# Advanced 4D Mesh Generation for Rotor-Stator Airfoils in Computational Fluid Dynamics

Henri Nela
Student Number: 2053620

11 May 2023

Supervised by Dr Andrew Lawrie
Department of Mechanical Engineering

**Abstract**

This Individual Research Project provides a comprehensive exploration of four-dimensional (4D) meshing, an emerging field in Computational Fluid Dynamics (CFD). With the goal of enhancing the accuracy and efficiency of fluid flow simulations, particularly in complex applications such as turbines, this work investigates an alternative to traditional approaches that use sliding and overlapping grids. The research focuses on the development of 4D unstructured meshes that amalgamate spatial and temporal domains, providing a more thorough representation of the physical system. Inspired by the PhD thesis "Mesh generation in four dimensions for adaptive refinement in space and time" by Siddharth Suresh Kamble, the study extends principles of mesh generation to the fourth dimension. Key components of the project include the design of a robust algorithm based on the Arbitrary Lagrangian-Eulerian (ALE) method, which involved the creation of the rotor and stator airfoils, the implementation of Poisson's equation for grid deformation in 2D, 3D, and 4D cases, the use of a new mesh generation technique developed by the PhD student and the derivation of control volumes, interface pressures and updated nodes position. Despite challenges in calculating the volumes of the control volume faces in a 4-dimensional space, this work has achieved success for the 2D case and made important steps towards the 4D application. The research contributes to the ongoing development of more precise and efficient CFD simulations for intricate physical systems, fostering advancements in turbo-machinery and beyond.

# DECLARATION

This project report is submitted towards an application for a degree in Mechanical Engineering at the University of Bristol. The report is based upon independent work by the candidate. All contributions from others have been acknowledged and the supervisor is identified on the front page. The views expressed within the report are those of the author and not of the University of Bristol.

I hereby assert my right to be identified as the author of this report. I give permission to the University of Bristol Library to add this report to its stock and to make it available for consultation in the library, and for inter-library lending for use in another library. It may be copied in full or in part for any bone fide library or research worker on the understanding that users are made aware of their obligations under copyright legislation.

I hereby declare that the above statements are true.



Certification of ownership of the copyright in a dissertation presented as part of and in accordance with the requirements for a degree in Mechanical Engineering at the University of Bristol.

# Contents

# 1 Introduction

## 1.1 Overview

Computational Fluid Dynamics (CFD) is an indispensable tool for simulating and understanding fluid flow in various applications, from biomedicine to turbo-machinery. Researchers and industry professionals are continually trying to improve the accuracy and efficiency of simulations, particularly in complex applications such as turbines [1][2]. One key challenge in this area is accurately simulating blade rows in rotating and stationary components interacting at high relative velocities. Traditional approaches, such as the use of sliding and overlapping grids, can introduce errors and limitations due to interpolation requirements in the overlapping regions [3]. An innovative approach with the potential to address these issues involves the use of four-dimensional (4D) unstructured meshes, which combine spatial and temporal domains, offering a more comprehensive representation of the physical system without the need for sliding or overlapping grids. The study of 4D meshing is an emerging and complex field that requires a deep understanding of higher-dimensional spaces and their intricacies.

In this thesis, the focus is on extending the principles of mesh generation to the fourth dimension, inspired by the PhD thesis *Mesh generation in four dimensions for adaptive refinement in space and time* by Siddharth Suresh Kamble. As an individual research project, this thesis presents a thorough investigation into the patterns and algorithms needed to create 4D meshes, as well as the implementation of the Poisson's equation for grid deformation in 2D, 3D, and 4D cases. In addition, the project encompasses the derivation of control volumes, interface force, and linear interpolations for the 4D mesh, enabling the storage of pressure data at each node. One of the critical steps in this work is the development of a robust algorithm based on the Arbitrary Lagrangian-Eulerian (ALE) method, capable of operating in four dimensions.

In summary, this thesis provides an in-depth exploration of 4D meshing and a potential technique to map the rotor and stator explicitly and meshed with a static 4D mesh. By focusing on the challenges and complexities associated with extending mesh generation to the fourth dimension, this work seeks to contribute to the ongoing development of more accurate and efficient CFD simulations for turbo-machinery and other complex physical systems.

## 1.2 Structure of the research project

The research project is structured as follows: Section 2 presents the method used to realise a mesh body fitting algorithm that can be developed for 2D, 3D and 4D simulations. Section 3 presents the object to define for the simulations: a pair of rotor-stator airfoils. Section 4 tries to build an algorithm to solve Poisson's equation for a regular grid. Section 5 attempts to reproduce the novel algorithm proposed by the PhD student to construct a 4 dimensional mesh starting from a 2 dimensional lattice. The steps for constructing the 4D Lagrangian flow solver that uses an approximation for compressible flows and controls the mesh density are achieved in Section 6. Finally, Section 7 outlines the conclusions of this research project, what could have been better, and possible errors. The potential advantages of this project and the possible works to be done in the future are exposed.

# 2 Methodology

The Section outlines the method and reasoning used in this research project. The project follows a clear and linear scheme in which the procedures derived from the PhD thesis are implemented. This study focuses on the realisation of an efficient algorithm for 4D body fitting mesh using Python as a programming language. To give more context, the PhD candidate develops a novel method in computational fluid dynamics by extending the Arbitrary Lagrangian-Eulerian (ALE) approach to higher dimensions in both

space and time. This can help tackle fluid dynamics problems involving intricate geometries and changing morphologies. The ALE approach was developed to combine the advantages of the Lagrangian and Eulerian methods. Lagrangian algorithms, primarily used in structural mechanics, involve tracking free surfaces and interfaces between different materials. While they effectively handle history-dependent constitutive relations, they struggle with large domain distortions without frequent re-meshing. On the other hand, Eulerian algorithms, which are popular in fluid dynamics, use a fixed computational mesh, and the continuum moves relative to the grid. This approach easily handles large distortions in continuum motion but suffers from imprecise interface definition and limited flow detail resolution. The ALE method allows the mesh on boundaries and interfaces to move with material deformation, effectively tracking interfaces in multi-material systems. This allows the computational mesh nodes to move with the continuum in a Lagrangian manner and remain fixed in an Eulerian manner. As a result, the ALE method can accommodate greater distortions while maintaining excellent resolution. This approach makes it possible to create an algorithm that creates an unstructured grid able to change its topology continuously without bending, while preserving properties [4][5].

The proposed algorithm is presented in Figure 1 through the schematisation of a flowchart, and all the work is condensed through the steps described. The body fitting algorithm is structured in 2 main steps. The first step is defining the objects (rotor-stator airfoils) and solve the Poisson's equation in a regular grid due to the load of the objects. This will result in a series of $\phi$ values throughout the N-dimensional grid which will determine the value of its deformation. The second step is to build an N-dimensional mesh through a new construction method. The control volume around each node of the mesh is then determined. This, through the gas law pressure, will allow to identify the pressure value which depends on the control volume. The resulting pressure value will be the combination of the pseudo-pressure and gas law pressure which will be calculated in the faces of the control volumes. A force will therefore be applied on each face and through a simplified version of the Navier-Stokes equation the velocity and for which the new position of the nodes will be identified. This process will be repeated for one time-step only.

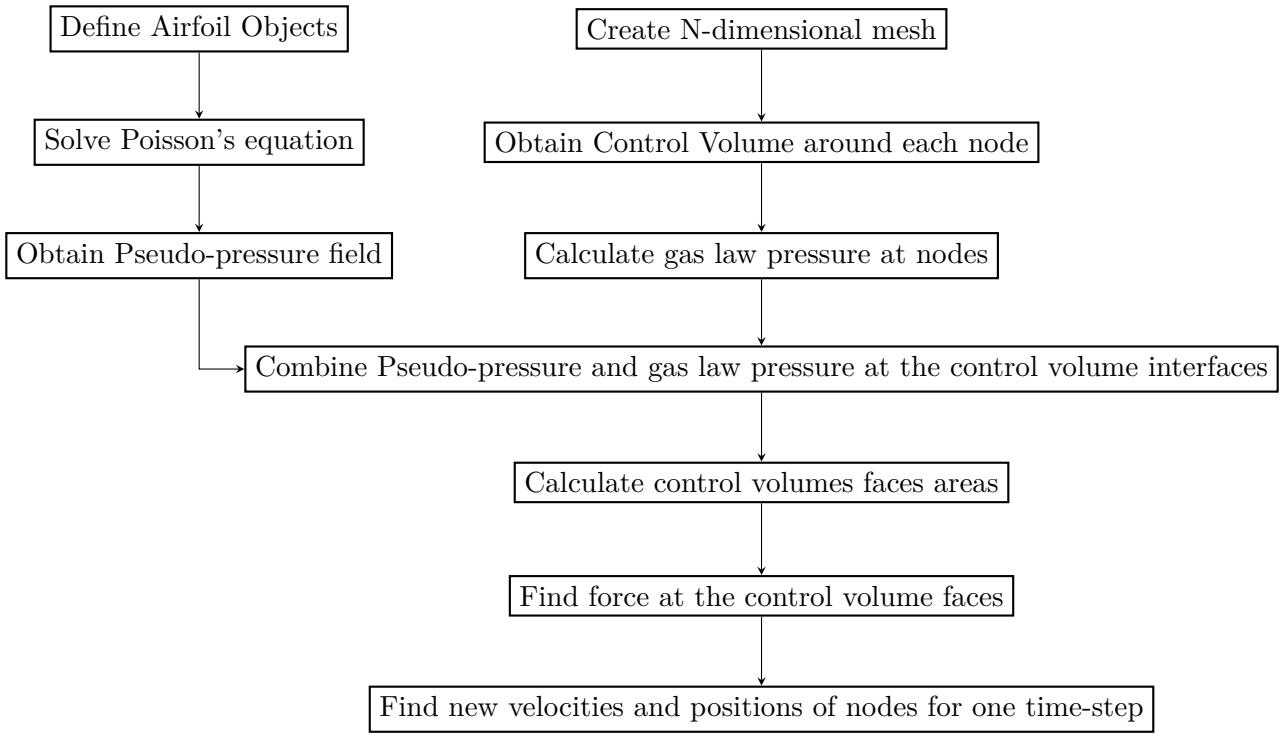



Figure 1: Flow Chart for body fitting in N-dimensional mesh

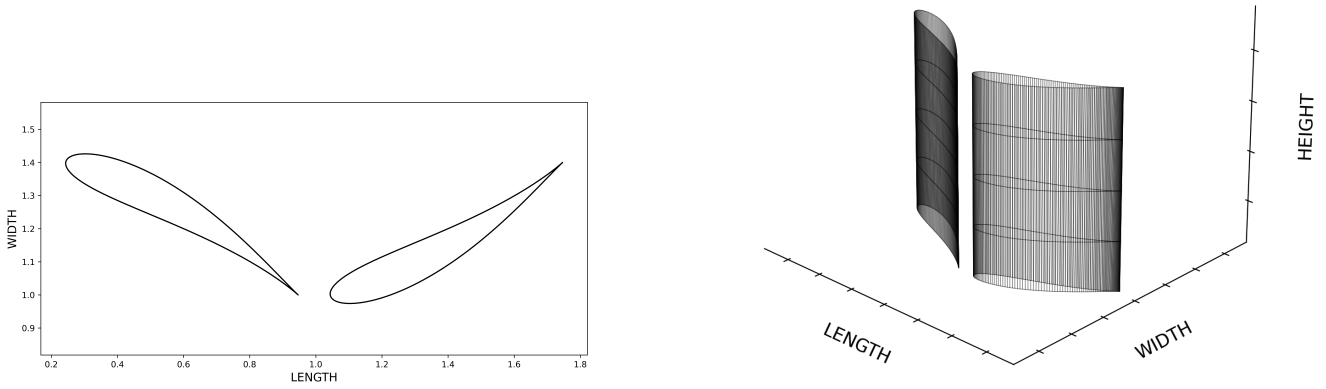
# 3 Object Definition

## 3.1 2D and 3D Airfoils

The object of interest to simulate is a pair of airfoils representing a part of the rotor and the stator. For convenience and ease it was chosen to create the airfoils through the classical Joukowski transformation which is a conformal mapping used in fluid dynamics and aerodynamics to study the flow around airfoils [6]. The transformation is used to convert a simple shape, such as a circle, in a complex shape, like an airfoil, as shown in Figure 2. The Joukowski transformation is given by the following complex function:

$$\zeta = \frac{1}{2}\left(z + \frac{a^2}{z}\right) \tag{1}$$

Here, $\zeta$ represents the complex plane of the transformed space (airfoil), $z$ is the complex plane of the original space (circle), and $a$ is the radius of the circle [7]. Through transformations such as translation, rotation and scaling the airfoils are aligned according to the x-axis and positioned in such a way as to potentially simulate the dynamics of a part of the rotor and stator as represented in Figure 2 for both a 2D and 3D profiles.




(a) 2D Plot

(b) 3D Plot

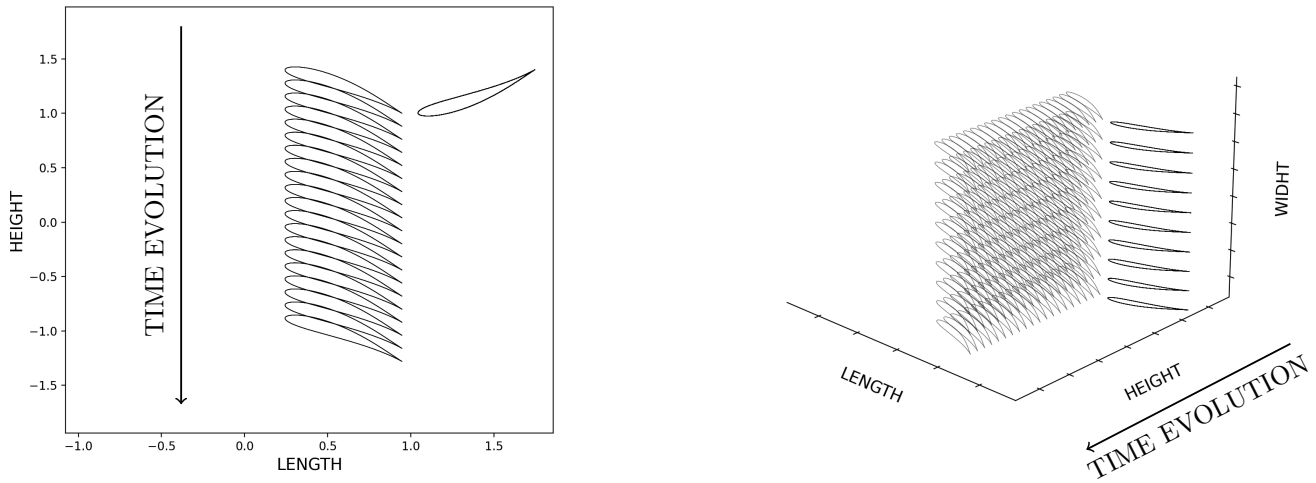
Figure 2: Representation of (a) 2D objects made by a combination of 2 Joukowski airfoil profiles and (b) 3D airfoil objects made by extrusion along z-axis of the two profiles. The 3D airfoils have 5 z-layers each 0.1 thickness.

## 3.2 4D Airfoils

In the context of mathematics and geometry, a 4D object is a concept that extends the idea of a 3 dimensional object by including an additional dimension. A more in-depth concept about 4D objects will be described in Section 5, but essentially, we cannot physically create or visualise a 4D object in its entirety. We can create projections or slices of 4D objects that help us understand their properties and structures. In this case the 4D airfoils are represented as a set of slices of the 3D object over time, as can be seen from Figure 3. They are 3D moving objects but in this context they are considered 4D static objects.

# 4 Poisson's Equation for regular grids

A requirement for building the adaptive mesh refinement algorithm is to solve Poisson's equation in a regular grid. In this section the equation and the best techniques to solve it will be presented.




(a) 2D Plot

(b) 3D Plot

Figure 3: 4D Airfoil visualisations. Figure 3a is a 2D representation of the rotor-stator airfoils extruded in t-direction. The stator is just remaining static for all the time layers. Figure 3b is a 3D representation of the airfoils of each time frame. The airfoils are extruded in z-axis and then extruded in time. For both plots the time axis is represented as a time evolution axis.

### 4.1 Overview

Poisson's equation, in this case is used to obtain the pressure field $\phi$ in N-dimensional grid due to the application of a force and it is described as:

$$\Delta\phi(x, y) = f(x, y) \tag{2}$$

where $\Delta$ is the Laplacian operator of the potential $\phi$ and $f$ is the force term. It can be also described in two-dimensional Cartesian coordinates as:

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = f(x, y) \tag{3}$$

in which we find the scalar potential $\phi$ and the force term $f(x, y)$. In our case the force $f$ is the input data while the scalar potential $\phi$ is the solution. An important thing is that the Poisson's equation is linear in both terms described [5]. This property will be useful in the definition of rotor and stator airfoils loading objects in Section 4.3.

### 4.2 Methods and solutions

The Poisson's equation discretisation techniques may be different and the one used for this case is the Finite Difference Method (FDM) and in particular the Cell-centered finite difference (CCFD) scheme. This discretisation method allows to store the computed values in the center cell of each square in the grid. So each cell of the grid can be represented simply by a value and this facilitates the way in which the data is kept, allowing to perform the calculation faster and saving memory [8][9]. But most importantly, the CCFD scheme allows to incorporate boundary conditions into the domain without any problems. This happens very well when the Neumann Boundary conditions (NBC) are defined ($\frac{\partial\phi}{\partial n} = 0$). They are positioned in the vertices of the grid cells and therefore define the edges of the grid in a precise way [5].

Solving linear partial differential equations (PDEs) such as Poisson's equation becomes like solving a linear algebra problem of the type $Ax = b$ and for this there are numerous iterative methods to solve it

[10]. They also differ in how the algorithm is described. The most popular methods are for example the Jacobi or the Gauss-Seidel iteration methods. To demonstrate it, a regular grid with uniform spacing $dx$ in the x-direction and $dy$ in the y-direction is defined. A discretisation through the cell centered finite difference scheme for the Laplacian of the potential phi is done, which is:

$$\Delta\phi_{(i,j)} \approx \frac{\phi_{(i+1,j)} - 2\phi_{(i,j)} + \phi_{(i-1,j)}}{dx^2} + \frac{\phi_{(i,j+1)} - 2\phi_{(i,j)} + \phi_{(i,j-1)}}{dy^2} \tag{4}$$

where $(i, j)$ are the indices of the grid points in the x and y directions, respectively. Based on the average of the neighboring values and the source term $f(i, j)$, the Jacobi method for example updates the potential phi at each grid point $(i, j)$. The iterated value is the component of the next approximation. This is described as:

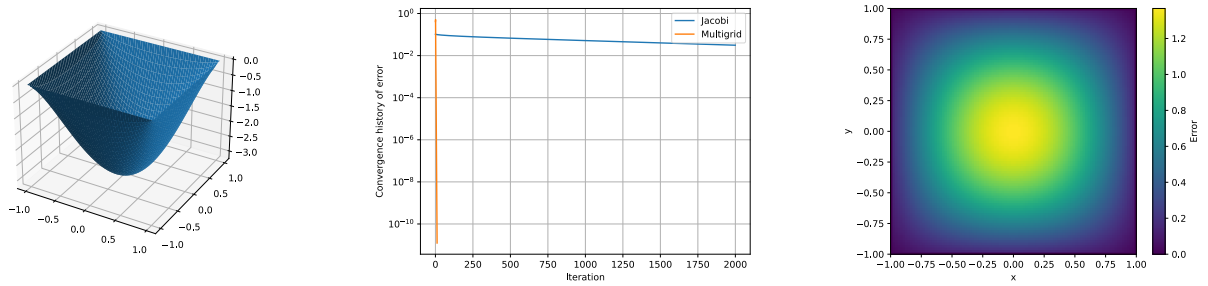
$$\phi_{\text{new}(i,j)} = \frac{\phi_{(i+1,j)} + \phi_{(i-1,j)} + \phi_{(i,j+1)} + \phi_{(i,j-1)}}{4} - \frac{f_{(i,j)} \cdot dx \cdot dy}{4} \tag{5}$$

where $\phi_{\text{new}(i,j)}$ is the updated potential $\phi$ at grid point $(i, j)$, $\phi_{(i\pm1,j)}$ and $\phi_{(i,j\pm1)}$ are the potential $\phi$ at neighboring grid points in the x and y directions, respectively, $f(i, j)$ is the force term and $dx$ and $dy$ are the grid space and x and y direction respectively. The process is similar for the Gauss-Seidel method except that the Gauss-Seidel method is faster because the approximate solution is updated immediately after the new $\phi$ is determined [11]. By applying a uniform force in the grid and defining the boundary conditions we can see the solution of Poisson's equation according to the first 3D graph of the Figure 4a. However, the solution to Figure 4a can be found much more efficiently. Known for its remarkable speed, the Multigrid method is based on the idea of accelerating the convergence of an iterative method (such as the Jacobi method) by solving the problem in a sequence of grids ranging from the finer one (grid of the original problem) up to the coarsest grid (a reduced problem size). By doing this, the multigrid method captures information from different scales of the problem. This is a repetitive process called "V-cycle", in which by applying the iterative method (also called point relaxation) to the finest grid a correction term is found which represents the fine-grid residual for the next one. And this happens until the problem is coarsened enough to be directly solved [12]. Using this method, the solution tends to converge faster as demonstrated in Figure 4b and by the fact that for a grid size of 300 x 300 cells the Jacobi method takes 191.15 seconds while the multigrid method takes 1.57 seconds. Building an algorithm for the multigrid method requires time and great programming techniques. Python offers several pre-built packages to solve these kinds of problems. AMG solver was used for this project. It is a multilevel technique that allows solving large-scale linear systems with optimal or near-optimal efficiency and is very powerful for solving the Poisson's equations for any grid [13].

## 4.3 Poisson's Solver for N-Dimensional airfoils

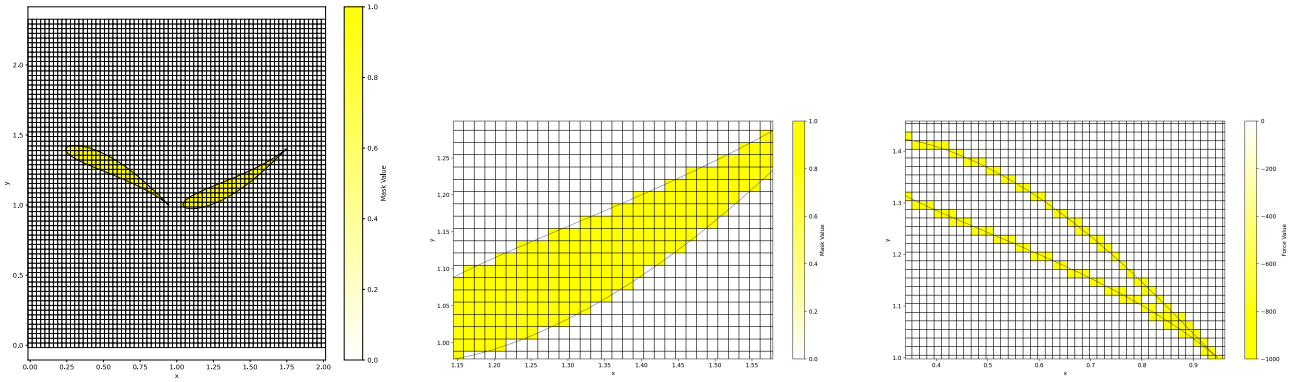
### 4.3.1 Mask definition

Once the discretisation and iteration techniques of the Poisson's equation have been defined, the rotor and stator airfoils that have been created in Section 3 have to be implemented. To do this, boolean masks have been created in 2D, 3D and 4D. Generically, a boolean mask is an array of boolean values (True or False) with the same shape as the data array it is applied to. The True values in the mask indicate the locations that meet a specific criterion, while the False values indicate the locations that do not meet the criterion. In the two airfoils case, each element in the mask corresponds to a point in the solution grid as can be seen in Figure 5a-5b. The steps in creating masks are not long and complicated in Python and essentially require writing the function that defines the shape of the airfoils and through the "path" function create the cells that make up the edges of the object as in Figure 5c. After creating the masks for each airfoil, they are combined together in the grid through a logical operator "np.logical.or" and the True value is the cells that are inside the airfoils [14].




(a) Convergence error (b) Convergence error (c) Error distribution

Figure 4: Figure 4a is a 3D visualisation of the solution of Poisson's equation on a regular grid (300 x 300 cells) with a uniformly distributed force of 100 over the grid domain. The Dirichlet' boundary condition DBC ($\phi$=0) has been applied on the domain boundaries. The equation was calculated using a cell-centered technique, first with the Jacobi method and then with the multigrid method, producing almost the same result. The coarsest level grid is 5 with only 3 unknowns (grid points) in this case and 6 levels in the hierarchy. The deflection shape is caused by the smoothing property of the iterative methods [13][5]. Figure 4b: Convergence history of the error (or residual) for both Jacobi and multigrid solvers. The figure shows how the error reduces over the iterations for each solver. For the Jacobi method, the error history is calculated as the L2 norm of the error (difference between consecutive solutions) at each iteration. For the multigrid method, the error history is the residual (the difference between the right-hand side of the equation and the product of the system matrix and the solution at each iteration). The y-axis is in logarithmic scale, which helps to better visualise the differences in convergence rates. The x-axis represents the number of iterations. It can be seen that the multigrid converges way more faster while the Jacobi even after 2000 iteration has still an error of $10^{-2}$. The Figure 4c represents the spatial distribution of the absolute error between the solutions obtained using the Jacobi and multigrid methods. This visualisation helps to identify areas where the error is higher. The color bar represents the magnitude of the error.




(a) Airfoil masks (b) Airfoil mask zoomed (c) Boundary cells mask zoomed

Figure 5: Representation of Airfoil masks: Figure 5a shows the combined airfoil masks in a grid cell of 100 x 100. The yellow cells are the ones inside the object where a force is applied. Figure 5b is zoomed version previews plot. Figure 5c represents the boundary cells of the objects (in this case the stator airfoil).

### 4.3.2 Force definition

The application of force, as can be seen from Figure 5 is defined within the object. This force itself represents the weight of the object and can be described as:

$$F = m \cdot g \tag{6}$$

where $m$ is the mass and as long it is coherent and constant can be every value like 10g and $g$ is the gravitation acceleration. This means that when the object is placed on the Poisson's solver grid, whatever the dimensionality of the grid is, there will be a force opposite to the weight of the object on the edges

of the object as represented in Figure 6c. Implementing this in Poisson's solver means subjecting the internal cells to a force equal to the weight of the object and at the same time an opposing force on the edges caused by the deformation of the grid. And as mentioned above, given that the equation is linear, multiple forces can combine (superimpose) their individual solutions to obtain the overall solution for the problem [5].

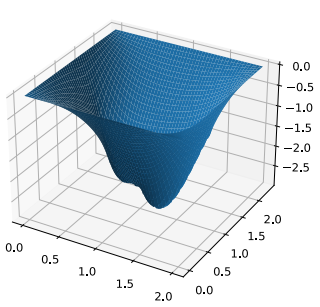
#### 4.3.3 2D-3D and 4D Poissons's solver

Once the objects have been defined, the procedures are the same. The multigrid AMG solver is automatic once the linear equation to be solved is defined. In this case, Equation 3 defined earlier is used for the 2D grid. In the case of 3D or 4D, they are respectively defined below in Equation 7 and 8.

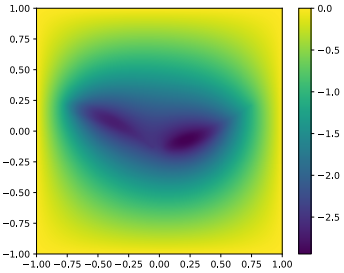
$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = f(x, y, z) \tag{7}$$

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} + \frac{\partial^2\phi}{\partial t^2} = f(x, y, z, t) \tag{8}$$

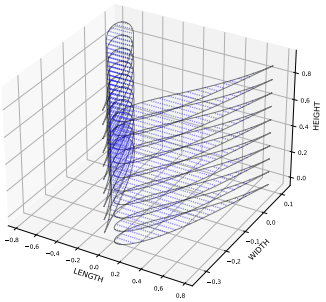
The DBC ($\phi = 0$) is set at the edges of the domain in any grid. The results of the Poisson's equation for a 2D, 3D and 4D grid are shown in Figure 6.



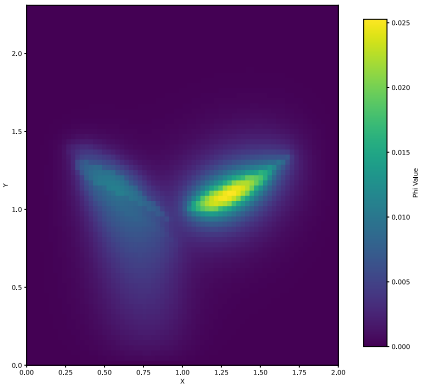
(a) 3D plot



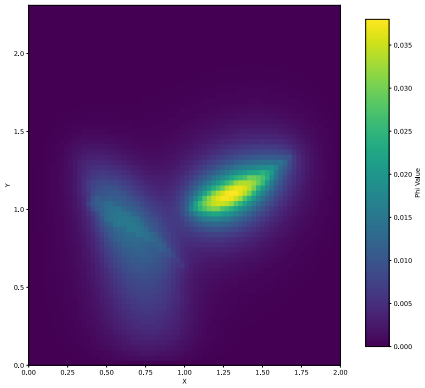
(b) Contour plot



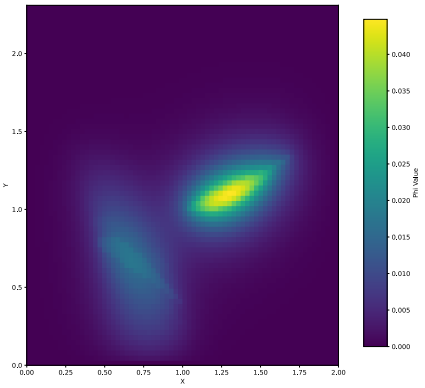

(c) 3D airfoils mask



(d) 4D Poisson's t=1



(e) 4D Poissons's t=3



(f) 4D Poissons's t=5

Figure 6: Figure 6a shows a 3D plot of the Poisson's equation solved for 2 airfoils objects in a 2D grid of 500 x 500. Figure 6b represents a contour plot of the 2D Poisson's equation solved with 6 multigrid levels. Figure 6c shows the mask created to define the 3D airfoils, extruded 10 z-layer of 0.1 each. Since the shape of the airfoils does not change in extrusion, the Poisson's equation is almost the same for every z-layer and thus similar to the 2D one. Figures 6d,6e and 6f represent the 4D Poisson's equation solved for a 70 x 70 x 70 x 20 grid solved in a multi grid. In this case the 3 plots take and analyse the object at z=5 for t=1,3 and 5. It can be clearly seen how the rotor is moving while the w-layer, which is the time dimension, changes. This plots demonstrate and approve the effectiveness of the Poisson's solver algorithm.

# 5 Construction of the N-dimensional mesh

In this section a strategy in the construction of a 4-dimensional mesh will be defined. It follows clear steps starting from the construction of a 2-dimensional mesh based on a lower 1-dimensional mesh. The same methodology is applied for making thus a 3-dimensional and finally 4-dimensional mesh. In Section 3, the rotor and stator airfoils in 4D arise from the movement of the 3D object over time. In that case, in fact, the 4th dimension is understood as something that adds a way to the 3rd dimension to change. Time evolution leads to the 3D object to change. The 4th dimension therefore in that case is a time dimension. It follows one direction, namely that of time. From now on, in the construction of the N-dimensional mesh, time will be considered as a spatial dimension as long as it is perpendicular to the other dimensions.

## 5.1 Method

The innovative method summarised below allows to obtain for each N-dimensional mesh, a patch, a lattice which is replicated. This project reaffirms the new approach, formulated by Siddharth Suresh Kamble in the PhD thesis "*Mesh generation in four dimensions for adaptive refinement in space and time*".

**Step 1:** Construct an even layer consisting of equally spaced nodes and connections between them as in Figure 7. This is the basis of the mesh in 1 dimension.


Figure 7: 1 dimensional grid made by 5 nodes equally spaces and connected between each neighbouring node.

**Step 2:** Build the base of the odd layer, i.e. the extrusion that leads to the second dimension. The extrusion starts from the nodes and goes to the centroids of the connections as shown in Figure 8 with the appropriate connections.

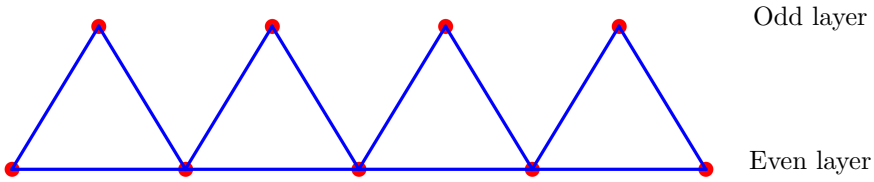



Figure 8: Extrusion of the centroid of each connection in the second dimension, considered as odd layer.

**Step 3:** Vertically extrude the nodes in the second dimension as shown in Figure 9.

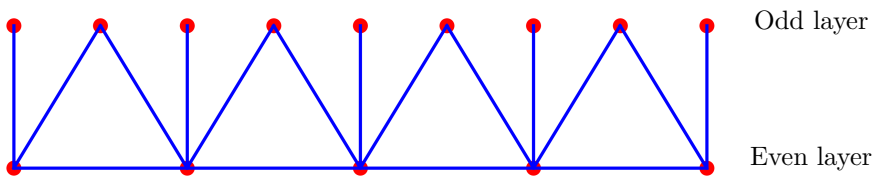



Figure 9: Extrusion of the even layer nodes in the odd layer.

**Step 4:** Connect the centroids to neighboring nodes of the same layer as shown in Figure 10.

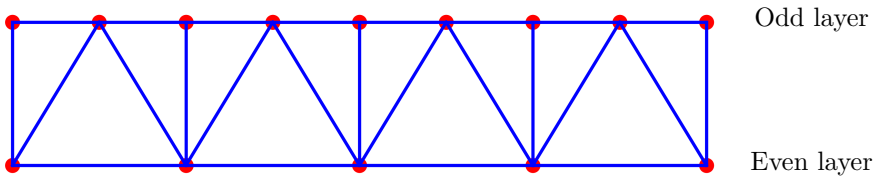



Figure 10: Connection of the centroid nodes with the neighbouring node.

**Step 5:** Mirror the current pattern according to the odd layer as shown in Figure 11. This creates a new even layer on which to start as a base and repeat Steps 1 through 5 again.

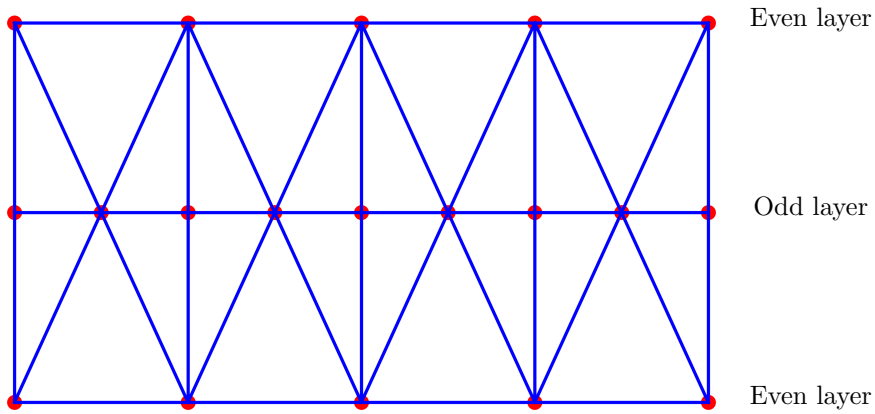
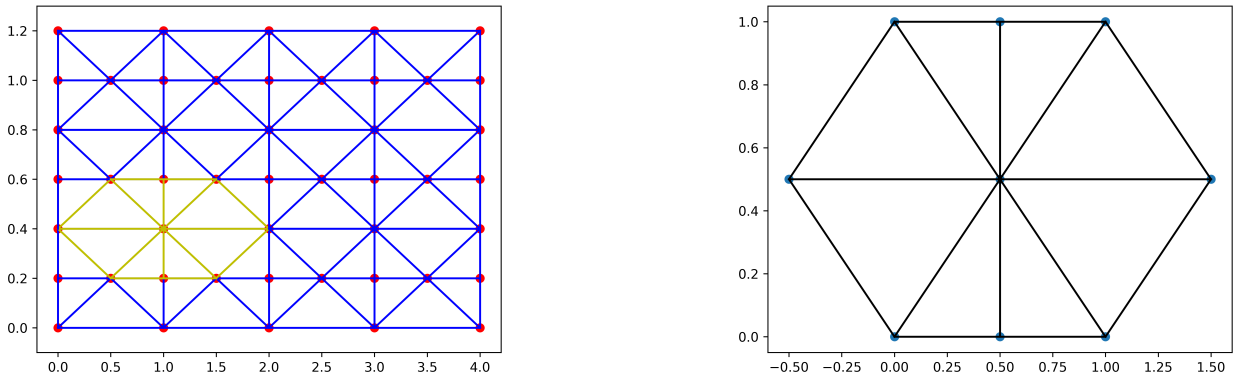



Figure 11: Patter produced by a mirror action about the odd layer.

## 5.2 2D mesh

Repeating this pattern, reveals a lattice of replication. In the 2D case, the lattice is the shape highlighted in yellow in Figure 12. This, made clearer in Figure 12b, constitutes the base, the even layer of the step to start building the 3D pattern.



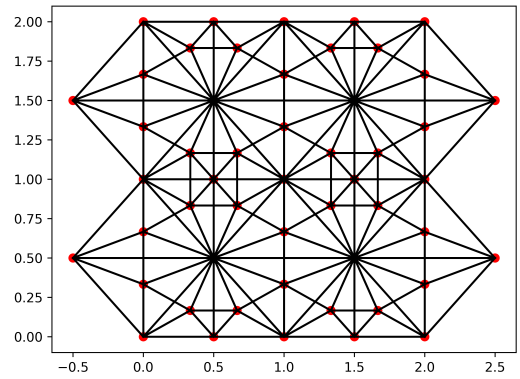

(a) 2D pattern

(b) 2D lattice

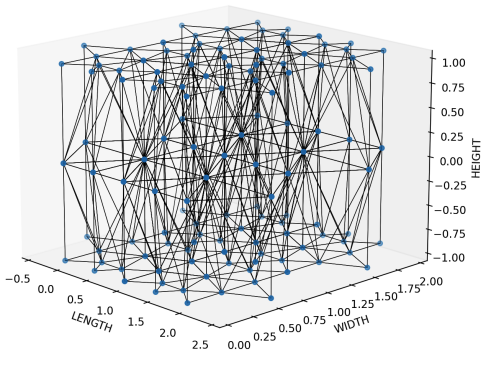
Figure 12: Lattice 2D with a hexagonal shape to be repeated as a pattern. The hexagonal lattice is also used as a base in the creation of a 3D mesh. The 2D mesh will be shown in its entirety in the next section as an example.

### 5.3 3D mesh

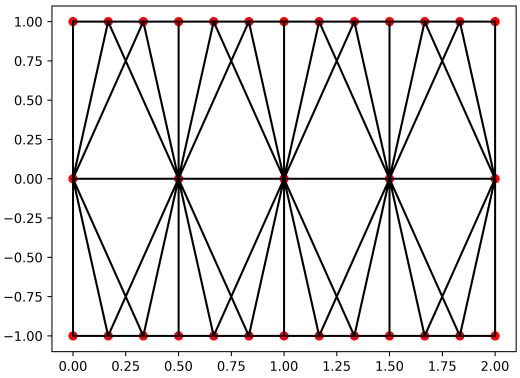
The 3D pattern follows the same steps explained above but the extrusion of the odd layer of Step 2 takes place in the third dimension. A complete visualisation of the 3D mesh can be found in Figure 13, where the mesh is shown from different views.



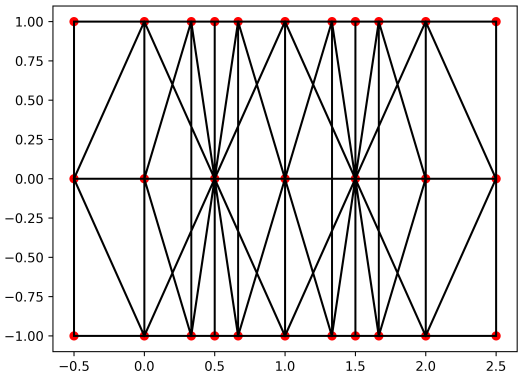

(a) Top view




(b) Orthographic view




(c) 2D pattern



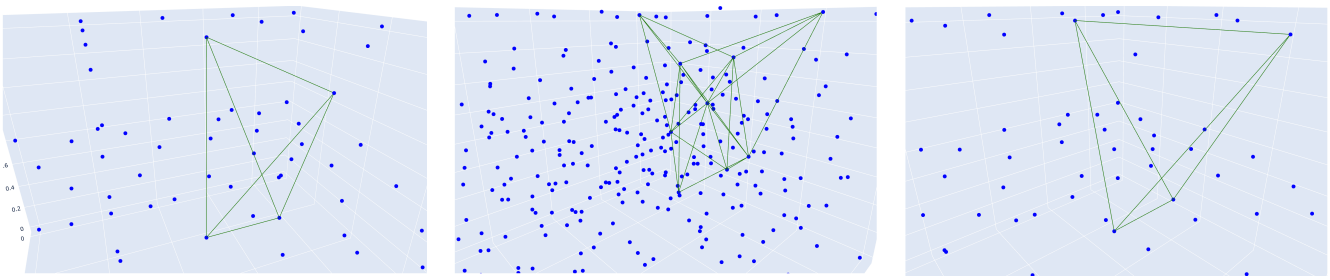

(d) 2D pattern

Figure 13: A 3D mesh made by 2 x 2 x 2 lattices. The "hexagonal" base was brought into the third dimension according to the steps for the realisation of the N-dimensional mesh and then only the repetition of the lattice takes place. The mesh is made up of nodes and connections and produce different patterns including that of tetrahedrons, a geometric figure useful in the creation of the 4-dimensional mesh. The 3D mesh is displayed in different perspectives in order to understand well its topology.

### 5.4 4D mesh

The creation of a 4-dimensional mesh is not a simple thing and requires a lot of knowledge of high-dimensional geometry, and mesh generation techniques such as Delaunay triangulation or octree-based methods [15]. In the case of this project, the 4D mesh proposed by the PhD student was explicitly used. The mesh in question is made up of nodes of 4 coordinates which constitute simplices in 4 dimensions. They are also called pentachoron and they consist in 5 vertices, 10 edges and 10 faces [16]. Each node has its own recognition ID index and each simplex also has an ID. The mesh data has been collected and different visualisation techniques have been used in order to understand its topology as can be seen from Figures 14 and 15 (Page 15). The 4D mesh is composed of 471 nodes and 3552 simplices. While analysing the mesh, it has been noted that there are so-called "disconnected nodes" that are located exactly at the

edges of the second 4th dimension. There are exactly 24 disconnected node. The reason for this is not explained but these nodes could serve as patch connections in case of enlargement of the mesh. In any case, attempts to connect these nodes manually has been made, bringing the mesh to have a total of 2676 simplices but it is only a hypothesis and needs validation so it will not be used in the body fitting mesh algorithm.



(a) w = 0.0 (b) w = 0.816497 (c) w = 1.632993

Figure 14: The following visualisation is built according to a Python algorithm that takes the 4D mesh and separates it into multiple 3D spaces based on unique fourth coordinate values. In fact, for each of them a 3D scatter is created which shows the nodes found in that 4-dimensional "layer" and the connections between them. This technique is a sort of "sliced 3D representation" of the mesh. In this case, in order to not make the visualisation difficult due to the many connections between the nodes, the node with ID 325 is chosen and its connections are shown. The node was explicitly chosen to explain how the central 4D layer acts as a pivot between the other two. It follows the basics of Step 5 in Section 5.1 where connections are mirrored according to the odd layer. The 3D scatter at the 4th coordinate of w = 0.0816497 is exactly the odd layer of this mesh. The various connections that lead to the creation of numerous tetrahedrons can be seen. In this algorithm, selecting a node, produces the connections that this node has in all 4-dimensional spaces. It is a very curious and exciting visualisation that could be used to better refine or analyse the mesh.

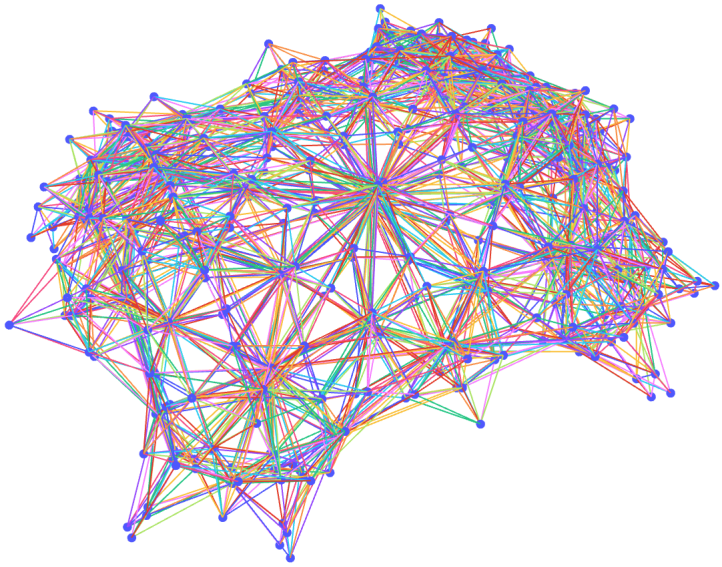


Figure 15: This "4D visualisation" is represented as a set of nodes and connections and the layout is computed using an algorithm that simulates a physical system. The purpose of this is purely visual. The visualisation technique is called 3D force-directed graph drawing, also known as "spring-layout". This technique tries to minimise the total energy of the system resulting in a layout where connected nodes are closer together while disconnected ones are pushed farther [17][18]. In this case, the 4D mesh is projected into a 3D space.

# 6 Body fitting of 4-dimensional mesh

Once the 4-dimensional mesh has been defined and the Python algorithm has been built so that it stores the data of each node of the mesh and of each simplex, we proceed linearly with the next steps of the Flow chart. At this point of the project, in fact, there are several pieces of information to keep, such as the values of the pressure field calculated by the Poisson equation for each cell of the 4-dimensional grid (Section 4) and the information of the mentioned 4D mesh. This data is essential and must be able to be stored for example in a list or dictionaries.

## 6.1 Control volume around each node

The node has a concept of existence in an adaptive mesh and is not able to move freely. When forces take over, the basis of the movement of the node is a concept called Control Volume (CV) [19]. Creating a control volume around each mesh node is essential to also retain data such as density and pressure. The steps used in this project for the identification and calculation of the CV are described below.

**Step:1** Determine centroid, median edges and face centroids in each 4D-simplex;

**Step:2** Create a loop for each node that takes every centroid, median edges and face centroid for each simplex;

**Step:3** Through a combination, determine all median dual simplices around each node;

**Step:4** Calculate the volume of each median dual simplex with determinant matrix;

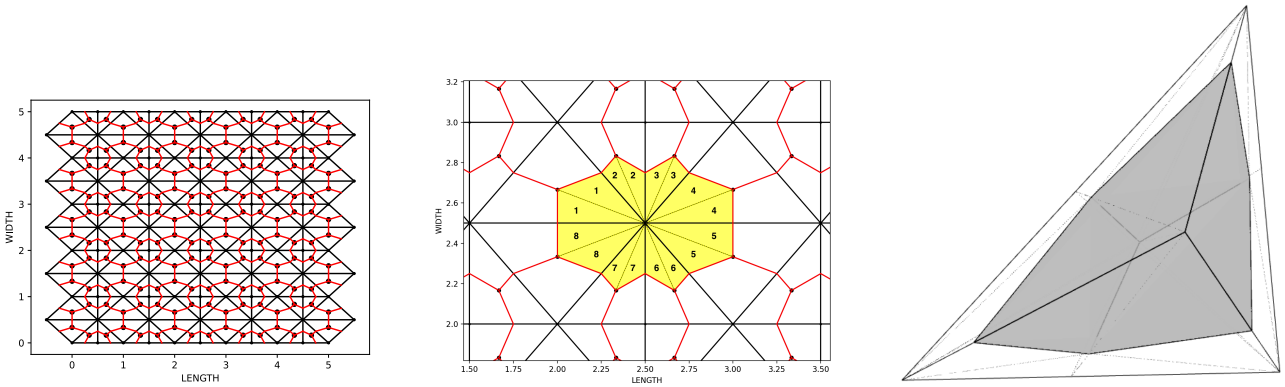
**Step:5** Sum all median dual simplices volumes around each node to get the CV.

### 6.1.1 Median dual 4D simplices

In the 4D mesh, each node is part of a series of simplices that surround it. These simplices form a cell around them. In the 2 dimensional case, this can can be seen as the triangles that form the hexagon in Figure 12b. The same reasoning occurs in the 3D mesh in which each node is surrounded by tetrahedrons. The CV in this case is the median dual cell of these triangles or tetrahedrons around. The median dual cell is a concept used in mesh generation, CFD and finite volume method (FVM). In the 2D mesh it consists in building a cell by connecting the centroids of the triangles with the neighboring ones as can be seen in Figure 16a. Thus any triangle, tetrahedron or pentachoron can be decomposed into sections of median dual N-simplices. This characteristic is governed by the following Equation 9:

$$MD_T^N = \alpha \cdot (T_f^N - T_f^{N-1}) \cdot N_V^N \tag{9}$$

where $MD_T^N$ is the number of median dual N-simplices for each vertex of an N-simplex, $T_f$ is the number of triangular faces associated with the N-simplex, $N_V$ is the number of vertices of a N-simplex and $\alpha$ is a multiplicator factor of 2 that is the number of median edges between a face centroid [5]. Again, in Figure 16b, a 2D triangle has 2 median dual triangles. For a 4D simplex, which has 5 vertices and 10 faces, this means that it has a total of 60 median dual 4D simplices and thus 12 per vertex. The sum of all median dual simplices around a node forms the CV. Theoretically, a median dual simplex in 4D can be formed by the node, centroid of the simplex and a combination of median edges and face centroids. An algorithm has therefore been built in order to identify the centroid, median edges and face centroids of each simplex and then through the itertools.combinations function to find the median dual simplices around each node.




(a) 2D median dual cell pattern (b) 2D median dual simplices (c) 3D median dual polytope

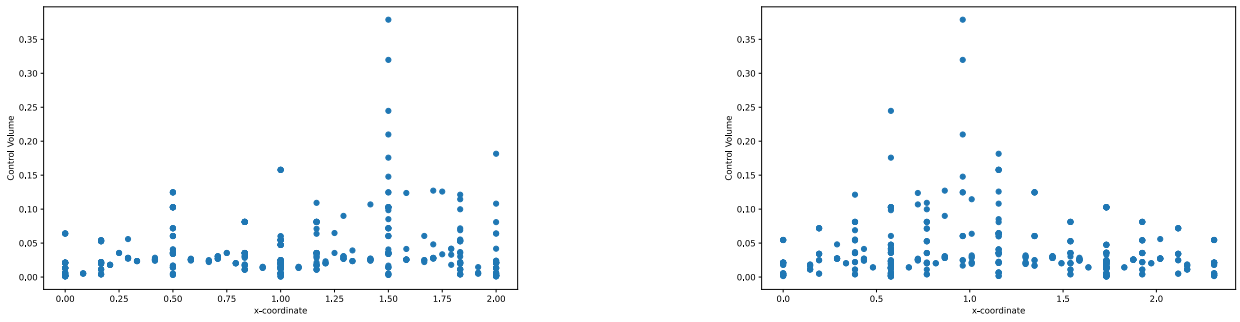
Figure 16: Figure 16a shows the representation in red of the median dual cells around each node of the 2D pattern. Each cell represents the CV of each node. next to it, Figure 16b represents the decomposition of the CV in median dual simplices in order to calculate the areas. Here it can be seen easily thank to the numbers, how each vertex of a triangle has 2 median dual simplices, making together a 2D polytope of 4 vertices. Figure 16c shows teh topology of this sector in 2D.

### 6.1.2 Volume of a simplex

The determinant matrix method was used to calculate the volume of the 4D simples. The following determinant in Equation 10 indeed gives the volume of a 4D simplex:

$$V = \pm \frac{1}{N!} \begin{bmatrix} a_x & a_y & a_z & a_w & 1 \\ b_x & b_y & b_z & b_w & 1 \\ c_x & c_y & c_z & c_w & 1 \\ d_x & d_y & d_z & d_w & 1 \\ e_x & e_y & e_z & e_w & 1 \end{bmatrix} \tag{10}$$

where $a$, $b$, $c$, $d$, $e$ are the vertices of each simplex and $N$ is the number of dimensions [5]. The values found for the control volumes promised a correct approach to the problem, as can be seen from Figure 17. The nodes in the middle of the mesh tend in fact to have more connections between other nodes and this makes the nodes construct a bigger CV.




(a) CV along x-direction (b) CV along y-direction

Figure 17: Both figure 17a and 17b represent the values of CV of then nodes of the 4D mesh along x and y direction. It can be noticed how the volumes increase in the middle of the mesh and decrease when they are in the boundaries where there are not many simplices connected between each other.

### 6.2 Gas law pressure at nodes

In the constructed N-dimensional mash, each node is considered to be a particle with constant mass. In fact, when simulations of this kind are performed, such as computational fluid dynamics (CFD), particle-based methods (e.g., Smoothed Particle Hydrodynamics) it is common to assign a mass to the nodes or to the control volume itself. The assignment of the mass can be chosen according to the problem to be solved and in this project, in which rotor and the stator airfoils needs to be simulated, each node could represent the particle of a compressible gas [20]. The value doesn't really matter right now for it to be constant throughout the project. It is therefore decided that the mass of each node m = 1. This immediately leads to calculate the pressure at each node according to the gas law pressure, described by the following Equation 11:

$$P_g = \left(\frac{m}{V}\right)^{\gamma} \tag{11}$$

where $P_g$ is the pressure at the node, $m$ is the mass of the node, $V$ is the control volume (CV) and $\gamma$ is the ratio of specific heats which it is equal to 1.4. Obtaining the pressure at each node of the 4D mesh allows to move on to the next step of the body fitting mesh algorithm: calculate the pseudo-pressure of Poisson's equation and gas law pressure at the control volume faces and combine them.

### 6.3 Control volume faces

As illustrated in Figure 16b, which simplifies the problem into two dimensions, the control volumes (CVs) at each node are in contact with each other. When a force is applied, the control volume facilitates movement of the node. This movement also affects the control volume, but its volume remains constant despite changes in its shape or topology. Therefore, interactions between the faces of adjacent control volumes can cause changes in the mesh over time. Consequently, it is necessary to identify the faces that are shared by adjacent nodes. Calculating the coordinates of the centroids of these faces is essential for determining the force that allows the boundaries of the control volume, and thus the node itself, to move.

The procedures outlined in Section 6.1 enable the calculation of the volumes of the median dual cells (control volumes) but do not definitively identify the coordinates that define these volumes. While a decomposition of each simplex into 60 parts allows volume calculation, obtaining the faces of the control volumes involves identifying common faces between the median dual cells of each simplex. This is illustrated in Figure 16b, which shows a 3D simplification of a tetrahedron. The following steps describe this process in detail:

**Step:1** Decompose the 4-simplex into median dual cells (polytopes) for each vertex. This is done by creating a polytope for each vertex that consists of the vertex itself, the centroids of the faces adjacent to the vertex, the midpoints of the edges connected to the vertex, and the centroid of the 4D simplex.;

**Step:2** After decomposing the 4-simplex into median dual cells, find the common 3D faces between the polytopes. This is achieved by computing the intersections of the vertex sets of each pair of polytopes. If the intersection contains at least 4 vertices, it is considered a common 3D face shared by the two polytopes.

Each face of the control volume would therefore be a polyhedral structure of a smaller dimension, i.e. 3D. And the volume control face area is the surface of the 3D polyhedron. This concept is complex to understand and visualise but again from Figure 16b, if connected by other tetrahedrons, the dark solid would create a 3-dimensional polyhedron which for the 3D mesh would be a control volume but for the 4D mesh it would only be the face of a control volume .

### 6.4 Pressure interpolation

To estimate the pressure value at the center of the face of each control volume (3D polyhedron) obtained previously, a linear interpolation method is used which in this multi-dimensional mesh is called tetra-linear interpolation. To do this, it is initially necessary to create a function that takes the values of the pseudo-pressure, the grid coordinates (x, y, z, w), and the target point (the centroid of each control volume face) as input arguments. After that, for each coordinate of the face centroid the points of the Poisson s grid that surround it are found. The face centroid is thus surrounded by a hypercube (a 4D cube which has 16 vertices) and through a combination of the nodes the interpolation weights based on the distances between the face centroid and the surrounding grid points are computed. Finally, the algorithm accumulates the weighted sum of the values of phi at these surrounding grid points to compute the interpolated value. The method is the same for every dimension. The following Equation 12 tries to explain this more clearly:

$$\phi_f = \sum_{(i,j,k,l)\in\{0,1\}^4} (1 - t_{i_x}) t_{i_y} t_{i_z} t_{i_w} \phi(i,j,k,l) \tag{12}$$

where $\phi_f$ is the interpolated $\phi$ value at the face, $\phi(i,j,k,l)$ is the value at the vertex with indices $(i,j,k,l)$ and $t_{i_x}, t_{i_y}, t_{i_z}, t_{i_w}$ are the interpolation factors for each dimension [21][5]. The same process is performed to interpolate the gas law pressure from the nodes at each interface. The resultant pressure at each side of the control volume is a sum of the pseudo-pressure and the gas law pressure.

### 6.5 Forces, node velocities and new positions

In this Section the last steps of the Lagrangian mesh deformation process are discussed. The main purpose is to calculate the forces at the centroids and update the node positions considering time step constraints. The force is calculated as follows:

$$F_f = P_f \cdot A \tag{13}$$

where $F_f$ is the force at the faces of the CV, $P_f$ is the pressure interface calculated by the sum of the interpolated values of pseudo-pressure and gas law pressure and $A$ is the area of the faces of the CV which are the surface areas of the tetrahedrons. To calculate the velocities and thus the positions of the nodes, a simplification of the Navier-Stokes equation, i.e. the Euler equation (Equation ), is used:

$$\rho \frac{\partial \mathbf{U}}{\partial t} = -\nabla P \tag{14}$$

$\rho$ is the fluid density, $\frac{\partial \mathbf{U}}{\partial t}$ is the rate of change of fluid velocity over time and $\nabla P$ is the change of pressure with respect to position [5]. The Euler equation relates the time rate of change of fluid velocity to the pressure forces acting on the fluid. This equation counts the absence of viscous and body forces. To solve this equation, integration over the control volume is computed. Applying the Gauss Divergence theorem and Euler's first-order integration scheme, the volume integrals are transformed into equivalent surface integrals and with a summation over the control volume faces, and several simplifications, the updated velocities are found. All this can be described by the following Equation 15:

$$\mathbf{U}_i = F_f \cdot \frac{\partial t}{m_i} \tag{15}$$

where $\mathbf{U}_i$ is the new velocity of each node at each time step $\partial t$. After calculating the velocities, the node positions are updated using the following Equation 16:

$$x_i^{n+1} = x_i + dt \cdot \mathbf{U}_i \tag{16}$$

where $x_i^{n+1}$ is the new position of the node at the next time-step, $x_i$ is the old position, $dt$ is the time step and $\mathbf{U}_i$ is the velocity of the node.

# 7 Conclusion and Results

A 2D body fitting mesh algorithm is finally implemented in Python. The code is implemented for just one time step. This means that the simulation is iterated until the parameters converge to a stable solution. Despite the challenges, the simulation has produced satisfactory results, with the pressures, areas, and node displacements aligning well with the physical system under consideration. The forces, being proportional to the control volume areas, and the locations with higher phi values cause the nodes to move and increase in density. Unfortunately, due to the high sensitivity of node movement at time steps larger than 0.125, these effects can not be clearly observed in Figure 18. The 2D mesh grid is too big to notice the adaptivity of the mesh. This is due to the limitations of the computational instruments used.

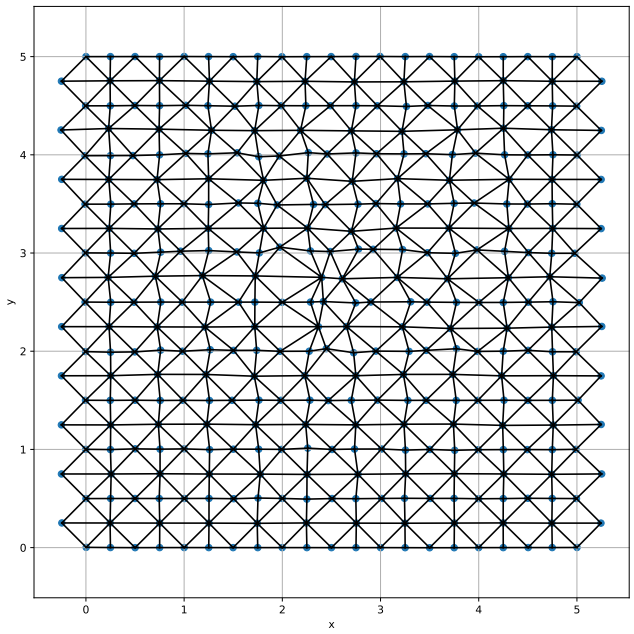



Figure 18: 2D body fitting mesh solved for 2 airfoils with a load of 100N. The Poisson's equation has been solved with multigrid for a grid size of 120 x 120 cells. The velocity, and new node positions are updated for a time step of 0.0125.

The adaptation of the algorithm for the 4D mesh was also successfully completed, albeit with very low resultant values of the node positions which are at around $10^{-18}$. This is attributed to numerical issues and the way floating-point numbers are represented in computers. Scaling the mesh has provided a partial solution by stabilising the results. The current challenge lies in locating the volumes of the control volume faces. These 3-dimensional polytopes are complex to define in a 4-dimensional space. Extensive research has been undertaken into methods for calculating the volumes of simplices, such as the Cayley-Menger method, but this work is ongoing. With a deeper understanding of the higher dimensional geometry and more time the body fitting mesh algorithm for a 4D mesh can be completely solved. The intricacies of the 4D meshing field and the challenges associated with implementing the algorithms and methodologies from the referenced PhD thesis have resulted in a wealth of knowledge and experience. This project has not only contributed to personal growth, but also holds the potential for significant contributions to the research field.

## Acknowledgements

I would like to thank Dr Andrew Lawrie for his supervision.

## References

[1] Y. D. Ram Kumar Raman and J. Raghuwanshi, "A review on applications of computational fluid dynamics," *International Journal of LNCT*, vol. 2, pp. 137–143.

[2] D. A. L. Supervisor, "Personal communication," 2023. Personal IRP Introduction.

[3] A. B. G. Bachler, H. Schiffermiiller, "A parallel fully implicit sliding mesh method for industrial cfd applications," *AVL List GmbH, Advanced Simulation Technologies Hans-List-Platz 1, A-8020 Graz, Austria*, pp. 501–508, 2001.

[4] J. Donea, A. Huerta, J.-P. Ponthot, and A. Rodríguez-Ferran, "Arbitrary lagrangian–eulerian methods," pp. 413–437, 2004.

[5] S. S. Kamble, *Mesh generation in four dimensions for adaptive refinement in space and time.* PhD thesis, 12 2022.

[6] M. F. Carla Cruz and H. Malonek, "3d mappings by generalized joukowski transformations," *Computational Science and Its Applications - ICCSA 2011*, vol. Part III, pp. 358–373, 2011.

[7] R. D. A. H.R. Malonek, "A note on a generalized joukowski transformation," *Applied Mathematics Letters*, vol. 23, p. 1174–1178, 2010.

[8] P. Wesseling, *NASA-CR-195045 Introduction to multigrid methods.* NASA Langley Research Center, Hampton, VA 23681-0001: ICASE, February 1995.

[9] P.-H. Maire, R. Abgrall, J. Breil, and J. Ovadia, "A cell-centered lagrangian scheme for two-dimensional compressible flow problems," *SIAM Journal on Scientific Computing*, vol. 29, pp. 1781–1824, 2007.

[10] L. N. L. N. Trefethen, *Finite Difference and Spectral Methods for Ordinary and Partial Differential Equations / Lloyd N. Trefethen.* Cornell University Dept. of Computer Science and Center for Applied Mathematics: Ithaca, N.Y, 1996.

[11] Y. Saad, *Iterative Methods for Sparse Linear Systems.* Other Titles in Applied Mathematics, SIAM, second ed., 2003.

[12] P. C. Hans Johansen, "A cartesian grid embedded boundary method for poisson's equation on irregular domains," *Journal of Computational Physics*, vol. 147, pp. 60–85, 1998.

[13] N. Bell, L. N. Olson, and J. Schroder, "PyAMG: Algebraic multigrid solvers in python," *Journal of Open Source Software*, vol. 7, no. 72, p. 4142, 2022.

[14] C. Bauckhage, "Numpy / scipy recipes for image processing: Creating fractal images," *B-IT, University of Bonn, Germany*, pp. 1–6.

[15] N. P. W. Joe F. Thompson, Bharat K. Soni, *Handbook of grid generation.* Boca Raton London new York Washington, D.C.: CRC Press, 1999.

[16] P. M. Howard Cheng, "Prescribing curvature on triangulated 3-manifolds," *The univeristy of Arizona*, p. 1.

[17] S. C. Teja and P. K. Yemula, "Power network layout generation using force directed graph technique," in *2014 Eighteenth National Power Systems Conference (NPSC)*, pp. 1–6, 2014.

[18] T. M. J. FRUCHTERMAN and E. M. REINGOLD, "Graph drawing by force-directed placement," *SOFTWARE-PRACTICE AND EXPERIENCE*, vol. 21, pp. 1129–1164, 1991.

[19] I. Sadrehaghighi, *Unstructured Meshing for CFD.* ANNAPOLIS, MD.

[20] J. J. Monaghan, "Smoothed particle hydrodynamics," *Astron. Astrophys, Access provided by University of Bristol on 03/04/23*, pp. 0–572, 1992.

[21] F. Lekien and J. Marsden, "Tricubic interpolation in three dimensions," *International Journal for Numerical Methods in Engineering*, vol. 63-2, pp. 455–471, 2005.