordinates (OpenGL)

One of the hardest problems during the development of the GUI was finding the cursor position at cartesian coordinates every time the user clicks on the viewer. The coordinates functions of OpenGL return pixels cordinates. Therefore, a process was developed to convert the coordinates from digital to Cartesian. If x is in the range $[a, b]$ and it is need to transform to y in the range $[c, d]$, the following linear relation gives

$$y = (x - a)\frac{(d - c)}{(b - a)} + c. \tag{1}$$

Thus, every time the user clicks on the viewer the digital coordinates are converted to Cartesian, using the previous transformation. However, an approximation of the actual position will be calculated, as it is impossible to click exactly over a specific node. This is reasonable if we take into acount that an average screen holds 1366 x 768 = 1,049,088 pixels resolution.

1.5.2 Zoom In – Zoom Out (OpenGL)

If the mouse pointer moves forward, it is considered as a zoom-in, while if it moves backward, it is considered as a zoom-out. Therefore, a variable that holds a zoom scale equal to 1 is multiplied by 1.1 during the zoom-out operation, and divided by 1.1 during the zoom-in. The value 1.1 was selected as the zoom ratio after testing several other values.

$$scale = scale\ x\ 1.1\ \ (zoom - out) \tag{2}$$

$$scale = \frac{scale}{1.1}\ \ (zoom - in) \tag{3}$$

As soon as the clipping plane is zoomed-in, the digital coordinates of the viewer change. As the pixel that was before in position (500, 200) after 8 scrolls will be at position(1072,430). Thus, the Cartesian coordinates that define the initial location of the graphics display camera should also be adjusted, through the "glOrtho" function.

1.5.3 About "glOrtho"

An orthographic projection matrix defines a cube-like frustum box that defines the clipping space where each vertex outside this box is clipped. When creating an orthographic projection matrix we specify the width, height and length of the visible frustum. All the coordinates that end up inside this frustum after transforming them to clip space with the orthographic projection matrix won't be clipped. The frustum looks a bit like a container (Fig. 5).

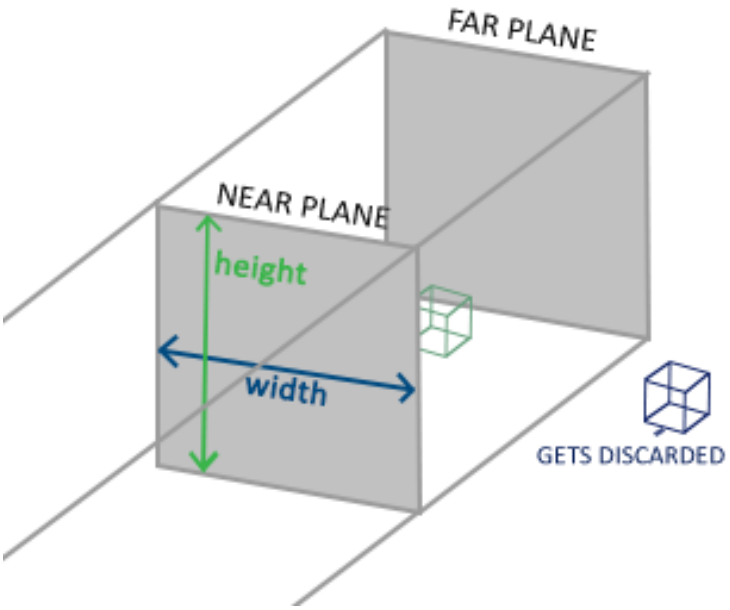


Figure 5: The frustum defines the visible objects [6].

The frustum defines the visible coordinates and is specified by a width, a height and a near and far plane. Any coordinate in front of the near plane is clipped and the same applies to coordinates behind the far plane. The orthographic frustum directly maps all coordinates inside the frustum to normalized device coordinates, since the *w* component of each vector is untouched. If the *w* component is equal to 1.0, perspective division doesn't change the coordinates.

To create an orthographic projection matrix, we make use of function "glOrtho":

$$glOrtho(\frac{Xmin}{zoom\ scale}, \frac{Xmax}{zoom\ scale}, \frac{Ymin}{zoom\ scale}, \frac{Ymax}{zoom\ scale}, Zmin, Zmax) \tag{4}$$

The first two parameters specify the left and right coordinates of the frustum and the third and fourth parameters specify the bottom and top parts of the frustum. With those four points we've defined the size of the near and far planes, while the 5th and 6th parameters define the distances between the near and far planes. This specific projection matrix transforms all coordinates between these x, y and z range values to normalized device coordinates [6].

1.5.4 Mouse Move (OpenGL)

In order to impart the sense of movement in 2D space, the visible components defined by "glOrtho" must be renewed. To this end, the following technique applies:

The difference between the previous position and the new is computed

$$d_x = x - (last\ position\ of\ x). \tag{5}$$

Translation into Cartesian coordinates

$$d_x = \frac{(xmax - xmin)}{width} * width. \tag{6}$$

Renewing the new position in Cartesian coordinates

$$x_{translate} = x_{translate} + d_x. \tag{7}$$

Re-initialization of function "glOrtho"

$$glOrtho\left(\frac{Xmin}{zoom\ scale} - X_{translate}, \frac{Xmax}{zoom\ scale} - X_{translate}, \frac{Ymin}{zoom\ scale} - y_{translate}, \frac{Ymax}{zoom\ scale} - y_{translate}, Zmin, Zmax\right). \tag{8}$$

1.6 Qt 5 (GUI)

Since creating a computational grid is a complicated process, it would be quite difficult for the user to interact with the software through the command line. In the software developed within this diploma thesis, a graphical environment was designed in order to simplify the corresponding procedures. The graphical environment should be easy to learn, simple in its structure, provide all necessary information and finally lead the user step by step on each process. In fact, the graphical environment is the intermediary between the code and the user. The user selects the available parameters through the GUI and the code runs on them. For the creation of the GUI the Qt 5 classes and functions packet applied. Qt's single source compatibility, its feature richness, its C++ performance, the availability of the source code, its documentation and all the classes and functions existence, make it ideal for GUI development [7].

1.7 The Free-Form Deformation Tool

The T2GR software incorporates the Free-Form Deformation (FFD) tool. The tool works complementarily with the initial mesh. A basic use of this tool is to create similar geometries and their grids without inserting a new geometry or re-creating the computational grid, but only by deforming the initial ones (geometry and grid). Since the user has produced the necessary grid on the initial geometry, through the use of the FFD tool the user can deform the current geometry and at the same time the current computational grid (Fig. 6). Thus, without changing the boundary conditions and the number of nodes in the grid, the user has the ability to rapidly produce similar cases and study similar problems (such as in the case of an automated design optimization procedure). In the field of computational engineering any time-saving procedure could mean dramatic decrease on computing or monetary cost [8]. A basic introduction on FFD technique will be presented in Chapter 4.

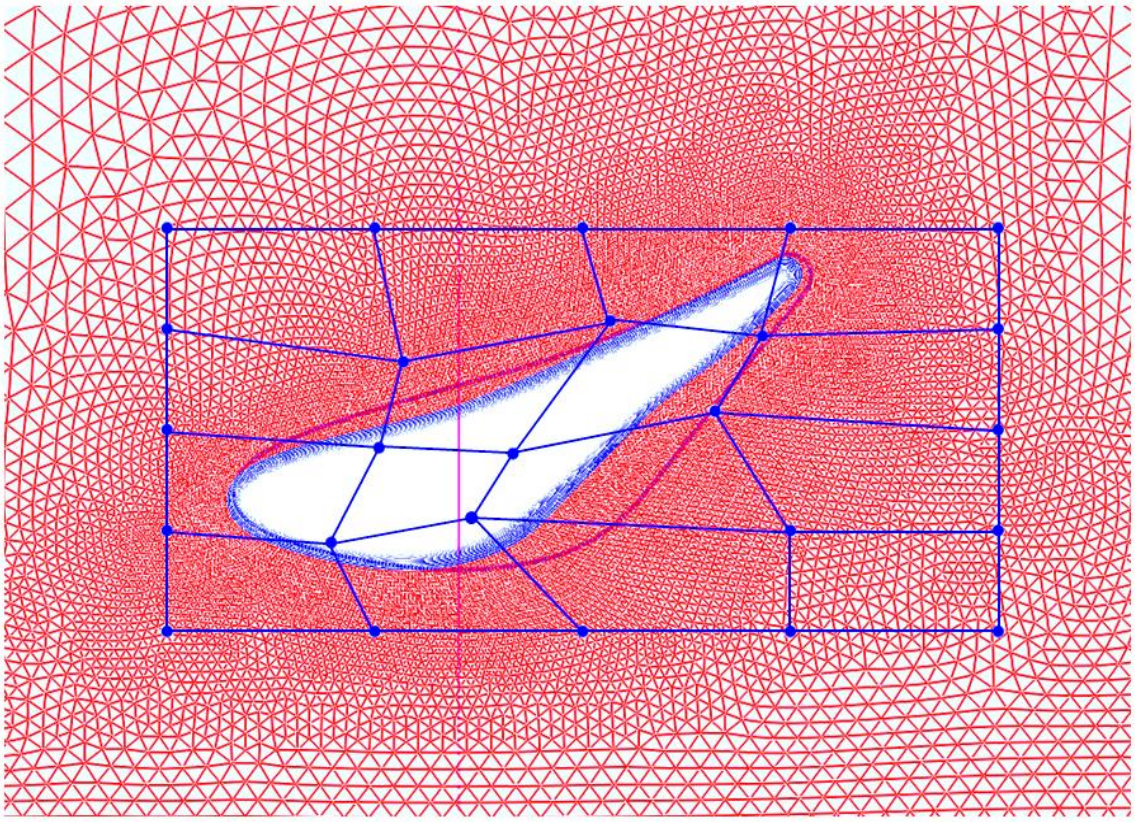


Figure 6: A deformed geometry and grid, using FFD.

Chapter 2: Quadrilateral Grid Generation

2.1 The Algebraic Method for Quadrilateral Grid Generation

The method is based on algebra and vector mathematics and does not require the solution of differential equations. The primary advantage of the method is that it provides explicit control of the physical grid shape and physical grid spacing. Additionally, it requires relatively few computations. Consequently, the application of interactive computer graphics in conjunction with the algebraic method is advocated for rapid generation of grids. The basic structure of the method is described below, while particular attention is given to elements quality, grid smoothing and nodal vectors inclination. Physical boundary topology requirements are also presented.

2.1.1 Normal Vectors Calculation for Each Edge

Firstly, the algorithm calculates for each boundary and for each edge of the boundary all normal vectors $E_{i.j}(boundary_i, edge_j)$. Thus, let $N_m(x_m, y_m)$ and $N_{m+1}(x_{m+1}, y_{m+1})$ be the set of nodes of the corresponding edge. The vector $E_{i.j}$ is computed as in (9), while its magnitude is given in (10); $E_{i.j}$ vector is then normalized and the result is the unit vector in (11). The vectors are normalized as for the subsequent calculations there is no need for their magnitude but only for their direction.

$$E_{i.j} = (y_{m+1} - y_m, x_{m+1} - x_m) \tag{9}$$

$$| \mathrm{E_{i,j}} | = \sqrt{(x_{m+1} - x_m)^2 + (y_{m+1} - y_m)^2} \tag{10}$$

$$\hat{E}_{i,j} = \frac{\mathrm{E_{i,j}}}{|\mathrm{E_{i,j}}|} \tag{11}$$

2.1.2 Nodal Normal Vectors Calculation

Each edge of a boundary consists of two nodes, and for each edge a normal vector is calculated. Initially, the normal (to the boundary) vector for a boundary node is taken equal to that of one of the adjacent edges. In order to achieve uniformity of the normal vectors along the corresponding boundary, it is necessary to apply a smoothing operator to them. The nodes along a boundary are divided, and half of them will be affected from the 1st node of the corresponding boundary and the remaining half will be affected from the last node of the corresponding boundary. The number of nodes, for which their normal vectors will be smoothed, is selected by the user.

Let the normal to node $k-1$ be (x_{k-1}, y_{k-1}), and the normal to its adjacent node of the boundary be (x_k, y_k). Let the pair of the corresponding normal vectors to the adjacent to node k edges be $(X_{k-1}, Y_{k-1}), (X_k, Y_k)$. Then, the smoothed normal vector $(x_{k,new}, y_{k,new})$ will be calculated as

$$x_{k,new} = SF * x_{k-1} + (1 - SF) * (X_{k-1} + X_k) + x_k \quad (12)$$

$$y_{k,new} = SF * y_{k-1} + (1 - SF) * (Y_{k-1} + Y_k) + y_k \quad (13)$$

$k = 1, 2 \ldots, n-1$

$SF \in [0, 1]$

Concerning the first and last nodes of each boundary, let the normal vector to the previous boundary last edge be $(X_{last}(pre), Y_{last}(pre))$ and for the next boundary first edge be $(X_0(next), Y_0(next))$. Then, the smoothed normal vectors at the first (0) and last (N) node of the boundary result as:

$$x_{0,new} = [X_0 + X_1 + X_{last}(pre) + X_{last-1}(pre)] + x_0 \quad (14)$$

$$y_{0,new} = [Y_0 + Y_1 + Y_{last}(pre) + Y(pre)] + y_0 \quad (15)$$

$$x_{N,new} = [X_{last} + X_{last-1} + X_0(next) + X_1(next)] + x_N \quad (16)$$

$$y_{N,new} = [Y_{last} + Y_{last-1} + Y_0(next) + Y_1(next)] + y_N \quad (17)$$

As before, every node vector is then normalized to become unitary.

In Figure 7 the smoothing was applied only to the 2 last nodes of the upper boundary. In Figure 8 smoothing was applied to all nodes of both boundaries, and the result is obviously better.

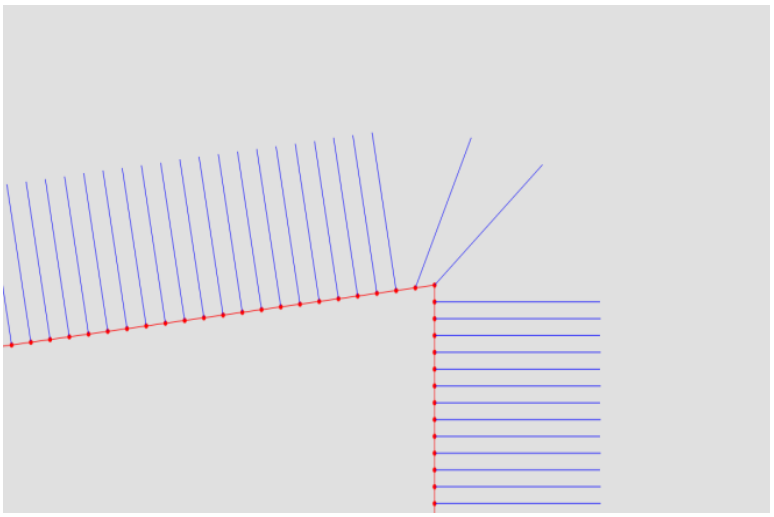


Figure 7: Smoothed (nodal) normal vectors, only for the last 2 nodes for the upper boundary.

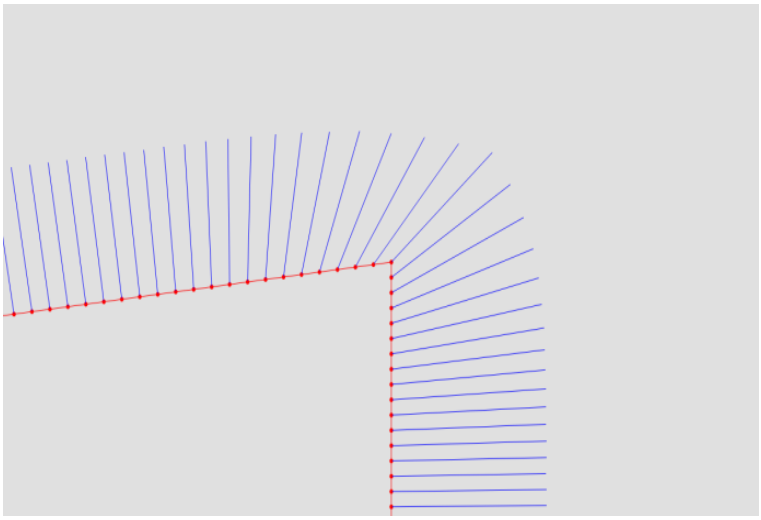


Figure 8: Smoothed (nodal) normal vectors.

2.2 Inflation Set-Up

In order to initiate the inflation (of the quadrilateral mesh) in the selected boundaries, it is first necessary to check which of the corresponding (inflated) boundaries are connected to each other. Looking at the first and last node of the corresponding boundary and the first and last node of the other inflated boundaries the software recognizes if there is a connection between them. In the case of a connection, all of the linked inflated boundaries are appended into a common list and the double nodes are removed. Thus, when this procedure ends, information about on which boundaries the inflation will be applied, which of these boundaries are connected and finally which are the inflation nodes, is available. Combined with the information of the growth rate, initial step, and number of steps, which is

provided by the user through the GUI, the loop for the creation of the quadrilateral grid can be initialized.

2.2.1 Quadrilateral Grid Generation

For the first inflation layer, the normal direction of the nodes has been already computed. All normal vectors are transformed to unitary ones, and then the high of the first layer is applied, in order to compute the corresponding nodes of the first inflation layer. After the construction of the first inflation layer, all subsequent layers can be created in the same way. In order to extend the (inflated) nodes to the desired direction, it is necessary to re-calculate the vertical vectors on each edge, for every new layer (so as to retain a smoothed inflation). As before, the direction of the normal vectors attributed to each node of the layer are computed and then a smoothing is applied. The height h_k of each layer k is computed using a geometric progression:

$$h_k = h_1 * (growth\ rate)^k \qquad (18)$$
$$k = 1, 2 \dots, K$$

2.3 Quadrilateral Grid Conforming to Adjacent Boundary

If the inflated grid has to conform to the geometry of an adjacent boundary (Fig. 9), then the previous methodology has to be modified.

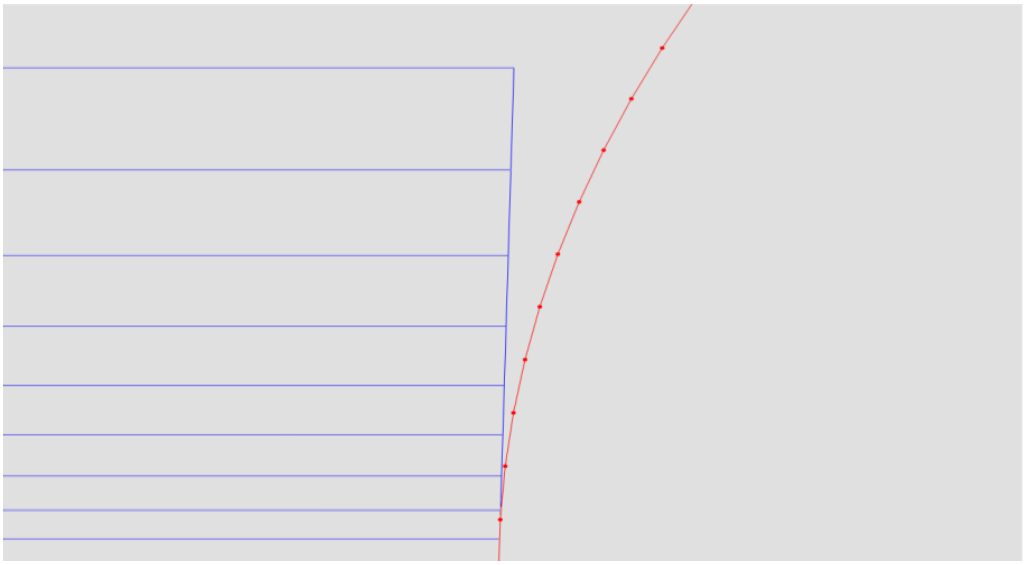


Figure 9: Non-conforming inflated grid.

The method that provides the grid with the correct inflating direction, to conform to the adjacent boundary, is based on B-Spline interpolation. The method of B-Spline interpolation will be described in a following chapter. However, in this section some elements of this method will be presented. Having the original nodes of the adjacent

boundary, a new node distribution is created along the corresponding boundary in order to obey to the inflation rate of the inflated boundary (Fig. 10). Firstly, using the existing nodes of the adjacent boundary, a B-Spline curve is computed to interpolate those nodes. Then, the geometric progression, used for the inflation, is used to compute new nodes along the B-Spline curve, which will serve as nodes of the produced quadrilaterals. The corresponding vectors along the B-Spline, will be also used at next for the smoothing operation of the normal vectors in the inflation procedure. The final result is demonstrated in Figure 10.

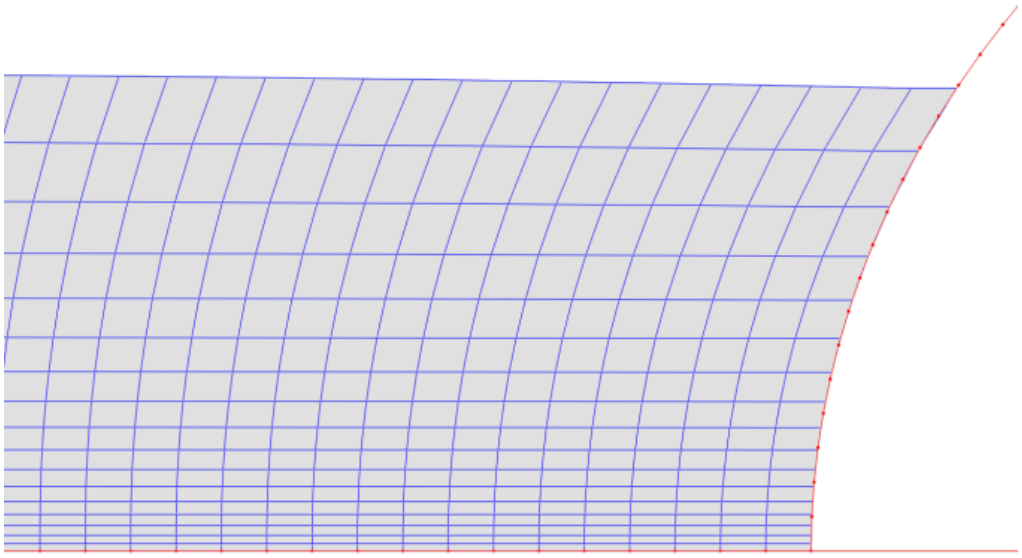


Figure 10: Conforming grid inflation.

2.4 Triangular Mesh Generation

In the next step of the algorithm and once the quadrilateral grid has been created, the boundaries on which the triangular grid will be created should be initialized. Thereafter, on the inflated boundaries, the triangular grid should be created, using as a starting boundary by the corresponding nodes of the quadrilateral grid. This procedure is quite simple; the software has to examine whether the first node or the last node of the last inflated layer is on an adjacent boundary (Fig. 10). If that is the case the double node of the adjacent boundary is removed from the boundary and this boundary takes as first node the corresponding node of the quadrilateral (inflated) grid. Triangular grid boundary conditions are then initialized.

Chapter 3: Delaunay Triangulation

3.1 The Delaunay Triangulation

The principle of the Delaunay Triangulation (DT) [9-11] is simple and powerful at the same time. Given a set of vertices, the convex hull around the vertices in the domain is tessellated such that each vertex is assigned the area that is closer to the vertex than to any other vertex. This tessellation is called the Dirichlet tessellation and the set of straight edges that define the borders between the different tiles is the Voronoi diagram. The rule of connection is to connect those vertices whose regions in the Dirichlet tessellation are adjacent or in other terms who share an edge of the Voronoi diagram. These edges are part of the median of the edges in the Delaunay triangulation and thus for each triangle there exists a point where the three edges of the Voronoi diagram intersect the Voronoi vertex. This point is by construction the center of the circle that goes through the three forming vertices of the triangle (Fig. 11).

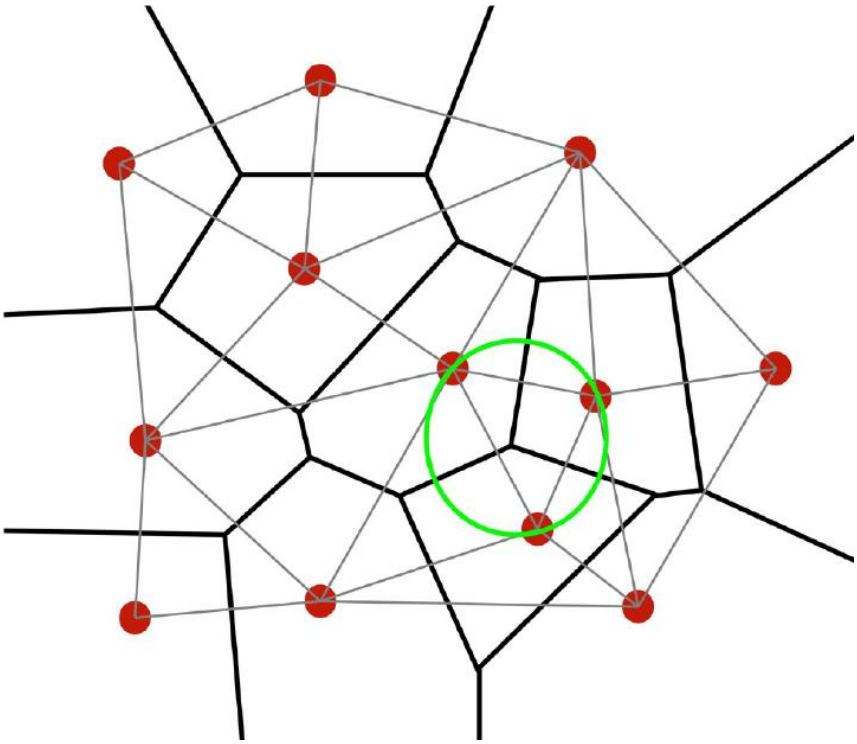


Figure 11: Voronoi diagram, Dirichlet tessellation [12].

In Figure 11 the Voronoi diagram is visualized with black thick lines, the circumcircle of a triangle with green line, and Delaunay triangulation with grey lines [12].

3.1.1 Quality of the Delaunay Triangulation

A few geometric properties of the Delaunay triangulation [13] are the following:

- The DT is unique, except for degenerate cases of more than three vertices that are co-circular.
- The circumcircle around a triangle does not contain any other vertex.
- The DT is a Max-Min triangulation: it maximizes the minimum angles in the grid.

There are many ways to generate Delaunay triangulation efficiently with high element quality. In this diploma thesis the methodology used was developed by Jens-Dominik Müller [12]. An extremely important feature of this methodology developed by Müller is that it was designed for the use of computational fluid dynamics (CFD) solvers. This has the effect of allowing the user to interfere with various parameters to affect the quality of the grid being created. Also, the grid is created with respect to constraints, such as maximum angle (one of the most classical conditions for mesh creation), regularity, size variation and the Laplacian normalization, an operator that when imposed on the grid, greatly improves its quality by smoothing it [12].

3.1.2 Generating the Delaunay Triangulation

Delaunay refinement methods insert new vertices recursively into a valid Delaunay triangulation. Thus any method that generates the Delaunay triangulation incrementally is suitable. Watson's algorithm [14] is more practical. It exploits the circumcircle criterion directly, by finding the region that is covered by all the triangles that contain in their circumcircles the new site that is to be inserted. The cavity can be found by locating a first cell that contains the new vertex and marching from neighbor to neighbor. All cells of the cavity have to share an edge, since the cavity is convex due to the circumcircle criterion. This cavity is re-triangulated by removing all the triangles covering it and reconnecting the edges of the cavity with the new vertex to a new valid DT (Fig. 12). For its simplicity, this is the method that was chosen by Müller [12] for Frontal Delaunay Method.

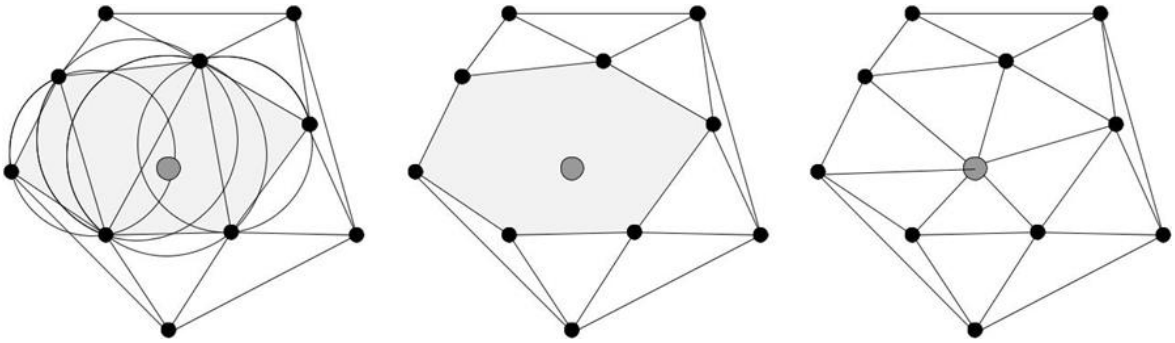


Figure 12: Vertex insertion with Watson's algorithm [14].
Part (a) shows the valid Delaunay triangulation with the new vertex that is contained in various circumcircles. The cavity that has to be re-triangulated is shown shaded in (b); (c) shows the re-triangulated cavity [12].

3.2 Frontal Delaunay Triangulation

The Frontal Delaunay method (FroD) incorporates ideas from the frontal vertex placement strategy of the Advancing Front Method (AFM) [15] to achieve the regularity and the smooth size variation desired into the Delaunay Triangulation (DT), providing the high quality point cloud with the optimal connectivity [12].

In FroD, vertices are generated and inserted in sets of one row at a time. This special treatment of each row distinguishes it from the independently developed method Rebay [16] that is otherwise very similar. After the insertion of each vertex, the DT is reestablished such that there exists a valid DT that covers the entire domain at any stage of the algorithm. However, the Delaunay criterion will not necessarily add the triangles between the new vertex and the frontal edge that they are projected from, as proposed by the insertion strategy, if a triangle is not part of the DT, it will not be formed. Other triangles are formed that close the DT around the newly inserted vertex [12].

In order to make these implicit triangles well-behaved, a minimum distance between vertices has to be imposed, say a certain fraction of the local mesh spacing. This prevents short edges leading to acute triangles. The required distance is taken from a background grid in this non-Euclidean space. The mesh-spacing has to be interpolated at the midpoint between the two points in question. In the case that tow vertices of the same row are too close, they are merged [12].

Chapter 4: Free – Form Deformation (FFD)

4.1 Introduction to FFD Technique

Free-form Deformation is considered as one of the most powerful, versatile and applicable parameterization and deformation techniques of two- as well as three-dimensional shapes, with major applications in a wide range of scientific and technological fields, such as engineering design optimization and computer graphics.

The fundamental idea of FFD is the indirect handling of the object by enclosing it into a parametric space (2D or 3D). Then, by deforming a parametric lattice (defining the parametric space), a deformation of the embedded object is achieved. Based on this idea, a large number of different FFD versions have been developed during the last years.

4.1.1 FFD Methodology Steps

Despite the numerous different FFD versions, either for 2D or 3D geometric models, according to Lamousin and Waggenspack [17] there are four main steps to implement any FFD technique.

Step 1: Construction of the Parametric Lattice.

The shape of interest, is embedded into a two- or three- dimensional parametric space (lattice), consisted by an ordered or arbitrary mesh of control points, depending on the particular FFD version and application. The topology of the parametric lattice should be such that it wraps the embedded shape under study, while the nature of the basis functions that form the FFD lattice has a significant impact on the handing of the embedded model.

Step 2: Embedding the Object within the Lattice.

This stage consist of the assignment of a unique set of parametric coordinates (u, v, w) to each point (x, y, z) of the enclosed shape, where u, v, w are the parametric variables that define the parametric coordinates system. Herein, it is useful to note that the parametric coordinates of each point of the model do not change during the deformation stage, with respect to the specific parametric coordinates system. Due to a lack of analytical methodologies for the aforementioned assignment problem, approximate methodologies have been developed to compute the parametric coordinates, such as Quadtree and Octree subdivision, for 2D and 3D problems respectively.

Step 3: Deforming the Parametric Space.

The deformation of the parametric space is implemented by the movement of the nodes of the control lattice. Especially, as long as the weighted FFD versions are concerned, a deformation could be also achieved only by the modification of the weights of the control points.

Step 4: Evaluating the Effects of the Deformation.

The parametric coordinates of the points (Step 2) are used with the deformed control lattice (Step 3) to evaluate the new locations of the embedded point set. The topology of the original model is then used to reconstruct the deformed object.

4.2 Two-Dimensional B-Spline FFD

In this work, a 2D B-Spline-based FFD version is utilized, which is defined as a mapping from a subspace V $\subset R^2 \rightarrow \bar{V} \subset R^2$. The main idea behind FFD is not to directly deform the shape of interest, but to achieve an indirect manipulation by embedding the object into a parametric control lattice; then by transforming the geometry of the particular lattice, every object enclosed to it undergoes the same deformation.

4.2.1 The Implementation Procedure

Step 1: Construction of the Parametric Lattice.

In this application of FFD, a 2D lattice, formed by a two-variate B-Spline surface, is chosen to take advantage of the benefits B-Splines offer, such as partition of unity, flexibility and convenient geometrical deformation via control points. Additionally, by using a B-Spline lattice, the alteration of a control point does not modulate the entire geometry of the enclosed object, so a focused deformation can be achieved. A planar B-Spline surface is obtained by taking a bidirectional net of control points, two knot vectors, and the products of the univariate B-Spline functions [18]

$$S(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)P_{i,j} \quad , \quad P_{i,j} = (x_{i,j}, y_{i,j}) \tag{19}$$

where $P_{i,j}$ are the control points of the FFD lattice, $n+1, m+1$ are the numbers of control points on each parametric direction and u, v are the parametric variables that define the parametric coordinates system. Let $N_{i,p}(u)$ be a B-Spline Basis function of p degree in u parametric direction and $N_{j,q}(v)$ be a B-Spline Basis function of q degree in v parametric direction, defined over the open and uniform knot vectors U, V respectively

$$U = \{u_0, u_1, \dots, u_{n+p+1}\}$$

$$V = \{v_0, v_1, \dots, v_{m+q+1}\}$$

Concerning the degrees p, q of the Basic functions, they must satisfy the following inequalities

$$1 \leq p \leq n, 1 \leq q \leq m$$

The value of each knot of the U knot vector is calculated by the following formula:

$$u_i = \begin{cases} 0, & 0 \leq i \leq p+1 \\ i-p, & p+1 \leq i \leq n+1 \\ n-p+1, & n+1 \leq i \leq n+p+1 \end{cases}$$

while the calculation of the values of the V knot vector is implemented respectively. The i-th B-spline basis function of degree p, written as $N_{i,p}(u)$, is defined by the utilization of the *Cox-de Boor* recursion formula, as follows:

$$N_{i,p}(u) = \frac{u-u_i}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u) \quad (20)$$

$$N_{i,0} = \begin{cases} 1 & if\ u_i \leq u \leq u_{i+1} \\ 0 & otherwise \end{cases} \quad (21)$$

Given that the parametric coordinates (u_t, v_t) of a point inside the parametric space are known, then the vector of the respective Cartesian coordinates (x_t, x_t) is calculated by the following equation:

$$R(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v)} \quad (22)$$

Step 2: Embedding the Object within the Lattice.

After the construction of the FFD lattice, a quadtree algorithm has to be implemented, so a unique parametric pair of coordinates (u_t, v_t) to be assigned in every single point (x_t, x_t) of the shape to be deformed. For each point of the object, the following algorithm is repeatedly applied [19]:

a. The parametric area is divided into four equal subareas.
b. The Cartesian coordinates of each subarea vertex are calculated.
c. These coordinates are compared to the Cartesian coordinates of the object's point under consideration, in order to identify the subarea in which the corresponding point lies.
d. The latter subarea is divided into four new equal subareas and steps *b-d* are

repeated for a prescribed number of subdivisions, or until a desirable accuracy is achieved. The desired parametric coordinates of the searched point are defined as the parametric coordinates of the center of the subarea, in which the point resides, resulting from the last subdivision [20].

In Fig. 13, a reference airfoil (DU-06-W-200) is embedded into an initial 2D FFD lattice formed by a two-variate B-Spline function. The parametric space is defined by the parametric coordinates (u, v) while a unique pair of parametric coordinates (u_t, v_t) has been assigned to each one of the k boundary points of the airfoil.

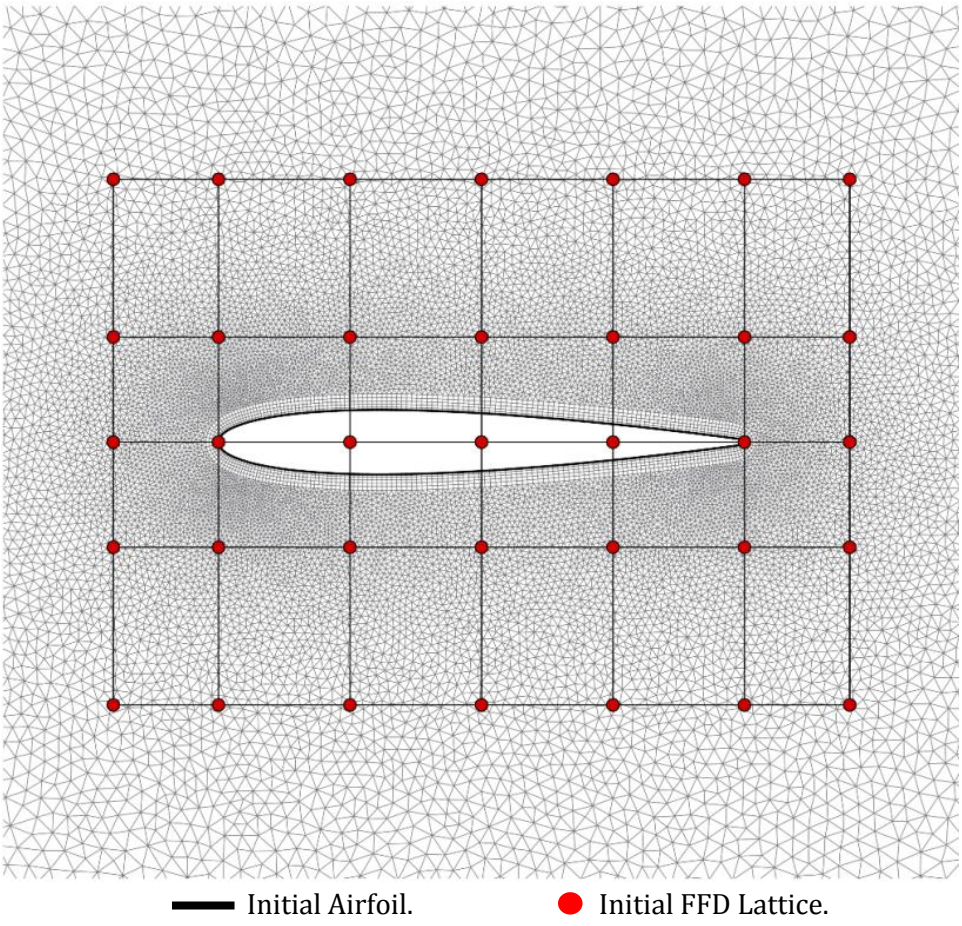
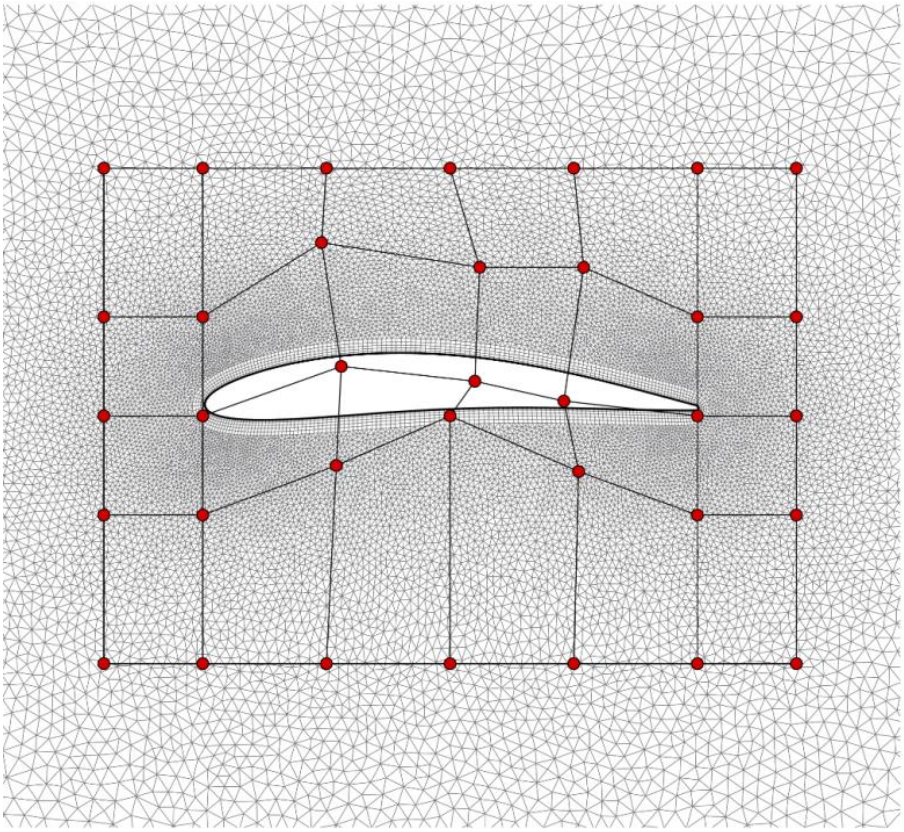


Figure 13: DU-06-W-200 airfoil in a 2D FFD lattice.

Step 3: Deforming the Parametric Space.

In Fig. 14 the deformation of the initial FFD lattice, caused by the movement of the B-Spline control points, is presented.



● Deformed FFD Lattice.

Figure 14: The movement of the control points causes the deformation of the lattice and the deformation of the corresponding space.

Step 4: Evaluating the Effects of the Deformation.

The recovery of the deformed shape, i.e. the calculation of the Cartesian coordinates of the deformed object, is achieved by importing the new Cartesian coordinates of the control points from Step 3 into Eq. (31) and implementing Eq. (31) for each one of the object's boundary points (and grid points), whose parametric coordinates are known from Step 2.

Chapter 5: T2GR Import & Export file formats

5.1 Exported Mesh "dpl" File Format

In this chapter the input and output file formats, which have been developed for the T2GR mesh generation software, will be described. The ".dpl" file format is one of the grid output formats of T2GR. The in-house RANS (Reynolds-Averaged Navier-Stokes) solver uses this format type for its input files. A ".dpl" file is a file format produced by the Delaundo software [12], but slightly adjusted for hybrid grids. An example is given below that demonstrates the structure of the ".dpl" file format.

22645 37413 3588 [the number of nodes: 22645, the number of triangular elements: 37413, the number of quadrilateral elements: 3588].

1 1258 1069 220 752 2246 2926
2 1092 785 116 1503 312 1593
3 2287 1386 2111 3925 5328 3829 …
[for each triangular element a running counter is used (1, 2, 3…), the nodes index at each corner in counterclockwise sense (1258, 1069, 220), and the neighboring elements index (752, 2246, 2926)].
…
37414 19046 19047 1 675 2478 37415 37713 0
37415 19047 19048 2 1 2477 37416 37714 37414
37416 19048 19049 3 2 2063 37417 37715 37415 …
[After all triangular elements, the quadrilateral element are listed. For each quadrilateral element a running counter is used (37414,…), the nodes index at each corner in counterclockwise sense (19046, 19047, 1, 675), and the neighboring elements index (2478, 37415, 37713, 0)].

-11.95076 0.3207567
-11.90385 0.3207575
-11.85718 0.3207597
-11.8106 0.3207618 …
[After all elements, all the nodes (x, y) coordinates are listed].

6 [: the number of boundaries – then all the boundaries are defined].

299 1 [the number of edges in the first boundary: 299, the number of boundary index: 1].
22346 22347 40703
22347 22348 40704
22348 22349 40705
22349 22350 40706 … [the index of each one of the two nodes of the corresponding edge: (22346, 22347), the index of the element adjacent to that edge: 40703].

150 2
22345 22645 41001
22045 22345 40702
21745 22045 40403
21445 21745 40104 …

50 3
437 438 1008
438 439 2098
439 440 2101
440 441 2102 …

50 4
487 488 2309
488 489 749
489 490 2312
490 491 2314
491 492 2316 …

50 5
537 538 1072
538 539 1066
539 540 2359
540 541 384
541 542 2371 …

100 6
587 588 1174
588 589 2210
589 590 2496

590 591 1172
591 592 74 …

5.2 Exported Mesh Case File Formats

The ".cas" file format is the second output format, included in T2GR software. Grid sections attributed with boundary conditions are stored in the ".cas" file, while an ".msh" file is a subset of the ".cas" file, containing only those sections relating to the computational mesh (no boundary conditions included). The currently defined grid sections are explained in the following sections:

Comment ID: 0

Header ID: 1

Dimensions ID: 2

Nodes ID: 10

Cells ID: 12

Faces ID: 13

Other (Non-Grid) Case Sections ID: 45

Each section is attributed with an ID, in order to initialize it and distinguish it [21].

5.2.1 Comment Sections

Comment sections can appear anywhere in the file (except within other sections), as:

(0 "comment text")

The purpose of comment sections is to precede each long sections or group of related sections by a comment section, explaining what is to follow. The comment sections make the content of the file readable and straightforward [21].

5.2.2 Header

Header sections can appear anywhere in the file (except within other sections), as:

(1 "exported from myMesher V1.03")

The purpose of this section is to identify the software that generated the file. Although this section can appear anywhere, it is typically one of the first sections in the file. Additional header sections indicate other software that may have been used in generating the file. This provides a history mechanism, showing where the file came from and how it was processed [21].

5.2.3 Dimensions

The dimensions of the grid appear as:

(2 ND)

where ND is equal to 2 or 3. This section is supported as a check that the grid has the appropriate dimensions, however this section is optional and its existence will not affect the case file construction [21].

5.2.4 Nodes

The nodes section appears as:

(10 (zone-ID first-index last-index type ND))

If zone-ID is zero, this provides the total number of nodes in the mesh. In this case, the "first-index" will be equal to 1, the "last-index" will be the total number of nodes, the "type" is equal to 0, *"ND"* is omitted, and there are no coordinates following (the parentheses for the coordinates are omitted as well), for example: 10 (0 1 2d5 0).

If zone-ID is greater than zero, it indicates the zone to which the nodes belong. In this case, "first-index" and "last-index" are the indices of the nodes in the zone. The values of "last-index" in each zone must be less than, or equal to, the value in the declaration section; "type" indicates the type of nodes in the zone. The following values are used to indicate the node type:

- 0 for "virtual" nodes.
- 1 for no (any) type.
- 2 for boundary nodes.

Nodes of type 0 are ignored but types 1 and 2 are read and written. "*ND"* is an optional argument that indicates the dimensionality of the node data, where *ND* =2 or =3. If the number of dimensions in the grid is 2, as specified in the Dimensions or in the node header, then only x and y coordinates are present in each line. The following is an example of a 2-dimensional grid:

(10 (1 1 2d5 1 2)(
1.500000e-01 2.500000e-02
1.625000e-01 1.250000e-02
.
.
.

1.750000e-01 0.000000e+00

2.000000e-01 2.500000e-02

1.875000e-01 1.250000e-02

))

As the grid connectivity is composed of integers representing pointers, using hexadecimal format conserves space in the file and provides for faster file input and output. The header indices are also in hexadecimal format, so that they match the indices in the bodies of the grid connectivity sections. The "zone-ID" and "type" are also in hexadecimal format for consistency [21].

5.2.5 Cells

The declaration section for cells is similar to that for nodes:

(12 (zone-ID first-index last-index type element-type))

When "zone-ID" = 0, it indicates that it is a declaration of the total number of cells. If "last-index" = 0, then there are no cells in the grid. This is useful when the file contains only a surface mesh, as it serves to alert the solver (*Fluent* for example) that it cannot be used in the solver. In a declaration section, the "type" has a value of 0, while the "element-type" is not present, for example (12 (0 1 3e3 0)) states that there are 3e3 (hexadecimal) = 995 cells in the grid. This declaration section is required and must precede the regular cell sections. The "element-type" in a regular cell section header indicates the type of cells in the section, as shown in the following table:

element-type	Description	Nodes/Cell	Faces/cell
0	Mixed		
1	Triangular	3	3
2	Tetrahedral	4	4
3	Quadrilateral	5	4
4	Hexahedral	8	6
5	Pyramid	5	5
6	Wedge	6	5
7	Polyhedral	NN	NF

*where *NN* and *NF* will vary, depending on the specific polyhedral cell.

Regular cell sections have no body, but they have a header of the same format, where "first-index" and "last-index" indicate the range for the particular zone, "type" indicates whether the cell zone is fluid (type = 1) or solid (type = 17).

A "type" = 0 indicates a dead zone and will be skipped when the file is read in solution mode in the solver. If a zone is of mixed type ("element-type" = 0), it will have a body that lists the "element-type" of each cell [21].

5.2.6 Faces

The face section has a header with the same format as that for cells, but with a section index of 13. The format is:

(12 (zone-ID first-index last-index bc-type face-type))

where [21]:

- "zone-ID" = zone ID of the face section,
- "first-index" = index of the first face in the list,
- "last-index" = index of the last face in the list,
- "bc-type" = ID of the boundary condition represented by the face section,
- "face-type" = ID of the type(s) of face(s) in the section.

A "zone-id" = 0 indicates a declaration section, which provides a count of the total number of faces in the grid. Such a section omits the "bc-type" and is not followed by a body with further information.

A non-zero "zone-id" indicates a regular face section and will be followed by a body containing information about the grid connectivity. Each line describes one face and appears as follows:

$$n_0 \qquad n_1 \qquad n_2 \qquad c_0 \qquad c_1$$

Where n* are the defining nodes (vertices) of the face, and c* are the adjacent cells. All cells, faces, and nodes have positive indices. If a face has a cell only on one side, then either c_0 or c_1 is zero. For files containing only a boundary mesh, both these values are zero.

"bc-type" indicates the ID of the boundary condition represented by the face section. The current valid boundary condition types are defined in the following table:

"bc-type"	Description
2	Interior
3	Wall
4	Pressure-inlet
5	Pressure-outlet
7	Symmetry
8	Periodic-shadow

9	Pressure-far-field
10	Velocity-inlet
12	Periodic
14	Fan, porous-jump, radiator
20	Mass-flow-inlet
24	Interface
31	Parent (hanging node)
36	Outflow
37	Axis

"face-type" indicates the type of faces in the zone, as defined in the following table:

"face-type"	**Description**	**Nodes/face**
0	Mixed	-/-
2	Linear	2
3	Triangular	3
4	Quadrilateral	4
5	Polygonal	NN

*where *NN* will vary, depending on the specific polygonal face.

5.2.7 Other (Non-Grid) Case Sections

The following sections store boundary conditions, material properties, and solver control settings. Grid generators and other preprocessors need only to provide the section header and leave the list of conditions empty as:

```
(45 (zone-ID zone-type zone-name domain-id) ( ))
```

The "zone-ID" is in decimal format, this is in contrast to the use of hexadecimal in the grid sections. The "zone-type" follows the same table as the "bc-type".

ANSYS Fluent allows the wall type to be assigned to face zones both on the inside and on the boundaries of the domain. Some zone types are valid only for certain types of grid components. For example, cell (element) zones can be assigned only one of the following types (fluid, solid). The zone-name is a user-specified label for

the zone. It must be a valid Scheme symbol and is written without quotes. The rules for a valid zone-name (Scheme symbol) are as follows:

- The first character must be a lowercase letter or a special-initial.
- Each subsequent character must be a lowercase letter, a special-initial, a digit, or a special-subsequent.

Some examples of zone sections, produced by grid generators and preprocessors, are as follows:

```
(45 (1 fluid fuel 1) ( ) )
(45 (8 pressure-inlet pressure-inlet-8 2) ( ) )
(45 (2 wall wing-skin 3) ( ) )
(45 (3 symmetry mid-plane 1) ( ) )
```

The "domain-id" is an integer that appears after the zone name, associating the boundary condition with a particular phase or mixture (sometimes referred to as phase-domains and mixture-domains) [22].

5.3 Obj Geometry Definition

"obj" is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer animation package. The "obj" file format is a simple data format that represents 2D & 3D geometries. However, in the current software only 2D axisymmetric geometries can be used for mesh generation. Each curve included in an "obj" file contains the control polygon points, the degree and the knot vector of the curve. These parameters define a B-spline curve.

Since an "obj" file can contain more than one curves, a special class named **model** has been developed. Within this class there are functions of reading and collecting data from the "obj" files and for the creation of the corresponding B-spline curves. Once the curves have been successfully created, the corresponding Cartesian coordinates are stored in appropriate structures to be visualized and made available to software tools and functions.

In summary, the developed software receives a list of parametric curves, then stores their data and creates the corresponding B-spline curves, finally refreshes the boundary structures by giving them the Cartesian coordinates. The corresponding "obj" file processing algorithm was developed entirely within this diploma thesis.

Chapter 6: Useful Algorithms

6.1 Distribution of Points using Geometric Progression

Geometric progression is the sequence in which no term is equal to zero and for two consecutive terms of α_ν , $\alpha_{\nu+1}$ holds

$$\frac{\alpha_{\nu+1}}{\alpha_\nu} = \lambda \tag{23}$$

where λ is a non-zero fixed quantity. The quantity λ is called the ratio of geometric progression. For better control over the nodes of the quadrilateral elements close to the corresponding boundaries, a function was created where, using the geometric progression, each layer of the quadrilateral elements is defined according to an initial step (height of the first layer) and a growth rate.

The general form of geometric progression is as follows:

$$\alpha_\nu = \alpha_1 \lambda^{\nu-1}. \tag{24}$$

The sum of the ν first terms of geometric progression (the total height of the inflation zone) is given as (for the different values of λ):

$\lambda > 1$

$$\sum_{i=1}^{\nu} \alpha_i = a_1 \frac{\lambda^\nu - 1}{\lambda - 1} \tag{25}$$

$\lambda = 1$

$$\sum_{i=1}^{\nu} \alpha_i = \nu a_1 \tag{26}$$

$\lambda < 1$

$$\sum_{i=1}^{\nu} \alpha_i = \frac{\alpha_1}{1 - \lambda} \tag{27}$$

For a desired total height of the inflation zone (the zone with quadrilateral elements), a_1 is the initial step and λ remains unknown. For $\lambda = 1$ and $\lambda < 1$ it is straightforward to compute λ from the total height of the inflation zone. However, for $\lambda > 1$ a polynomial of ν_{th} degree results for λ. In order to find the root of the polynomial, the Newton Raphson methodology is used.

6.2 Newton Raphson Methodology

In numerical analysis, Newton's method (also known as the Newton–Raphson method), named after Isaac Newton and Joseph Raphson, is a method for finding successively better approximations to the roots of a real-valued function. The Newton–Raphson method in one variable is implemented as follows [23]:

The method starts with a function f defined over the real numbers x, the function's derivative f' and an initial guess x_0 for the root of function f. If the function satisfies the assumptions made in the derivation of the formula and the initial guess is close, then a better approximation x_1 is given as

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{28}$$

Geometrically $(x_1, 0)$ is the intersection of the x-axis and the tangent of the graph of f at $(x_0, f(x_0))$. The process is repeated until a sufficiently accuracy value is reached

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{29}$$

Fig. 15 explains graphically one step of the Newton-Raphson method.

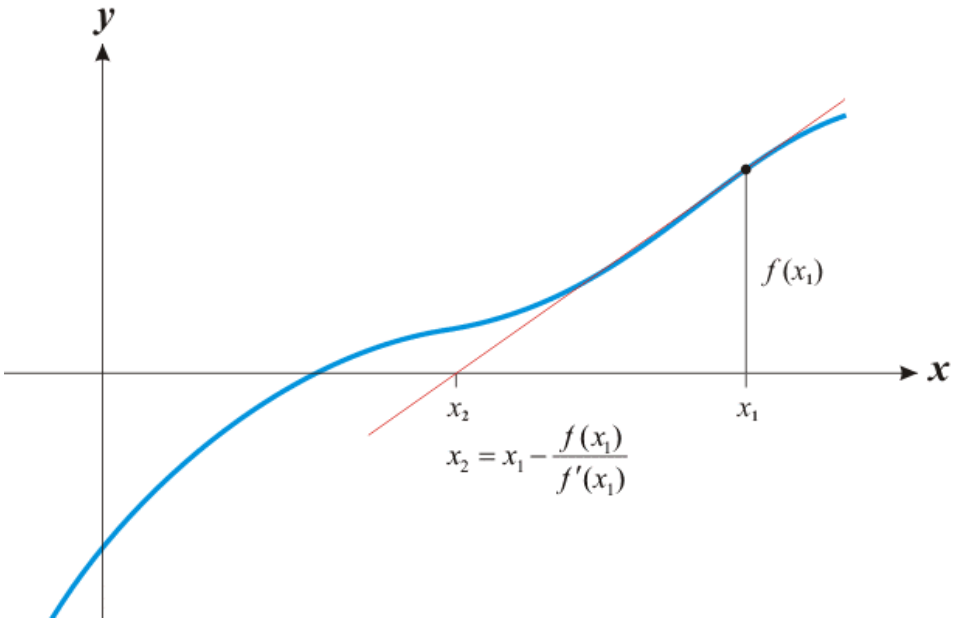


Figure 15: Graphical representation of Newton-Raphson method [24].

In Fig. 15, we start from x_1 approximation of the root, and move to x_2 approximation of the root of f, using the Newton-Raphson method. In the T2GR it was necessary to find the root of the polynomial in Eq. (30)

$$\sum_{i=1}^{\nu} \alpha_i = a_1 \frac{\lambda^{\nu}-1}{\lambda-1} \Rightarrow \sum_{i=1}^{\nu} \alpha_i (\lambda - 1) = a_1(\lambda^{\nu} - 1) \Rightarrow \lambda^{\nu} - \lambda \frac{\sum_{i=1}^{\nu} \alpha_i}{\alpha_1} + \frac{(\sum_{i=1}^{\nu} \alpha_i - \alpha_1)}{\alpha_1} = 0$$

$$f(\lambda) = \lambda^{\nu} - \lambda \frac{\sum_{i=1}^{\nu} \alpha_i}{\alpha_1} + \frac{(\sum_{i=1}^{\nu} \alpha_i) - \alpha_1}{\alpha_1} \tag{30}$$

$$f'(\lambda) = \nu\lambda^{\nu-1} - \frac{\sum_{i=1}^{\nu} \alpha_i}{\alpha_1} \tag{31}$$

We use as initial guess $x_0 = 5$ and as termination condition the accuracy of the root at least equal to $1 \cdot 10^{-13}$. The root approach formula changes when its multiplicity is greater than one. In this case, the mathematical type of root approach changes as follows

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)} \tag{32}$$

where m is the multiplicity of the root. This process helps to improve the speed of convergence towards the root. The software developed in this work does not apply this technique.

6.3 Cosine Distributions

The usage of the cosine distribution is common in aerodynamic design, as it is ideal for local densification of airfoil nodes near the leading and/or trailing edges of the airfoil. The densification of the surface nodes also affects the density of the computational grid on these areas. In order to create a double-cosine distribution on the nodes of a boundary, the interval [0, π] is first divided into as many segments as the desired ones on the corresponding boundary. Then, for each segment within [0, π] the corresponding cosine value is calculated, which corresponds on the x-coordinate of the node along the unitary chord of the airfoil. For a simple cosine distribution either the [0,π/2] interval is used, or the [π/2, π], based on the desired density of nodes close to the leading or the trailing edge of the airfoil.

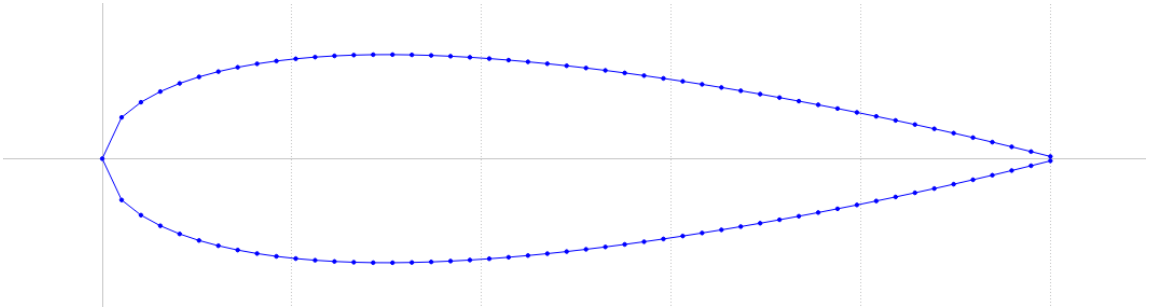


Figure 16: NACA0012 (equal spacing).

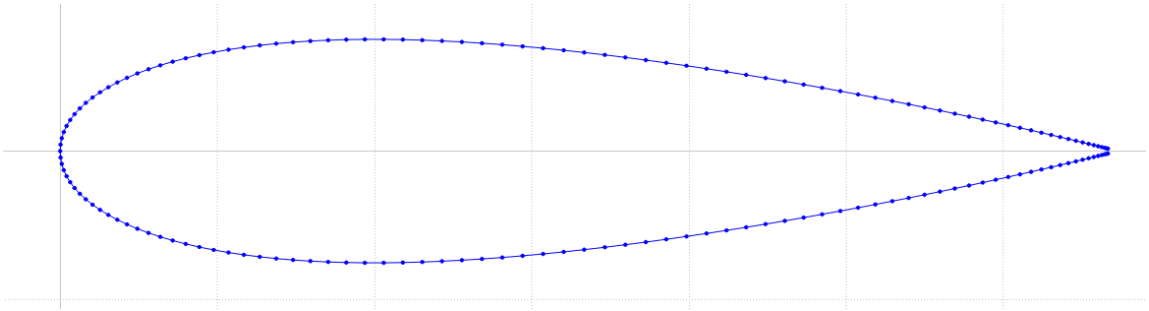


Figure 17: NACA0012 (double cosine distribution).

In Figure 16 a NACA0012 airfoil is presented with an equal between the nodes in x-direction. It is obvious that the geometry representation is much more accurate in Fig. 17, where a double cosine distribution has been applied for the airfoil surface nodes in x-direction.

6.4 B-Spline Interpolation

Spline interpolation is a form of interpolation where the interpolant is a special type of a piecewise polynomial called a spline. Spline interpolation is often preferred over polynomial interpolation because the interpolation error can be made small, even when using low degree polynomials for the spline. Suppose we are given a set of points $\{Q_k\}, k = 0, \dots, n$ and we want to interpolate these points with a p^{th} degree non-rational B-spline curve. If we assign a parameter value, $\bar{u}_\text{k}$, to each Q_k, and select an appropriate knot vector $U = \{u_0, \dots, u_m\}$, we can set up the $(n+1)\cdot(n+1)$ system of linear equations [18]

$$Q_k = C(\bar{u}_\text{k}) = \sum_{i=0}^{n} N_{i,p}(\bar{u}_\text{k}) P_i. \tag{33}$$

For the creation of the above system of linear equations we should define $\bar{u}_\text{k}$ and U. Let U (the knot vector) be a set of $m + 1$ non-decreasing numbers (knots), $u_0 <= u_1 <= u_2 <= \dots <= u_m$. The knots can be considered as division points that subdivide the interval $[u_0, u_m]$ into knot spans. All B-spline basis functions are supposed to have their domain on $[u_0, u_m]$. We frequently use $u_0 = 0$ and $u_m = 1$ so that the domain is the closed interval [0,1]. Their choice affects the shape and parameterization of the curve. In this diploma thesis the calculation of $\bar{u}_\text{k}$ is as follows:

$$\bar{u}_\text{k} = \bar{u}_{\text{k-1}} + \frac{|Q_k - Q_{k-1}|}{d} \qquad k = 1, \dots, n-1 \tag{34}$$

$$\bar{u}_0 = 0 \qquad k = 0 \tag{35}$$

$$\bar{u}_{\mathrm{n}} = 1 \qquad k = n \tag{36}$$

$$d = \sum_{k=1}^{n} \sqrt{|Q_k - Q_{k-1}|} \tag{37}$$

This is the most widely used method, and is generally adequate. It also provides a good parameterization to the curve, in the sense that it approximates a uniform parameterization. Hence, to define B-spline basis functions, we need one more parameter, the degree p. The i^{th} B-spline basis function of degree p, $N_{i,p}(\bar{u}_{\mathrm{k}})$, is defined recursively as follows:

$$N_{i,0} = \begin{cases} 1 & if\ u_i \le u\ \le u_{i+1} \\ 0 & otherwise \end{cases} \tag{38}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i}\, N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{39}$$

The knots can be easily calculated with the technique of averaging

$$u_0 = \cdots = u_p = 0 \qquad j = 0, \dots, p \tag{40}$$

$$u_{m-p} = \cdots = u_m = 1 \qquad j = m - p, \dots, 1 \tag{41}$$

$$u_{j+p} = \frac{1}{p} \sum_{i=j}^{j+p-1} \bar{u}_i \qquad j = 1, \dots, n - p \tag{42}$$

With this method the knots reflect the distribution of the $\bar{u}_{\mathrm{k}}$. Furthermore, combining all this equations, leads to a system, which is totally positive and banded with a semi-bandwidth less than p, that is $N_{i,p}(\bar{u}_{\mathrm{k}}) = 0$ if $|\, i - k\, | \ge p$. Hence, it can be solved by Gaussian elimination without pivoting [18].

Chapter 7: Generating Mesh through the GUI

7.1 T2GR Software

By launching T2GR software, the application's main window is shown, as depicted in Fig. 18. Before the creation of a new project, the main window is composed only by the menu bar, which is a graphical control element, containing drop down menus and the status bar. The latter is a graphical control element that poses an information area, located at the bottom of the main window. At this stage, the menu bar contains only the *File*, *About* and *Help* menus; however, more drop down menus will be added dynamically to the menu bar after the creation of a new project, depending on the user's actions, in order to support the software's functionality.

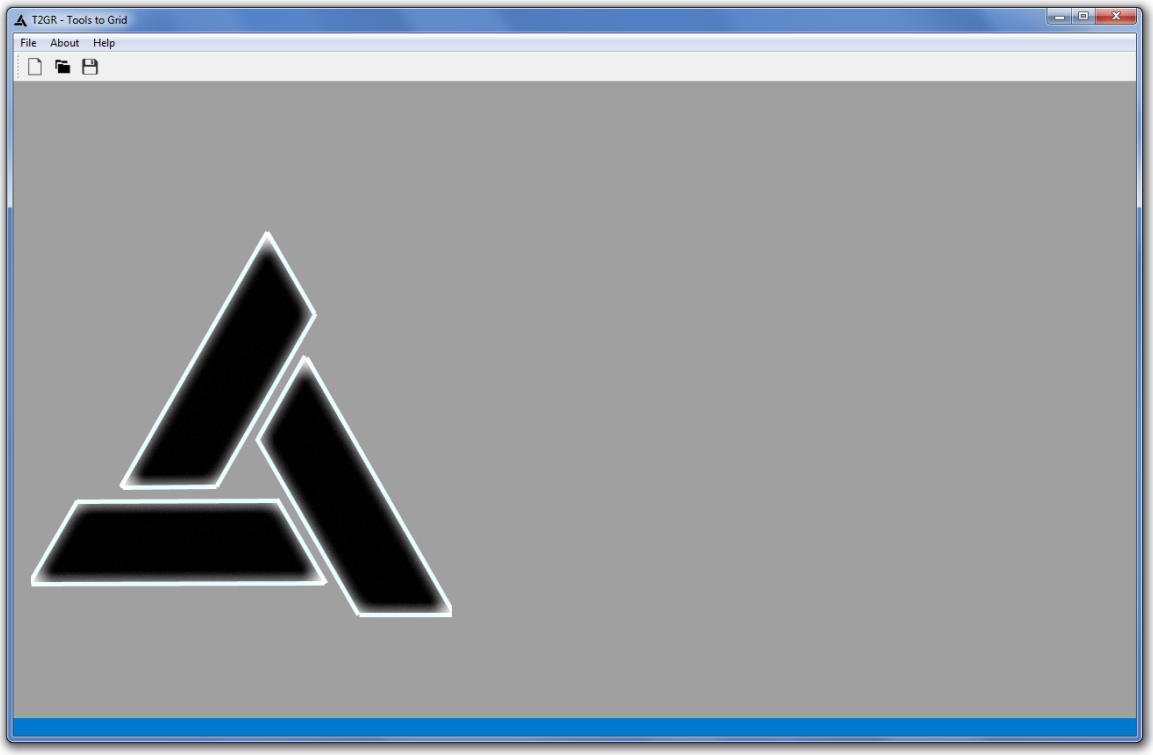


Figure 18: The main window of T2GR software.

From the File menu the user can chose one of the following actions:

- *New*: Creates a new project.
- *Open*: Opens an existing project.
- *Save*: Saves the current project.
- *Exit*: Exits the application.

7.2 New T2GR Project

In order to create a new project, two options can be used; the user can click on the empty file symbol, following the sequence File ->New Project, or use a short cut key (ctrl + N). After the user creates a new project a new dialog appears, with the OpenGL viewer on the right side and the main functionalities of the software on the left side of the screen. In addition to the new dialog that appeared, three new drop down menus appear and three new tools on the main toolbar, as shown in Fig. 19.

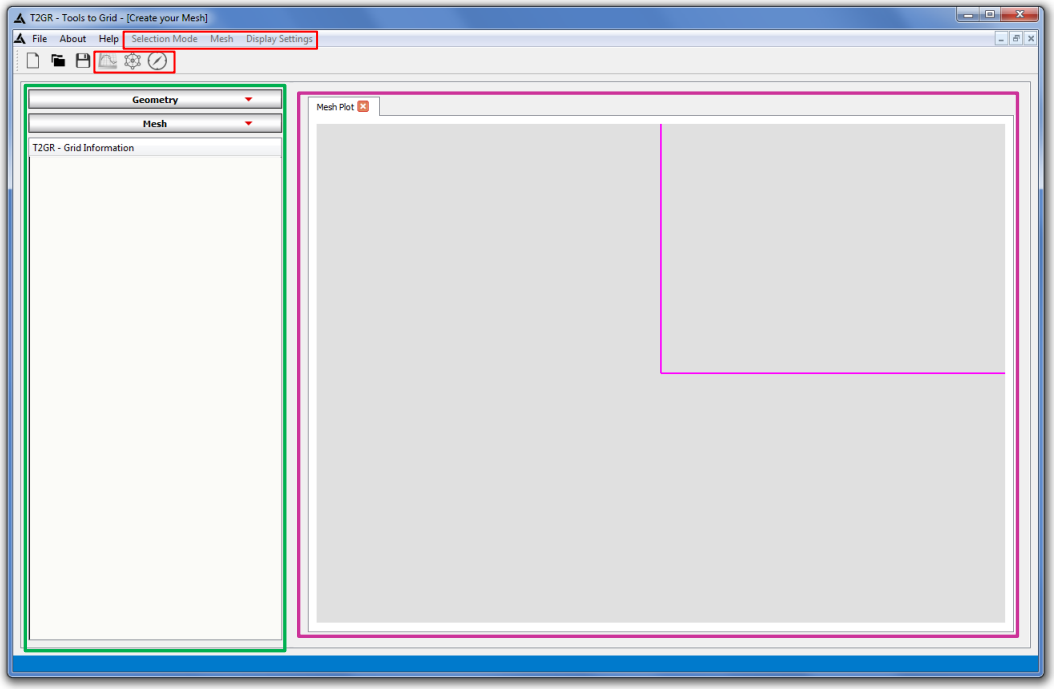
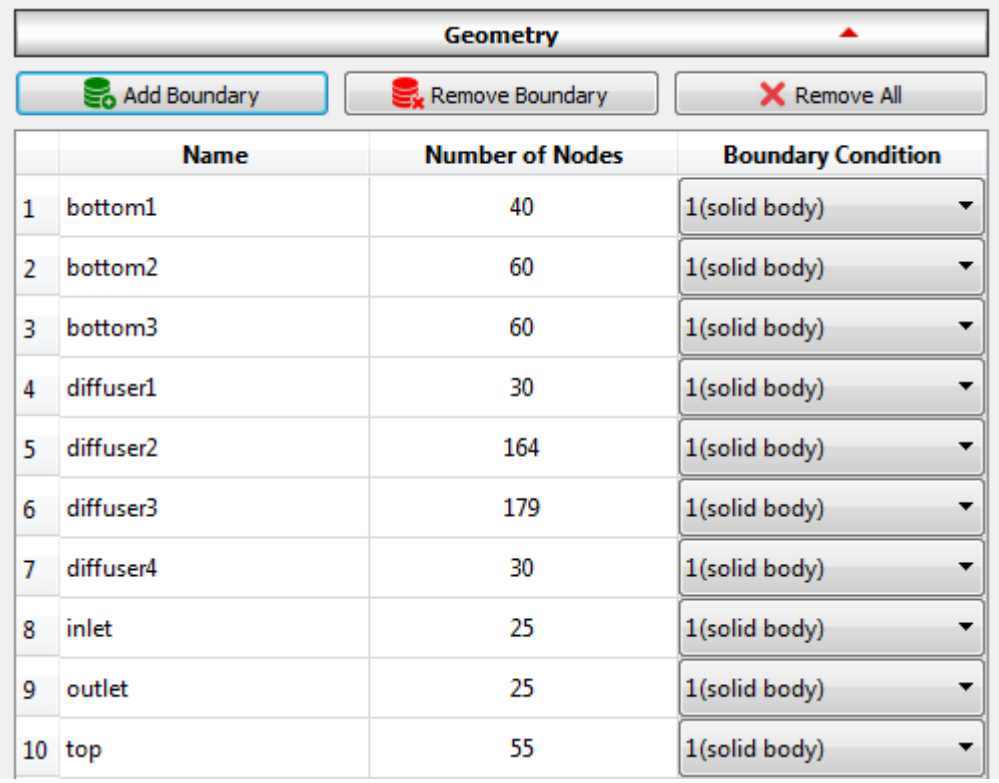


Figure 19: The main window after the creation of a new project.

7.3 Functionalities

The software offers the user a comprehensible graphical environment with step-by-step guidance. Initially, the user can add and remove boundaries through "txt" and "obj" files, by simply clicking the *Add Boundary* or *Remove Boundary* buttons at the *Geometry* Tab (Fig. 20).



	Name	Number of Nodes	Boundary Condition
1	bottom1	40	1(solid body)
2	bottom2	60	1(solid body)
3	bottom3	60	1(solid body)
4	diffuser1	30	1(solid body)
5	diffuser2	164	1(solid body)
6	diffuser3	179	1(solid body)
7	diffuser4	30	1(solid body)
8	inlet	25	1(solid body)
9	outlet	25	1(solid body)
10	top	55	1(solid body)

Figure 20: Geometry Tab.

As shown in Fig. 20, the necessary information is provided for each boundary, such as *name*, *number of nodes*, *boundary condition*. Whenever the user adds or removes a file from the boundaries table, the viewer automatically refreshes the graphics on the screen.

There are two options for computational grids (*triangular*, *hybrid*). The *Mesh* tab includes these two options (Fig. 21). Each of these selections works in a different way, with the corresponding requirements depending on the grid type.

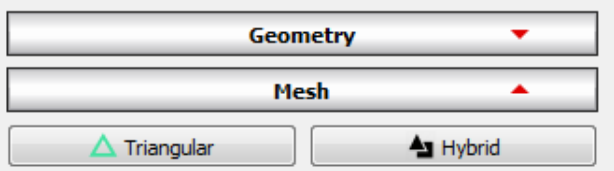


Figure 21: The Mesh Tab.

7.3.1 Triangular Mesh Generation

For the triangular grid generation, the parameters to triangulate are displayed step by step. The first step is to set boundaries' nodes orientation (Fig. 22). In the second step (after the user clicks *Next*), the control file parameters dialog is shown (Fig. 23).

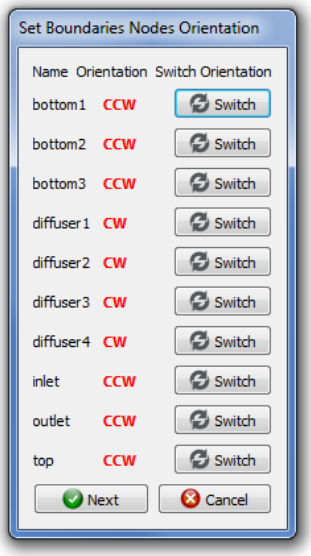


Figure 22: Boundaries' nodes orientation dialog box.

The control file dialog includes all the parameters for the construction of the triangular grid (Delaundo code).

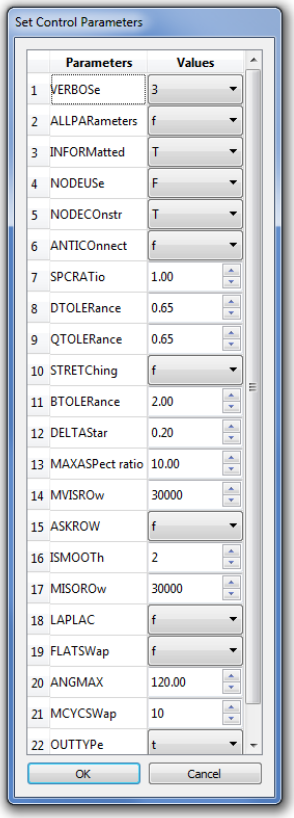


Figure 23: Control file parameters for the triangular grid.

[http://www.ae.metu.edu.tr/tuncer/ae546/prj/delaundo/DelaundoUserManual.pdf]

7.3.2 Hybrid Grid Generation

In order to create a hybrid grid, the user first has to choose the inflated boundaries. The user controls (*smoothing factor*, *orientation*, *frontal smoothing gradient*, *end smoothing gradient*) the direction in which the quadrilateral mesh will be created, the slope of the first layer and the nodes orientation (Fig. 24).

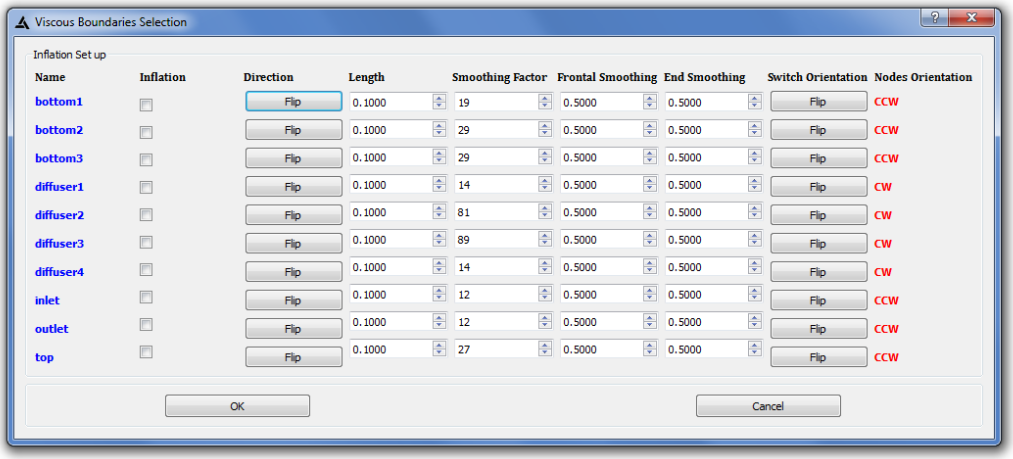


Figure 24: Inflation Setup.

In the next dialog that appears, the user controls the parameters of the quadrilateral mesh (*initial step*, *growth rate*) as in Fig. 25.

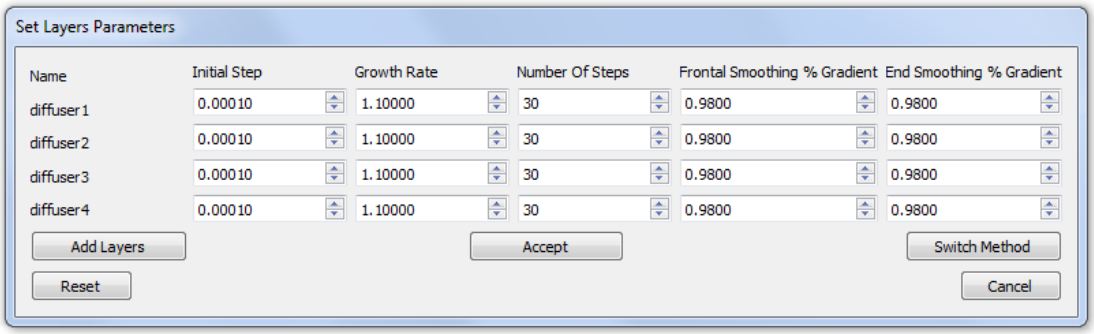


Figure 25: Inflation Setup (layers parameters setup).

It should be noted that the quadrilateral mesh is created and dynamically altered. That means the user can influence the above parameters and directly modify the shape of the grid (Fig. 26).

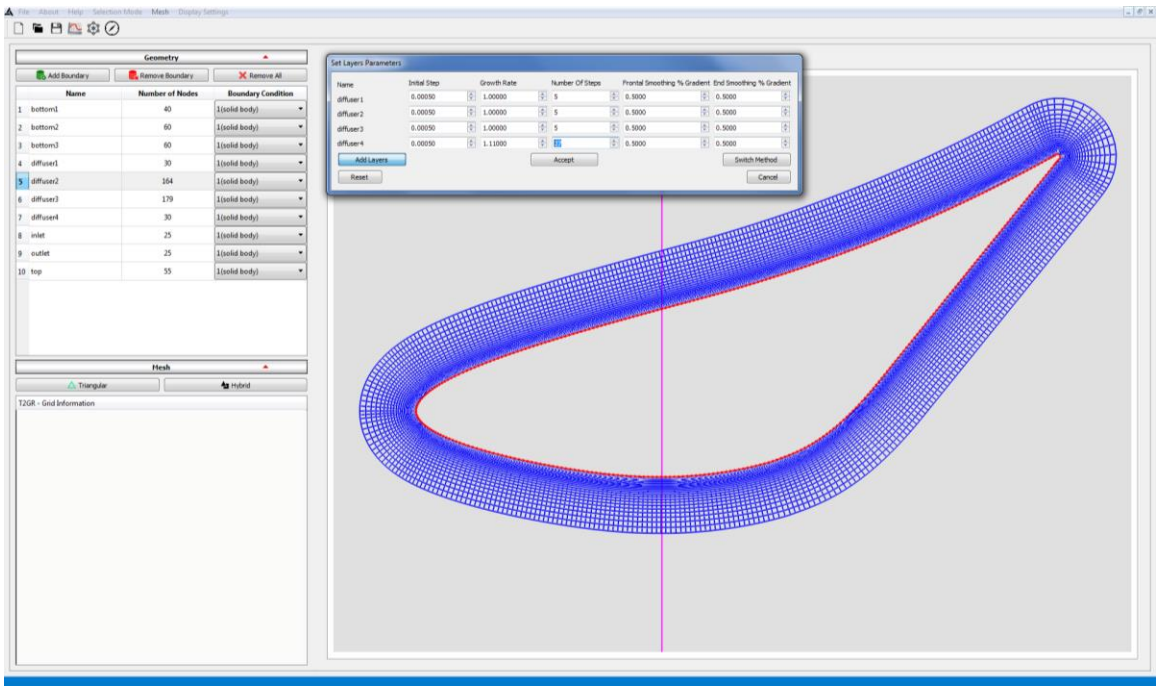


Figure 26: Quadrilateral mesh procedure.

Changes are finalized as soon as the *Accept* button is pressed. The last dialog that appears is to initialize the parameters for creating the triangular mesh (Fig. 23). Thus, pressing the *Ok* button displays the hybrid mesh. Once the mesh has been created, either triangular or hybrid, all of its necessary information (number of nodes, number of triangles, number of quads) appear on the left side of the screen. In addition, the mesh drop down menu features are now available to the user. If the user selects this menu while the grid has not been successfully created, a suitable message box informs the user for the reason the mesh menu is deactivated. The blue color refers to the quadrilateral mesh while the red refers to the triangular mesh (Fig. 27). After the mesh has been created, the user can *hide*, *display*, and *delete* the mesh from the *Mesh* menu.

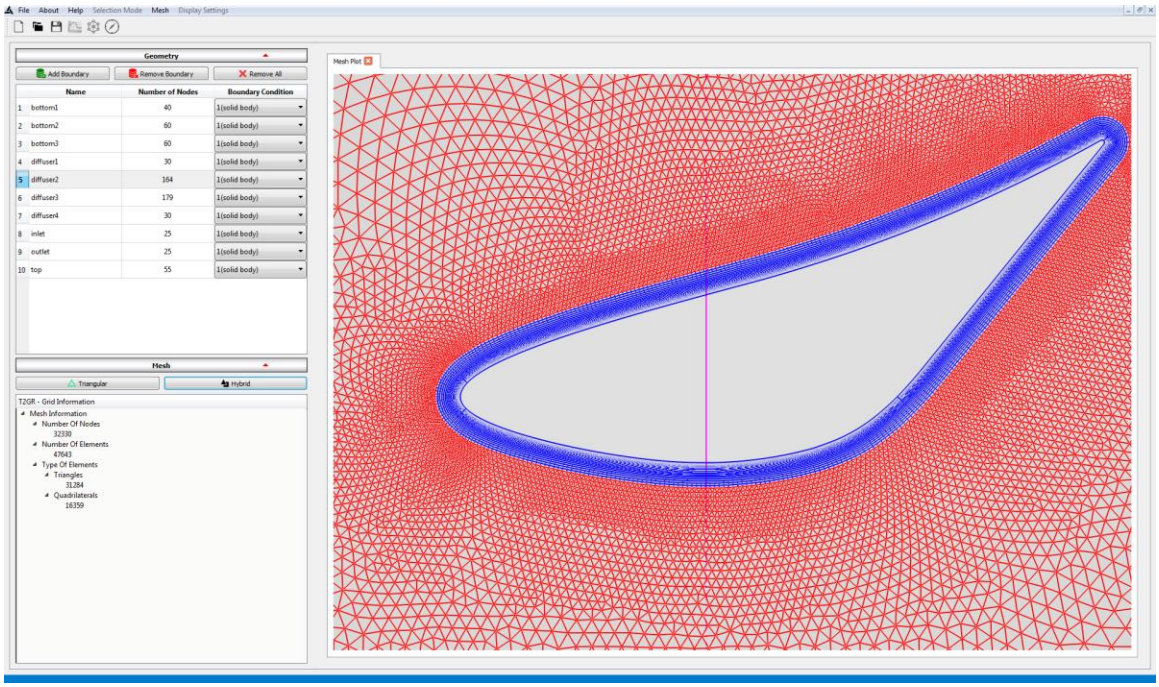


Figure 27: Hybrid mesh generation.

7.4 T2GR Tools

The T2GR software provides the user a variety of tools to edit either the grid or the geometry. These tools are based on geometric methodologies, such as B-Spline interpolation, Free Form Deformation (FFD) and geometric cosine distribution. All methodologies will be described, step by step, for their proper implementation.

7.4.1 Boundary Tools

On any of the available boundaries, the number of nodes and spacing can be changed, which is extremely useful for mesh generation. Thus, through the main toolbar and the *tools* (*B-spline interpolation*, *spacing*) these operations can be initialized. It should also be mentioned that modifications occurring on any boundary are automatically saved. For example, when a modification is made to the distribution of the nodes and then a modification is made to the number of nodes, the software redistributes the new number of nodes, according to the current distribution. In addition, if the user clicks the *discard* button in the B-spline dialog, it resets the geometry to the initial state. As a result, the user has a useful tool to process the imported geometries.

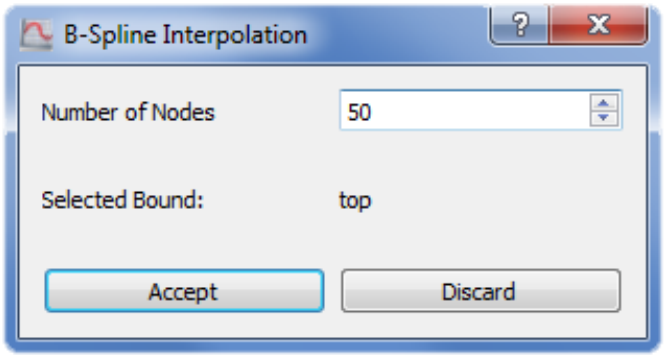


Figure 28: B-Spline Interpolation tool.

The B-spline interpolation tool is used whenever the user wants to modify the number of nodes on a selected boundary.

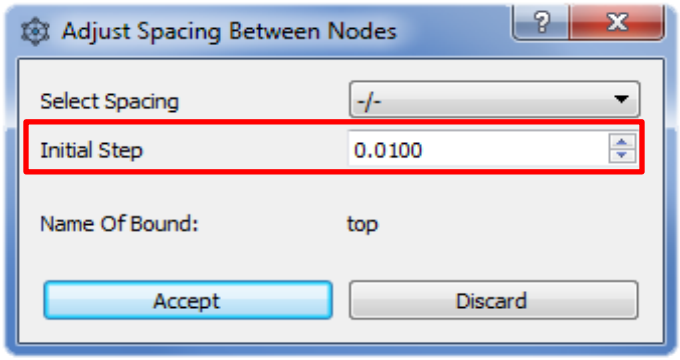


Figure 29: Adjust spacing between nodes.

The initial step spin box (Fig. 29) is used only for the geometric progression distribution, as the value it defines will be the initial length of each edge. If the user selects –/– then an appropriate message box will appear, informing the user that the option does not match to any of the available distributions. These functions are available for any boundary selected by the user through the selection mode. The selection of the boundary is performed by simply clicking on it or near it. The user can always distinguish which boundary is selected as it changes color (Fig. 30).

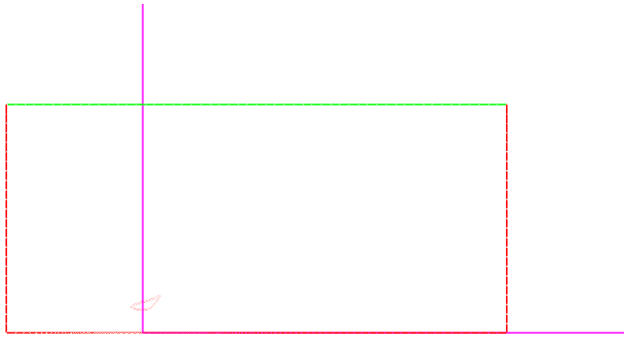


Figure 30: Selected boundary (green color) and non-selected boundaries (red color)

Last but not least, a styling menu was added in order to help users to personalize their T2GR workspace. The user, through *Display Settings* menu, has the ability to change the color through an RGBA (Red, Green, Blue, Alpha) pallet, on boundaries, nodes and the background. Furthermore, there is also an option to increase or decrease the width of lines and the size of points (Fig. 31).

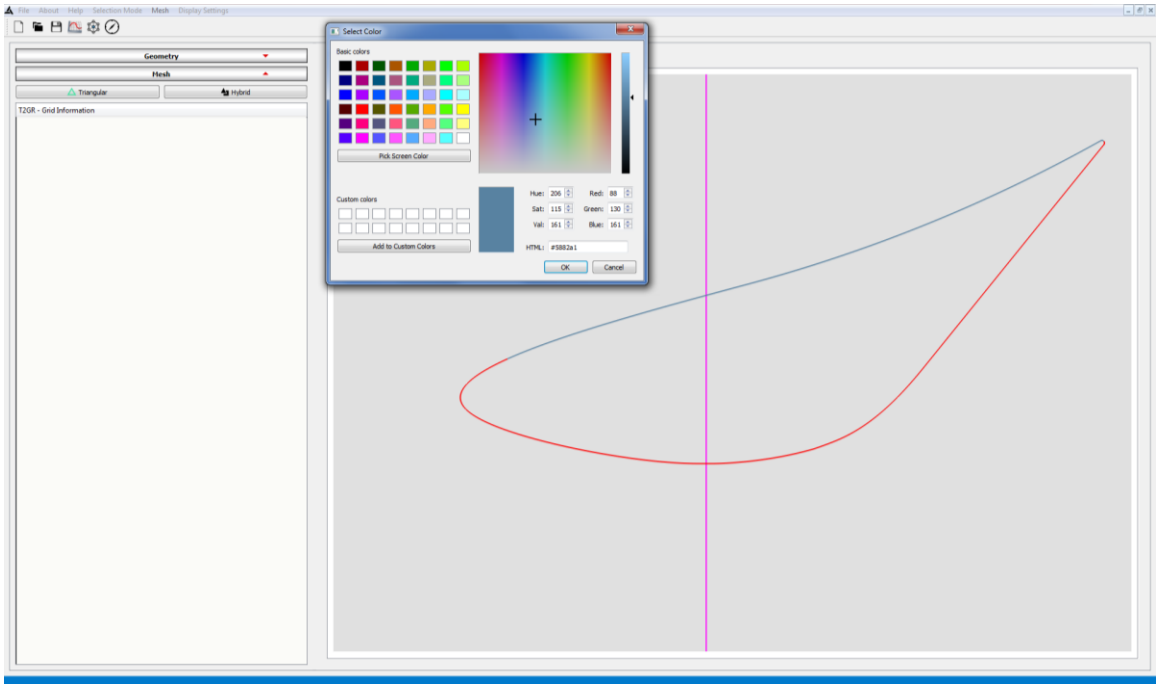


Figure 31: The RGBA pallet and the applied color.

7.4.2 FFD Tool (Grid)

Once the grid has been created, the user can now use the FFD tool. First, the user has to select two points on the grid, with the cursor, to determine the size of the surface lattice, as shown in Fig. 32.

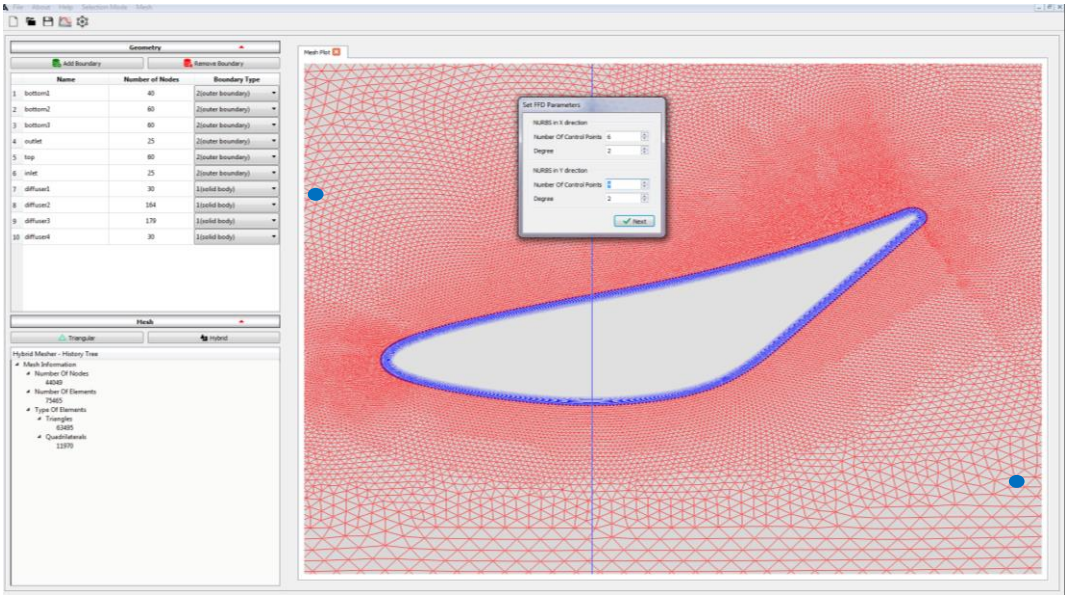


Figure 32: Surface lattice initialization.

On the dialog that appears the user should select the number of control points needed on the x-axis and the number of control points on the y-axis. After clicking "*Next*" the surface lattice is created (Fig. 33).

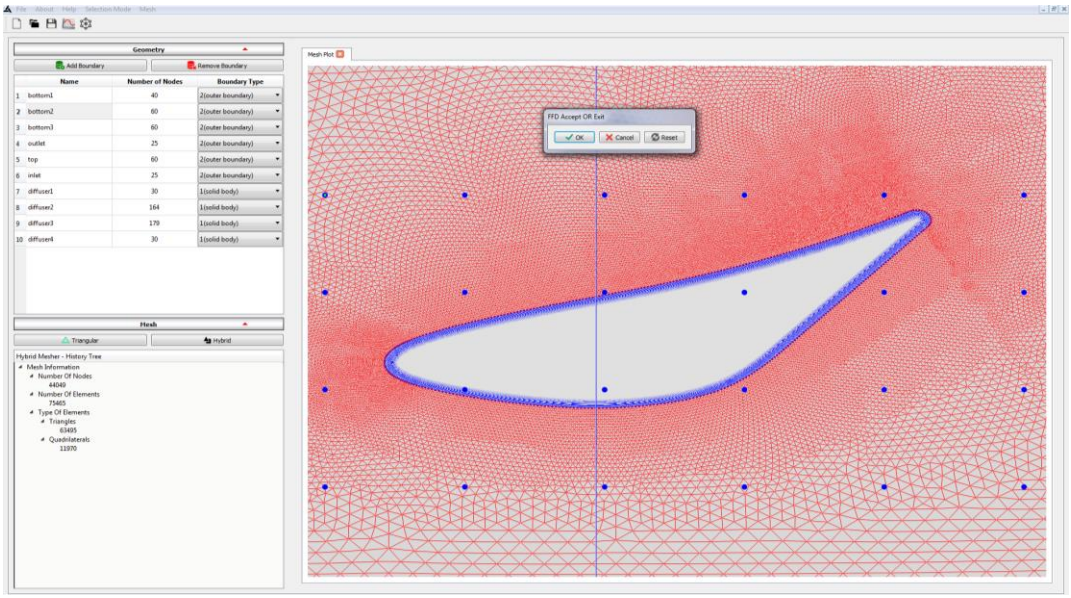


Figure 33: Surface lattice FFD tool.

The user now has the ability to deform the grid by moving any point from the surface FFD lattice (Fig. 34). Finally, the dialog that appears provides the user with the ability to replace the initial mesh data with the deformed mesh data.

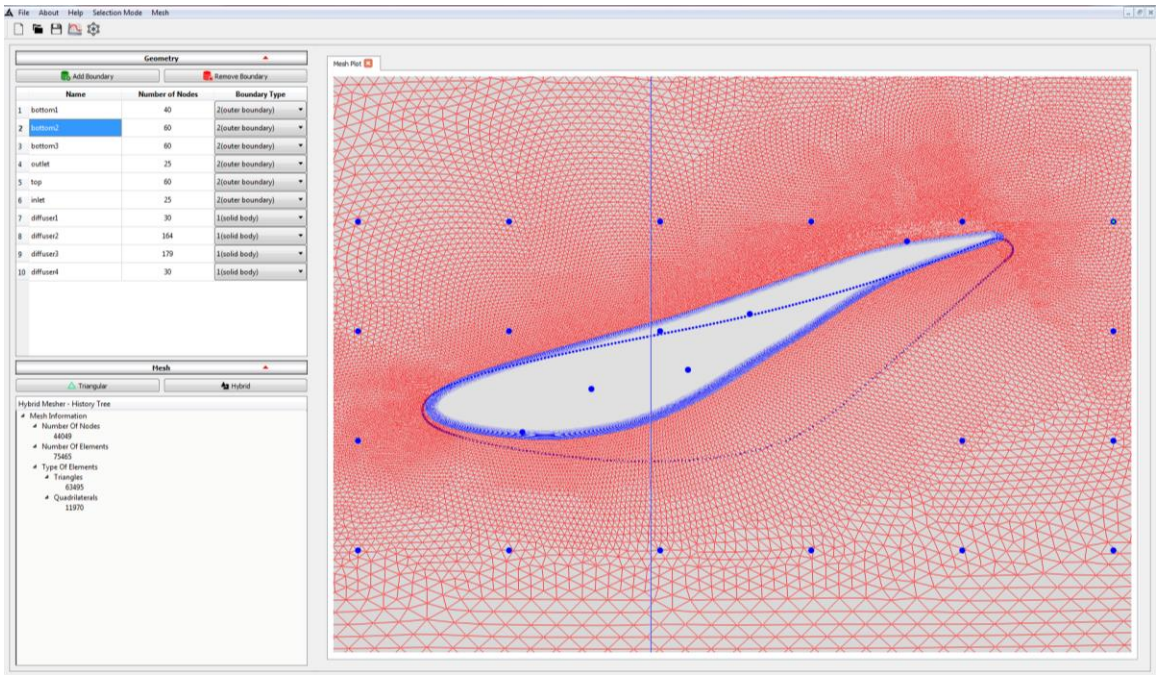


Figure 34: Mesh deformation after moving points on the FFD lattice.

7.5 T2GR Overview

Software Abilities:

- Editing the boundaries of the computational domain.
  - Insertion or export of any boundary.
  - Visual personalization tools.
  - Increase or decrease the number of nodes, using a B-Spline curve interpolation.
  - Modification of the nodes distribution (spacing) on every boundary.
    - Cosine Distribution.
    - Geometric Distribution.
- Mesh processing.
  - Inflation on any bound that is chosen by the user.
  - Vector slope adjustment at nodes.
  - Adjustment of quadrilateral grid inclination.
  - Smoothing in regions with strong geometry change (sharp angles).
  - The user can choose any of the following parameters: growth rate, initial step, number of layers.
- Mesh Generation.
  - Triangular
  - Hybrid
- Mesh Export Format.

- .dpl (Delaundo)
- .cas (Ansys FLUENT)

- Geometries Import Format.
  - .dat (point cloud)
  - .obj

Thus, it is apparent that the user has the flexibility to handle the geometry but also holds the option of his intervention in the creation of the grid. One of the most significant factors on mesh generation is the nodes' distribution on the boundaries. For that reason two different methodologies have been developed (cosine distribution, geometric distribution). Each methodology serves different needs. For example, on an airfoil the double cosine distribution is ideal. In contrast, for geometries with slight changes in gradient the geometric distribution is preferable; by giving an initial step, automatically a growth rate is calculated, and a new distribution for the corresponding nodes is applied. Also, the user's involvement in the inclination of the inflated grid provides an extra control of its quality.

Chapter 8: Software Validation

8.1 Validation Procedure

In order to certify the ability of the proposed software to produce computational grids of good quality, flow simulations were performed for three different reference problems, using the computational grids created by the present software. The flow simulation for all the following cases was accomplished through the use of an in-house incompressible Navier-Stokes solver, based on Vertex-centered, Finite Volume (FV) methodology.

8.1.1 Laminar Flow around a Sphere

Initially, simulation of the incompressible laminar flow around a sphere with a Reynolds number equal to 100 was performed. Under these conditions, the flow around the test body is steady and axisymmetric. Therefore, although the flow is actually three-dimensional, using the ability of the CFD solver to deal with 2D-axisymmetric flows, it can be solved with a 2D computational grid, setting the appropriate boundary conditions. The 2D hybrid grid constructed around the sphere is shown in Figure 35 and consists of 59,967 triangular elements, 1957 quadrilateral ones, with a total number of nodes equal to 32,474.

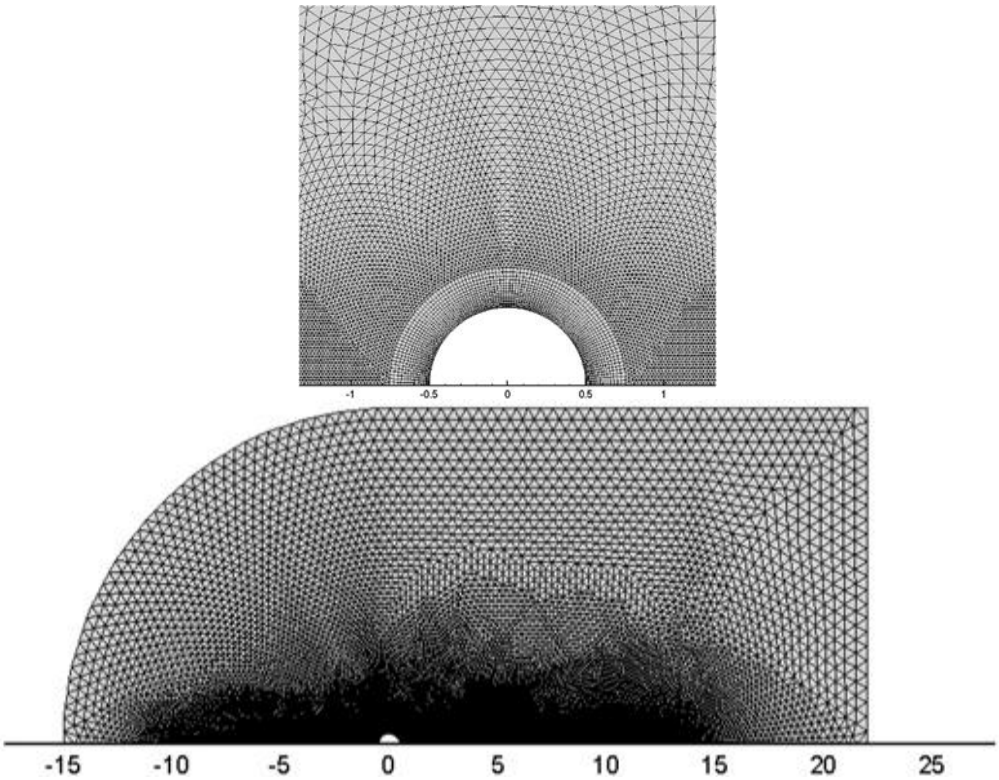


Figure 35: The computational grid used for simulating the axisymmetric flow around a sphere.

Fig. 36 shows the dimensionless pressure and velocity fields around the sphere, as well as the computed streamlines. Finally, for the quantitative evaluation of the results, Fig. 37 presents the distribution of the pressure coefficient on the surface of the sphere, compared to the corresponding distributions resulting from the work of other researchers. There seems to be a fairly satisfactory agreement between the current results and the reference ones.

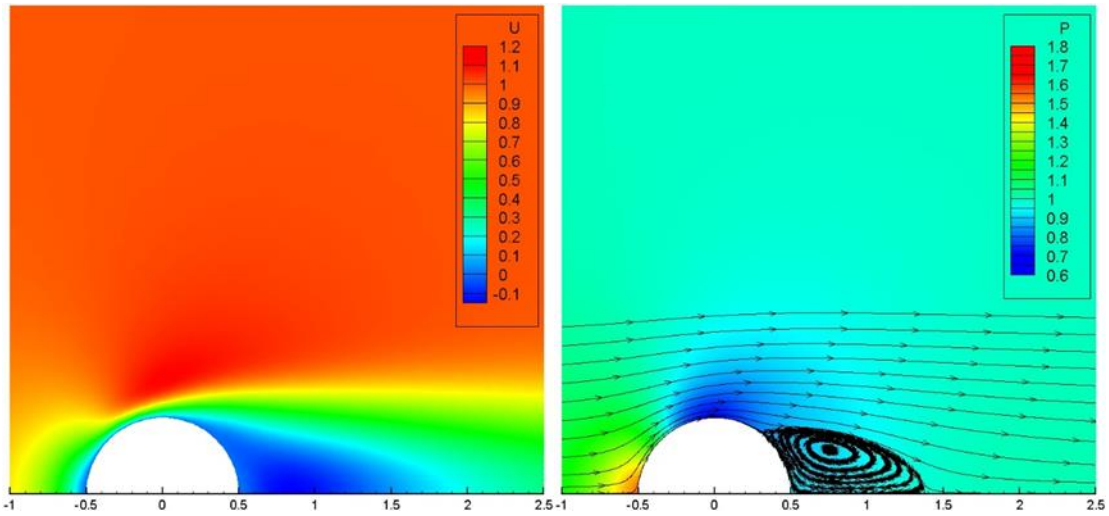


Figure 36: Non-dimensional fields of velocity and pressure

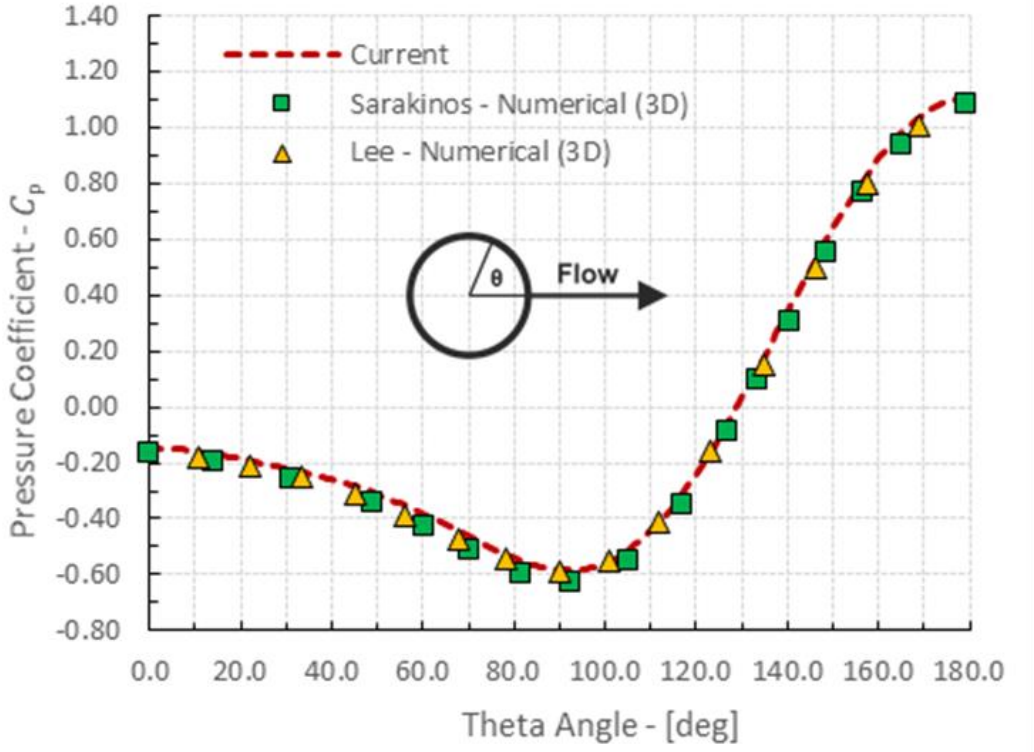


Figure 37: The distribution of the pressure coefficient on the sphere surface.

8.1.2 Turbulent flow around a NACA0012 airfoil

The incompressible turbulent flow around a NACA0012 airfoil was simulated for a Reynolds number equal to $3 \cdot 10^6$. The produced grid is shown in Figure 38, and consists of 20,790 triangular elements, 9,384 quadrilateral elements, with 19,985 nodes.

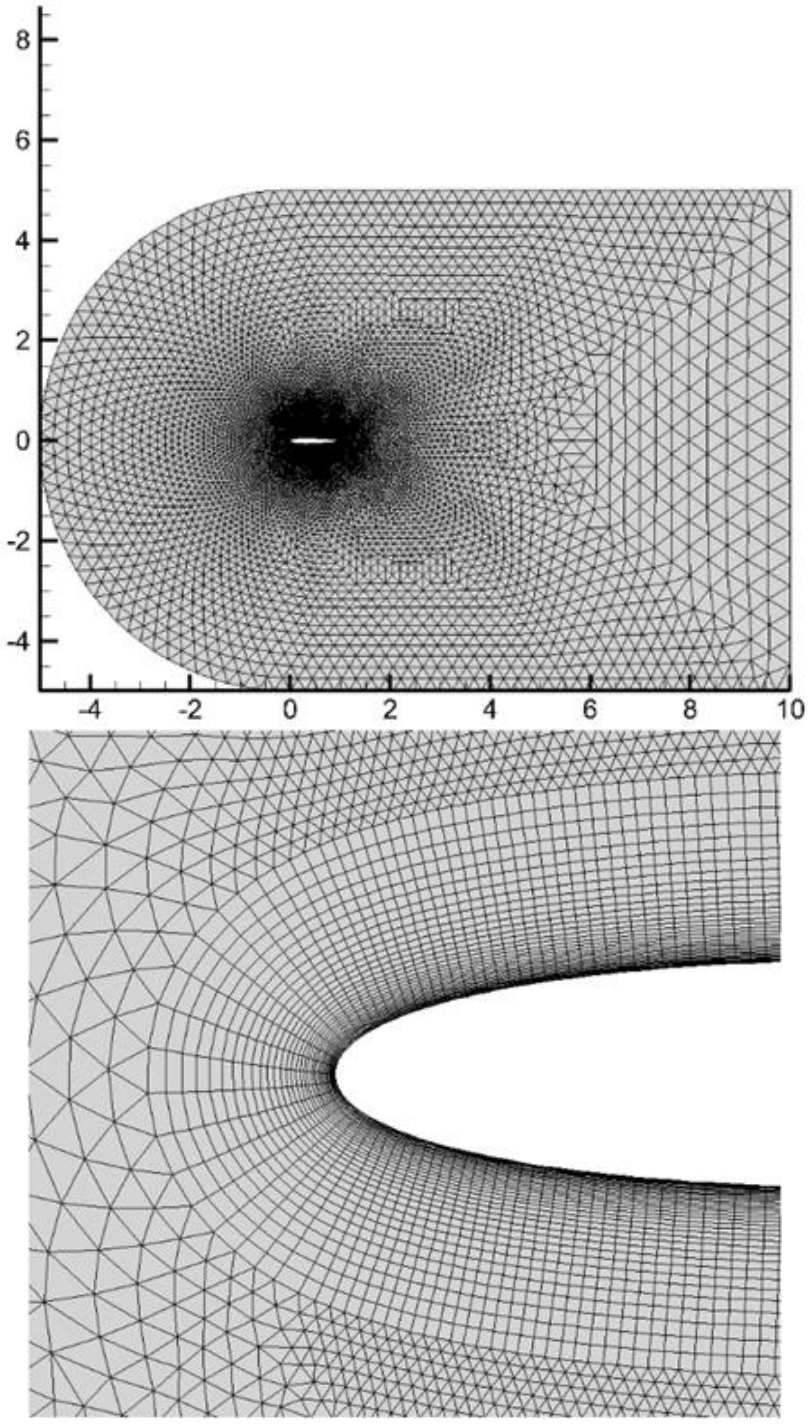


Figure 38: The computational grid for simulating turbulent flow around NACA0012 airfoil.

Figure 39 shows the dimensionless pressure and velocity fields around the airfoil. Finally, for the quantitative evaluation of the results, Figure 40 shows the distribution of the pressure coefficient on the surface of the airfoil, compared to experimental measurements. As can be seen, the two distributions of the non-dimensional pressure coefficient fit quite well.

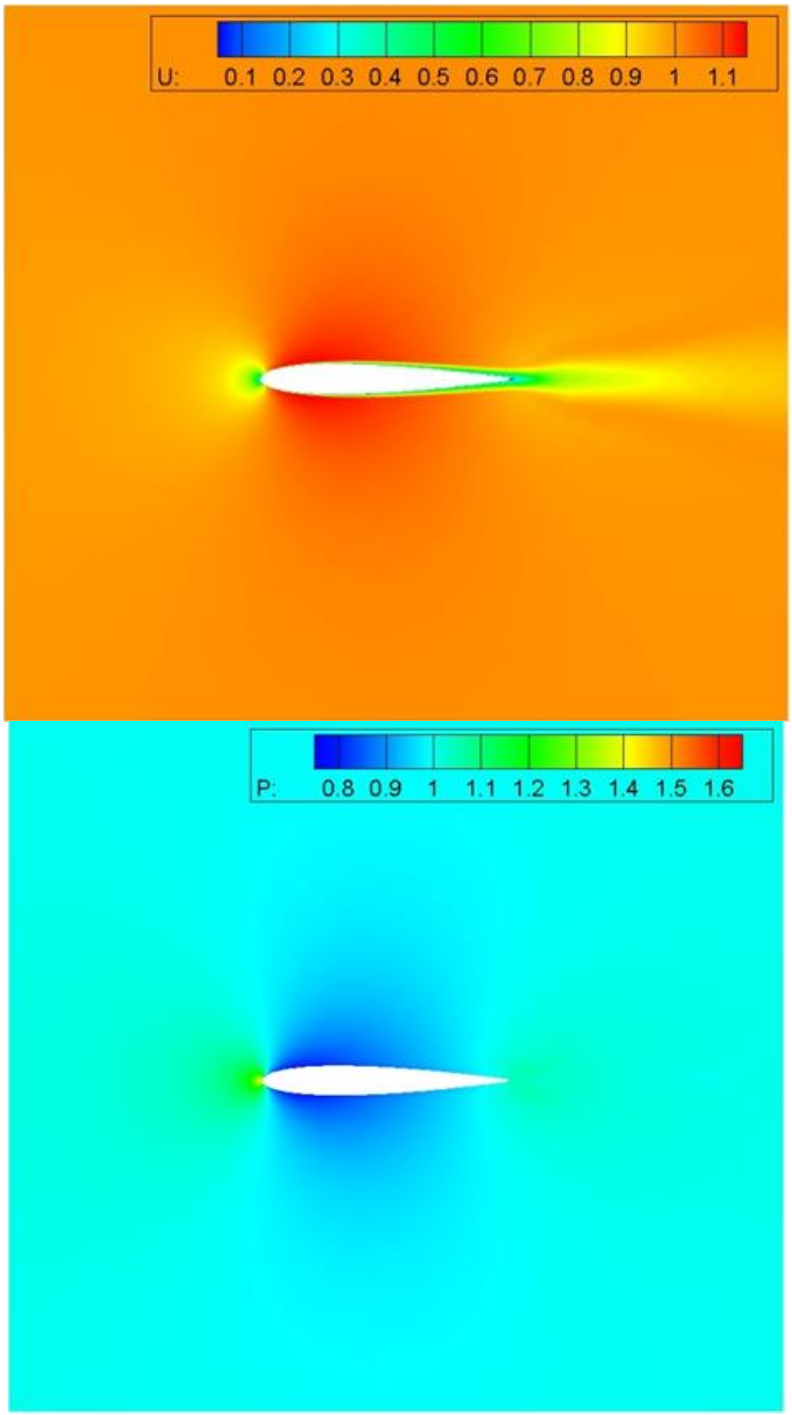


Figure 39: Dimensionless velocity and pressure fields around the NACA0012 airfoil.

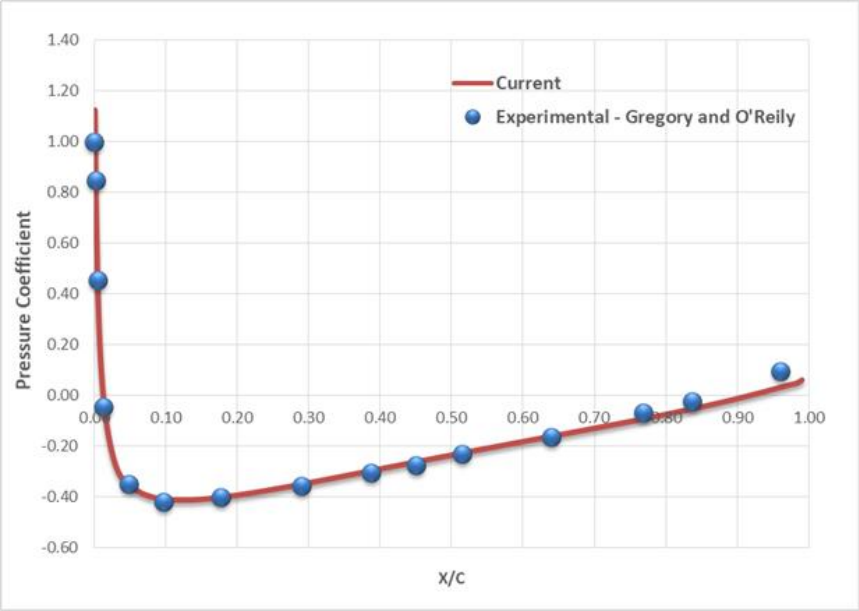


Figure 40: Distribution of the pressure coefficient on the surface of NACA0012 airfoil.

8.1.3 Turbulent flow around an axisymmetric diffuser

Finally, the incompressible turbulent flow around an axisymmetric diffuser for Reynolds number equal to 67,000 was simulated. For the specific reference problem, as in the case of the flow simulation around the NACA0012 airfoil, the calculation of the turbulent kinetic energy was made through the use of SST (Shear Stress Transport) turbulence model. The 2D hybrid computational grid is shown in Fig. 41, consisting of 97,999 and 15,264 triangles and quadrilaterals respectively, with a total number of nodes equal to 64,837.

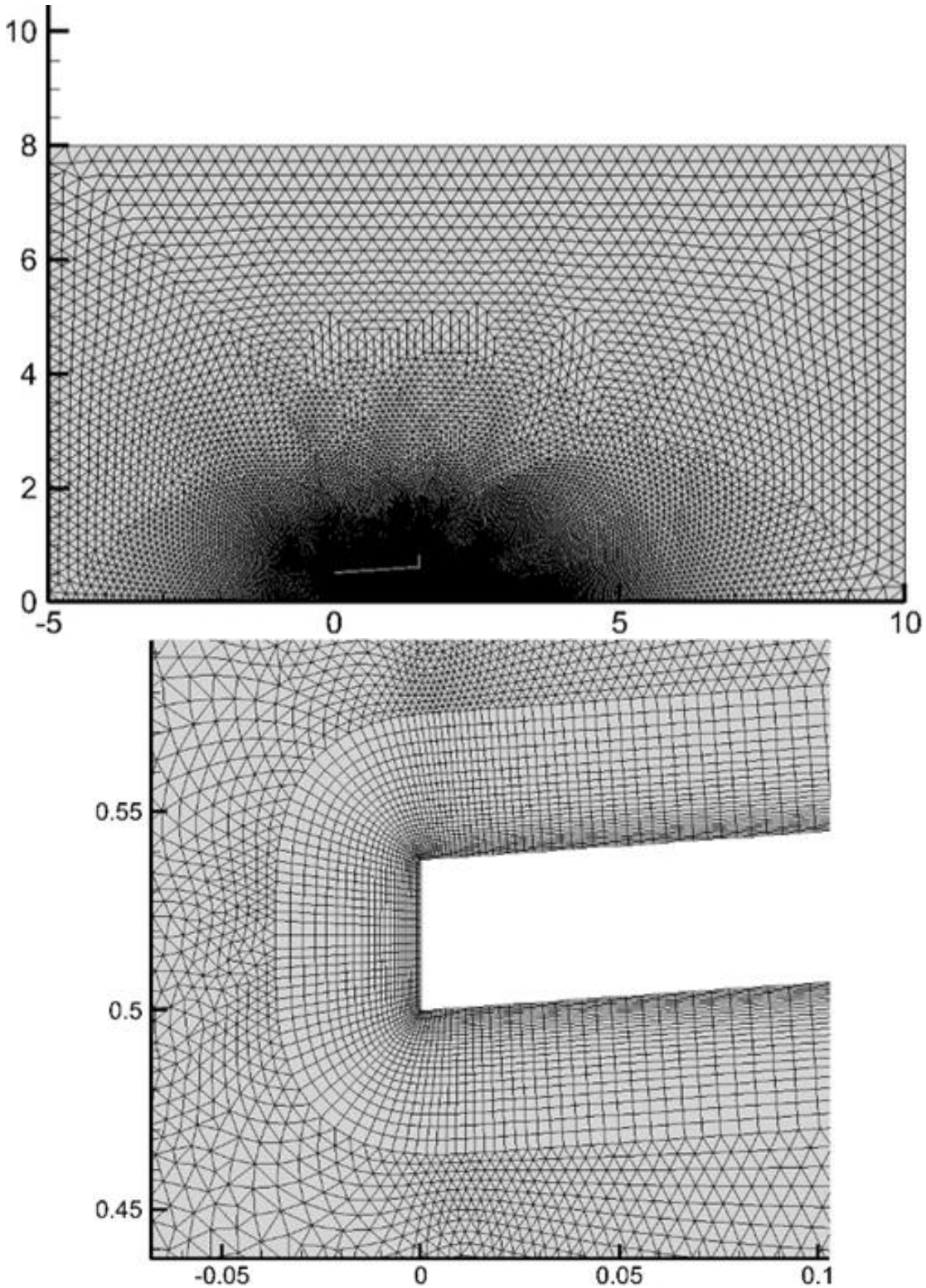


Figure 41: The computational grid used to simulate the turbulent flow around the axisymmetric diffuser.

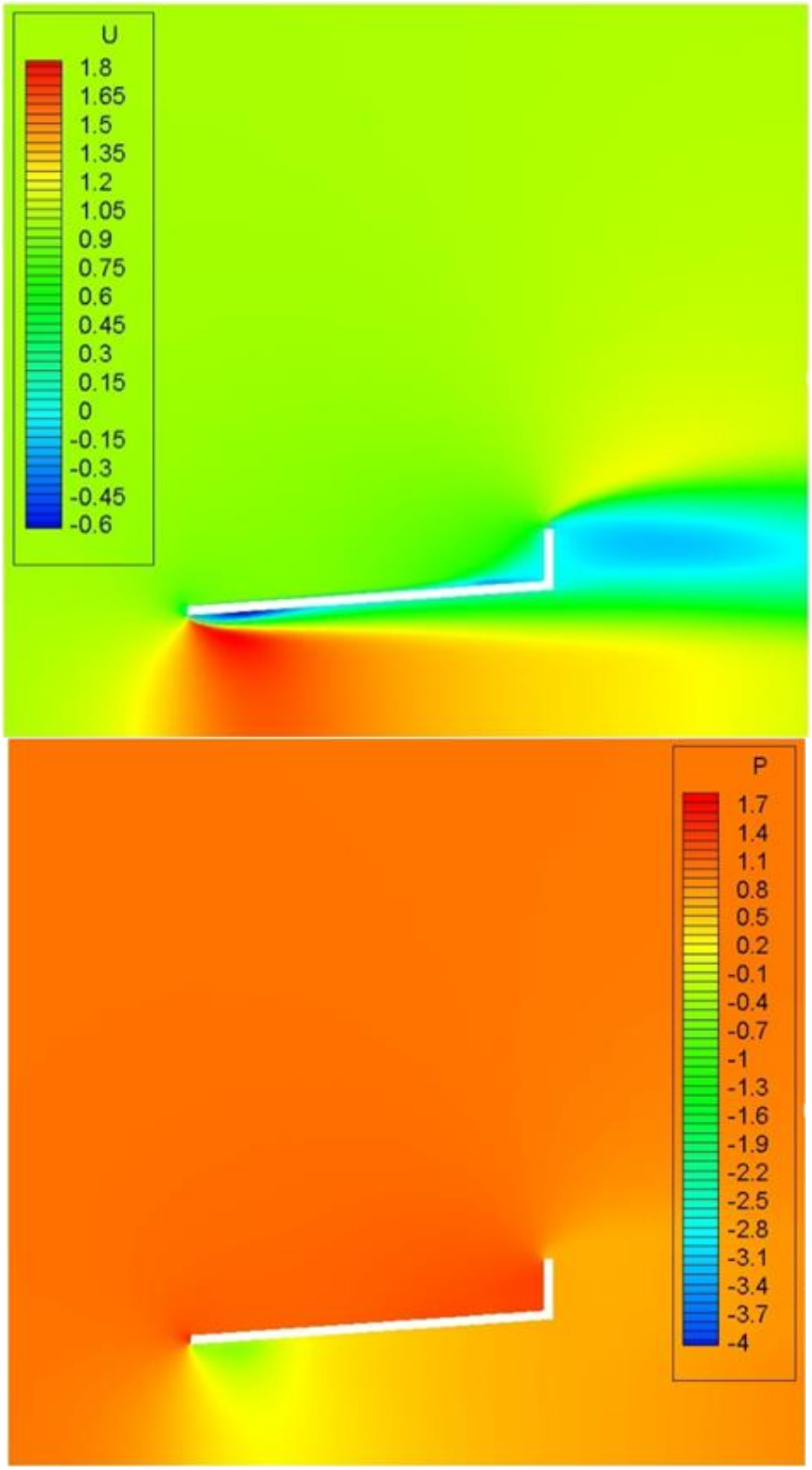


Figure 42: Non-dimensional velocity and pressure fields around the axisymmetric diffuser.

Finally, Fig. 43 presents the distributions of the non-dimensional pressure coefficient and the non-dimensional acceleration of the velocity on the axis of symmetry of the axisymmetric diffuser, compared to the corresponding numerical and experimental data derived from the research work of Abe & Ohya [25].

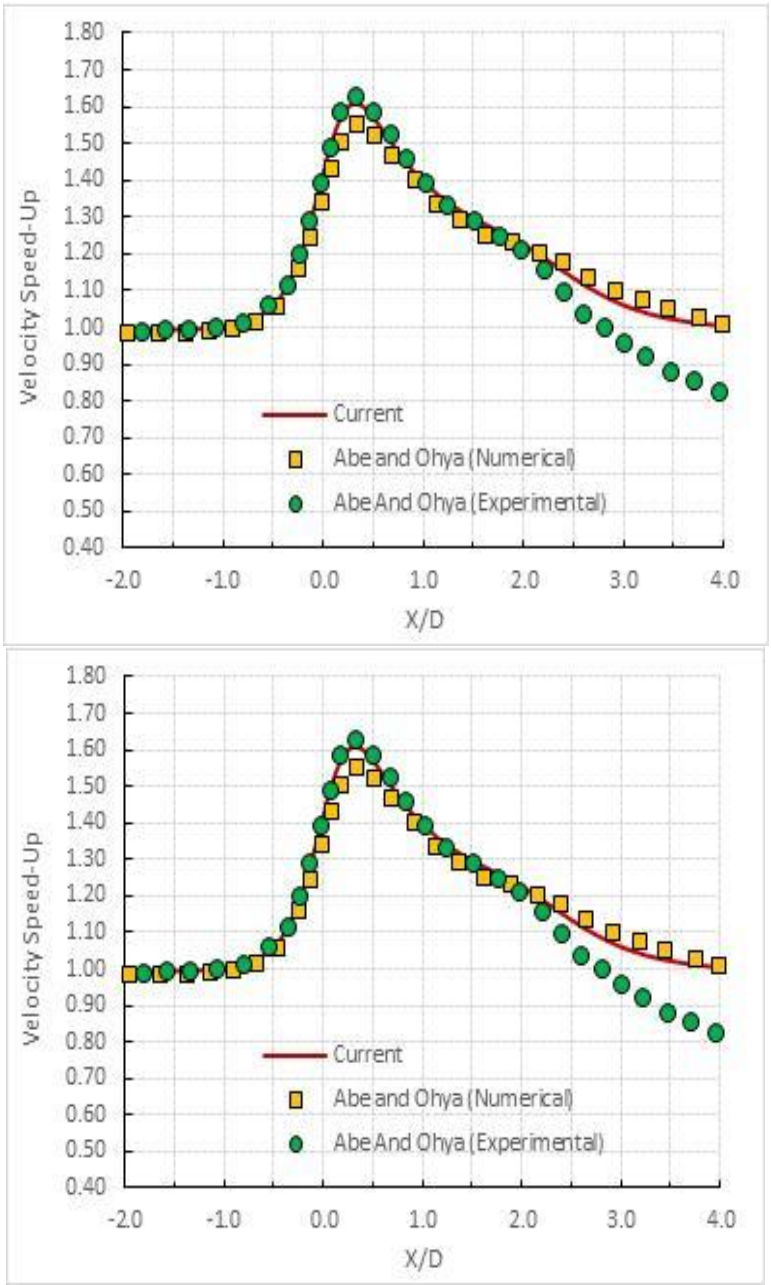


Figure 43: Distributions of the non-dimensional pressure coefficient and the non-dimensional acceleration of the velocity on the axis of symmetry [25].

Chapter 9: Pseudo Code and Flowcharts

9.1 Hybrid Mesh Generation Pseudo Code

```
for Each Viscous Bound{
        for Each Edge of The Bound
          Calculate Perpendicular Vector
      for Each Node of The Bound
          Calculate Smoothed Extended Nodes
          Calculate First Step Nodes
      for Each Step != 1{
          if Quad Mesh Connects Neighboring Bound
              Reconstruct Neighboring Bound
              Set Surface Nodes (n-1) Step Nodes
          for Each Edge of Surface
              Calculate Perpendicular Vectors
          for Each Node of Surface Nodes
              Calculate smoothed extended Nodes
   }
}
for Each Quad Mesh{
   If Quad Mesh Nodes are External
          Set as Surface Nodes for Viscous Bound
}
Create .pts File
run Delaundo
If Delaundo Creates Triangles{
   Read Dpl File
   for each triangle element{
      Store Index of Neighbors
      Store Nodes
      Store Index of Nodes
      Store Index of Element

   }
}
Create dpl file with a single mesh
```

9.2 Mesh and Bounds Connectivity Flowchart

The flowchart in Figure 44 shows the case where the grid being developed is linked to a neighboring boundary. Four alternative approaches are followed, in order to achieve the alignment of the grid with the adjacent geometry. The “if” statements

represent logical operations with Booleans and the blue boxes represent functions that take place whenever necessary.

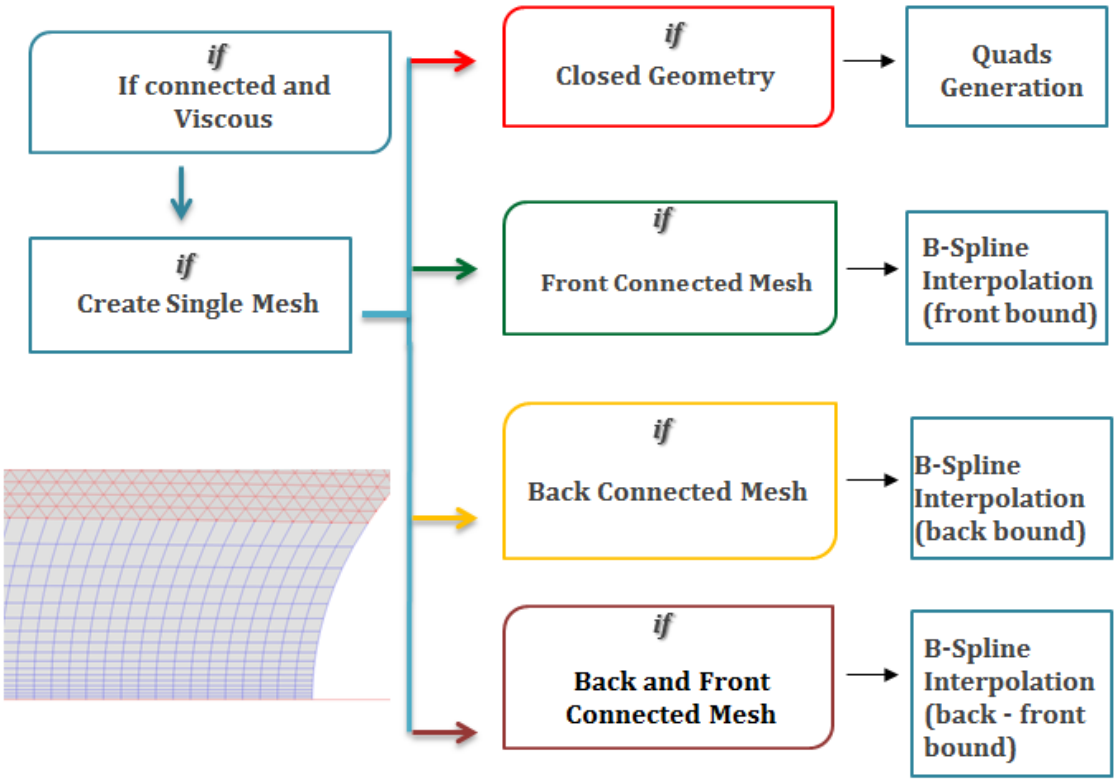


Figure 44: Mesh and bounds connectivity flowchart.

9.3 Bounds Reconstruction Function

The flowchart in Figure 45 shows the algorithm used to produce the reconstruction of the neighboring boundaries associated with the quadrilateral mesh. The methodology is based on B-spline interpolation and geometric progression for nodal distribution.

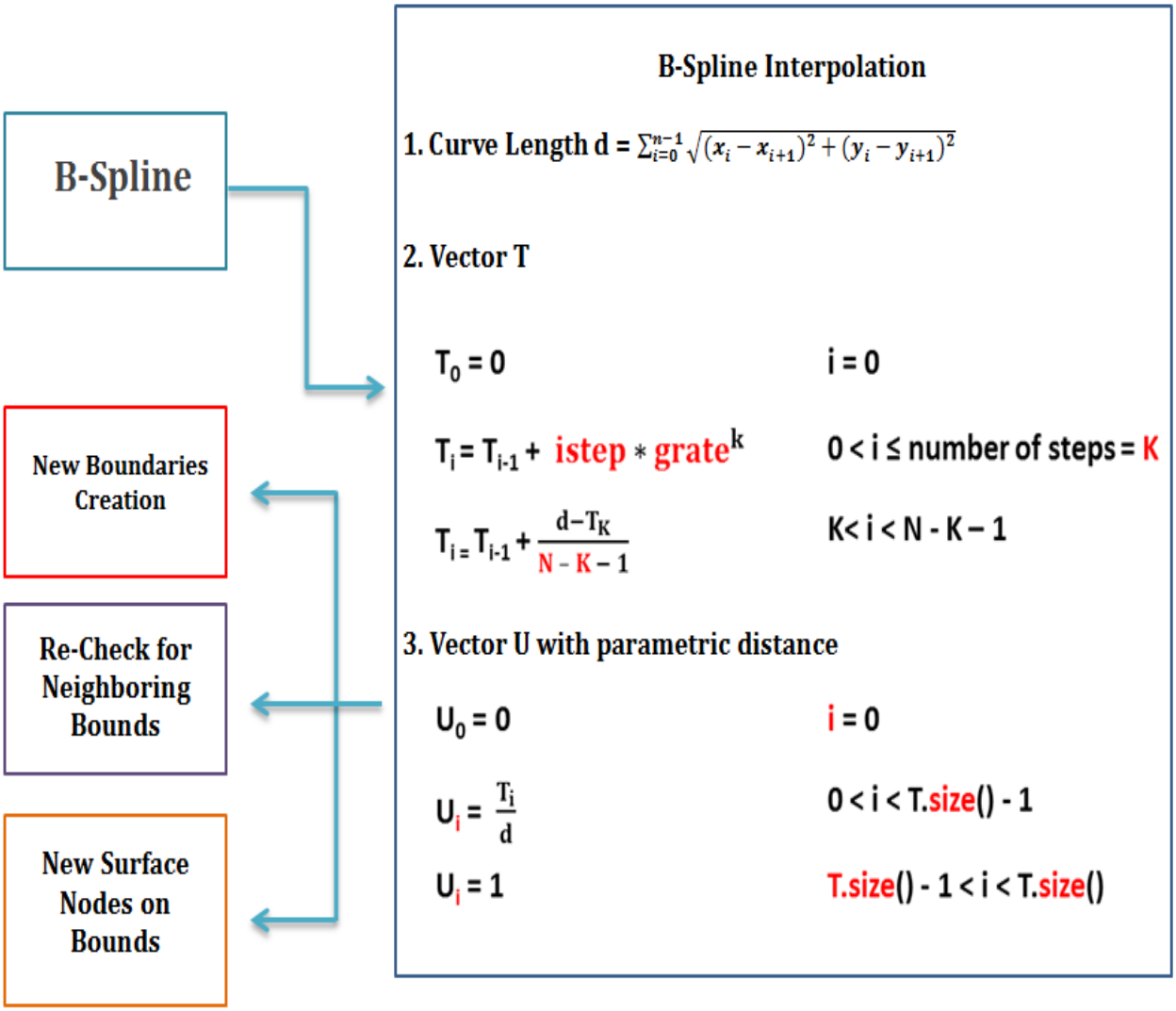


Figure 45: Boundaries reconstruction function flowchart.

9.4 Numbering Quadrilateral Elements

Since the quadrilateral and triangular computational grids have been successfully constructed, the algorithm automatically joins the two grids to create a single hybrid computational grid. The flowchart followed by the algorithm is contained in Fig. 46.

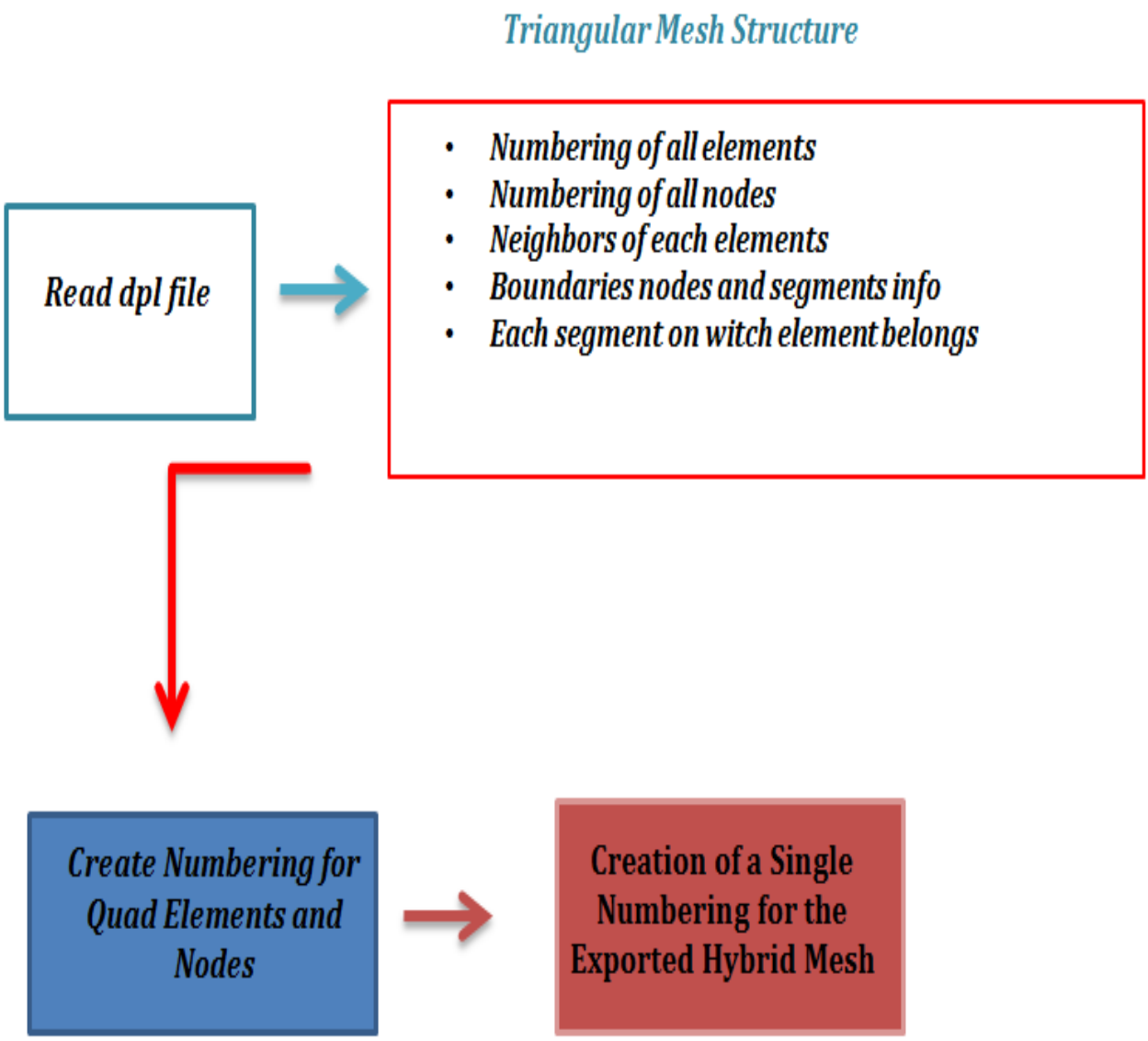


Figure 46: Joining quadrilateral and triangular grids in a single one.

.

Chapter 10: Hybrid Grid Cases

10.1 NACA0012

Number Of Nodes	56113
Triangular Elements	87966
Quadrilateral Elements	11662
Number of Layers	17
Initial Step	$5 * 10^{-4}$
Structured Mesh Growth Rate	1.12
Unstructured Mesh Growth Rate	0.8
Unstructured Mesh Laplacian Smoothing	True
Viscous Wall Nodes Orientation	Clockwise (CW)
Cage Boundaries Nodes Orientation	Counter-Clockwise (CCW)
Switch Method Function (Structure Grid)	Activated
Frontal Smoothing Gradient	0.35
End Smoothing Gradient	0.55

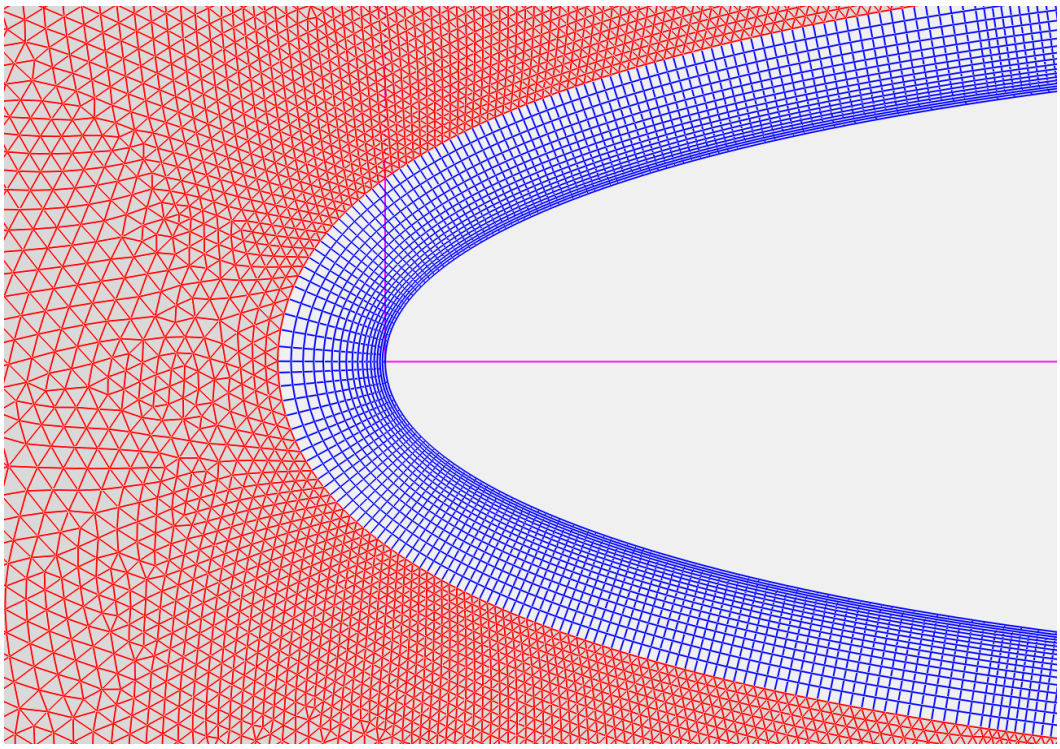


Figure 47: NACA0012 (leading edge).

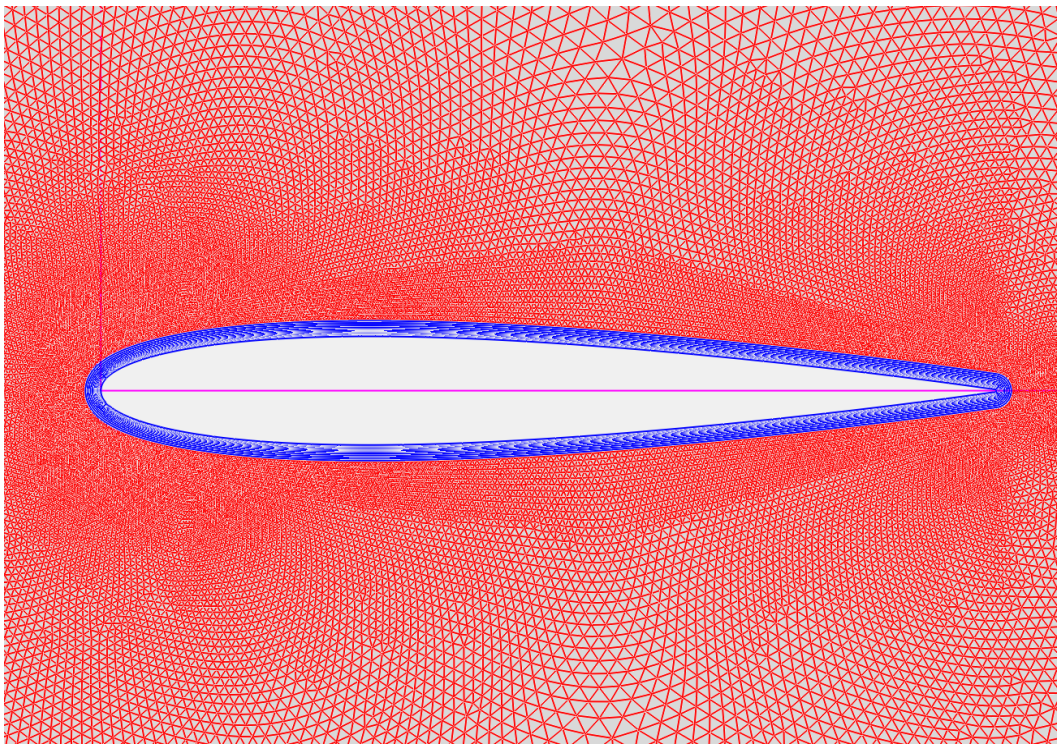


Figure 48: NACA0012 with hybrid mesh (close view).

10.2 Axisymmetric Diffuser

Number Of Nodes	33634
Triangular Elements	30700
Quadrilateral Elements	17955
Number of Layers	45
Initial Step	$5 * 10^{-5}$
Structured Mesh Growth Rate	1.11
Unstructured Mesh Growth Rate	0.5
Unstructured Mesh Laplacian Smoothing	True
Viscous Wall Nodes Orientation	Clockwise (CW)
Cage Boundaries Nodes Orientation	Counter-Clockwise (CCW)
Switch Method Function (Structure Grid)	False
Frontal Smoothing Gradient	N/A
End Smoothing Gradient	N/A

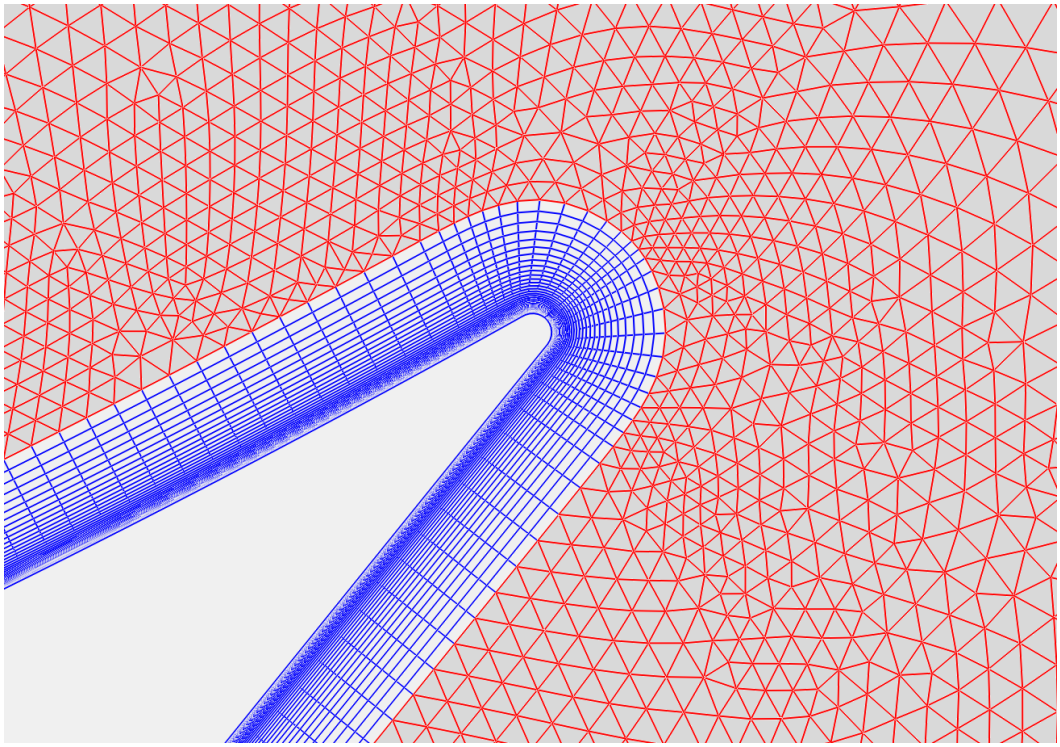


Figure 49: Axisymmetric Diffuser (trailing edge).

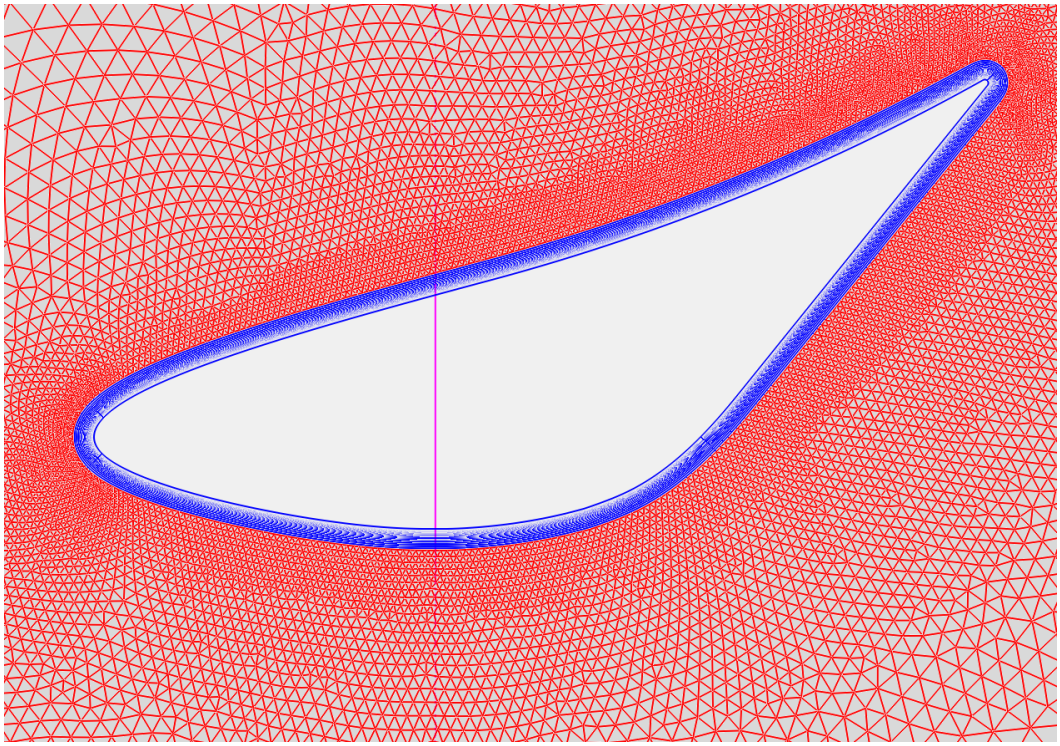


Figure 50: Axisymmetric Diffuser (close view).

10.3 Axisymmetric Sphere

Number Of Nodes	33634
Triangular Elements	30700
Quadrilateral Elements	17955
Number of Layers	17
Initial Step	$5 * 10^{-3}$
Structured Mesh Growth Rate	1.12
Unstructured Mesh Growth Rate	0.2
Unstructured Mesh Laplacian Smoothing	True
Viscous Wall Nodes Orientation	Clockwise (CW)
Cage Boundaries Nodes Orientation	Counter-Clockwise (CCW)
Switch Method Function (Structure Grid)	True
Frontal Smoothing Gradient	0.95
End Smoothing Gradient	1.0

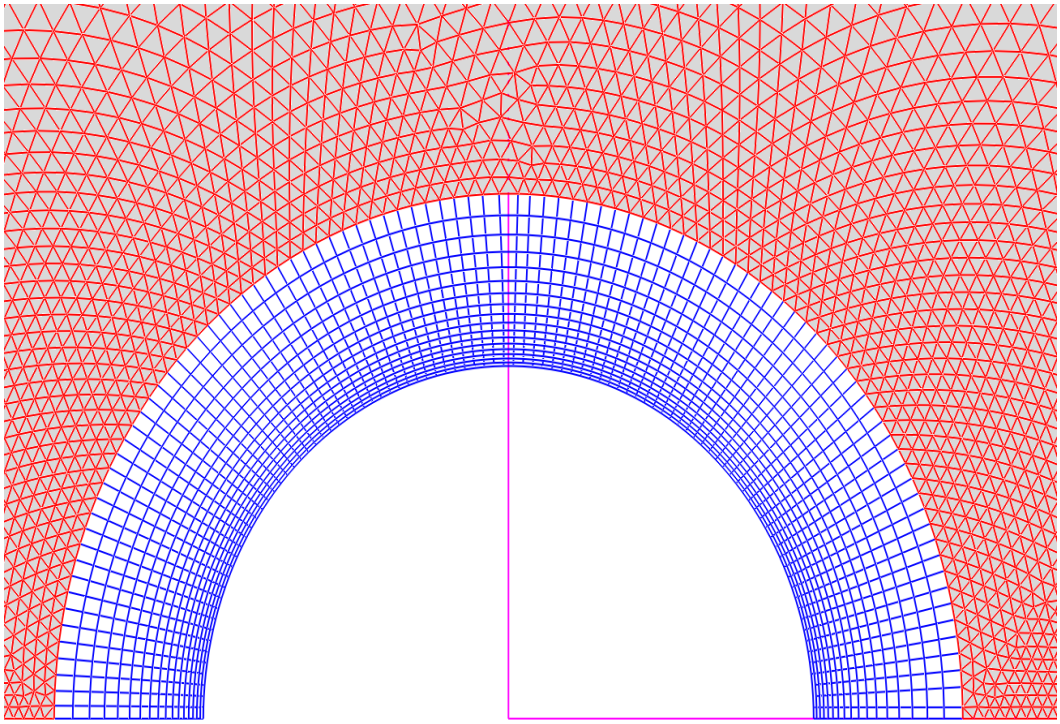


Figure 51: Axisymmetric Sphere.

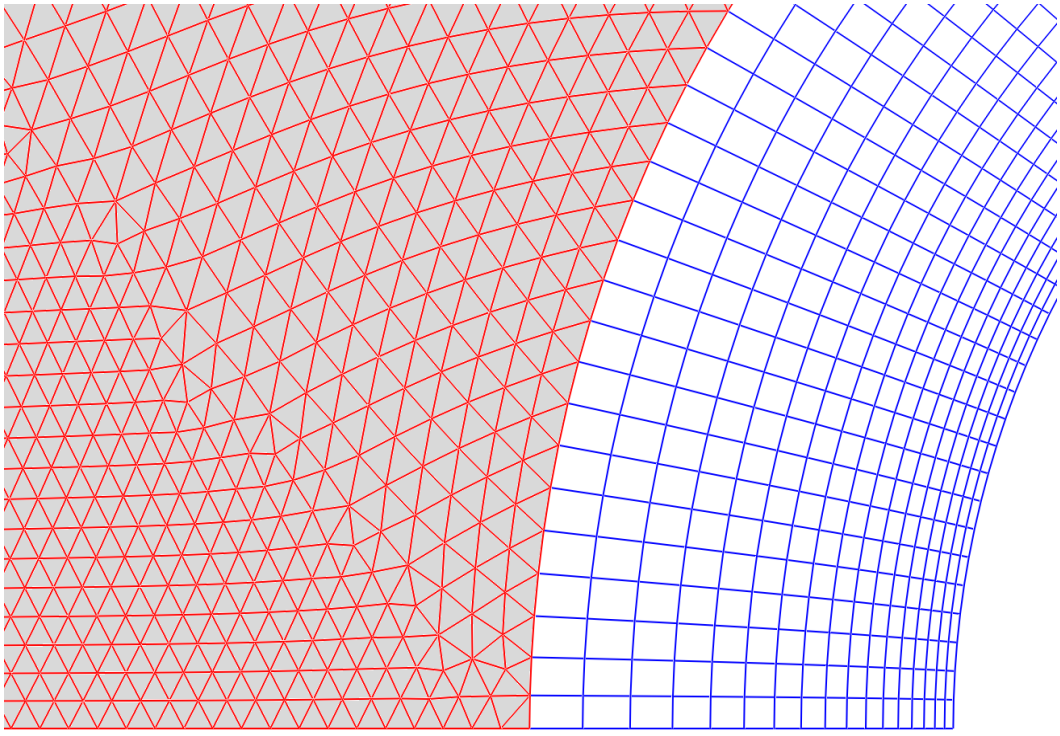


Figure 52: Axisymmetric Sphere (leading edge).

Chapter 11: Conclusions

In this diploma thesis a software was developed for the construction of 2D hybrid unstructured grids. The target was the development of a user-friendly software that produces grids of high quality for accurate fluid dynamics simulations. The construction of the inflation layers (utilizing quadrilateral elements) is based on an algebraic technique, which provides a fast and easy way to distribute the layers of the grid.

Since mesh generation is inherently a complicated procedure, it is of major importance the development of a graphical user interface (GUI), for rendering the grid generation procedure straightforward for the inexperienced user. Such a GUI was successfully developed within this work.

The fidelity of the geometric algorithm has tested through three different test cases, involving flow simulation around axisymmetric bodies, using an in-house incompressible 2D-axisymmetric solver. Each one of the utilized test cases has different geometrical characteristics, different grid-density requirements and different types of boundary conditions. Complicated flow phenomena, such as vortices and flow separation regions, have been successfully simulated, using the produced computational grids. As a result, all three simulations successfully confirmed the fidelity of the algorithm and its ability to produce hybrid unstructured grid with proper characteristics for use with flow simulation solvers.

The incorporation of the FFD tool greatly enhances the use of the developed software in design optimization procedures. The FFD tool is very useful in rapidly deforming the geometry at hand, along with its computational grid, without the need for new grid generation.

The developed software has a very wide region of applications. However, many new characteristics and abilities are to be added in the future. Some of them are the following:

- Introduction of grid-quality metrics.
- Surface mesh generation (on non-planar surfaces described as B-spline and/or NURBS parametric surfaces).
- Introduction of grid-refinement tools.

References

[1] G. Papadakis, *Lecture Notes on Computational Fluid Dynamics*, King's College, U.K.

[2] M.J. Marchant & N.P. Weatherill, "Adaptivity Techniques for Compressible Inviscid Flows", *Computer Methods in Applied Mechanics and Engineering*, 106, pp. 83-106, 1993.

[3] J. Tu, G.H. Yeoh, C. Liu, *Computational Fluid Dynamics: A Practical Approach*, 3rd edition, Butterworth-Heinemann, 2018.

[4] https://www.design-engineering.com/cfd-automeshing-1004028397-1004028397/

[5] A. Khawaja, T. Minyard, Y. Kallinderis, "Adaptive hybrid grid methods", Computer Methods in Applied Mechanics and Engineering, 189(4), pp. 1231-1245, 2000.

[6] https://learnopengl.com

[7] J. Blanchette, M. Summerfield, *C++ GUI Programming with QT 4*, 2nd edition, Prentice Hall, 2008.

[8] S. Menzel, M. Olhofer, B. Sendhoff, "Application of Free Form Deformation Techniques in Evolutionary Design Optimisation", 6th World Congress on Structural and Multidisciplinary Optimization (WCSMO6), Rio de Janeiro, 30 May - 03 June, Brazil, 2005.

[9] G.L. Dirichlet. Uber die Reduction der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen, *Z. Reine Angew. Math.,* 40(3), pp. 209-227, 1850.

[10] G. Voronoi. Novelles applications des parametres continus a la theorie des formes quadratiques, *J. Reine Angew. Math*., 134, 1908.

[11] B. Delaunay. Sur la sphere vide. *Bull. Acad. Science USSR VII: Class. Sci.-Mat. Nat*. pp. 793-800, 1934.

[12] J.D. Muller, *On Triangles and Flow*, PhD. Thesis, The University of Michigan, 1996.

[13] D.T. Lee & B.J. Schachter, "Two algorithms for constructing a Delaunay triangulation", *International Journal of Computer & Information Sciences*, 9(3), pp. 219-242, 1980.

[14] D.F. Watson, "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes", *The Computer Journal*, 24(2), pp. 167-171, 1981.

[15] J. Peraire, M. Vahdati, K. Morgan, O.C. Zienkiewicz, "Adaptive remeshing for compressible ow computations", *Journal of Computational Physics*, 72, pp. 449-466, 1987.

[16] S. Rebay, "Ecient unstructured mesh generation by means of Delaunay triangulation and Bowyer-Watson algorithm", *Journal of Computational Physics*, 106(1), pp. 125-138, 1993.

[17] H.J. Lamousin, N.N. Waggenspack, "NURBS-based free-form deformations", *IEEE Comput. Graph. Appl*., 14(6), pp. 59–65, 1994.

[18] L. Piegl, W. Tiller, *The NURBS Book*, Springer-Verlag, Berlin Heidelberg, 1997.

[19] E.I. Amoiralis, I.K. Nikolos, "Freeform Deformation versus B-Spline Representation in Inverse Airfoil Design", *J. Comput. Inf. Sci. Eng*., 8(2), 2008.

[20] N.M. Patrikalakis, T. Maekawa, *Shape Interrogation for Computer Aided Design and Manufacturing*, Springer Berlin Heidelberg, 2010.

[21] https://www.sharcnet.ca/Software/Ansys/17.0/en-us/help/tgd_usr/tgd_user_format_gridsect.html?fbclid=IwAR08jOCsPpvNa3a4HtdsFkz1evOBLeg-L8F2AAOp44QskhB0Dp-7GQ9-I-Y]

[22]https://www.sharcnet.ca/Software/Ansys/17.0/en-us/help/flu_ug/flu_ug_format_case.html?fbclid=IwAR10V5S9tOK88gZCN50P1WJtbbEYy_nDEm1B6gNwxjWcqMcPLH1PHmM-5tk

[23] S. Abbasbandy, "Improving Newton–Raphson method for nonlinear equations by modified Adomian decomposition method", *Applied Mathematics and Computation*, 145(2-3), pp. 887-893, 2003.

[24] www.codewithc.com

[25] K. Abe, Y. Ohya, "An investigation of flow fields around flanged diffusers using CFD", *Journal of Wind Engineering and Industrial Aerodynamics*, 92, pp. 315–330, 2004.